FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Yavaa

Supporting Data Workflows from Discovery to Visualization

Dissertation
zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt dem Rat der Fakultät für Mathematik und
Informatik der Friedrich-Schiller-Universität Jena

von
SIRKO SCHINDLER
geboren am 1983-05-04 in Ilmenau, Germany

Gutachter:

1. **Prof. Dr. Birgitta König-Ries**
   Friedrich-Schiller-Universität Jena, 07743 Jena, Germany

2. **Prof. Dr. Paul Groth**
   Universiteit van Amsterdam, 1012 WX Amsterdam, Netherlands

3. **Prof. Dr. Axel Polleres**
   Wirtschaftsuniversität Wien, 1020 Wien, Austria

Tag der öffentlichen Verteidigung: 06. September 2022, Jena

Dedication

To those that paved the way
and
those that will follow after.

Zusammenfassung

Über die letzten Jahre war zu beobachten, dass immer mehr Datensilos aufgebrochen werden, um die darin enthaltenen Daten der eigenen Organisation oder sogar der breiten Öffentlichkeit zur Verfügung zu stellen. Dieser Trend ist dabei nicht auf einzelne, progressive Bereiche beschränkt, sondern erfährt Unterstützung aus einer breiten Masse an Beteiligten. Bei diesen setzt sich zunehmend die Erkenntnis über die Vorteile durch, die sich aus der Bereitstellung und Integration Daten verschiedenster Quellen ergeben: In der Wissenschaft werden Rohdaten als Teil regulärer Artikel oder sogar als eigenständige Publikationen veröffentlicht. Dadurch können Arbeiten verifiziert werden und es wird anderen möglich diese weiterzuführen. Regierungen erlassen Gesetze, die vormals interne Datenschätze öffentlich zugänglich machen. Dies soll sowohl eigene Transparenz fördern als auch den Boden für neue Geschäftsideen bereiten. Selbst Unternehmen teilen strukturierte Daten über ihre Angebote und Produkte, um deren Verbreitung zu fördern und so letztlich den eigenen Gewinn zu steigern.

Bevor Nutzer diesen Reichtum an Informationen anzapfen können, gilt es für sie aber noch einige Herausforderungen zu bewältigen: Daten sind oft nur in Form schier endloser Tabellen verfügbar, die so für die meisten Menschen kaum handhabbar sind. Zwar kann Informationsvisualisierung hier Abhilfe schaffen, bestehende Systeme sind aber häufig auf eine geringe Zahl von Visualisierungen beschränkt und lassen Nutzer bei der Auswahl und Nutzung derer weitgehend allein. Ähnliches gilt für die Bearbeitung der Daten, wo oft nur sehr wenige Möglichkeiten angeboten werden, um Datensätze an die eigenen Bedürfnisse anzupassen. Hilfestellungen, um gängige Fehlerquellen zu vermeiden, sind ebenfalls spärlich gesät. Die Situation wird weiter erschwert, sollten die gewünschten Daten über mehrere Anbieter verteilt sein. Erst kürzlich wurden Werkzeuge verfügbar, die sinnvoll auch über die Grenzen einzelner Anbieter hinweg nach Daten suchen lassen. Einfache Wege diese Datensätze auch miteinander zu verknüpfen fehlen allerdings weiterhin. Selbst wenn schlussendlich alle diese Hürden überwunden wurden, fehlen den Ergebnissen meist Informationen zu Herkunft und Bearbeitung. Ohne diese werden aber selbst ansonsten überzeugende Visualisierungen schnell in Zweifel gezogen.

Die Voraussetzungen für lebendigen Austausch und Nutzung von offenen Daten sind also vorhanden. Die Zugangshürden bleiben aber gerade für Nutzer ohne einen entsprechenden Hintergrund unverhältnismäßig hoch. Das Ziel dieser Arbeit ist es diese Hemmnisse abzubauen. Durch geeignete Werkzeuge und Hilfestellungen soll das Maß an notwendigen Vorkenntnissen und Fähigkeiten soweit reduziert werden, dass einer breiteren Öffentlichkeit ein einfacher Zugang zu bereits verfügbaren Daten ermöglicht wird. Die Arbeit umfasst dabei den gesamten Prozess beginnend bei der Identifikation passender Datensätze, über notwendige Anpassungen, bis hin zur Erstellung geeigneter Visualisierungen. Die wesentlichen Beiträge können dabei wie folgt zusammengefasst werden:

- *Semantisches Metadatenmodell zur Beschreibung tabellarischer Daten.* Bestehende Ontologien zu Metadaten tabellarischer Datensätze wurden erweitert, um eine Beschreibung der Primärdaten zu ermöglichen. Dadurch können einzelne Spalten bspw. mit einem Wertebereich oder einem semantischen Konzept näher beschrieben werden. Weiterhin lassen sich, so notwendig, Informationen zu verwendeten Codelisten oder Maßeinheiten für eine korrekte Interpretation der Inhalte hinterlegen.

- *Kombination von Datensätzen.* Neben herkömmlichen Stichwortsuchen lassen sich Anfragen auch als strukturelle Beschreibung des gewünschten Datensatzes formulieren. Ergebnisse breschränken sich hierbei nicht auf einzelne Datensätze, sondern umfassen ggf. auch die Kombination mehrerer Datensätze. Dabei enthalten sie nicht nur die benötigten Datensätze sondern auch Anweisungen, wie diese durch Verkettungen von Union-, Join-, Filter- und Aggregationsschritten umzusetzen sind.

- *Konsistente Operationen.* Während der Ausführung von Operationen, insbesondere direkt von Nutzern formulierter Anweisungen, wird die Konsistenz der Ergebnisse sicher gestellt. Ein Schwerpunkt liegt dabei auf Maßeinheiten, die auch ohne weitere Nutzerinteraktionen stets korrekt angewendet werden, was u.U. auch automatische Umrechnungen mit einschließt. Sollten mehrere Umrechnungen in einer Anweisung nötig werden, so wird diese im Hinblick auf eine Minimierung der notwendigen Umrechnungen umstrukturiert. Dies beschleunigt die Ausführung und erhöht die numerische Stabilität der Ergebnisse.

- *Empfehlung von Visualisierungen.* Auf Basis des aktuellen Datensatzes werden geeignete Visualisierungen vorgeschlagen. Selbige sind anhand ihrer Anforderungen an zugrunde liegende Datensätze beschrieben, was u.a. die Anzahl der Spalten bzw. deren Datentyp und Rolle einschließt. Statt statischer Grenzwerte werden hierbei Funktionen genutzt, die den Übergang zwischen erlaubten und nicht unterstützen Werten beschreiben. Falls keine Visualisierung für den gesamten Datensatz verfügbar sein sollte, werden auch Empfehlungen gemacht, die Veränderungen am Datensatz selbst enthalten.

- *Sammeln von Provenienzinformationen.* Alle Schritte, die zur Erstellung eines Datensatzes oder einer Visualisierung beitragen, werden automatisch erfasst. Dies ermöglicht es einerseits die Herkunft eines spezifischen Ergebnisses zu dokumentieren, um dessen Glaubwürdigkeit zu steigern. Andererseits lässt sich damit der entsprechende Prozess auch auf aktualisierten Daten erneut ausführen. Das zugrunde liegende Modell sammelt Informationen auf Ebene einzelner Spalten, um so eine Balance zwischen Kompaktheit und Ausdrucksmächtigkeit zu finden.

Die Machbarkeit der vorgeschlagenen Konzepte und Algorithmen wurde anhand einer prototypischen Umsetzung, Yavaa, demonstriert. Diese stellt eine einheitliche Oberfläche zur Verfügung, die Nutzer unabhängig ihrer Vorkenntnisse auf dem Weg vom Finden passender Datensätze bis zur Erstellung passender Visualisierungen begleitet. Auf Basis der von Eurostat [web1] angebotenen Datensätzen wurde daraufhin eine Nutzerstudie durchgeführt. Die Ergebnisse belegen den Mehrwerts eines solchen integrierten Systems. Im Vergleich zu anderen Systemen, konnte hier nicht nur die Nutzererfahrung sondern auch die Qualität der erstellten Ergebnisse verbessert werden. Die Dissertation schließt mit einer Zusammenfassung und dem Ausblick auf mögliche Fortführungen der hier vorgestellten Konzepte.

Abstract

Recent years have witnessed an increasing number of data silos being opened up both within organizations and to the general public. This trend is not limited to some progressive niches, but a great variety of actors embraces the chances arising from sharing and integrating data: Scientists publish their raw data as supplements to articles or even standalone artifacts to enable others to verify and extend their work. Governments pass laws to open up formerly protected data treasures to improve accountability and transparency as well as to enable new business ideas based on this public good. Even companies share structured information about their products and services to advertise their use and thus increase revenue.

Exploiting this wealth of information holds many challenges for users, though. Oftentimes data is provided as tables whose sheer endless rows of daunting numbers are barely accessible to most humans. Information Visualization can mitigate this gap. However, offered visualization options are generally very limited and next to no support is given in applying any of them. The same holds true for data wrangling. Only very few options to adjust the data to the current needs and barely any protection are in place to prevent even the most obvious mistakes. When it comes to data from multiple providers, the situation gets even bleaker. Only very recently tools became available to search for datasets across institutional borders reasonably. Easy-to-use ways to combine these datasets are still missing, though. Finally, even if all obstacles can be overcome, results generally lack proper documentation of their provenance. So even the most compelling visualizations can be called into question when their coming about remains unclear.

As outlined in the previous paragraphs, the foundations for a vivid exchange and exploitation of open data are set, but the barrier of entry remains relatively high, especially for non-expert users. This thesis aims to lower that barrier by providing tools and assistance, reducing the amount of prior experience and skills required. It covers the whole workflow ranging from identifying proper datasets, over possible transformations, up until the export of the result in the form of suitable visualizations. The main contributions can be summarized as follows:

- *Semantic metadata description for tabular data.* Existing ontologies for metadata of (not only) tabular datasets have been extended to account for details on the primary data included. In particular, this allows augmenting individual columns, among other properties, with a concept to describe their contents and a range of values. Further if applicable, a link to the corresponding codelist for abbreviations or the unit of measurement used is attached to interpret the provided values correctly.

- *Dataset combinations.* Going beyond mere keyword search, queries can be posed as structural descriptions of a target dataset. Queries will be answered not necessarily from a single dataset but might involve the combination of multiple ones. Combinations include not only the datasets required but consist of instructions on how these datasets have to be integrated by use of possible union-, join-, filter-, and aggregate-operations.

- *Consistency of operations.* Throughout operations triggered, especially, those authored directly by a human user, the consistency of the resulting datasets is ensured. In particular, unit consistency will be maintained without further user intervention required. If any conversion becomes necessary during an operation, it will be applied automatically. Further, if multiple conversions are required due to more complex formulae, the respective formulae will be restructured to minimize conversions applied and thus to improve the accuracy of results as well as overall performance.

- *Visualization recommendation.* Suitable visualizations will be proposed based on a given dataset's characteristics. Visualizations are described by constraints on possible input datasets, such as the number of columns or their datatype and role. Instead of using fixed thresholds to separate matching inputs from unsuitable ones, this relies on transition functions to provide a range of accepted values. Further, recommendations may include suggestions for modified versions of the given dataset if no suitable visualization can be found that encompasses the dataset as a whole.

- *Provenance tracking.* All actions involved in creating a dataset or its visualization will be recorded automatically. On the one hand, this provenance information documents the origins for a specific result and thus may support its credibility. On the other hand, it can re-execute the respective workflow with possibly updated data later on. The underlying provenance model is able to capture relationships on a per column level in order to balance between conciseness and descriptivity needed.

The feasibility of these proposed approaches and algorithms has been demonstrated in a prototypical implementation: Yavaa. It provides a uniform interface to support non-expert users in their way from fetching datasets to match their task up until displaying them using proper visualizations. Using this prototype and the data offerings of Eurostat [web1], a user study has been conducted. Its results support the claim that such an integrated system spanning the whole workflow and including proper recommender systems improves the user experience and quality of created visualizations compared to commonly used applications. The thesis concludes with a summary and an outlook of future research directions to extend the proposed concepts.

Acknowledgments

Over the course of my PhD work, many people have accompanied me on my path and supported the thesis – be it directly or indirectly. Completing such a task would not have been possible without their assistance, encouragement, and, more often than not, patience. I want to thank all of them for their continuous support.

First and foremost, I am grateful to Prof. Dr. Birgitta König-Ries, who supervised the thesis. She provided me with the opportunity and has been a constant source of inspiration and feedback ever since. In numerous discussions, she challenged my assumptions, supported my ideas, or pointed out new directions – whatever was needed at the time. Above all, I want to thank her for her almost stoic patience throughout all the delays throughout the thesis work.

I would also like to thank Manfred Hauswirth, who made my stay in Galway possible and planted the seeds that grew into this thesis. My thanks also go to Prof. Dr. Paul Groth and Prof. Dr. Axel Polleres, who agreed to serve as external reviewers. I appreciate them taking the time to assess this thesis despite their numerous other responsibilities.

I also owe gratitude to all the colleagues that were a part of the Fusion group during my time there. Alsayed, Andreas, Andreas, Carola, David, Elli, Fedor, Frank, Felicitas, Franziska, Friederike, Jan Martin, Javad, Jesus, Jitendra, Kob, Leila, Martin, Michael, Nora, Roman, Samira, Sheeba, Steffen, Sven, and many more - you all create the great environment that Fusion is. Special thanks go to Jan Martin Keil with whom I had the pleasure to dive deep into the abyss of unit ontologies – the ever longer answer to “what would you suggest?”

Further, I want to thank all the students I had the pleasure of working with either during courses or their theses. Not least because of teaching and explaining, you truly understand the ins and outs of your domain. There is no such thing as a stupid question; sometimes asking is the best way of finding gaps and inspiring new directions. In particular, I want to thank Maximilian Stiede, who improved the (meta)data collection required for the evaluation. Similarly, I want to thank Markus Steinberg for his participation in the journey through units and quantities.

The final study would never have been possible with Christoph Bergmann and Christian Streubel. Their feedback and pedantry on the prototype pushed it to the play where it is now. Thanks also have to go to all the survey participants, who contributed their time and effort during the early time of a global pandemic.

Last but not least, I am in great debt to my family and friends. They have unconditionally supported me over many (and even more) years. You enabled this long journey! Even though some may have doubted, this thesis has finally been finished.

Ehrenwörtliche Erklärung

Hiermit erkläre ich,

- dass mir die Promotionsordnung der Fakultät bekannt ist,
- dass ich die Dissertation selbst angefertigt habe, keine Textabschnitte oder Ergebnisse eines Dritten oder eigenen Prüfungsarbeiten ohne Kennzeichnung übernommen und alle von mir benutzten Hilfsmittel, persönliche Mitteilungen und Quellen in meiner Arbeit angegeben habe,
- dass ich die Hilfe eines Promotionsberaters nicht in Anspruch genommen habe und dass Dritte weder unmittelbar noch mittelbar geldwerte Leistungen von mir für Arbeiten erhalten haben, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen,
- dass ich die Dissertation noch nicht als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht habe.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts haben mich folgende Personen unterstützt:

- Prof. Dr. Birgitta König-Ries

Ich habe die gleiche, eine in wesentlichen Teilen ähnliche bzw. eine andere Abhandlung bereits bei einer anderen Hochschule als Dissertation eingereicht:
Ja / Nein.

_____ Jena, 17. Oktober 2022

[Sirko Schindler]

Table of Contents

Part I

Prolog

CHAPTER 1

OVERVIEW

In his 1886 novel *Fathers and Sons* [1], Turgenev wrote: "The drawing shows me at one glance what might be spread over ten pages in a book."[1] Those "ten pages" might not be enough to cover even a tiny fraction of the data available today, but the key message has lost none of its relevance. Encoding intractable, barely accessible datasets in concise, easily understandable graphic representations is still considered the prime solution to communicating complex ideas and relations. The systematic study of designing and creating useful graphs is the main topic of Information Visualization (InfoVis). Its origins can be traced back at least 150 years to the works of Snow [2] and Minard [3]. More formally, InfoVis "is the static or dynamic presentation of information in an external representation such that the information can be processed by efficient human visual mechanisms. [...] The key idea of information visualization is to make use of people's powerful visual system to efficiently process information that otherwise requires more cognitive effort" [4]. At the crossroads of computer science, graphics, behavioral sciences, and cognitive science, the focus is on visual representations and interaction patterns that aid the understanding of abstract data[2].

The importance of InfoVis becomes even more apparent in the face of the wealth of information that is constantly published. The current situation of the information landscape is driven by at least two different factors: First, the volume of data being created is increasing at an ever-growing rate [7]. This is not only caused by the increasing capabilities of employed hardware [8], but also by the prevalence of digitization of more and more processes [9]. Second, this growth is not confined to a specific area but can be seen across almost all aspects of human life. An easily

[1] The more recent and commonly referred to proverb reads, "a picture is worth a thousand words."

[2] In contrast, Scientific Visualization (SciVis) is based on datasets with a given spatial structure like wind flows or medical examinations. However, the exact border between both areas, if it exists at all, is subject to extensive discussions [5, 6].

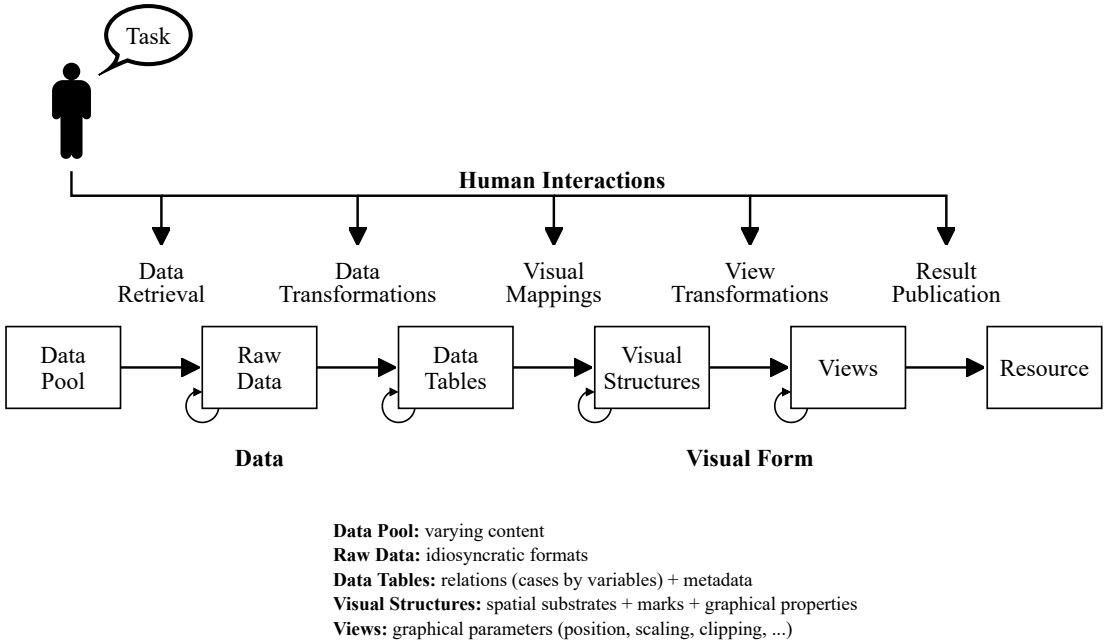


Figure 1.1: Visualization Reference Model (extended from [19]).

observable witness to this trend is the number of publicly available datasets from sources as diverse as Open Government Initiatives [10, 11, 12], science [13, 14], or businesses [web2, web3]. In some areas like public administration, access to documents and thus data is even mandated by international conventions like the Tromsø Convention [15]. This publicly available data can only give a glimpse of what is created and processed within closed systems, though. Nevertheless, the core challenge remains the same in all scenarios: how can this wealth of data and information be made accessible to humans – be it the general public or a restricted audience?

The selection of a fitting visualization is but the final step in a more extensive workflow in order to communicate one's ideas. The overall process starts far earlier, as depicted in Figure 1.1. It begins with a *task* or *idea* that should be conveyed with the final result [16]. This is followed by multiple steps of data preparation. In particular, it involves a *data retrieval* step to acquire the required *raw data* out of the available *data pool*. This source is then subjected to a series of *data transformations* that harmonize its structures and possibly add new, derived information to it. The resulting so-called *data tables* form the input to *visual mappings* creating the *visual structures* of a visualization. If the respective environment permits it — e.g., the visualization is presented in the context of an interactive website —, users may apply *view transformations* to adapt the visualization to their specific need. The control elements for these *views* oftentimes follow established interaction paradigms like Shneiderman's "overview first, zoom and filter, then details-on-demand" [17] or so-called *Magic Lenses* [18]. Regardless of whether the visualization is interactive or not, a static snapshot may be exported. This snapshot can then be used independently of the previously used tools as a resource to support the initial idea.

For a novice user, each of these steps bears its own perils[3]. After conceiving the idea about a visualization project, acquiring appropriate data oftentimes proves to be the first obstacle [22]. With the shift towards publishing data, also the number of providers of data has increased tremendously [14]. There are some, mostly domain-specific efforts to provide an aggregated catalog of datasets like GFBio for the (German) biological and environmental research community [web4] or `data.gov.uk` for governmental data from the United Kingdom [web5]. The overall landscape for structured data on the web still seems rather fragmented, though. This situation resembles the state of the conventional world wide web in its early years, before the advent of search engines. In the meantime, products like Bing [web6], Baidu [web7], or Google [web8] have established themselves as the main portals to the general-purpose world wide web, offering information access with little previous knowledge required. However, applying their techniques to dataset retrieval does not yield the same effect. Search strategies for structured data differ from conventional search requests and support for their particular requirements is currently limited in traditional search engines [23, 24]. Apart from queries that target metadata descriptions, there is also a need to query the contents of a dataset, its primary data. In particular, users are interested in its spatial and temporal extent but also in the range of other covered values. Recent efforts like Google's Dataset Search [25] attempt to address this discrepancy but still focus on datasets as a whole and do not allow for effective search within the primary data. Other approaches index the entire table's content, applying techniques from document-retrieval [26, 27]. However, such an approach neglect most of the table's inner structure. This loses not only the connection between different fields of individual tuples but is also limited to categorical columns omitting all numeric ones.

Another issue in dataset retrieval can be caused by differences in the terminology used to describe a dataset or its contents. Knowing the domain and the dataset, data curators might use different terms in the description compared to data consumers who might be less familiar with the overall topic. This Semantic Gap [28] can greatly diminish search results when those are retrieved by a plain keyword-based algorithm. Other approaches make use of the Semantic Web [29] and knowledge graphs [30] to exploit connections between different terms, thus creating a semantic(ly enhanced) search [31]. Here, the Semantic Web Graph is mostly used in one of two ways: On the one hand, the graph can be used to either support the construction of a search request (e.g., [32, 33]) or extend the original search request by including synonyms, hyponyms, or hypernyms, so-called query expansion (e.g., [34, 35]). These approaches can be used on top of keyword-based technology stacks and thus require fewer changes to an existing search infrastructure. On the other hand, search can be viewed as finding connected subgraphs connecting nodes matching the search request with nodes that represent the search results (e.g., [36, 37]). Common to many approaches is the fact that they almost exclusively focus on metadata descriptions of the datasets

[3]For a broader overview of challenges in using Open Government Data, kindly refer to [20, 21].

and omit their primary data[4]. Again, a common understanding between data producers and consumers is needed to a certain degree — this time not about the common use of terms, but about what is considered useful metadata and as such should be included in a dataset's description.

Once one or multiple suitable datasets are identified, the next step is to import them into the tool of choice for further processing. The basic problem at this point seems to be widely solved: Most commonly used tools support a wide range of file formats[5] and thus are applicable to many datasets. However, while adhering to common file formats some, data providers choose file structures that are not commonly supported. Instead of a plain table format, a pivot-table structure was chosen, e.g., at Eurostat [web1]. Converting pivot tables to flat table structures is rarely supported by popular spreadsheet software and thus impedes the use of these datasets. For similar reasons, web services of any kind (e.g., [web13] or [web14]) are exempted from support in many tools. The variety in possible data structures returned by these services can not be tackled by a one-size-fits-all solution. Then again, removing all these potential data sources from the available data pool severely limits the range of possible analysis tasks.

After the dataset is imported into the tool of choice, users can freely engage with its contents beyond the available metadata information[6]. However, many data providers make extensive use of abbreviations throughout their datasets that might not be easily understandable to the uninitiated. Consider the example of the European Nomenclature of territorial units for statistics (NUTS) [41], which provides the codes for geographical regions in Eurostat's datasets [web1]. The codes for countries are rather understandable, as they are modeled after the respective countries' names and are oftentimes similar to other standards such as ISO 3166-1 [42]: *FR* for France or *ES* for Spain (Span. España). For further subdivisions, this system switches to a more abstract scheme using alpha-numeric suffixes to the respective country code. This results in codes like *BG412* for Bulgaria's capital Sofia or *PL71* for the Polish region of Lódzkie. For novice users, these codes are barely accessible and are not usable without proper code lists.

A similar discrepancy in terminologies can become apparent when working with datasets by multiple providers. For different historical or legal reasons, these providers adhere to different standards to encode their data values. Continuing the previous example of codifying geographical regions, the National Institute of Standards and Technology (NIST) recommends the usage of the ISO 3166-2 [43] since 2008 [web15]. Here, the respective codes for some countries' subdivisions

[4]There are approaches like the RDF Data Cube Vocabulary [web9] that model entire datasets – both metadata as well as primary data – in terms of a semantic graph. However, due to the overhead introduced by RDF models [web10], this does not scale for materialized storage. Ontology-Based Data Access (OBDA) [38] techniques can offer a solution to this, however, at the cost of limiting the semantic expressiveness of the models used.

[5]Notable exceptions can be found among XML [web11] or JSON-based [39] file formats. For example, SDMX [40, web12] is an XML-based format commonly used in governmental statistics. It features a multidimensional data model, which differs from the two-dimensional table layout that is usually supported. Consequently, additional transformations would be required to translate between the data models. These transformations and the oftentimes rather specialized application area of these formats most likely prevented their support beyond their traditional ecosystem.

[6]While some providers offer some means to interact with primary data, this is oftentimes limited to a very narrow set of operations. Thus, for a fully interactive exploration and interaction with primary data, other tools are required.

might look similar to NUTS at a first glance. However, a closer inspection reveals that the assignments differ substantially: Sofia is represented by *BG-22*, while Lódzkie is assigned the code *PL-10*. With no general adoption of a single standard in sight, these differences in value encoding can severely hamper data integration. Oftentimes, there is no reference mapping between such encoding standards available, so users are left to manually map data entries to each other. Besides being a rather tedious and error-prone task, this does also not scale beyond a very limited number of data entries.

Available data is rarely in the form needed for a specific task. More often than not, users have to apply multiple transformations before they can achieve their actual goal – a process called data wrangling [44] similar to "Extract, Transform, Load" (ETL)-tasks in Data Warehousing [45]. Besides the union- and join-operations used in the aforementioned data integration, this also includes filtering the data by certain criteria or applying computations based on existing columns. On a technical level, these operations are supported by most tools. However, ensuring the consistency of these operations is left up to users. A significant source of errors is the handling of units of measurement, in particular when integrating data from different providers. Here, mistakes are often hard to spot and can slip by unnoticed. Nevertheless, the omission of a crucial unit conversion can substantially alter the result [46, 47] or lead to catastrophic failure [48].

At some point, the dataset is ready and prepared: all relevant data is assembled, necessary computations have been carried out, and superfluous parts have been removed. Now the decision has to be made among a variety of available visualizations [web16]. All of them differ in their technical requirements like number and type of columns in the dataset [49, 50] as well as the type of message they are meant to convey [51, 52]. As the number of possible choices is huge, most applications choose one of two strategies: A simplistic approach might restrict the selection of visualizations offered. Popular spreadsheet software like LibreOffice [web17] serves as an example here by offering just eight different visualizations to choose from [web18]. On the other end of the spectrum, tools allow users to create basically every possible visualization imaginable provided a sufficient level of expertise [web19, web20]. No matter the approach, most tools offer little if any assistance in selecting a fitting visualization for the dataset at hand or mapping columns to visual artifacts. Users have to rely solely on their own expertise, which might fail them at times [web21]. This also extends to interactive visualizations, where there is no support but individual users' experience on what interactions patterns exist and can be applied to the chosen visualization type.

An often neglected but nevertheless vital part of any data-driven workflow is the documentation of individual steps, also called provenance or lineage of the final data product. At a time when the validity of statements is more and more questioned [53] and rumors spread quickly regardless of their veracity [54], the ability to trace back to the origins of a visualization or dataset is essential in (re-)establishing trust. Instead of relying exclusively on the assurances of the data product's creator, everybody can verify all sources the methodology applied to the raw datasets for

themselves. Furthermore, a detailed provenance record can be used to reenact the workflow and (hopefully) reproduce the results. In particular, in many branches of science, this can contribute to fighting the so called “reproducibility crisis” [55, 56, 57]. Again, tool support for keeping track of changes is generally lacking. Standard spreadsheet software offers no means of capturing provenance information beyond the undo/redo-functionality. However, even this information is oftentimes lost once the respective tool is restarted. The use of scripting languages provides some improvements: If the whole workflow was performed using the same language, there is at least a chronological record of applied modifications. But as source datasets are oftentimes downloaded separately, their sources are not part of that script and, hence, its provenance record. Although scripting languages provide some kind of data provenance, this approach also lacks compatibility to established standards like W3C’s PROV suite [web22] that enable analyses beyond the scope of a single environment.

The previous discussion indicates that many basic components for the visualization workflow presented in Figure 1.1 are present. If supported, they are scattered over multiple tools, though. For decent results, quite a lot of expertise in a diverse range of areas is required. Besides the usage of the respective tools, this includes knowing where to find appropriate data, how to consistently transform it to the desired form, and which visualization suites the respective data and message. For novice users, each of these challenges can prove to be quite a barrier. Even if they can master the technical requirements, plenty of possible pitfalls remain on their way between idea and final visualization.

1.1 Objectives

The previous section outlined key challenges users have to face when preparing a visualization to support their storytelling. This thesis contributes to enabling novice and intermediate users to overcome these challenges and create meaningful visualizations. The very same techniques can also help advanced users to reduce time spent on repetitive steps and allow them to concentrate on the creative parts of their tasks. Hence, the main objective of this thesis can be stated as follows.

Objective 1. (Thesis objective) ***Development of a unified platform to support visualization workflows from start to finish.***
The platform has to support at least the following steps: *data retrieval, data transformation, visual mapping, and provenance tracking*. All steps have to be accessible through a single, unified user interface that allows novice and intermediate users to access all functionalities with little to no prior expertise in visualization workflows.

However, unified access to all functionality alone does not necessarily enable inexperienced users to create suitable visualizations. As established before, at several stages of the workflow, users require a certain level of expertise to produce meaningful results. However, this knowledge can not be assumed for all users and is, in particular, lacking for novice users who, by definition, have next to no prior experience with common best practices. Hence, the developed platform also has to establish means to support users at critical stages in visualization workflows by either providing suitable recommendations or highlighting (and thus preventing) potential errors. This leads to the following additional set of objectives.

Objective 2. ***Development of integrated search and data access capabilities to query and aggregate over multiple, heterogeneous data providers.***
Heterogeneity across datasets andor data providers – e.g., concerning file formats, file structure, or terminology – has to be harmonized transparently to users. If an information request can only be fulfilled by using multiple datasets, their integration has to be performed in a semi-automatic fashion.

Objective 3. ***Development of techniques to ensure the validity of data transformations.***
During data transformations, the system has to ensure the consistency of the derived dataset. If users trigger an operation that would violate given consistency constraints, the system should try to fix the issue. If an automatic correction is not possible, it has to reject the execution of said operation and issue a corresponding error to the user.

Objective 4. ***Development of a recommender to suggest suitable visualizations and visual mappings for given datasets.***
The system has to determine applicable visualizations for a given dataset and rank them according to their suitability. Upon user selection of a suitable visualization, the system has to suggest possible mappings of data columns to visual artifacts taking into account their respective characteristics.

Objective 5. ***Consistent tracking of provenance over the entire data visualization life-cycle.***
The provenance and dependencies of all applied operations starting with data retrieval and up to the creation of the final visualization product have to be tracked consistently. Users have to be provided the option to export this historical data in a standards-compliant way.

When applied to the entirety of possible visualizations and types of data, these objectives go beyond the scope of just a single thesis. So within the presented work, certain restrictions are placed on the considered inputs and outputs that the developed platform is able to cope with. It only covers tabular data and InfoVis-related visualizations. In particular, this excludes any kind of multimedia data like images or videos as or within datasets. Similarly, most visualizations that are commonly attributed to SciVis are out of scope, including, e.g., any kind of isosurface-plots or 3D-renderings. While not discussed explicitly, some of the presented concepts may be transferred beyond the scope, though. However, this extension of concepts is left for future work.

1.2 Thesis Structure

This thesis is structured into four parts: Prolog, Dialog, Genesis & Analysis, and Epilogue. They reflect the split into prior considerations, conceptual contributions, evaluation through a prototypical implementation, and final considerations.

Part I continues to lay the basis for the presented efforts. In Chapter 2 the previously outlined objectives are translated into functional and non-functional requirements to guide the remainder of the thesis. Subsequently, in Chapter 3, common strategies to create visualizations are discussed and evaluated.

Part II contains the conceptual contributions of this thesis as well as necessary background knowledge. Chapter 4 outlines the overall approach taken and thus motivates later discussions. This is followed by foundational aspects, namely data types in Chapter 5 and data models for tabular data in Chapter 6. Having laid this groundwork, data models to describe visualizations and datasets are proposed in Chapter 7 and Chapter 8, respectively. Chapter 9 provides the means to consistently and efficiently handle units of measurement across computations. Chapter 10 introduces an approach to fulfill search requests by combining multiple datasets. Chapter 11 presents a framework to recommend visualizations for a given dataset and rank them according to their suitability. Finally, Chapter 12 concludes the conceptual aspects with a provenance model to capture all activities throughout the proposed workflow.

Part III is dedicated to applying the proposed concepts and assess them in different scenarios. Chapter 13 introduces the corresponding implementation, Yavaa, and reviews major decisions throughout its development. This prototype is then subjected to the evaluation discussed in Chapter 14, covering both a user study as well as performance benchmarks of the developed implementation.

The final Part IV closes the thesis. Chapter 15 revisits the contributions of this thesis in the light of the initially posed objectives and requirements, including a discussion of limitations in the proposed approach. Chapter 16 summarizes the thesis, before in Chapter 17 possible future directions are put forward that allow to continue and extend the work presented here.

CHAPTER 2

REQUIREMENTS

Having previously outlined current challenges in common visualization workflows, now the requirements they impose upon possible solutions shall be considered. The discussion of functional requirements will follow the general order of steps given in the reference model of Figure 1.1. The derived requirements will be used across multiple sections of this thesis: Existing solutions will be reviewed in Chapter 3 in light of these requirements. The design of the proposed system described in Chapter 4 and the subsequent chapters will be guided by the goals established here. Finally, the prototypical implementation will be evaluated in Chapter 14, before Chapter 15 will revisit the requirements posed here to judge the outcome of this thesis.

2.1 Functional Requirements

Dataset portals are just about as numerous as data providers [14]. From a user perspective finding the proper portal is almost as difficult a task as identifying the actual dataset [23]. So it is essential that search capabilities are not restricted to a single data provider but can integrate multiple ones in a one-stop portal. But merely making the datasets available through a single interface is not sufficient. As terminologies used oftentimes vary between providers [58], these differences also have to be mediated to provide a truly integrated search.

Requirement 1. Search across Providers
The system has to provide a unified search interface across different data providers. This also includes a translation of provided inputs into the idioms used by the respective providers.

Users' information needs are ultimately served predominantly from the primary data. Yet, traditional search systems often focus on a select few metadata fields like title and (human-readable) description [59]. Sometimes users can place restrictions on the spatial or temporal extent of their search, but most other primary data characteristics remain hidden [24, 58, 60]. For an effective result-driven search, users have to be able to place constraints on all primary data variables [23]. Instead of having to manually inspect candidate datasets as returned by a keyword-based search, users should be able to define relevant constraints as part of the search query and thus increase the task-oriented precision of returned results.

Requirement 2. Search in Primary Data
The system has to support the definition of constraints on the primary data as part of search requests.

Only a few visualization tasks can be served from a single dataset. More often than not, multiple datasets, possibly from different providers, need to be combined [61, 62, 63, 64]. The system should acknowledge this fact and also consider combinations of multiple datasets in order to fulfill a certain request. Proposed combinations have to be valid, i.e., proper join-conditions among the involved datasets have to exist, such that the result meets the posed requirements.

Requirement 3. Search by Combination
If a given search request can only be fulfilled by using multiple datasets, this combination has to be suggested as a possible result.

As the system has most likely already established the required join-conditions for a combined result, it is redundant for users to repeat this task when selecting this result. Upon user request, the system should automatically materialize the proposed combination result. User interaction should be limited to ambiguous cases like deciding on required aggregation functions when multiple meaningful options exist.

Requirement 4. Materialize Combination-Results
If a search result was given as the combination of multiple datasets, means have to be provided to materialize this combination.

Not limited to automatic combinations of datasets but also affecting manual joins are the vocabularies used to encode the primary data. If two datasets are combined that use different vocabularies for the same concepts, the system has to automatically harmonize occurring terms and present a consistent result to its users [65, 66]. Besides accelerating the overall process, this can also prevent errors in the results, e.g., when the same or similar abbreviations represent different concepts in different vocabularies.

Requirement 5. Mediate Abbreviations
Heterogeneous abbreviation schemes possibly in use by the involved data providers have to be harmonized transparently for users.

Another consequence of Requirement 4 is that the system has to be able to take care of any heterogeneity pertaining to the storage of data [66]. In particular, structures and formats used by specific providers have to be transparently aligned to enable a seamless integration into a single dataset as the basis for further operations.

Requirement 6. Support Heterogeneous File Formats
When loading primary data, the system has to support the consumption of multiple, heterogeneous file formats transparently to users.

Requirement 7. Support Heterogeneous Dataset Structures
When loading primary data, the system has to be able to automatically harmonize different data structures for further processing.

The vocabularies used by providers to encode their primary data can be arbitrarily abstract. The target audience most likely is not familiar with the intricacies of each and every scheme in use [67]. The system has to translate the used abbreviations wherever possible to terms users are familiar with [65]. This only pertains to the presentation to users and does not have to extend to an internal representation, which may remain unchanged.

Requirement 8. Translate Abbreviations
Encoding schemata in use by the different providers have to be consistently translated into human-readable labels.

InfoVis is an inherently iterative process [19]: Data is prepared and visualized. Upon inspection of the visualization, further adjustments to the underlying data may become necessary, triggering another cycle of modifications. The system has to support such iterations by providing means to actively transform the data to get it into a suitable shape. In order for the process to be truly interactive, the results of all modifications have to become immediately visible to users.

Requirement 9. Allow for Modification of Data
The system has to allow users to adapt datasets to their current needs by, e.g, filtering, aggregating, or adding new columns derived from given formulae.

Requirement 10. Provide Immediate Feedback on Operations
The results of applied operations have to be apparent to users without unnecessary delays.

Data modifications are a primary source of errors not only in visualization workflows but in any data-driven process (e.g., [68, 69, 70]). During any data modification, the system has to check operations' validity, or critical errors might inadvertently find their way into the analysis. Here, it does not matter whether this operation was triggered directly by the user or is the consequence of some automated process like the aforementioned combination of datasets. In case a possible error is detected, the system should attempt to automatically remedy the situation and apply a proper compensatory actions. Only if this is not possible should the operation be canceled and users be presented with the respective error.

Requirement 11. Ensure Validity of Operations
During both system-initiated as well as user-driven modifications of the datasets, the system has to ensure the validity of operations. Possible errors have to be transparently corrected. Only if this can not be achieved shall users have to intervene.

Given the plethora of available visualizations, users may easily get overwhelmed with their choices [71, 72]. However, not all visualization will be equally suitable for the current dataset. For example, some might not even be applicable as the number and types of variables in the dataset does not fit the requirements of the visualization. The system has to account for the current dataset and only recommend visualizations that can be generated from the current dataset.

Requirement 12. Recommend Visualizations
The system has to provide recommendations for meaningful visualizations given a certain dataset.

In general, for a selected visualization, there are also several possible configurations that determine how the tuples of a dataset are converted into visual artifacts. However, this configuration-space is subject to certain restrictions, as, e.g., not every variable can be translated into every characteristic of the visual artifacts. Disregarding these restrictions either consciously or accidentally can lead to misleading visualizations [73]. The system has to provide guidance on how to translate the given dataset's variables into visual artifacts of a selected visualization.

Requirement 13. Recommend Variable to Artifact Mappings
The system has to provide recommendations for a meaningful translation from the current dataset and into the selected, applicable visualization.

For a unified platform, merely recommending visualizations and mappings is not enough. The system also has to be able to execute those recommendations and create the respective visualizations. Users should only be required to select and, if necessary, adjust the declarative definition of said visualization.

Requirement 14. Materialize Visualizations
The system has to be able to create visualizations based only on declarative definitions provided by users.

There are plenty of steps involved in creating a visualization, with the final visualization allowing for only limited assumptions about the process. In order to trace back the origins of a visualization, the system has to record all data sources involved as well as the operations applied to them. The entirety of those records then constitutes the provenance record for the visualization. Adequate provenance records for a visualization form the basis for reproducing it if needed [55] and enable tracking down possible errors in both analysis and data sources [74].

Requirement 15. Track Provenance
Each operation that alters a dataset has to induce an entry into the provenance record of this dataset. The corresponding entry has to include all settings that affected the respective operation, including at least inputs, outputs, and any parameters necessary.

Provenance records in and of themselves are rather technical constructs and barely accessible for most human users. On the other hand, visual representations such as workflow graphs have emerged as a comparatively accessible way to inspect dependencies among and the order of operations. To ease user access to the provenance record of a given visualization, the system has to provide a visual representation of that record, e.g., in the form of a workflow graph.

Requirement 16. Visualize Provenance Records
The accumulated provenance records for the current dataset have to be presented visually to users. This representation has to include at least the applied operations, their logical order, and their respective inputs and outputs.

Attesting to the validity of a visualization within the tool that created it is of only limited use in the communication with others. Visualization pipelines have become an integral part of the scientific process like analysis pipelines before [74]. As a means to maintain, share, and reuse visualization pipelines, a standards-based exchange of provenance records is required.

Requirement 17. Share Provenance Records
The system has to provide the means to export the complete provenance record for a given workflow in a standardized format consumable by other applications.

While human inspection is one use case for provenance tracking, another is given by the reenactment of the respective workflow. While, in general, there is the distinction in repeating, replicating, or reproducing the original workflow [web23, 75], a single system will first have to focus on the former two[1]. To achieve replication of a previous result, the system has to be able to consume a previously exported provenance record and execute the workflow described within. The result should then resemble the original visualization or datasets. This will hold true only under the assumption that the data sources did not change since the original execution. However, those aspects are beyond the control of the described system.

Requirement 18. Allow for Reenactment of Workflows
Based on a previously exported provenance record and under the assumption that the data sources did not change in the meanwhile, the system has to be able to execute a previous workflow with the same results.

2.2 Non-Functional Requirements

As outlined by Objective 1, the main audience of the proposed system are users with no or only limited experience in visualization workflows. Particularly in those scenarios, the usability of the provided functions is of greater concern than their range. Hence, any system primarily catering to non-expert users has to ensure that all functionalities are easily accessible.

Requirement 19. Usability
The system has to be accessible for novice to intermediate users.

Like many other domains also the areas touched by the described system are in constant flux. In order for a system to stand the test of time, it has to be retrofitted with new components on a regular basis to follow current trends. Two areas, namely visualizations as well as file format and structure adapters, are particularly crucial in this regard. The system has to allow to develop and integrate new components in those areas with only reasonable efforts required.

[1]The definitions of [web23, 75] require a different experimental setup for *reproducing* a workflow. In a strict interpretation this requires a second, independent system with a comparable feature set.

Requirement 20. Extensibility
The system has to be structured in a way that allows to easily add new components for critical areas. In particular, this includes adding data providers, data wrappers, and visualizations.

2.3 Summary

Table 2.1 summarizes the set objectives and lists the requirements derived from them. The thesis objective, Objective 1, is not listed separately but assumed to be the underlying motivation to all requirements. The last two non-functional requirements arise directly from the thesis objective and from common best practices in developing user-facing software.

Primary Objective	Requirement
2: Support of Multiple Providers	1: Search across Providers
	2: Search in Primary Data
	3: Search by Combination
	4: Materialize Combination-Results
	5: Mediate Abbreviations
	6: Support Heterogeneous File Formats
	7: Support Heterogeneous Dataset Structures
3: Validity of Data Transformations	8: Translate Abbreviations
	9: Allow for Modification of Data
	10: Provide Immediate Feedback on Operations
	11: Ensure Validity of Operations
4: Visualization Recommendations	12: Recommend Visualizations
	13: Recommend Variable to Artifact Mappings
	14: Materialize Visualizations
5: Provenance Tracking	15: Track Provenance
	16: Visualize Provenance Records
	17: Share Provenance Records
	18: Allow for Reenactment of Workflows
	19: Usability
	20: Extensibility

Table 2.1: Summary of objectives and requirements.

CHAPTER 3

COMMON STRATEGIES

The need for visualization arises in a wide range of areas, be it as a means of analysis, communication, or any other purpose. Consequently, the solutions developed so far differ just as much. In an attempt to outline the current landscape, this chapter will describe several commonly used approaches that shall serve as representative examples for their respective domain. Naturally, such an overview will need to make a selection among the myriads of available tools and systems. One could argue that one system or the other should also be included. The choice presented here was made to highlight the different environments, assumptions, and solutions present without letting the list grow too far. Another factor for the selection is the amount of information that is publicly available and illustrates a system's inner workings. After describing the approaches individually, all of them will be evaluated along the criteria previously established in Chapter 2.

The approaches presented in the following are loosely ordered by the efforts required before using them. This is determined not solely by expertise needed but also by whether there is software to be installed locally instead of using a ready-made web interface. The descriptions will start with Eurostat [web1] to outline the capabilities provided by many data portals. This is followed by LibreOffice Calc [web17] as a prototypical spreadsheet software. Next, a web-based, specialized data integration and visualization tool is given in Google Fusion Tables [web24]. Business Intelligence (BI) software is generally used to analyze the data holdings of companies and support their business decisions. Tableau [web25] will be discussed as an example here. On the border between scripting and tools providing a graphical user interface lie Jupyter Notebooks [web26], which are described afterward. Finally, scientific workflow management tools will be illustrated by means of a general one, Taverna [web27], as well as one specifically targeted at visualizations, VisTrails [web28].

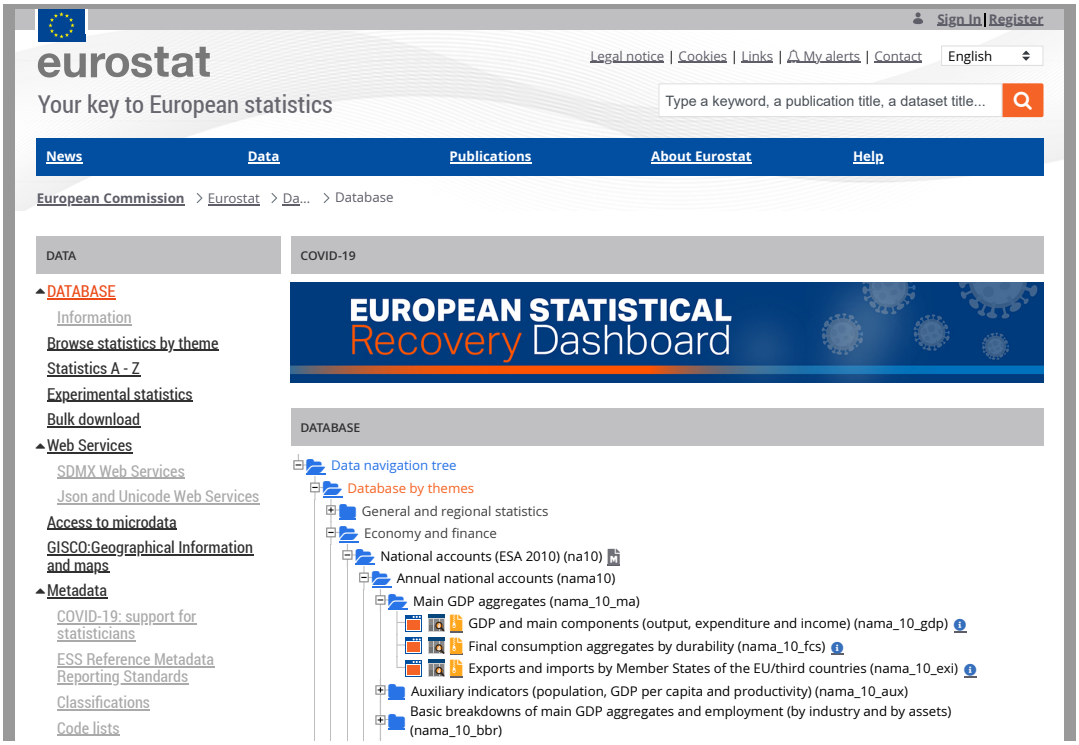


Figure 3.1: Eurostat: Catalog-based search interface (screenshot from [web1]).

3.1 Eurostat

As part of its operations, Eurostat offers data holdings and other publications via a web-based, multi-lingual (English, German, French) interface [web1]. The data offered contains statistical information about various socio-economic indicators with a focus on the European Union but also including other selected countries and regions. Eurostat offers free reuse for its data, including potential commercial uses [web29]. Primary data is organized into tables, each focusing on a single measurement and having at least two dimensions. The default two dimensions common to all tables are *geo* and *time* for the geopolitical entity the measurement refers to and the period of time, respectively. In addition, other dimensions might be covered, including, e.g., the unit of measurement, classification (e.g., age cohorts), or different variations of the respective indicator (e.g., unadjusted vs. seasonally adjusted data).

There are arguably three different ways to search for suitable datasets. First, the bulk download facility [web30] is geared towards automated access. It offers the raw data as well as metadata in a mere file listing labeled by the respective dataset code. In addition, there are files providing a table of contents, including the time of the last update for a specific dataset. These tables of content are published in the same three languages as the portal and in a machine-readable XML file. Finally, code lists are provided to map between used code-entries and human-readable labels.

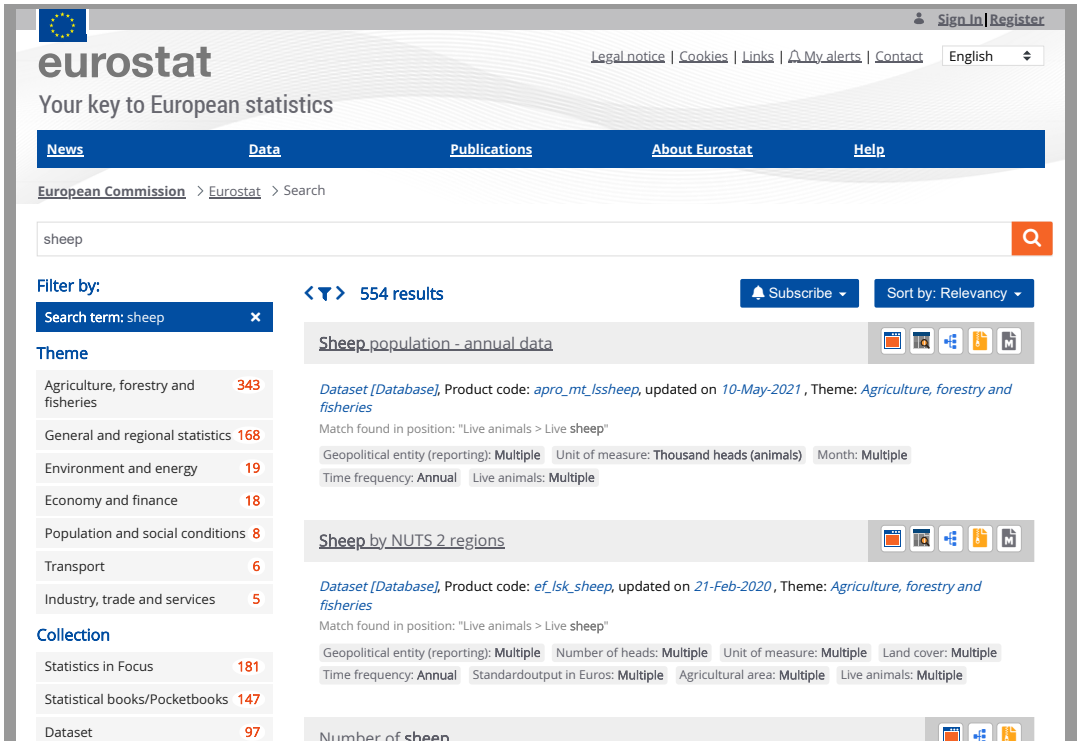


Figure 3.2: Eurostat: Keyword-based search interface.

The two other search facilities are targeted to human consumption: a catalog-based (cf. Figure 3.1) and a keyword-based search (cf. Figure 3.2). The catalog interface provides various multi-level classifications of datasets that can be traversed via a tree-like menu. On the other hand, the keyword-based interface provides a full-text search based on titles and descriptions but apparently omitting other metadata fields as well as the primary data. Once an initial search has been executed, the results can further be filtered using three different facets: Theme, Collection, and Publication date. As the keyword-based search also extends to non-dataset resources, one can restrict the search using the Collection-facet and exclude resources like news entries or manuals.

Once a proper dataset has been identified, it can be downloaded for local analysis but also immediately accessed through Eurostat's Data Browser[1] (cf. Figure 3.3). It consists of three sections: The topmost section includes the general metadata like title, description, and time of the last update. The center section offers means of interaction, as discussed in the following. Finally, the lower section shows the current representation of the data. In general, data can either be shown in a tabular format or using one of three visualizations: line chart, bar chart, or map.

By default, the center interaction area provides controls to select a certain subset of the data. Values are filtered by selecting the requested ones out of a list of all available values (cf. Figure 3.4). The same interface is also used to select possible values from the time column.

[1]In addition, there is a second web application called Data Explorer. However, this appears to be a legacy application offering only a subset of the Data Browser's functionality and, hence, will be omitted from discussions here.

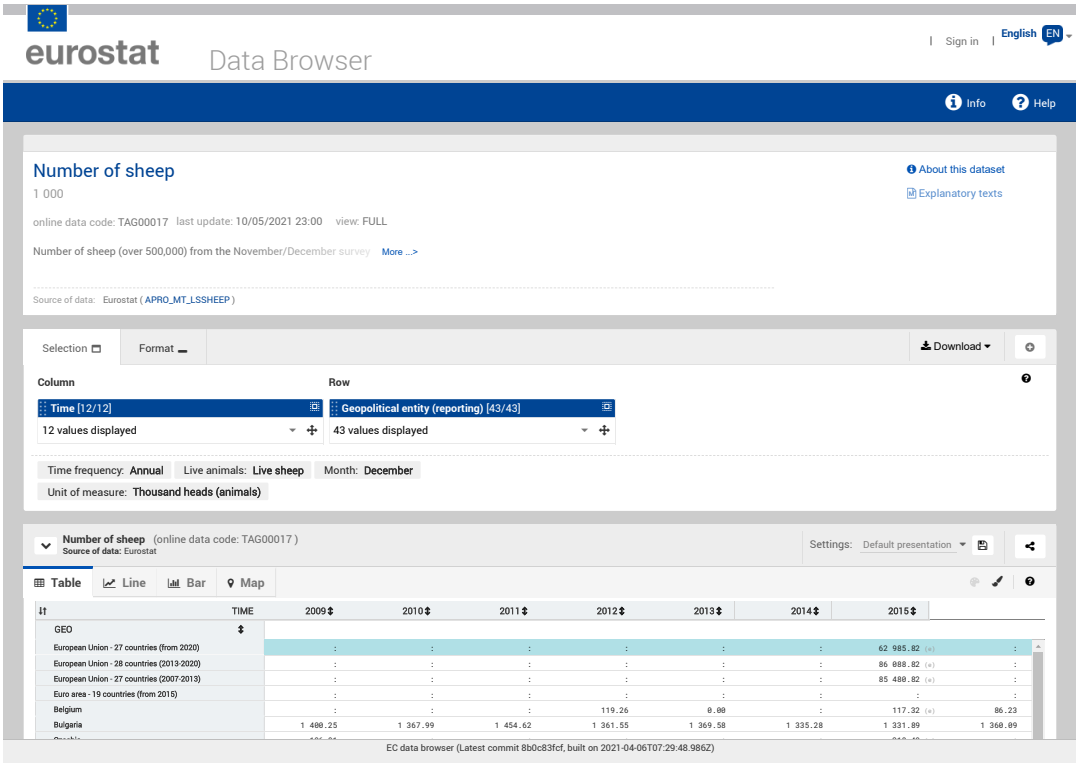


Figure 3.3: Eurostat: Data Browser.

Filters can only be applied to multi-valued dimensions. Measurements as well as single-valued dimensions (see gray boxes in the center section of Figure 3.3) can not be adjusted. Besides filtering, multi-valued dimensions can also be assigned to different visual aspects of the current representations. For example, in the table view, one is able to assign dimensions to be shown in rows or columns, whereas in the bar chart a similar interface is used to distinguish between x and y axes. On occasion, dimensions are fixed to a particular aspect, e.g., in the map representation, the geographical area is bound to visualize the geographic entity referred to in the dataset.

In order to visualize more dimensions, both bar charts as well as maps offer the option to define one dimension to serve as *series*. This provides a shortcut to easily switch between different values of the assigned dimension (cf. bottom row of options in Figure 3.5) instead of adjusting the filter each time.

Each representation also includes a range of formatting options available once the corresponding representation has been selected (cf. different tabs in the lower section of Figure 3.3). There is a common set of options across all representations to control, e.g., the display of labels (cf. Figure 3.6(a)). Additionally, each representation has an individual set of options (cf. Figures 3.6(b) to 3.6(e)), like the sorting of entries in a bar chart or the color scale in a map.

Settings that lead to a particular visualization can be stored and retrieved from the browser's local storage or shared via a custom link. Storing settings in local storage is not persistent, though, and may be removed when clearing the browser's history. Links, on the other hand,

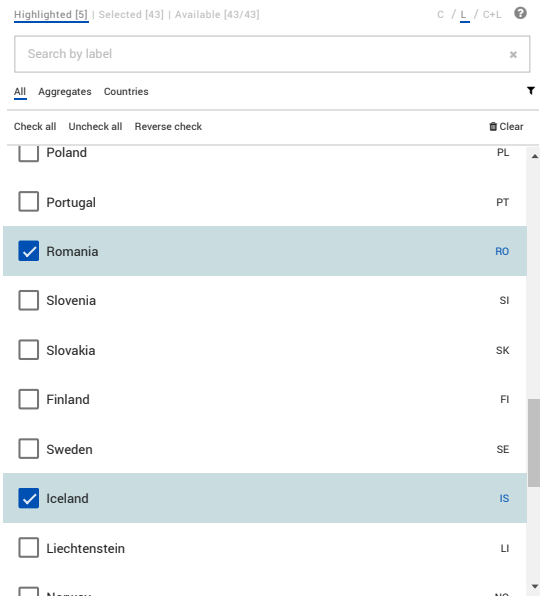


Figure 3.4: Eurostat Data Browser: Filter interface.

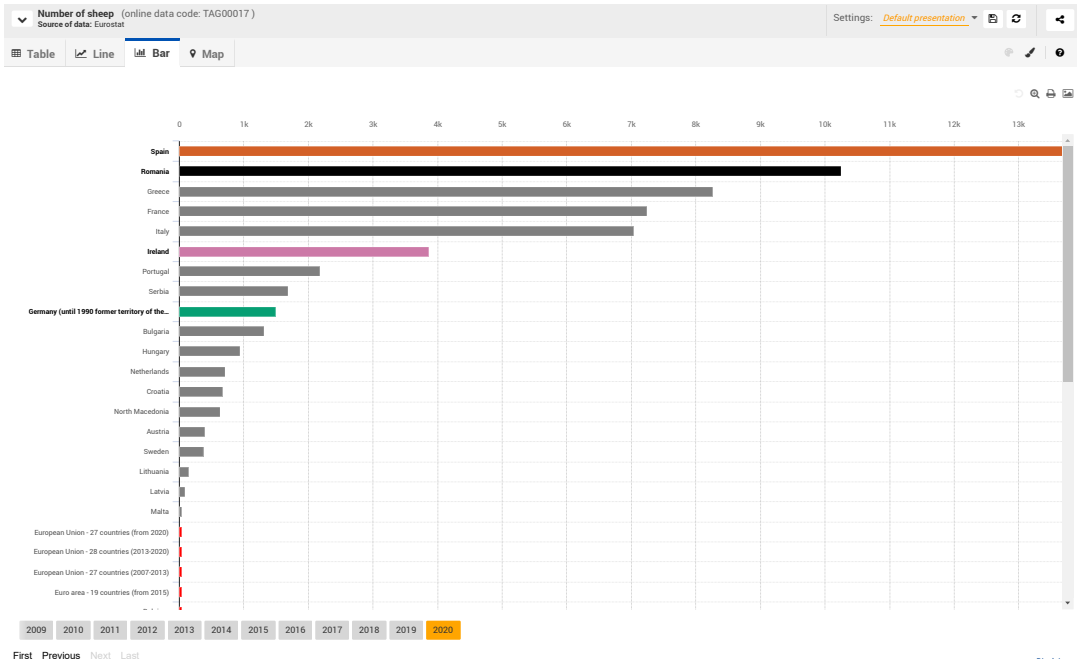
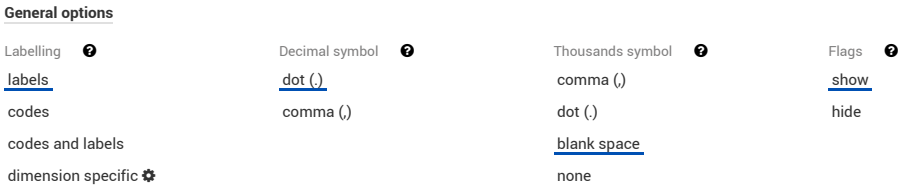
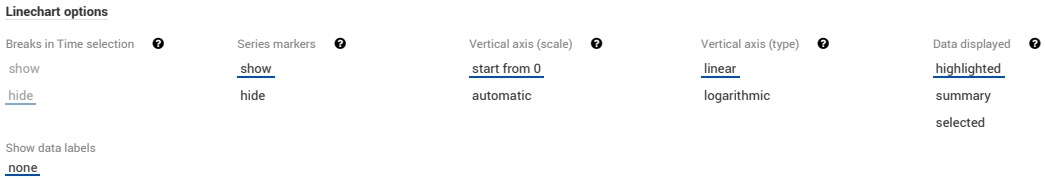


Figure 3.5: Eurostat Data Browser: Bar chart visualization.

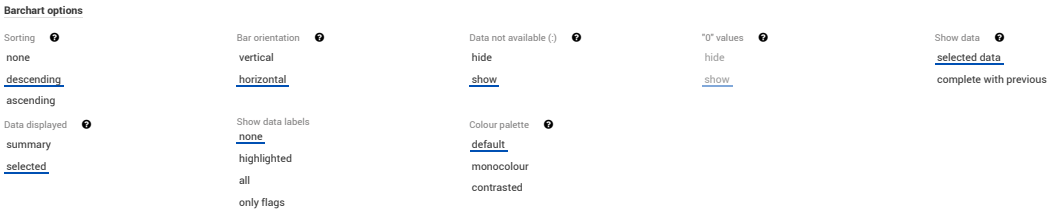


(a) General options.

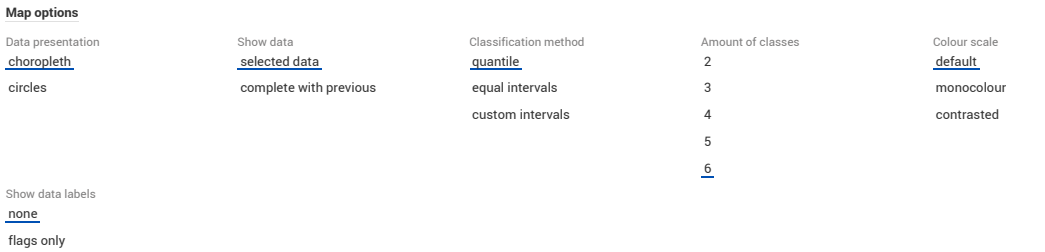
(b) Table formatting options.



(c) Line chart formatting options.



(d) Bar chart formatting options.



(e) Map formatting options.

Figure 3.6: Eurostat Data Browser: Formatting options.

encode all settings in a unique identifier and can thus be freely shared with others. The final representations can also be exported to files. The format depends on the particular representation: Tabular views can be exported to Excel spreadsheets [web31], SDMX [web12], or TSV [web32]. Visual representations vary in the supported formats but include at least one raster graphic and one vector-based format each.

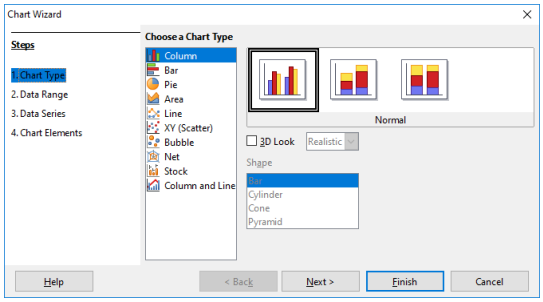
3.2 Spreadsheet Software

To many users, spreadsheet software is the first choice when it comes to data-related tasks. As a consequence, many different variants were created over time ranging from desktop products like commercial Mircosoft Excel [web31] and open-source Calc [web17] in the Document Foundation's LibreOffice suite to web-based tools like Google Sheets [web33]. Despite the differences in existing implementations, their core functions are similar. Although LibreOffice Calc [web17] in version 6.0 forms the basis for the following discussion, the statements can be generalized to other spreadsheet software products.

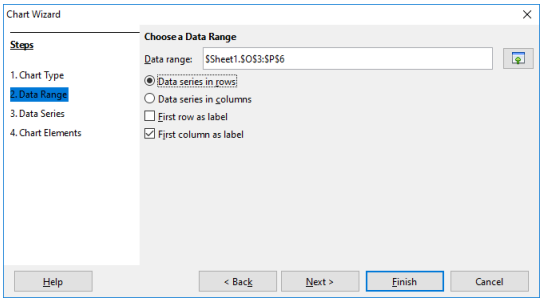
Unlike in other data-related software, spreadsheets do not enforce a strictly tabular structure on the data. Instead, an (in theory) limitless 2-dimensional grid of cells is used, which is usually called a sheet. Users can choose which part of this grid to use for their actual data. Oftentimes, rows and columns are left empty or are used for metadata like headings or descriptions of the primary data. This approach also allows for multiple tables to be part of the same sheet separated by empty rows or columns or other markings within the cells in between. The native document formats like the *Office Open XML File Formats* [web34] or the *Open Document Format* [web35] allow to include multiple sheets within a single file. This allows for even greater flexibility in storing data using these formats.

The offered flexibility is a double-edged sword. On the one hand, it enables applications like spreadsheets. Here, user can enter their data in a subset of the cells. The remainder then performs various computations, including visualizations of the respective data. On the other hand, automatic processing is hindered. As data may be found anywhere in the spreadsheet in various forms, automatic extraction into other systems is non-trivial [76, 77].

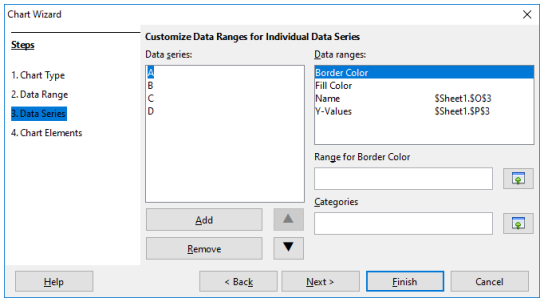
Derived data is created by setting the content of cells to formulae whose result is then shown accordingly. These formulae can refer to any other cell within the same file by the use of its "coordinates." The coordinates are given by (optionally) the sheet and a combination of column (via characters) and row (via numbers). Hence, the descriptor `$Sheet1!B10` denotes the cell in the tenth row of the second column in a sheet called `Sheet1`. Rectangular ranges of cells are defined by their upper left and lower right corner, whose coordinates are separated by a double colon. Within a formula, users are able to make use of a plethora of functions, including most common



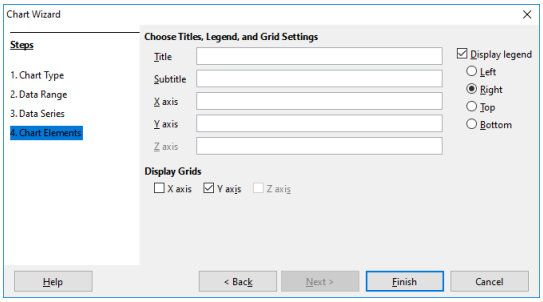
(a) Chart Type.



(b) Data Range.



(c) Data Series.



(d) Chart Elements.

Figure 3.7: LibreOffice Calc: Chart Wizard.

functions like summation or min/max estimation. To improve usability, formulae's coordinates are usually copied relative to the source cell. So if, e.g., a formula references a cell two columns to the left, but on the same row, this will hold true also for all target cells this formula is copied to.

Joining two tables is also accomplished using a function that performs a lookup in some other area of the file. Here, two alternatives are offered dependent on the table structure: `VLOOKUP` and `HLOOKUP`. While `VLOOKUP` assumes rows to form a single tuple, `HLOOKUP` does the same for columns. Both functions take four parameters: The first is the value to search for. This can either be the coordinate of a cell or a regular expression. The second parameter refers to the area (table) to search in. Here, the first column will be traversed in the search, while all later columns can be referenced in the third parameter. This third parameter defines the column (`VLOOKUP`) or row (`HLOOKUP`) whose value to take. The respective index is relative to the defined area and not the whole sheet. The final parameter indicates whether the first column is sorted, which will speed up the search. Joins on multiple columns are not supported but have to be emulated by first creating a primary key consisting of just one column and subsequently using that in the lookup. In general, joins can also be performed across multiple files. As the values of the source cells are not copied to the target but are just referenced, in this case, a link to the other file is created. This link can easily be broken if the files are moved or the referenced file is deleted.

For the creation of visualizations, a four-step wizard is provided (cf. Figure 3.7). The first step allows selecting one of ten general chart types. In the second step, the area is selected that contains the data to be visualized. The third step allows setting additional parameters. Their

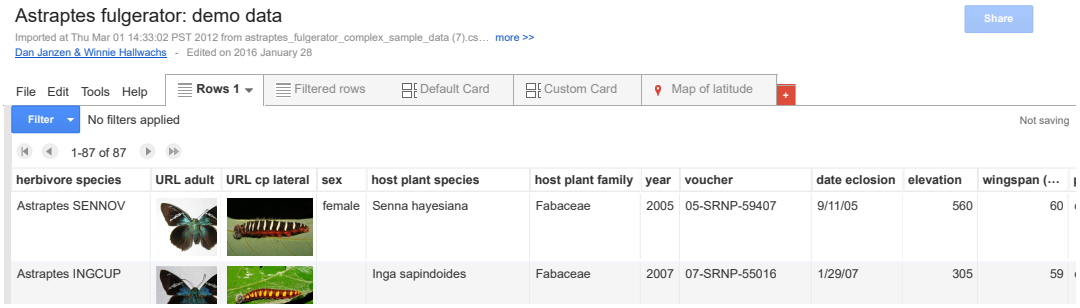


Figure 3.8: Google Fusion Tables: Data view using given demo data.

exact extent depends on the chart types selected but commonly includes naming the data series or assigning a specific color. In the final fourth step, users are able to make some adjustments to the graph as a whole. This includes adding titles for the graph as well as for the axis or enabling grid lines to be shown.

3.3 Google Fusion Tables

Google Fusion Tables[2] [web24, 78] described itself as "a service for data management, integration and collaboration." The system allowed users to upload their data or use previously shared tables. Data was stored in a cloud-based BigTable [79]. In addition to the primary data, also a basic set of metadata was maintained. This included the name and description of the respective table, its schema, and the assigned permissions in form of access control lists. The schema of the table consisted of column names and their respective types. The type model distinguished text and numerical columns. Furthermore, there was a detection for geolocation like latitude and longitude.

Once the data was uploaded, users could inspect it in a tabular layout as shown in Figure 3.8. Users could create new views on the data using filters. The respective forms here depended on the selected columns' types and allow to either include or exclude rows based on specific values. There was no option to compare values in different columns to one another and all filters were connected using an *AND* relation. The results of these filter operations were not materialized but stored as a view over the original data alongside the respective permissions.

Data could also be augmented using other tables, both user-provided ones as well as those publicly shared through Google Fusion Tables. To discover matching tables, a function "Find a table to merge with ..." was provided (cf. Figure 3.11(a)). Although details were not documented, related data seemed to be found by using a column's contents. Users selected a single column as the join condition and the systems searched for tables containing similar values. The results list also contained information about the coverage, i.e. how many rows did have matching rows in the

[2]The service was discontinued by the end of 2019 [web36, web37].

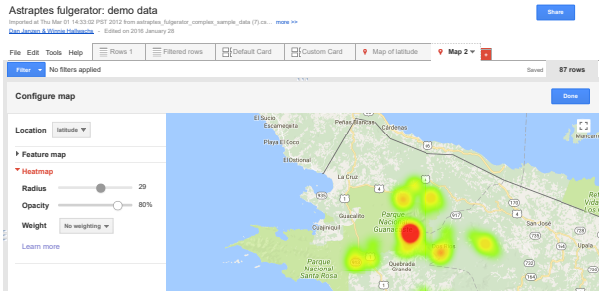


Figure 3.9: Google Fusion Tables: Map visualization.

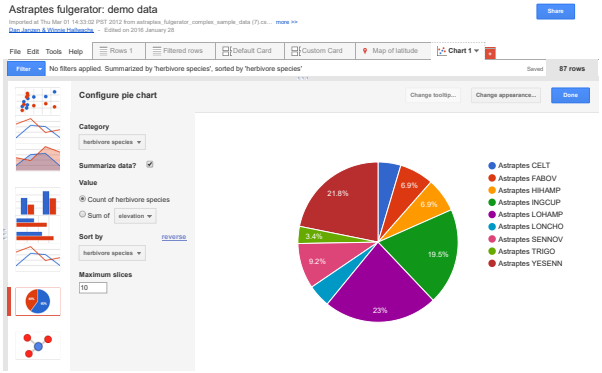
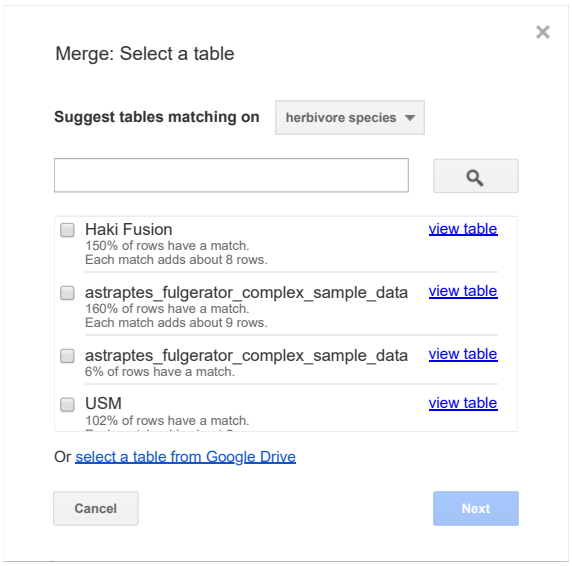


Figure 3.10: Google Fusion Tables: Chart visualization.

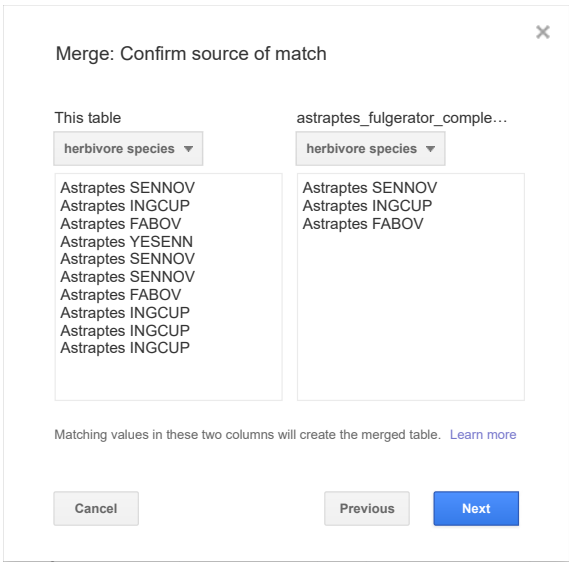
target table. After reviewing matching values (cf. Figure 3.11(b)), users could select the columns to be added (cf. Figure 3.11(c)). Join-conditions consisted of only one column pair. In addition to the filter-limitation to single column predicates, this did not allow to use multi-column join predicates like *country* and *year* for some governmental statistics.

Users were also able to visualize the given data. They were able to choose between map-based visualizations (cf. Figure 3.9) and a set of statistical charts (cf. Figure 3.10). For each visualization type, the used columns could be selected. Some visualizations also provided basic parameters. In the heatmap view, e.g., users were able to set a radius and the opacity of the highlighted areas. Some statistical charts also offered options to aggregate the given data. The graph in Figure 3.10, e.g., shows the number of rows for the respective species.

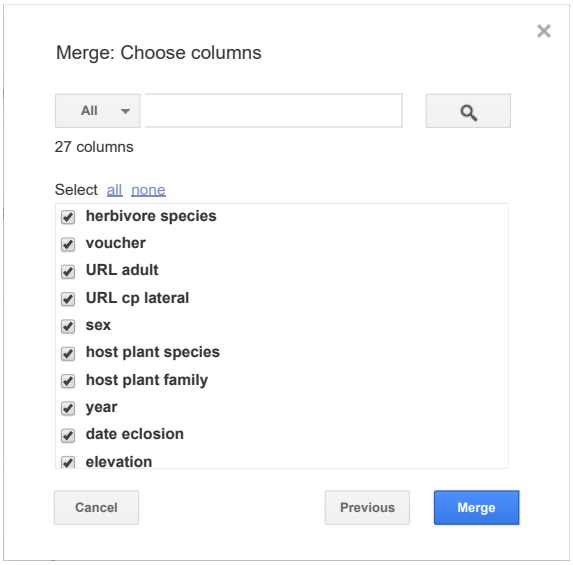
Charts and tables created could be shared both within Google Fusion Tables and beyond. For users that wanted to embed the maps or charts into their websites, a JavaScript snippet was provided to embed the object in any HTML site. The data itself could also be accessed through the Fusion Table REST API [web38]. Using that API users were able to programmatically access all functionalities. In addition to the operations available through the front end, SQL-like queries could be issued to retrieve and modify the stored data.



(a) Select a table



(b) Confirm source of match



(c) Choose columns

Figure 3.11: Google Fusion Tables: “Find a table to merge with ...”.

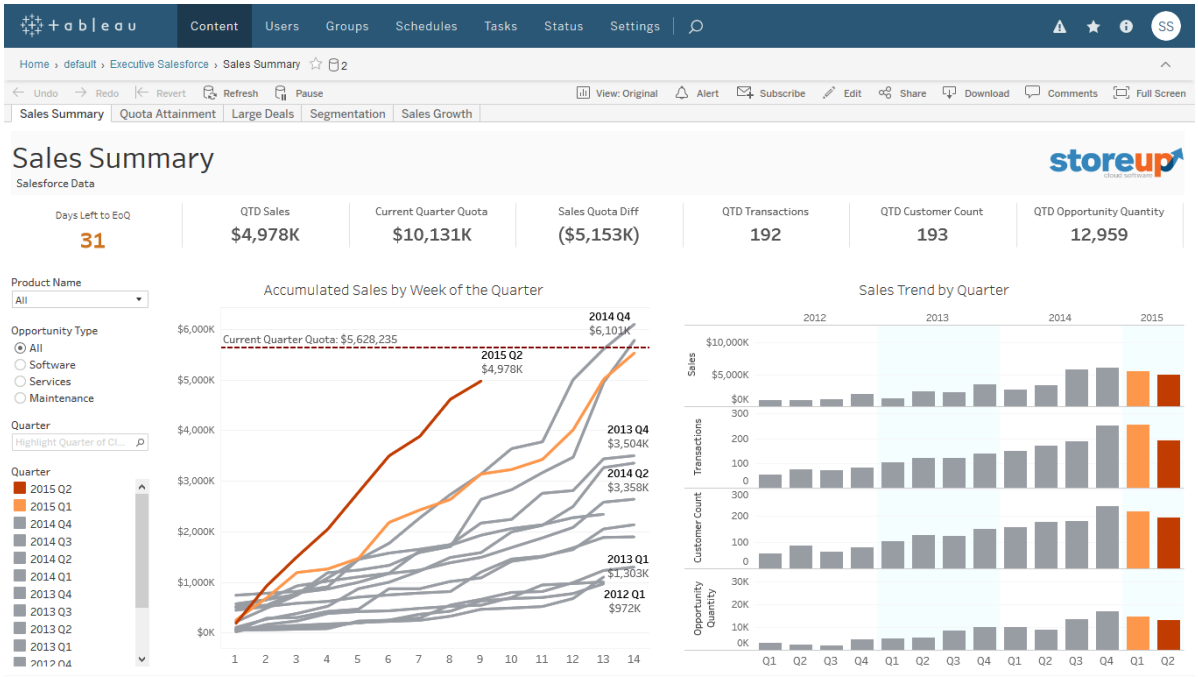


Figure 3.12: Tableau: User-created dashboard.

3.4 Tableau

Tableau [web25] is the commercialized successor to Polaris [80]. It allows users to create interactive visualizations from a variety of data sources. The following discussion refers to the online version of Tableau corresponding to the 10.5 release of the desktop version. At this point only features from a user perspective are described. Underlying principles and approaches will be the subject of later discussion within the related work of the conceptual aspects. Example figures were created from datasets supplied as samples by the software itself.

The general idea for users is to create dashboards that consist of multiple visualizations as shown in Figure 3.12. It consists of multiple so-called *sheets* each depicting a particular aspect of the underlying dataset. In addition, users are able to add static content like headings and controls to apply additional filters to the data (shown on the lower left-hand side of Figure 3.12). Upon changing any value within this filter, the visualizations on the dashboard are adapted on the fly. For this purpose, neither data nor the actual visualization is transferred, but a series of drawing commands in form of a JSON [39] serialization, which are then executed on the client's machine.

To design such dashboards users first have to create a workbook. A workbook represents a visualization project consisting of multiple datasets and dashboards. The dataset overview is shown in Figure 3.13. It is possible to upload files in various formats including Excel [web31] and text-based formats like TSV [web32]. Furthermore, it is possible to establish live connections to cloud-based databases of various vendors using, e.g., SQL [81]. On-premise databases can be

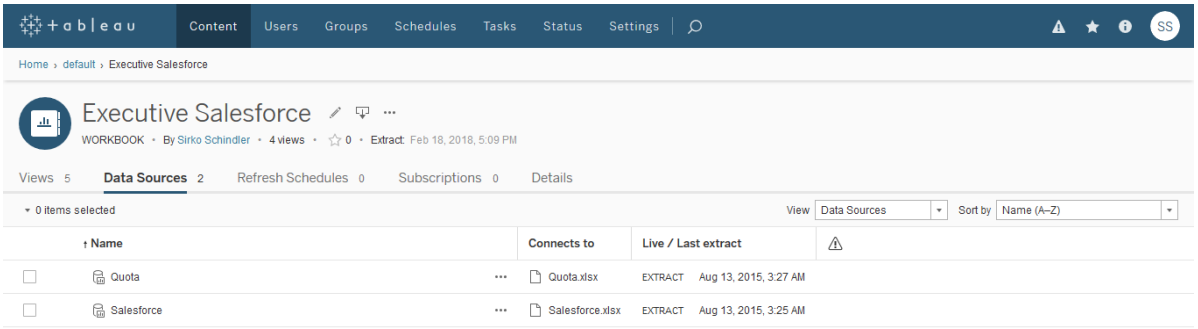


Figure 3.13: Tableau: Datasources.

accessed using Tableau Bridge. Data sources included can also be filtered. This will remove a subset of the rows for all sheets of the workbook. As discussed later, it is also possible to add filters on a sheet level, which will only not affect any other sheet beside the assigned one.

The interface to create or edit a particular sheet is given in Figure 3.14. In the left column, the available datasets are listed – in the example, these are “Quota” and “Salesforce”. Below the datasets, the list of available fields is shown. This list always corresponds to the selected dataset above and is separated into dimensions, measurements, and parameters along the OLAP naming conventions [82].

In the second column and the top row, fields are assigned to aspects of the visualization, which are shown live in the remaining area. The main parts here are “Columns” and “Rows” on top as well as “Marks” in the second column. Fields can be dragged into any of these shelves. The “Columns” and “Rows” shelves refer to the horizontal and vertical partitioning of the visualization. The actual result depends on the current visualization chosen as well as the data types and order of the respective fields. In the example of Figure 3.14, both fields are numeric and are mapped to the scales for the x- and y-axis. If, for example, a categorical field is prepended to the column shelve, the horizontal axis is first split into separate panes according to the values of that categorical field. Within each pane, the data is filtered according to that value and the resulting visualization is shown. For more details, see the respective parts of Chapter 7.

In the “Marks” of the interface, fields can be mapped to other aspects of the visualization. From the drop-down menu, the general mark type of the visualization can be chosen. If the automatic option is selected, Tableau will determine the shown visualization by the fields assigned to rows and columns. For example, a date field in columns and a measurement in rows will lead to a line chart. Other options (cf. Figure 3.16) allow users to explicitly select the mark type. In general, the mark type directly corresponds to the visualization shown. So, e.g., the mark type “Line” results in a line graph and the “Bar” type in a variation of a bar chart. Some mark types behave differently depending on other options set. Per default, a tabular heatmap is shown, if the mark type “Square” is selected and some measure is assigned to color. Assigning another field to size, however, changes the visualization to a treemap.

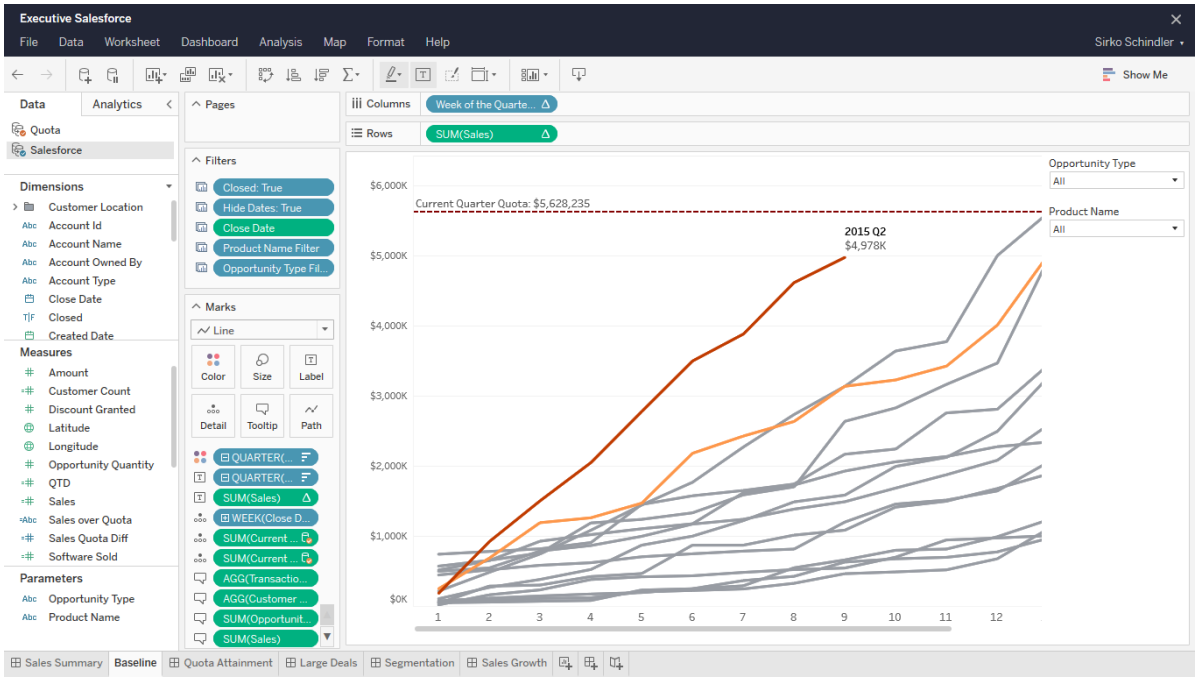


Figure 3.14: Tableau: Editing interface.

Additional information can be encoded using the options below the mark type. Some options are not applicable to all mark types. So, e.g., an area mark type does not allow for additional information to be encoded in size or a path to be added. Assigning fields to the details type allows separating marks according to that field. This can, for example, transform a bar chart into a stacked bar chart including one more dimension. By default, Tableau adds a tooltip to each data point in the visualization. It shows the values encoded in the graph and, if added, all fields assigned to the tooltip type. Finally, the path option allows connecting separate marks by a path, whose order is determined by the assigned field. An example from the Tableau documentation is the connection of individual measures on a map to show the route hurricanes follow.

Users are also able to define their own calculated fields that depend on the fields as given by the data sources. To this end, Tableau provides a number of functions spanning standard operations on numbers, strings, and dates. Some operations like the Pearson correlation can only be applied to data coming from particular sources. Here, the interface to create such calculated fields is text-driven as shown in Figure 3.15.

The filter pane allows adding filters on particular fields to just this view. The respective characteristics of the filter depend on the type it is applied to. Date and number fields allow defining a range of values to be retained, while for categorical fields the values can be chosen from a list. In addition to being predefined by the author, filters can also be presented to users. This allows to interactively explore the underlying data at runtime of the dashboard. Figure 3.14 shows these user-accessible filters on the right-hand side.

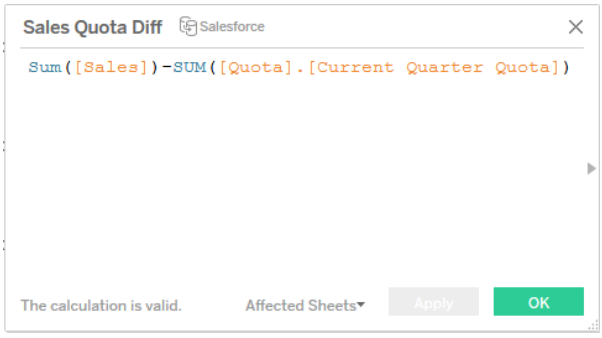


Figure 3.15: Tableau: Calculated fields interface.

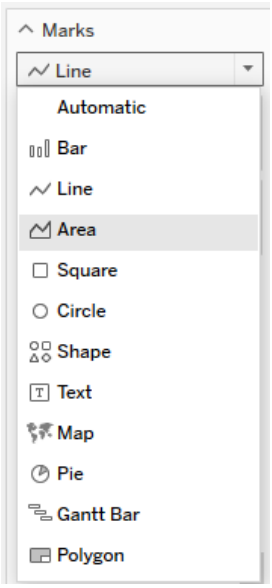


Figure 3.16: Tableau: Mark types.

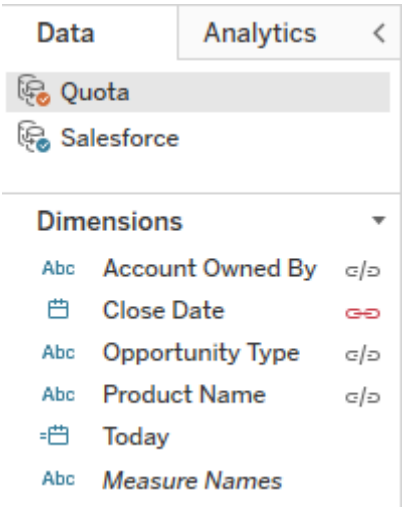


Figure 3.17: Tableau: Joining datasets.

When working with multiple datasets in a sheet, the one first added is designated the primary dataset and highlighted using a blue icon next to the dataset. Other datasets are considered secondary and have an orange icon (cf. Figure 3.17). Two datasets can be joined by first selecting a secondary dataset and then choosing the respective fields. Candidate fields for the join are automatically selected by their respective labels. Figure 3.17 shows the respective dimension list for “Quota”. In total, four fields have been determined as possible join conditions. In the example, only one of them, “Close Date”, is active. Field labels can be changed. So if the source datasets do share values in fields of different labels, these differences can be mitigated. If the values differ, they can be adjusted by creating a calculated field.

3.5 Jupyter Notebooks

Jupyter Notebooks [web26, 83] are a collection of tools to enable so-called computational notebooks. The underlying ideas can be traced back at least four decades to Knuth who wrote in 1984 [84]: “Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want the computer to do”. He implemented that idea into a system called `WEB` that allowed for extensive documentation of code using formatted comments within the code itself. Subsequently, code and documentation could be separated and compiled individually leading to an executable binary as well as a human-readable document. Computational notebooks extend this approach and allow for intermixing executable code fragments with text, images, and other media meant for human consumption. The aforementioned Jupyter Notebooks are one such implementation that is primarily focused on using Python [web39] as the coding language of choice, but allows for other languages as well[3].

The interface of a Jupyter Notebook is implemented as a web application (cf. Figure 3.18). It is comprised of a number of cells and their corresponding output, if applicable. There are three types of cells [web40]: First, code cells contain a code fragment to be executed. Once available, the output of a code cell is shown alongside it. Second, markdown cells contain formatted text, written in Markdown [web41] and using a few extensions. Markdown cells may also include media files like images. Finally, raw cells allow adding content that is not processed by the notebook itself.

Code cells are not executed inside the browser but are instead sent to the server[4]. Here, a so-called kernel executes the respective fragment and returns the results back to the application. In general, kernels are wrappers for the language-specific components exposed by a Jupyter Notebook. Similarly, access to files and the Internet is not run from within the client’s system but is instead executed on a separate machine running the kernel. In particular, this requires all datasets and other dependencies to be available on that machine. The order of execution among the cells is not fixed, so for better or worse users can execute cells in any order.

Through code cells, Jupyter Notebooks offer access to all aspects of the underlying programming language. Consequently, all available libraries of that language can also be used as part of a notebook’s code. Python’s vivid data science community in particular has developed a broad range of libraries[5] over time that can be used to implement various visualization workflows. An example is the SciPy ecosystem [web46] that includes some of the most popular Python libraries for data science applications: NumPy for numeric computations [web47], pandas providing manipulation and analysis tools [web48], and matplotlib for visualization [web49].

[3] Its predecessor, IPython [85], used Python exclusively, but support for other languages has been added since [83].

[4] As of June 2021, JupyterLite has been announced [web43, web44] to remove the dependency on a dedicated server. Instead, notebooks and all of their contents are to be translated to WebAssembly [web45] and thus can be run directly in the browser.

[5] While performance-critical parts are usually developed in languages like C or C++, libraries include Python wrappers that allow for seamless integration into any Python script.

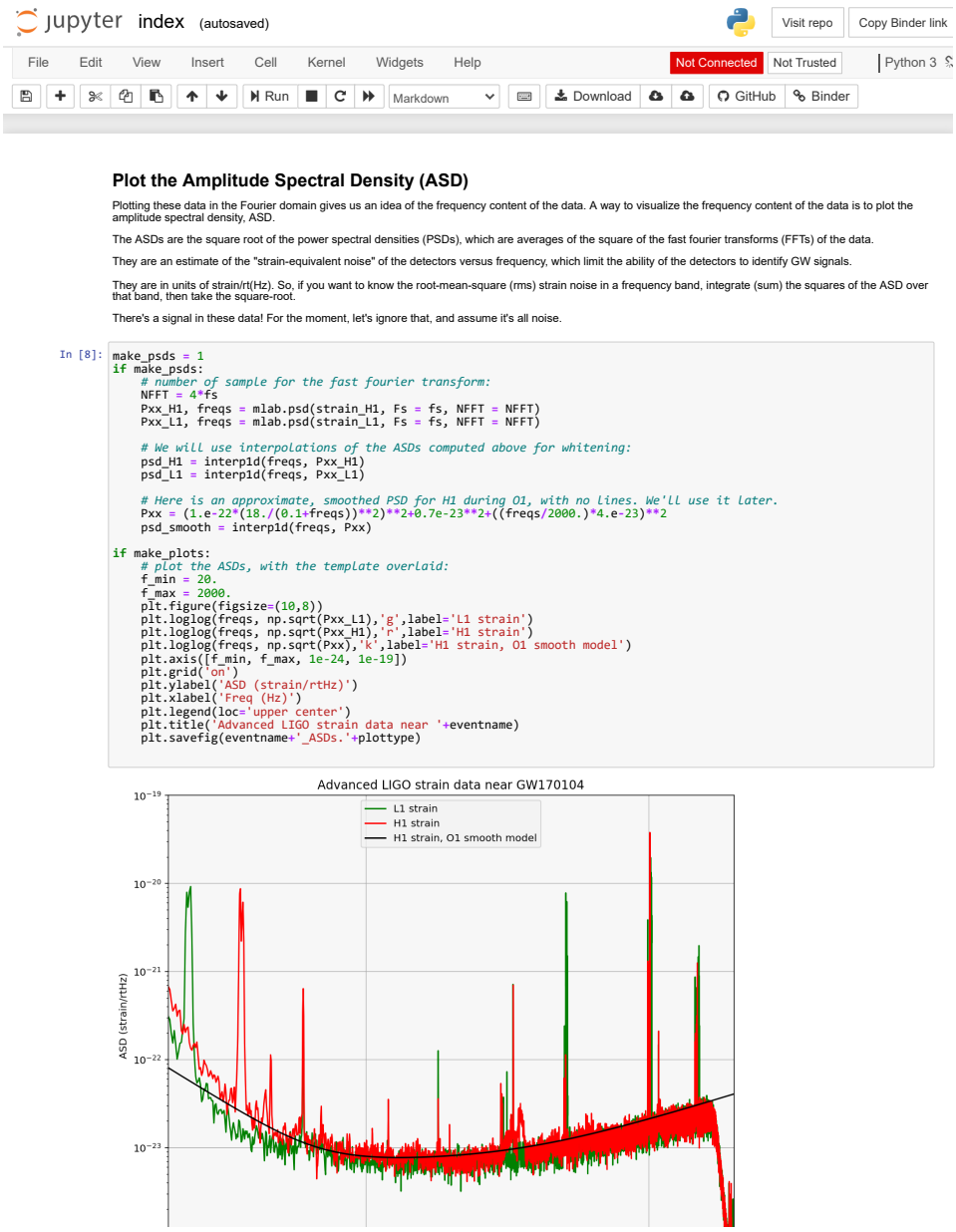


Figure 3.18: Jupyter Notebook: Interface (example from [86, web42]).

```
{
  "nbformat": 4,
  "nbformat_minor": 1,
  "metadata": {
    "kernelspec": {
      "display_name": "Python 3",
      "language": "python",
      "name": "python3"
     },
    "language_info": {
        "name": "python",
        "version": "3.6.0"
    },
    /* ... */
  },
  "cells": [
    /* ... */
  ]
}
```

Listing 3.1: Structure of ipynb: Top level structure (extracted from [web42]).

```
{
  {
    "cell_type": "markdown",
    "metadata": {},
    "source": [ /* ... */ ]
  },
  {
    "cell_type": "code",
    "execution_count": 1,
    "metadata": {
     "collapsed": true
    },
    "outputs": [],
    "source": [ /* ... */ ]
   },
   /* ... */
}
```

Listing 3.2: Structure of ipynb: Cell level structure (extracted from [web42]).

Jupyter Notebooks are stored in form of ipynb files which are in essence a JSON-encoded description of a notebook [web50]. Listing 3.1 illustrates the top-level structure of such files. Besides fields to specify the standard's version used, the data content is divided into metadata and cell description. According to [web50], all metadata fields are optional and may be ignored. Additionally, any extension may add new metadata fields as it sees fit. The content of the actual notebook is defined by the sequence of cells. In general, cells consist of a type, metadata information, their source, and possibly their output (cf. Listing 3.2). Outputs in turn are given by their mime-type and a base64-encoded representation of their content.

Besides using Jupyter Notebooks as a tool (e.g., [86, 87, 88]), scientific interest up to now focused on two areas: extending notebooks with additional ways to interact with data (e.g., [89, 90]) and viewing notebooks as means of reproducible science (e.g., [91, 92]). While notebooks allow capturing the steps leading to a particular result, they do not capture discarded attempts during their development or other aspects of retrospective provenance (cf. Chapter 12). Initially developed for purely script-based environments, noWorkflow was ported to Jupyter Notebooks and allows capturing detailed operating-system-level provenance[6] [91]. Similarly, ProvBook tracks the provenance of a notebook on the level of the notebook itself, i.e. it captures executions of cells together with the current source code, start and end time, and the corresponding output [92]. Both systems allow inspecting the collected provenance data through the notebook itself, with ProvBook also offering means to compare between different runs of a single cell.

Computational notebooks promised to add both reproducibility and a narrative to analysis workflows. Studies on their real-world use indicate that this promise has not yet been fulfilled, though [93, 94]. From a sample of about $1.1m$ notebooks gathered from Github [web51], only about 25% could be executed at all [93]. The most prominent reason for failing to execute a notebook was found to be missing dependencies or an unknown version thereof. Another study focused, among other aspects, on interviews with academic scholars using Jupyter Notebooks [94]. Many of the scholars displayed difficulties providing proper narratives for their notebooks. In particular, the lack of interest on behalf of collaborators did not incentivize proper explanations. Furthermore, real-world notebooks were characterized as messy and in need of "polishing" before being published and distributed beyond personal use.

3.6 Taverna

Under the umbrella of the myGrid project [95, web52], a number of tools are developed to help scientists in their daily work with *in-silico experiments*[7]. The project originates in Bioinformatics, where there is a rather large base of web services providing different functionalities. Due to the lack of proper tools, most scientists used these services either manually by copying the outputs of one service to the inputs of another or they wrote specialized scripts to handle this. One of the tools developed is the Taverna workflow suite [web27, 96], which provides a common model for workflows and means for sharing and reusing them across the borders of individual working groups.

On October 20th, 2010, Taverna became an Apache Software Foundation [web53] Incubator project [web54]. The Incubator project was announced to be retired on February 20, 2020, so no further development under the umbrella of the Apache Software Foundation is expected. There

[6] For more details on noWorkflow, kindly refer to Chapter 12.

[7] The term *in silico* refers to experiments conducted in form of a computer simulation of some sort. It contrasts the terms *in vivo* and *in vitro* which refer to experiments or observations of living organisms or components of organisms respectively.

have been releases on a set of core libraries of Taverna including a command-line client (version 3.1.0-incubating as of June 30, 2016 [web55]), the server (version 3.1.0-incubating as of January 18, 2018 [web56]), and the workflow specification language SCUFL2[8] (version 0.15.1-incubating as of March 3, 2016 [web57]), but there has been no comprehensive release of the Taverna suite during the lifetime of the Incubator project. Consequently, the following discussion will mostly refer to the pre-Apache-era release.

The Taverna workflow model is focused on processors and data links [98]. A workflow specifies input and output ports, which represent the necessary inputs (files, databases, user input, etc.) for this workflow to be executed as well as the generated output(s). Each port is uniquely identified by a combination of the workflow name and a name for the specific port[9]. Processors represent software components, which can either be called as web-services or are present locally in form of scripts. Similar to their respective counterparts in the workflow definition, inputs and outputs of processors are also mapped via ports. As the general interface of workflows and processors is compatible, one can also use pre-existing workflows as part of a new workflow definition. Taverna offers a collection of predefined processors like reading from or writing to files or connecting to a database. Furthermore, there is a tight integration of web-service registries like BioCatalogue [99, web58], which is also a myGrid project.

The connections between processors are given by data-links, which connect an output with an input port using `receivesFrom` and `sendsTo` properties. In addition, there are so-called "control links". Control links allow delaying the execution of a processor until a certain condition is met. One use case mentioned is waiting for asynchronous operations like writing to a database before proceeding with the workflow execution [web59].

An example of a workflow defined using the Taverna Workbench is shown in Figure 3.19. The workflow [web60] takes a list of NCBI [web61] gene identifiers and returns the information about them stored in the KEGG database [100]. Figure 3.19(a) shows the design perspective, which lets users edit their workflows. The upper left part offers options to add new processors to the workflow. The lower left and right parts represent the current workflow. There is both a visual graph representation as well as a tree view, which orders the elements by their respective types.

After executing the workflow, one can inspect the results in the result perspective, which is shown in Figure 3.19(b). The upper left part now offers references to previous runs. In the upper right part within the graph representation of the workflow users may select specific components (in the example the `add_ncbi_to_string` processor is selected). Upon selection, the lower part

[8] Simple Conceptual Unified Flow Language [97].

[9] To be exact, each workflow will have an identifier in the Taverna namespace `http://ns.taverna.org.uk/2010/researchObject/[uuid]/workflow/[workflowName]/` with `[uuid]` and `[workflowName]` replaced with the respective workflow's information. Accordingly ports identifiers are given by the template `workflow/[workflowName]/[portType]/[portName]`, where `[portType]` is either `in` or `out`. Processor identifiers are build in a similar manner using the template `workflow/[workflowName]/processor/[processorName]/` and the same extension for ports.

of the perspective allows users to review input (red triangle) as well as output (green triangle) parameters of this component. That way in case of errors in the execution users can track down the failing component.

Once a workflow is created, it can either be executed locally using Taverna Workbench (cf. Figure 3.19(b)) or remotely using Taverna Server. While in version one of Taverna there had to be a centralized enactment controller, version two dropped this requirement. Each processor is mapped to an object, which starts its processing in a separate thread as soon as data on all input ports is available [98].

Once a workflow is executed, its progression can be observed (cf. Figure 3.19(b)). This includes inspecting all input and output values as well as any intermediate results created. One is also able to export the data concerning a specific run into a so-called provenance bundle. This bundle includes a number of files describing

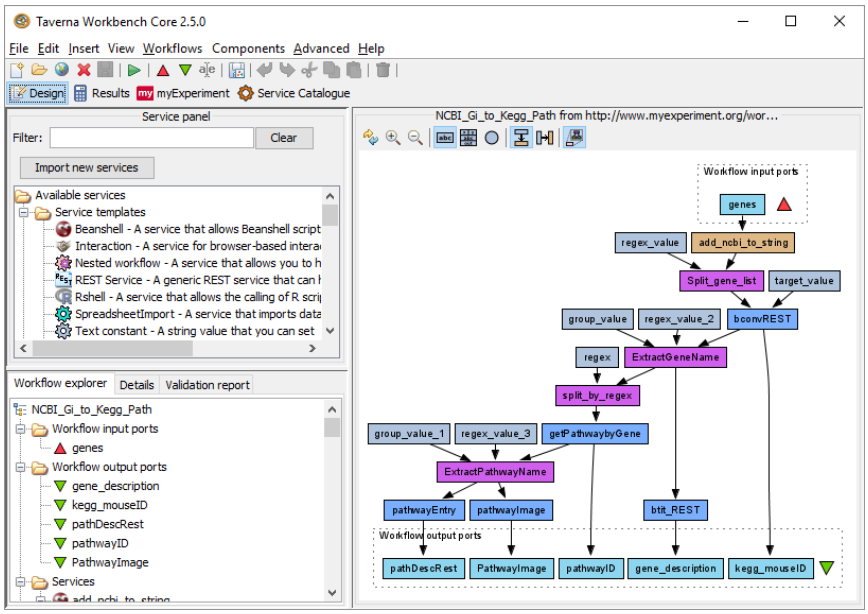
- input, output, and intermediate results as text files,
- the executed workflow as a SCUFL2 [web62] bundle, and
- provenance information about the particular run using a PROV-O dialect [101] (cf. Subsection 12.1.3) in form of a Turtle file [web63].

The PROV-O description of the workflow run connects all stored result text files as part of the workflow with the processors that created them. Furthermore, it keeps track of the execution times of processors and stores hashes for all result files.

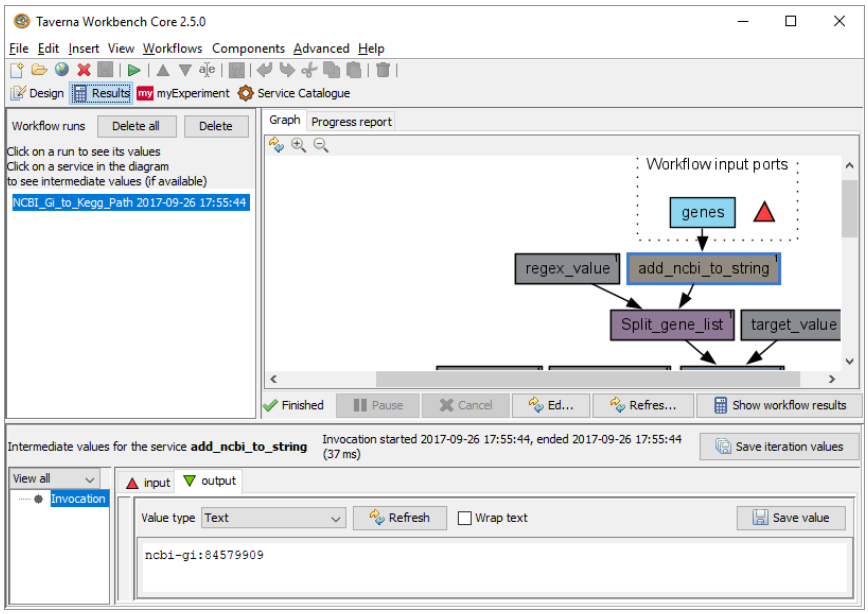
The SCUFL2 [web62] description is stored as an RDF/XML file [web64]. This ontology closely models Taverna's view on workflows as it contains processor and data-link definitions as well as input and output ports for both the workflow and the included processors. Other processor-specific information like required parameters are stored separately per processor in so-called annotation files. In a mailing list entry [web65] Stian Soiland-Reyes, one of the Taverna authors, states that "SCUFL2 is not meant as a generic workflow language, but as a way to generalize the Taverna workflow model" and points towards [98] for further explanation.

3.7 VisTrails

VisTrails [web28, 102] is a scientific workflow management system with an emphasis on workflow provenance. It keeps track of the whole process which leads to a final workflow. This includes branches that resulted in dead ends as well as the eventual "successful" paths. This results in provenance information about the workflow itself in contrast to most other approaches, which just collect provenance information about the execution of a workflow and its respective results. To separate this from other kinds of provenance, this information is called *workflow provenance* or *workflow evolution* [103].

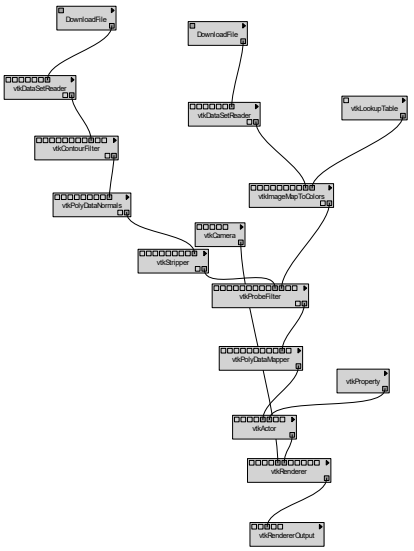


(a) Design Perspective

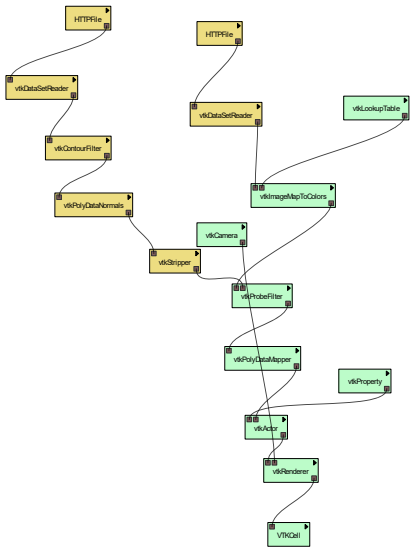


(b) Result Perspective

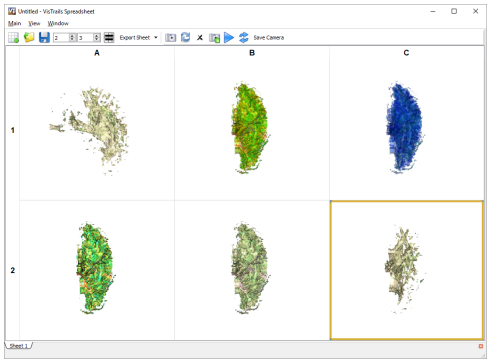
Figure 3.19: Taverna Workbench: Perspectives for an example workflow [web60].



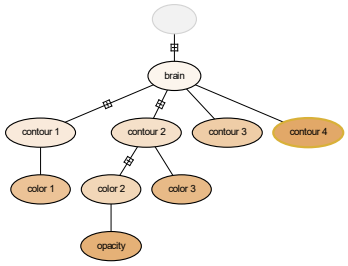
(a) Pipeline.



(b) Execution provenance.



(c) Generated output.



(d) History of workflow evolution.

Figure 3.20: VisTrails: Example workflow
(generated using “brain_vistrail.vt” example bundled with VisTrails).

Another feature is the separation between workflow generation and execution. That way VisTrails opens the possibility of automatic workflow generation by scripts and exploring the parameter space by multiple batch executions of the same workflow using different parameter sets. Figure 3.20 shows the main parts of the VisTrails user interface: A pipeline, the execution provenance, the respective workflow history, and the generated output.

The creation of a workflow is similar to other workflow systems with one exception: A workflow is modeled as a directed acyclic graph, so no loops are possible within a workflow. Furthermore, the workflow is modeled as a data flow. Operations on the data are encapsulated into modules like *HTTPFile* for accessing files using the HTTP protocol [104] or *PythonCalc* as a wrapper for simple arithmetic operations. Most of the bundled modules, however, come from wrappers for

VTK [105, web66]. VTK is a basic visualization toolkit with a focus on three-dimensional graphics. It places an abstraction layer on top of system-dependent graphic interfaces. That way it can easily be ported between different operating systems.

The output ports of modules are connected to input ports of other modules in order to build up an acyclic graph representing the workflow. The final data products, which in most cases will be images of some sort, are then shown in a spreadsheet. This spreadsheet distinguishes VisTrails from other workflow systems. Each run of the workflow may be assigned to a different cell inside this spreadsheet, thus making it possible to compare different visualization modules or variations on the parameters easily. The spreadsheet in Figure 3.20(c), e.g., shows variations on the parameters used, while the underlying dataset and workflow remain the same.

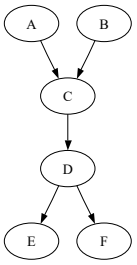
Figure 3.20(b) shows the (execution) provenance perspective of VisTrails. In this example, the nodes, for which cached results have been used, are colored yellow, while others are colored green. Upon selecting a particular node additional information is available like the start of the execution and its end. The provenance information can be exported using PROV (cf. Subsection 12.1.3) or OPM [106]. This export includes only the data for a single execution and not the evolution of the workflow.

VisTrails also includes a repository for past workflow executions. Using this repository parts of a workflow may be replaced with cached data products from previous runs [107]. As a prerequisite VisTrails requires modules to work stateless: The output of a module is only determined by its current input and parameters. Past input data or previous executions must have no influence beyond their own processing. A data product is then defined only by the (sub)workflow and the parameters that lead to its creation. That way VisTrails can replace parts of the current workflow with a cache lookup, if the intermediate result is already known, and thus save execution resources.

Another advantage of saving workflows' creation history is, that the very same process enables for concurrent collaborative work on a single workflow [108]. Different scientists may work on the same workflow at the same time by creating separate paths in the workflow's history. It is not possible to delete or modify parts of the history. This feature of a VisTrails history is called *monotonicity* [108]. It prevents cases when one user removes or alters parts of the workflow another one is currently working on.

Inside a workflow, each module has a timestamp. As it can not be enforced that these timestamps are globally unique, each local repository keeps a mapping between its local timestamp and the global ones. That way it is also possible to change the global timestamps in a commit process without interfering with internal identifiers.

Applying graph algorithms to the workflow graph enables new ways of creating workflows like creating workflows by analogy [109]. Given three pipelines p_a, p_b, and p_c a new pipeline p_d shall be created, so that p_d extends p_c in the same way as p_b does extend p_a. For simplification, p_b has to be derived from p_a. Then a function δ_{ab} is defined to describe the actions that transformed



path	vertex
A → C → D	E
A → C → D	F
B → C → D	E
B → C → D	F
C → D	E
C → D	F
D	E
D	F

Figure 3.21: VisComplete: Path summary (from [110]).
Node `D` is chosen as an anchor node and subsequently all paths ending in `D` are extracted. These are called *upstream paths*. Additionally the *downstream vertices* `E` and `F` are appended to each path.

p_a into p_b. Furthermore, a mapping between the two pipelines p_a and p_c is given. This mapping relies on the graph representation and may only be partial if nodes of p_a do not match nodes in p_c or vice versa. Connecting both functions p_d is generated from p_c by applying δ_{ab} to p_c with respect to the mapping found. Using the same pipeline difference function δ the authors describe a "query by example" approach, where users only define fragments of a pipeline they are searching for and the system searches for pipelines that include or resemble that fragment.

Another approach is taken by [110] with VisComplete. This VisTrails extension provides users with suggestions on how the current pipeline can be extended similarly to the autocomplete functionality in many web applications. To gather data for the suggestions the system analyzes the repository of previous pipelines. Individual pipelines get split into a multitude of paths[10]. An example for such a path summary can be seen in Figure 3.21. This path summary is generated in forward as well as reverse direction with respect to the direction of the edges. This is necessary to allow for suggestions in both top-down and bottom-up approaches. Additionally, information about the in- and out-degree of nodes is saved along with the connection types for pairs of modules as two modules may be connected using different (combinations of) ports. Having extracted paths from many different pipelines, VisComplete maintains an extensive database on how (statistically) likely a node v is to follow a path P.

The suggestions are then generated by analyzing the current pipeline, extracting its paths and repeatedly augmenting these paths with the most likely nodes. This process is recursive in the sense that adding a new node to a path generates a new path that may itself be augmented as well. That way a multitude of possible extensions is generated. In order to restrict the complexity of this calculation and only present a few, but better-suited suggestions to the user, for each extension a confidence value is calculated using Equation 3.1. This confidence relies primarily on the frequency a node occurred in conjunction with a path in the past. In Equations 3.1, `v` represents a newly added node, `P` is the set of corresponding paths and `G` is a workflow graph

[10] Remember that workflow graphs are directed and acyclic, at least within VisTrails. Bearing in mind that a workflow is also finite, there is a finite number of paths per workflow.

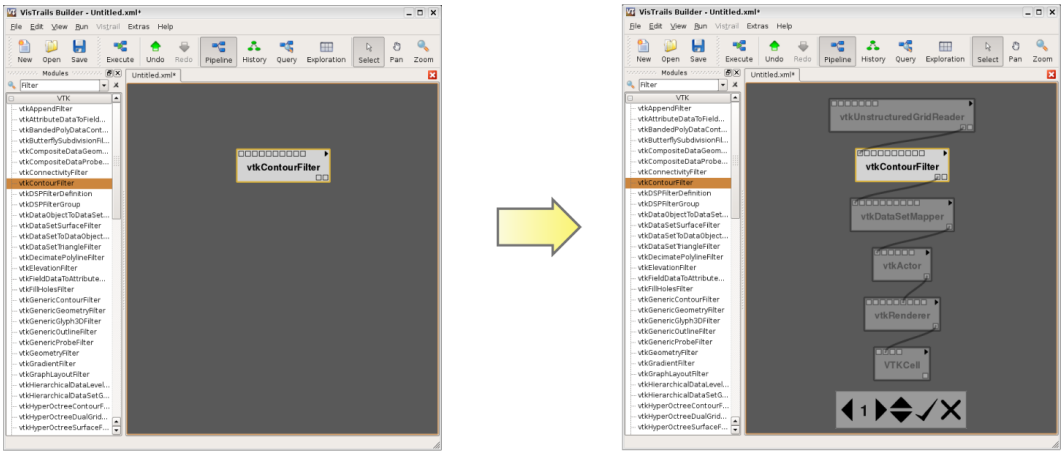


Figure 3.22: VisComplete: Interface example (from [110]). The left-hand side shows a node added by the user. On the right-hand side, a suggestion is given. Suggested completions are displayed as semitransparent nodes. Users may navigate through the set of suggestions and choose the appropriate one using the controls on the bottom of the screen.

created by adding multiple nodes.

$$c(v) = \frac{\sum_{P \in \text{upstream}(v)} \text{count}(v \mid P)}{\sum_{P \in \text{upstream}(v)} \text{count}(P)} \tag{3.1}$$

$$c(G) = \prod_{v \in G} c(v)$$

It is noted that the recommendation might be adjusted by using weights based on user preference. There are, however, no further details given on how these user preferences are modeled or collected. The description of this adjustment is restricted to giving two motivating examples [110]:

> For example, if a user has been working on volume rendering pipelines, completions that emphasize modules related to that technique could be ranked higher than those dealing with other techniques. In addition, some users will prefer certain completions over others because they more closely mirror their own work or their own pipeline structures.

The confidence is then used to prune intermediate results from rare cases. The final set of suggestions is sorted by the calculated confidence values before being presented to the user. The interface to present users with the suggestions is shown in Figure 3.22.

Besides the creation of new workflows, keeping provenance information about existing ones and enabling their reproducibility are major goals of VisTrails [103]. As already mentioned VisTrails not only keeps track of the workflow execution but also the workflow evolution. An

example of such workflow provenance is shown in Figure 3.20(d). Using this mechanism users can go back in the history and explore other variations of the workflow without needing to worry about storing and maintaining the different versions themselves.

One obstacle for reproducibility is the aging of software [111]. With respect to workflows, this means, that some modules for specific steps within the workflow might get updated or replaced. This also extends to cases, where multiple modules are replaced by a single one or the other way around. To reproduce earlier results, one could maintain copies of the complete execution environment including current versions of all tools. Despite the obvious storage consumption, this also prevents the use of newer, improved modules. In VisTrails there is the option to upgrade[11] workflows [112]. In its essence, this approach compares the versions of the modules as given within the workflow to those that are present in the current environment. If there are mismatches, it will try to replace (or upgrade) the version present in the workflow, so that it can be executed in the current environment. Developers of modules may also include so-called "developer-defined upgrades", which gives them more freedom to manage this upgrade procedure including the option to insert multiple modules.

Similar concerns regarding aging can also be raised for data, which is used within a workflow. Files used as input might have changed since the last execution or are not available on the machine. The latter might be caused by the workflow being created at a different location than it is to be executed. For VisTrails, a package has been developed to cope with this problem [113]. By using this package all data files involved in a workflow's execution are put into a repository. Files are given identifiers and version numbers, so they can be uniquely identified. Within the workflow, instead of storing a local path to the file, in addition to this id, a hash value is maintained. So even when the workflow is executed on a different machine, the execution engine has access to the same input data. As all intermediate and final results are stored in that repository, those files can be used to increase the performance of the workflow. Later if in a workflow steps have been changed, instead of executing the whole workflow again, the intermediate result from the last run can be retrieved and the workflow may start with executing the changed parts.

Beyond the core workflow topics, there are efforts towards executable publications using VisTrails [114]. While traditionally publications just feature static figures and tables, the attempt here is to include links to workflows and input data, so that reviewers and other readers may reproduce the original result. The idea is to embed links to workflow and data into each figure. Interested parties can then retrieve them online and execute the workflow to validate the published results. For computationally expensive workflows, a reference to the aforementioned caching mechanism for intermediate results is given [113].

[11] There is also a similar approach to downgrading workflows.

3.8 Discussion

The strategies presented before were developed with different goals in mind and, hence, differ substantially in the features offered. They range from tools specifically designed to exploit the data offerings of a single provider (e.g., Eurostat) to general-purpose tools that can be applied to basically any computational workflow (e.g., Jupyter Notebooks or Taverna). A consequence of these differences in focus is varying support for the requirements previously defined in Chapter 2. A summary of this varying support is given in Table 3.1 and will form the basis for the following discussion. If in doubt, the assessment was made in favor of the respective system in order to not skew discussions towards the approach presented in this thesis.

Search is a rather neglected aspect of most tools. Even when present, it is confined to the data holdings within one system: Eurostat offers basic keyword- and catalog-based search interfaces, but only for its own data holdings. Similarly, in Google Fusion Tables only datasets previously uploaded to this system were available through a keyword-based search or as a means to augment the current table. All other systems fully rely on users identifying proper datasets with the help of tools not part of their own ecosystem. Consequently, also advanced features like searching across multiple providers, inside the primary metadata, or even using multiple datasets in a single response are outside of their scope. However, if datasets have been manually gathered, their combination can be materialized in about half of the tools. With the exception of Google Fusion Tables, this requires users to possibly prepare datasets and subsequently define proper join conditions manually. This may include harmonizing abbreviations in case datasets have been acquired from different providers and/or are employing different coding schemes.

When it comes to the heterogeneity in file formats and structures, most systems are able to cope with a variety of such. As Eurostat focuses on its own data collections, it does not feature support to include additionally provided data and thus is labeled as having no support for other file formats. All other systems support a wide selection of file formats. Similarly, different dataset structures are quite well supported, as most systems provide proper means to transform data to the structure required. There are two exceptions, though: Google Fusion Tables required datasets to be uploaded in a tabular format. Other structures like pivot tables (cf. Chapter 6) are not supported. Spreadsheet software is somewhat in between in this regard. While it is technically possible to load other data structures, converting between different structures is rather cumbersome in comparison. Overall, this aspect of data retrieval is rather well supported, though.

In the next step, modification of data, all systems provide at least the basic means to do so. Only Eurostat is an exception, as it offers merely filtering dimensional values and no other operations beyond that. With regard to getting immediate feedback from operations, workflow engines like Taverna and VisTrails fall somewhat short. Here, the definition of a workflow is decoupled from its execution. So, instead of having the result immediately available, users have to run a workflow in order to retrieve its results. For smaller workflows and datasets, this is usually

	Eurostat	Spreadsheet Software	Google Fusion Tables	Tableau	Jupyter Notebooks	Taverna	VisTrails
Requirement 1: Search across Providers	○	○	○	◐	○	○	○
Requirement 2: Search in Primary Data	○	○	○	○	○	○	○
Requirement 3: Search by Combination	○	○	◐	○	○	○	○
Requirement 4: Materialize Combination-Results	○	○	●	○	◐	◐	◐
Requirement 5: Mediate Abbreviations	○	○	○	○	○	○	○
Requirement 6: Support Heterogeneous File Formats	○	●	●	●	●	●	●
Requirement 7: Support Heterogeneous Dataset Structures	○	◐	○	●	●	●	●
Requirement 8: Translate Abbreviations	●	○	○	○	○	○	○
Requirement 9: Allow for Modification of Data	◐	●	●	●	●	●	●
Requirement 10: Provide Immediate Feedback on Operations	●	●	●	●	●	◐	◐
Requirement 11: Ensure Validity of Operations	○	○	○	○	○	○	○
Requirement 12: Recommend Visualizations	○	○	○	●	○	○	⊕
Requirement 13: Recommend Variable to Artifact Mappings	○	○	○	●	○	○	⊕
Requirement 14: Materialize Visualizations	●	●	●	●	●	●	●
Requirement 15: Track Provenance	○	◐	○	◐	⊕	●	●
Requirement 16: Visualize Provenance Records	○	○	○	○	⊕	●	●
Requirement 17: Share Provenance Records	○	◐	○	◐	⊕	●	●
Requirement 18: Allow for Reenactment of Workflows	○	◐	○	◐	●	●	●

Table 3.1: Support for requirements in common strategies.
●... full support; ◐... partial support; ○... no support; ⊕... support via extensions

more of an annoyance than an actual obstacle. It might lead to situations, though, where large workflows are defined without checking intermediate results. Mistakes during such workflows might not be easily recognizable in the final result and as such can go unnoticed.

Pretty much no support is given by any tool in translating abbreviations or ensuring the validity of operations. Only Eurostat implicitly translates all abbreviations used within their datasets to human-readable labels. In all other systems, this has to be done manually by fetching the respective dictionaries and replacing individual values. The situation for the validity of operations beyond mere checking of data types is even bleaker. The responsibility for validating each operation and applying possible corrections beforehand is completely put on individual users. This opens up workflows to all kinds of inadvertent mistakes without any advice being given.

In terms of visualization, all systems again provide the means to produce reasonable graphs and charts. On the way to those results, though, barely any recommendation or assistance is given. Only Tableau and – using extensions – VisTrails help users in identifying proper visualizations for their data. VisTrails' extensions rely on analyzing past behavior and as such will not be able to recommend new visualizations or mappings. On the other hand, Tableau allows users to pick a subset of columns and proposes a suitable visualization including the corresponding mappings to represent them.

Tracking of provenance is an essential part of workflow systems and, hence, supported by both Taverna and VisTrails. Similarly, through the use of proper extensions, the history of computations can also be tracked on different levels within Jupyter Notebooks. Tableau supports a weaker notion of provenance. It is possible to maintain the connection between a created dashboard and the underlying data sources. As such, it documents the provenance of a dashboard to some degree and allows for sharing or recreating it. However, when the underlying data source changes, also the results change without further notice. This behavior was labeled as partial provenance support in Table 3.1. Finally, spreadsheet software allows documenting basic workflows within a single document[12]. So following the argument made for Tableau, support is labeled partial here as well. As discussed later in Chapter 12, both these approaches can be seen as prospective provenance, whereas the approach taken by Taverna, VisTrails, and Jupiter Notebooks covers both prospective as well as retrospective provenance. The other two systems, Eurostat and Google Fusion Tables, do not support any meaningful way of tracking provenance at all.

The basic means to create visualizations are present within all the systems discussed here. However, there is only very limited support available to use those features without extensive knowledge and expertise. Especially, non-expert users can easily be overwhelmed by the number of features but have next to no guidance in how to use them properly. Some shortcomings stand out in particular. First, pretty much no tool covers the whole visualization lifecycle as previously outlined in Figure 1.1. Only Eurostat can cover the entirety of steps necessary but

[12]A particular sheet can fetch data from other sheets, formulae can be defined and are evaluated at runtime, etc.

does so at the cost of largely restricting their individual capabilities. Second, search interfaces are usually externalized and not part of visualization-supporting tools. Users often have to switch between tools which opens the door for all kinds of interoperability issues. Furthermore, search is usually restricted to the dataset level, neglecting the contents of primary data. It also assumes that the partitioning of data is the same for both producing as well as consuming it. Third, supporting users during the workflow via recommenders and other assistants is overall a quite rare trait. With the exception of BI software (Tableau) and some extensions in workflow management systems (VisTrails), no system offers to support its users throughout the process. Finally, provenance-related features are mostly restricted to workflow engines or, with limitations, full-fledged programming languages. Both these options are usually only used by professional users, so provenance is neglected for those not sharing this level of expertise. In summary, the basic means to create meaningful visualizations are available for everyone in theory. However, in practice, non-expert users are confronted with quite some obstacles and receive next to no support in making the right choices along the process.

Part II

Dialog

Chapter 4

Approach

After reviewing existing solutions that cover (parts of) the workflow outlined in the reference model of Figure 1.1, now the approach taken within this thesis shall be described. This includes core decisions made on the conceptual level but postpones their impact on a technical level to later chapters. Similarly, more detailed analyses of the individual components are subject to the following chapters and are only referenced here.

Figure 4.1 provides a high-level view of the workflow that is to be supported: At the start, an information need leads to a first corresponding dataset via *Search*. This dataset is then subjected to several *Modifications* resulting in another dataset. Through a process of *Visualization*, this later dataset is, in turn, converted into its final representation in form of a graph. *Provenance* information is kept throughout all of these steps to document the process. Each requirement of Chapter 2 refers to one or more of these activities or the system as a whole. Consequently, the remainder of this chapter will first consider generic system aspects, before discussing each activity individually. The chapter will close with some final considerations and a summary of the approach.

4.1 Overall System

Objective 1 calls for a unified platform. This can be interpreted in at least two different ways. The Unix community, among others, favors a collection of independent tools, each particularly well suited for a very specific task [115]. The inputs and outputs of said tools shall be designed to be interoperable with yet unknown other programs. This approach most certainly contributed to the success of Unix-like operating systems and their ecosystems, as it lets users combine existing tools into new toolchains rather quickly. However, the sheer mass of available tools can quickly become intimidating, especially to novice users. A different strategy is to integrate the available

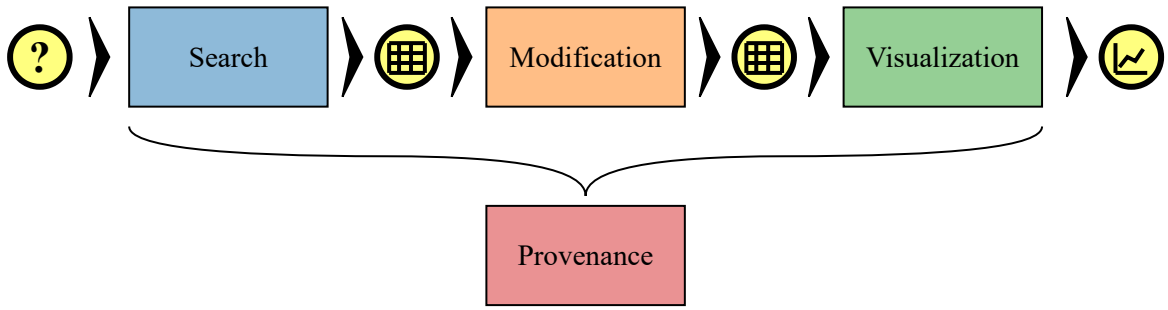


Figure 4.1: Conceptual workflow overview. Entities (circles) are transformed through activities (rectangles).

functionality into a single, oftentimes graphical, user interface. Examples can be found among large web platforms like Facebook [web67] or the online office suite provided by Google [web68]. Here, a unified user interface provides access to a variety of functionalities. With all parts of the interface sharing the same design philosophy and interaction paradigms, new sections of the portal can be understood rather quickly based on past experiences.

The intended audience for the developed system is envisioned to be among novice and intermediate users (cf. Requirement 19). Consequently, a low barrier of entry is essential. Here as well, integrated web platforms shine, as they do not require users to locally install certain tools and possibly figure out their respective dependencies, but provide immediate access to the intended functionality. For these reasons, the system described here will use a web interface to expose access to the main components established previously: search, data modification, visualization, and provenance.

4.2 Search

One core purpose of the system is to seamlessly work with data from multiple, independent sources (cf. Objective 2). Depending on the point in time when data is integrated, systems are generally classified into physical data integration (data warehousing) or virtual data integration [116]. In data warehousing, data is processed in so-called “Extract, Transform, Load” (ETL) pipelines to be accumulated in the namesake warehouse [45]. During this process data from all sources is converted to adhere to a single schema which can then be easily queried from a central location. On the other end of the spectrum, virtual data integration delays the primary data integration until a query has been issued [117]. The query is then broken up, translated, and relayed to the respective data providers, before the partial results are combined again and the materialized result is returned.

The supposed environment for the proposed system is characterized by an immense number of datasets that differ in their level of detail and also partially overlap in their content. Hence, the creation of a single global schema needed for physical data integration seems to be an

insurmountable task. While this would enable the combination of all datasets, the majority of those combinations either are meaningless or will never be requested, anyways. Besides the impracticability of storing all the data at a single location, this motivated the selection of a virtual integration approach for the system at hand. The necessity of a global schema is thus replaced with a local one driven by a user-defined query (cf. Requirement 3). Besides substantially reducing the number of involved datasets[1], the target schema will also be given by the query itself.

For enabling search requests across multiple providers (cf. Requirement 1) the decision is again between a centralized and a decentralized approach. Both approaches provide a unified interface through which queries can be issued. In decentralized query processing or federated search [118], the query is distributed to the presumably independent search engines of each provider. Individual results are subsequently merged and transparently returned to the user. In contrast, a centralized system will accumulate a metadata catalog over the datasets provided by each source and answer user queries from this knowledge base.

The decision between both approaches primarily depends on the capabilities of the providers' search engines. For the system at hand, we require search capabilities that include the primary data of datasets (cf. Requirement 2). As this is a barely available feature at the moment [119], a centralized approach is chosen going forward. Depending on the capabilities of the respective data provider, the system will either harvest or crawl the available datasets[2] and consolidate the acquired metadata into a common model stored within a metadata repository.

The acquired metadata has to contain information of the datasets' primary data, in order to enable corresponding search queries. This information is intended as a summary of the primary data and not a copy of it, so reasonable simplifications are made to reduce the overall amount of metadata stored. The tabular primary data in question can be structured into columns (cf. Chapter 6), which present themselves as a useful level of aggregation for this purpose. In the proposed metadata model (cf. Chapter 8) for each column, its range will be stored. For quantitative and time columns[3] this translates to the respective minimum and maximum values, whereas for categorical columns a list of occurring values is maintained. The inherent order of quantitative and time values allows omitting specific values in favor of such a range. Categorical columns do not allow for such aggregation. However, unlike their quantitative counterparts, individual values are oftentimes repeated rather frequently within a single dataset and thus a list of appearing values likely is small enough to be stored.

A side effect of this collection of categorical values is the support for mitigating differences among the data providers (cf. Requirement 5). As stated before, the coding schemes among the data providers might differ, resulting in possible cases of synonymy and homonymy among

[1]The majority of datasets will be unrelated to the user query and as such can be discarded early on.

[2]We will use the term *harvesting* when cooperative data providers offer designated web services or files for fetching metadata in a machine-accessible form. On the other hand, *crawling* denotes the process of extracting metadata from any other source, such as the datasets themselves or websites intended for human use.

[3]A detailed discussion of the data types mentioned throughout this thesis can be found in Chapter 5.

used abbreviations. Using the respective code lists, the abbreviations used by the respective data providers can be mapped to one another. Within the presented approach these mapping rules are realized using techniques of the Semantic Web [29]. Abbreviations are translated to Internationalized Resource Identifiers (IRIs) for the respective code list. If there are no IRIs for the respective code list publicly available, custom ones will be minted in a private namespace and augmented with labels taken from the respective code list. In the next step, IRIs referring to the same concept are linked to one another using `owl:sameAs` statements, thus providing the required mappings. Whether this mapping is conceived automatically or manually as well as the precise manner of doing so, is beyond the scope of this thesis. For the remainder of the thesis, the existence and maintenance of such mappings is assumed as a given.

The previous process does not only relate different code lists but also provides the means to translate abbreviations to human-readable labels (cf. Requirement 8). This can be broadened further by including links to authoritative sources from Linked Data Cloud sources [120] like Wikidata [121, web69]. Assuming such extended mappings are available to a sufficient degree, these outside sources can provide alternative labels and multi-language support. Besides the ability to customize the output of datasets seamlessly to the current user's locale, this also broadens the scope of keywords supported in the search interface.

On the back of the outlined metadata repository, the proposed system is able to provide primary data search. In the past, two general strategies have been proposed for this: Keyword-based search considers the dataset as a document and then applies techniques from document retrieval [31]. In a nutshell, this looks for occurrences of values and ranks the datasets with more occurrences over those with less. However, this will only provide a set of matching datasets and leaves anything else to the user. Another strategy is generally referred to as "query by example" in the database community [122]. Here, users describe their information need by providing a rough table structure including column headers and possibly values. Systems will then try to complete this table based on the datasets they have access to.

The proposed system follows the latter approach and allows users to define a set of columns required to be present in the result (cf. Figure H.18). Paralleling the aforementioned range definitions of datasets' metadata descriptions, users can further specify a range of values for each column. These ranges are interpreted as constraints on the desired content of the result. Consequently, missing ranges for specific columns put no further restrictions on possible results.

To answer the posed query, the system will first fetch a list of candidate datasets from the metadata repository (cf. Chapter 10). Candidates are those datasets that share at least one dimension and one measurement column with the request[4]. For each of these candidates, the overlap with the current request is computed and the largest overlap is chosen as part of

[4]The notions of "dimension" and "measurement" follow their respective definitions in the OLAP community, as discussed in Chapter 6.

the result[5]. Subsequently, this overlap is removed from the request definition and the process is repeated until no more suitable candidates are available or the request has been fulfilled completely. If there are various iterations of this cycle, the posed query is answered using the entirety of chosen candidates (cf. Requirement 3). Furthermore, if multiple datasets are involved in providing an answer to the request, a sequence of join- and union-operations among the respective overlaps is created to represent their combination. This sequence forms a workflow that can directly be executed once the query response has been confirmed and possibly augmented by the issuing user (cf. Requirement 4).

Actually fetching the data is done through the use of proper wrappers [123] that match the data providers' data format and structure. The information about which wrapper is suitable for a particular dataset is retrieved from the respective metadata description and, hence, does not require any user interaction (cf. Objective 2). For better maintainability (cf. Requirement 20), wrappers are split into two groups building upon one another: The first group parses the source data format as given by the provider to access individual data elements (cf. Requirement 6). A second group then converts the given data structure into the format used throughout the system (cf. Requirement 7). Here, categorical values are given by a label and their IRI which is created in the same fashion as in the metadata descriptions before. Furthermore, on the column level, the system will maintain more crucial metadata: All columns link to a describing concept as well as their current range. Quantitative columns will also include the unit of measure used, while time columns preserve their respective precision.

4.3 Modification

Once a dataset or a combination of multiple datasets is properly loaded, users can inspect the primary data through a familiar, spreadsheet-like interface (cf. Figure H.12). From this interface, they can issue further commands to adapt the dataset like aggregating, filtering, or adding derived columns (cf. Requirement 9). Operations are always carried out over the dataset as a whole to simplify the interface and certain operations like the conversion of units[6]. This does not pose a restriction on the general capabilities of the system. The modification of individual entries could still be facilitated via conditional operations, that apply a change only when the preconditions are met, e.g., the row number is equal to a given target number. The results of an operation are immediately shown in the interface giving users direct feedback of their actions (cf. Requirement 10). In cases where the result of an operation does not match the expectations, corresponding undo and redo features are provided.

[5] Computing the overlap can involve aggregations to reduce the number of dimensions within the candidate dataset to match the ones in the request. The particular kind of aggregation function is of no consequence here and is left to the user. Kindly refer to Chapter 10 for more details.

[6] Kindly recall that the unit of measurement is maintained on a column level. Hence, changing the unit requires the conversion of all individual values.

This mode of applying operations also allows for easier validation. Compared to data structures in non-restrictive spreadsheets, the data within the proposed system stays coherent and most properties are shared between all cells of a column. Consequently, operations can be validated on the column rather than the individual cell level consuming significantly fewer resources. The proposed architecture will focus on the consistent handling of units as one important example of ensuring the validity of operations and thus the resulting datasets (cf. Objective 3 and Requirement 11). As mentioned before, the unit is shared for all cells of a quantitative column – a property maintained by applying bulk operations on datasets as a whole. So instead of ensuring unit consistency on a cell level, any operation can be validated before execution on the column level. This is far less computationally expensive and thus improves the overall responsiveness of the system (cf. Requirement 10).

Ensuring unit consistency should be handled as transparent and unobtrusive to users as possible (cf. Requirement 19). In particular, in case of an impending mistake, the system should automatically take care of fixing the root cause, if possible at all. Only if such a fix is infeasible, should a user request be rejected with an error message. With respect to units of measure, missing conversions can be added automatically, while dimensional errors still need user intervention. However, due to limited precision arithmetics, any additional conversion increases the chance that rounding errors may distort the results. Hence, the proposed system will analyze formulae used within operations and inject only the minimum number of conversions necessary (cf. Chapter 9).

4.4 Visualization

The final step of the workflow is the visualization of the results. For this work, the recommendation of suitable visualizations (cf. Objective 4) will be limited to leveraging available data characteristics. Other factors like the semantic meaning of the content or specific preferences towards certain visualizations or representations are generally domain- or even task- and user-specific, whereas the techniques and tools described here are intended to be domain-independent. The creation of a knowledge base describing the individual peculiarities of individual domains, tasks, or users is beyond the scope of this thesis and thus only considered on a conceptual level.

In general, recommending items from a given collection can be seen as a two-step process: First, inapplicable items are removed from consideration, before, second, the remaining items are ranked according to their suitability in the current situation. For visualizations, the notion of applicability boils down to the compatibility with respect to types (cf. Chapter 5) between a dataset's columns and a visualization's visual artifacts. A visualization is applicable to a dataset if each mandatory visual artifact can be mapped to a specific column of the dataset. The reciprocal condition, each column of the dataset can be mapped to a visual artifact, is an example of an aspect to influence the ranking but not prevent the general applicability of the visualization. Assuming that all remaining columns in the dataset are still significant within the context of

the current task[7], visualizations capable of representing a larger subset of columns or even all of them should be ranked higher. Another factor to influence the ranking is the multitude of values. Some visual artifacts or combinations thereof can only be used to represent a certain number of states (e.g., see [49]), before the quality of a visualization starts to deteriorate up to the point of being completely useless. The number of states is no single threshold beyond which a visualization is no more applicable. In fact, the transition from a well-received visualization to an overburdened one is quite fuzzy.

The recommender proposed within this work (cf. Chapter 11) tries to accommodate all these factors. It relies on a black-box model of visualizations that are characterized by a set of optional and mandatory requirements. These requirements are formalized in possibly multiple descriptions for a visualization that lists the visual artifacts of said visualization alongside their characteristics (cf. Chapter 7). Here, the type and role of the visual artifact mirror the respective notions for datasets (cf. Chapter 8). Furthermore, the number of representable states is given by a threshold function to describe the aforementioned fuzzy transition between suitable and unsuitable inputs. After users assembled their dataset, they are able to call upon the visualization component. The recommender analyzes the given dataset and attempts to create suitable mappings for all visualizations. In the process, inapplicable visualizations will be determined and subsequently hidden in the interface. For applicable visualizations, a score is computed to represent the degree to which columns could be mapped and the requirements of the visualization could be fulfilled. Users can then select from a list of possible visualizations ranked by these scores (cf. Requirement 12), adjust the mapping if needed (cf. Requirement 13), and trigger the generation of the selected visualization (cf. Requirement 14). Allowing users to adjust the mapping between columns and visual artifacts accounts, e.g., for cases where visual artifacts are not distinguishable based on the mentioned criteria like the x and y axes in a scatter plot.

4.5 Provenance

For each result, no matter if intermediate or final, information about its provenance has to be maintained (cf. Objective 5). In its most general form, this includes all resources used, the actions taken, as well as the information accessed to create a particular result[8]. The information accessed is rather hard to determine, unless users are confined to a very restrictive environment. Hence, a narrower interpretation of provenance will be adopted subsequently. It includes only information available within the proposed system but omits any failed attempts. Hence, the tracking of provenance is confined to the resources used as well as the successful actions taken to arrive at a certain result.

[7] A column not significant anymore is, e.g., one that only contains one value for all its cells as the result of a filter operation.

[8] In some approaches like [web28, 102] (cf. Section 3.7) it even includes unsuccessful attempts that preceded the finally successful one.

The granularity of such tracking follows directly from the need to reenact a previous workflow (cf. Requirement 18). To this end, any action has been documented that modified the dataset and whose result did not get abolished later on. Having in mind that each action will operate on the level of columns (cf. Chapter 6) and result in a new dataset, both the cause and the level of detail for individual provenance records are defined. For each dataset, the system will create an individual record including the action that led to its creation, its parameters, possible inputs in form of other datasets, and the time of execution. The latter becomes especially important if external input datasets are not under version control. In such cases, the time of access is the only way to document the state of said datasets short of maintaining a complete copy. Following the trail of provenance records across multiple intermediate datasets allows tracking the provenance of a specific result (cf. Requirement 15). In an ideal world, this could be extended to each external dataset as well, thus providing a complete record up until the initial means of data collection or creation. However, in practice, the assumed independence and heterogeneity of providers, for now, will make this impossible in almost all situations.

The accumulated provenance records serve at least three purposes: First, they can be exported in a machine-readable, interoperable format (cf. Requirement 17). By adhering to commonly accepted standards (cf. Chapter 12) this enables other tools to leverage or analyze the workflows created. In the long run, this may also contribute to overcoming the aforementioned limitations in provenance tracking across providers. Second, based on the exported provenance record the system will later be able to reenact a given workflow (cf. Requirement 18). Leveraging the documented actions as well as parameters and assuming the initial data is still available from the respective provider, each step can be executed once again and thus replicate the original result. Finally, the provenance records can be used to visualize a workflow to users (cf. Requirement 16). Following commonly adopted visual metaphors like flow charts or workflow graphs, this enables users to revisit and inspect the steps that led to the current result (cf. Section 13.5).

4.6 Final Considerations

In a system such as the one outlined above, there is a number of components that implement different variations of a single task. For example, all included visualizations (cf. Appendix F) in an abstract sense perform the same kind of transformation from a dataset to an image. Components like these are also the most likely candidates to see new additions and changes over time. Besides the already mentioned visualizations, this includes in particular the wrappers needed to access specific data providers. The prototypical implementation (cf. Chapter 13) will gather available variations of a specific component in so-called repositories. A repository is a collection of variations for a specific type of component, usually consisting of a standardized description and an implementation. At runtime, this repository will be accessed to determine which variations are available and offer them as options in the respective user interfaces. This way, developers

may add, e.g., a new visualization without necessarily having a complete understanding of the system as a whole (cf. Requirement 20). A loose coupling like this also improves the overall maintainability of the implementation as single components can be developed and tested without affecting other parts of the application.

4.7 Summary

This chapter outlined the proposal made in this thesis. This includes the description of measures taken to address the requirements identified previously (cf. Chapter 2). Table 4.1 summarizes this effort by juxtaposing requirements and the corresponding proposed solutions. The latter will be described throughout the remainder of this thesis. For now, Table 4.1 will point to the respective chapters for a more detailed discussion of concepts and to individual sections of Chapter 13 for a description of implementation details as part of the developed prototype, Yavaa.

Requirement 1: Search across Providers	Centralized metadata store (cf. Chapter 8 and Section 13.2) Wrappers for data access (cf. Chapter 13)
Requirement 2: Search in Primary Data	Tabular data model (cf. Chapter 6) Primary data summary in metadata (cf. Chapter 8)
Requirement 3: Search by Combination	Query by example and fulfill request from multiple datasets (cf. Chapter 10)
Requirement 4: Materialize Combination-Results	Search results as executable workflow (cf. Chapters 10 and 12)
Requirement 5: Mediate Abbreviations	Map categorical values to semantic concepts (cf. Chapter 8)
Requirement 6: Support Heterogeneous File Formats	Wrappers for data access (cf. Chapter 13)
Requirement 7: Support Heterogeneous Dataset Structures	Wrappers for data access (cf. Chapter 13)
Requirement 8: Translate Abbreviations	Labels via mappings to semantic concepts (cf. Chapter 8)
Requirement 9: Allow for Modification of Data	Column-oriented operations (cf. Section 13.3)
Requirement 10: Provide Immediate Feedback on Operations	Interactive user interface (cf. Section 13.9)
Requirement 11: Ensure Validity of Operations	Column-oriented operations (cf. Section 13.3) Implicit management of unit consistency (cf. Chapter 9)
Requirement 12: Recommend Visualizations	Data-driven visualization recommendation (cf. Chapter 11)
Requirement 13: Recommend Variable to Artifact Mappings	Filtering of vis. by data types (cf. Chapters 5 and 7) Ranking of vis. by data characterstics (cf. Chapters 8 and 11)
Requirement 14: Materialize Visualizations	Modules to materialize selected visualization mappings (cf. Section 13.7)
Requirement 15: Track Provenance	Implicit tracking of applied operations (cf. Chapter 12)
Requirement 16: Visualize Provenance Records	Visualize provenance in a flow chart (cf. Section 13.5)
Requirement 17: Share Provenance Records	Serialize provenance according to PROV-standards (cf. Chapter 12 and Section 13.8)
Requirement 18: Allow for Reenactment of Workflows	Re-execute provenance records (cf. Section 13.8)

Table 4.1: Functional requirements and corresponding solution approaches.

CHAPTER 5

DATAMODEL - DATA TYPES

Data types are one of the essential parts of data management. They classify the processed values, so the system can easily decide how to present them or whether certain operations are valid. This classification is generally driven by one of two factors.

The first approach focuses on the characteristics of the values themselves. One of these characteristics can be, whether the set of values exhibits some kind of order. Another option would be to ask for the existence of a meaningful definition of mathematical operations like summation or multiplication.

The second approach to classification is driven by the kind of operations the data is subjected to or how it is to be stored. An example is the introduction of the type Boolean, where the goal is to support logical operations like *AND* or *NOT*. This results in two distinct values, *true* and *false*, required to support said operations. In practice, these values may take multiple lexical forms like *T*/*F*, *yes*/*no*, or any other binary set. However, during processing all these variations map to the same two values.

Oftentimes both approaches end up with similar results, but one major exception to this is the distinction into fixed-point and floating-point arithmetics in many systems. This is not driven by data characteristics but by technical limitations in current hardware.

This chapter shall examine existing type schemes and derive one to be used henceforth in this thesis. As discussed later, these data types are a cornerstone in the translation between an actual dataset (cf. Chapter 8) and the visualization to represent it (cf. Chapter 7).

5.1 Related Work

Spreadsheets and relational databases are a common way to store tabular data. Although there are plenty of different systems, all of them support roughly the following data types in one way or the other [web70, web71, web72, web73, web74, web75, web76, web77]. Most of them only differ in the respective labels used and the respective length or precision.

Number Numerical values. Generally distinguished in integer, fixed-point and floating-point.

Text Strings of arbitrary characters.

Boolean Two possible values *true* and *false*. Sometimes implemented as an integer, where zero represents *false* and all other values stand for *true*.

Date and Time Values describing a specific point or period in time. Oftentimes distinguishing between dates, times, and combinations thereof.

Collection Lists of values of the same type.

Null Special keyword representing a missing value.

The last data type mentioned, *Null*, denotes a missing value as defined in SQL [81, Section 3.1.1.12]. It is explicitly included in OLAP systems as well as in relational databases. Spreadsheets, on the other hand, do not model this as a separate type or value, but instead, just leave the respective cell empty. The actual reason why a certain value is missing is not captured by *Null*. For this reason, users oftentimes chose to use other designated values instead of null. For a measurement that can have no negative values, e.g., users may specify to use multiple negative values to indicate why a measurement was not possible. Examples may be a malfunction of the measuring device or simply a forgotten measurement, which is labeled as such later on.

In SQL three different subtypes are defined for the data type *collection*: *collection*, *array*, and *multiset* [81, Section 4.4.5.1]. These types are used to represent an (un)ordered list of values of the same data type. Spreadsheets also define a data type *array* but omit the unordered equivalents. Oftentimes spreadsheets also do not define a separate data type for date and time but use a numerical timestamp which is then displayed to the user with a suitable formatting.

Another classification of data types is given by Stevens in [124]. For the classification four operations are used: determining equality, rank-ordering, determining equality of differences, and determining equality of ratios. For rank-ordering, no assumption is made on the distances between two values. Especially it is not assumed, that all values can be equidistantly placed on a linear scale. To illustrate the difference between the latter two, think of temperature scales like Celsius and Fahrenheit. Given four temperatures ω_1 to ω_4 the test for equality of differences is independent of the used scale. So, if $\omega_1 - \omega_2 = \omega_3 - \omega_4$ holds true in one scale, it will do so in the other. For the respective ratios, however, $\frac{\omega_1}{\omega_2} = \frac{\omega_3}{\omega_4}$, the respective zero point in both scales

is important. As both scales use a different zero point (0 °C = 32 °F and 0 °$F \approx -17.78$ °C) temperature values, in general, are not subject to determining the equality of ratios. An example where both, determining equality of ratios and differences, can be applied are lengths, where there is a shared absolute zero point. If both of the above equations hold true, they will do so no matter if one uses a meter or feet as a scale. Using these four operations, the following four data types – called scales in this context – are defined:

Nominal For two values their equality can be determined.
Example: Color names, which possess no inherent order.

Ordinal There is an ordering defined over the set of possible values.
Example: Rating schemes like the Five-Star-Scheme, where different options are not placed on an absolute scale but can be compared to one another.

Interval Differences between values can be tested for equality.
Example: Temperature scales like Celsius or Fahrenheit, where no absolute zero point is given.

Ratio Ratios between values can be tested for equality.
Example: Length measurements, which share an absolute zero point.

Another different classification of data types can be found in [125]. While nominal and ordinal scales are adopted as described before, interval and ratio scales are subsumed as *numerical scales*. Numerical scales, however, have been distinguished into two types:

Discrete Numerical The set from which values are drawn is either finite or countable infinite.
Example: Counts of any sort like the number of offspring in a litter or the value of various coins.

Continuous Numerical The precision of values is only bound by the ability to measure them and not by the scale that is used. In theory, there can be as many different values as there are real numbers.
Example: Most physical measurements like weight, length, or temperature.

This distinction has also been adopted by Tableau [80, 126], which presents among other things an algebra to map multidimensional data to visualizations[1] that also includes dates and times as separate types:

C Categorical (discrete and continuous)

Cdate Categorical date (date or date&time)

[1]This grammar will be described in more detail in Chapter 7.

Q Quantitative (continuous)

Qd Quantitative dependent (measure)

Qi Quantitative independent or Qdate (dimension)

Date and time are not exclusively listed as subtypes to either categorical or quantitative, but depending on context can be part of both.

VizAssist [127] follows the general consensus of data types by using *numeric*, *ordinal*, *nominal*, and *time* as data types. In addition, some more specific data types are included that convey a special meaning when used within a visualization.

imageURL Representing the location of images that can be used within a visualization.

URL A general hyperlink is either added directly into a visualization or used as a target for other elements serving as anchors.

country A geographical location as used, e.g., in maps.

source, target Used to represent graph nodes and edges.

Furthermore there a graph-specific versions of the common data types *numeric*, *ordinal*, and *nominal* for both nodes and edges: *Nodenumeric*, *Nodenominal*, *Edgenumeric*, and *Edgenominal*.

5.2 Discussion

Reviewing both implementations as well as more theoretical classifications it seems for most areas there is a broad consensus regarding the data types. There is a common distinction between numeric and non-numeric data types. Date and time are oftentimes seen as a separate data type, although sometimes regarded as a subtype of either numeric or non-numeric depending on the respective usage.

The numeric data types are distinguished into discrete and continuous. SQL follows a slightly different path here: numeric types are either *exact numeric* like integer or fixed-point or *approximate numeric* like floating-points. The reasoning for this distinction is mostly based on the technical limitations of the adopted IEEE standard 754 [128] for floating-point arithmetics, which on occasion introduce errors due to the limited precision involved. A simple example to illustrate the shortcomings of this standard is given in Inequality 5.1.

$$(0.1 + 0.2) - 0.3 > 0 \tag{5.1}$$

Using any exact computation the left-hand side evaluates to zero, but as none of the involved numbers has an exact representation using IEEE 754, the result of the computation evaluates to about $5.551e-17$.

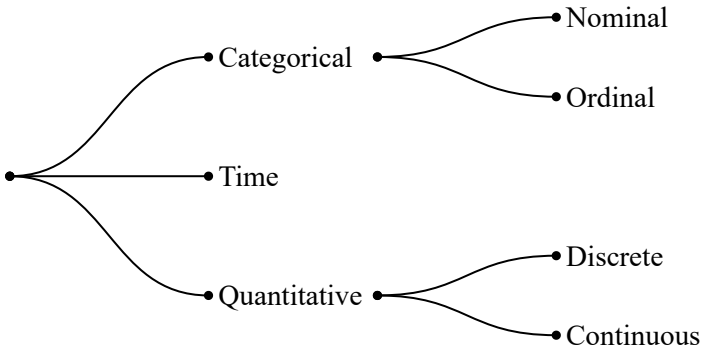


Figure 5.1: Hierarchy of data types.

For non-numeric data types, many implementations omit the distinction between *ordinal* and *nominal*. One explanation for this is probably the difficulty to actually capture the order of multiple values. Most of the time, values are just seen as a sequence of characters that carry no inherent ordering. Systems implementing a separate *ordinal* data type have to resort to some other source for obtaining the ordering of values.

Some types can also be restated as a constraint on those basic data types. *Boolean* in SQL, e.g., could be stated as a non-numeric type restricted to two values *true* and *false*, which in some contexts carry special meaning. Similarly, most of the special types in VizAssist can be seen as non-numeric types that carry some additional information that is exploited in the mapping to a visualization.

Here, one can see a major decision that has to be made when deciding upon data types: Should the data types be based on the characteristics of the data values – as done by Stevens [124] or Tableau [80] – or are the possible operations performed on them the driving factor – like done in VizAssist [127] or SQL [128][2]? The result of this consideration directly affects the number of data types to be defined. As can be seen from the examples mentioned above, the number of data types is much higher when driven by operations rather than characteristics.

5.3 Approach

Following the argument at the end of Section 5.2, within this work data types will be determined by data characteristics rather than applied operations. Interpretations of values that go beyond their inherent characteristics will be achieved in conjunction with additional meta-information as stated in Chapter 6. The data types used are arranged into a hierarchy as shown in Figure 5.1.

Categorical Values represent certain entities. No arithmetic operations can be applied.
Superset of nominal and ordinal data types.

[2]The decisions in SQL are mostly driven by the efficiency of storage rather than the efficiency of computations. For the sake of argument here storage of data is seen as one kind of operation.

Nominal Values represent separate entities. No further restrictions apply.
Example: Color names.

Ordinal Values represent separate entities and have an inherent order.
Example: Ratings like “good”, “mediocre”, and “bad”.

Quantitative Values describe a measured observation of some sort in form of a number.

Continuous Values are drawn from a certain range of values. Any value of that range is valid.
Example: Most physical measurements like length or frequency.

Discrete Values are drawn from a set of numerical values that is either finite or countable infinite.
Example: Counts.

Time Values describe a point in time[3]. The precision of what is considered a “point in time” may vary, though.
Examples: Specific months like *April 2010* or specific times like *1969-07-20T20:17:40*[4].

The definitions of the nominal and ordinal data types follow the ones given by Stevens [124], while the definition of the quantitative types is modeled after the ones known in statistics as presented in [125]. Regarding time the decision was made to introduce a distinct datatype similar to SQL [81]. The main reason is that time values exhibit characteristics of both the other major data types. Values like *April 2010* can be seen as standalone entities which would result in a categorical data type. On the other hand, some arithmetic operations like average or difference are applicable which would put them into a numerical data type. So instead of having to decide upon context which data type a time value belongs to like done in Tableau [80], within this work time is seen as a separate data type.

Time intervals are not modeled as a separate type but are assumed to be split into two (time) values: One denoting the start and one the end of the respective period. This removes the necessity for an additional type and defining appropriate operations. All operations possible on a specific time interval like calculating duration or ordering by start can be restated accordingly. The above definition of time does also not specify the precision of a given point in time. Socio-economic indicators like “production per month” can be defined using a single time value although they actually represent a period of time. So following that approach, each time value itself actually represents a period of time.

[3] In contrast, measuring the span of time a certain observation lasts is considered quantitative.
[4] Timestamp conforming to ISO/IEC 8601 [129].

CHAPTER 6

DATAMODEL - TABLES

Each system has to have an internal model of the data and objects it deals with. This model is tailored towards the operations performed on the respective data. So the most frequent operations are sometimes executed more efficiently at the expense of more uncommon ones. Some of these models are closely related so that instead of focusing on just a single model, a system can choose a set of models and use the appropriate model for the task at hand. This, however, comes at the cost of either maintaining both models throughout the execution or transforming the underlying data from one model to the other when needed.

For the system discussed here the main type of data to deal with is statistical, tabular data, while the necessary operations arise from the requirements stated in Section 1.1. Other systems concerned with data processing are using different approaches ranging from relational databases [130] over XML-dialects like SDMX [40] to formats closely modeling plain tables like CSV [131].

In this chapter first, the different models behind those systems are described in Section 6.1. In real-world implementations, the distinctions may oftentimes not be as clear-cut as presented here, though. The attempt is to focus on the fundamental ideas rather than the actual implementations. After a brief discussion in Section 6.2, different aspects of approaches are integrated in Section 6.3 to form the data model used within this thesis including terms and distinctions used throughout.

Metadata descriptions will not be discussed here, but are subject to Chapter 8.

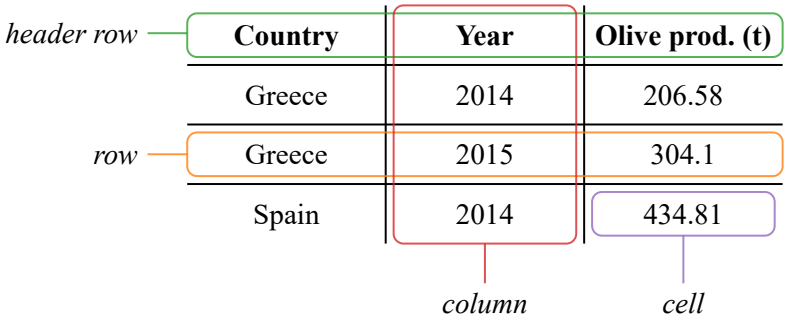


Figure 6.1: Basic table structure; Naming based on [web78]; Data: [data1].

6.1 Related Work

One basic model is a mere table, which is used by, e.g., CSV [131] and similar encoding schemes like TSV [web32]. A table like the one shown in Figure 6.1 is comprised of rows and columns. Individual values are contained within cells. The total of all cells in the same row forms a single record. Cells in the same column represent instances of the same aspect. This aspect is usually given within the first row - the so-called header row. In this row, cells contain the name or label of the column instead of a concrete value. The standards usually make no assumption about the type of data stored within a column. Only certain characters are prohibited or need to be escaped as they are used to separate cells and rows within the persisted data.

Neither CSV nor TSV requires the number of cells per row to be constant within a single document. Both standards, however, do not make any statement, how to handle mismatches between given column labels and the number of cells in a given record.

A modification to this basic scheme is used by so-called pivot tables[1], crosstables, and contingency tables. If the same phenomenon has been observed under varying conditions, e.g., at different locations and times, a table can be restructured as shown in Figure 6.2. Columns describing the conditions will have just a few different values compared to the total number of records in the table, so they are moved to the first row (*column area*) and first column (*row area*) respectively. The actual observations are then placed in the remaining cells (*value area*) so that they share the row or column with their respective conditions. The names row and column area are a result of the fact, that the values there apply to the whole row and column respectively. The cell in the first row and column serves no distinct purpose but is used for column titles of the row and column area, the overall table title or left empty altogether.

The benefits of this approach are the reduced redundancy and thus the more compact display. Furthermore, additional columns or rows can be appended to also include aggregates of the given data like, e.g., the sum or average for certain conditions. The label *ALL* in the row or column area

[1] Credited to Pito Salas by [132] and confirmed by himself [web79].

Country	Year	Olive prod. (t)
Greece	2014	206.58
Greece	2015	304.1
Spain	2014	434.81
Spain	2015	540.48

Olive prod. (t)

Year \ Country	Greece	Spain	ALL
2014	206.58	434.81	641.39
2015	304.1	540.48	844.58
ALL	510.68	975.29	1485.97

column area

row area

value area

Figure 6.2: Transformation of a plain table to a pivot table. Naming based on [132]; Data: [data1].

represents the aggregate over all possible values of the respective aspect. By the same logic the cell, that is addressed by *ALL* in both row and column area, represents the aggregate over all values of that table.

The basic approach of pivot tables is extended by Gray et al. in [82]. Columns are distinguished into dimensions and measurements or facts. Dimensions characterize the conditions of an observation, while measurements describe the actual observations. Where pivot tables used the dimension columns to span a two-dimensional area, here a multi-dimensional cube, also called OLAP[2] cube, is created. Each dimension of that cube represents one-dimensional column of the data and the actual cells within the cube are filled with the respective values from the measurement columns. An example of such an OLAP cube is given in Figure 6.3. The values of the dimensions can be part of a hierarchy. If a column describes, e.g., a location of some sort, then the levels of the hierarchy may be city → region → country → continent. This allows for easier access to subsets of the data, so in the given example one can easily query for the data relating to a single region without having to explicitly state which cities are actually part of that region. Like in pivot tables most aggregates are precomputed to speed up analytical queries.

[2]On-Line Analytical Processing [133].

Country	Year	Usage	Olive prod. (t)
Greece	2014	table use	206.58
Greece	2014	oil	1574.48
Spain	2014	table use	434.81
…	…	…	…

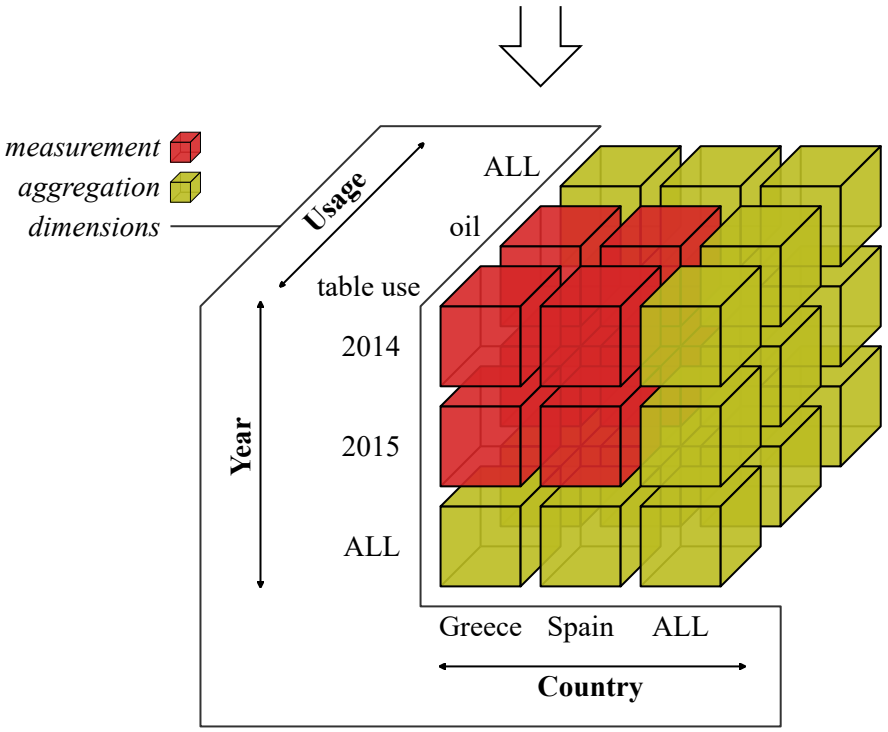


Figure 6.3: Transformation of a plain table to an OLAP cube; including precomputed aggregations, but omitting dimension hierarchies;
Data: Eurostat [web1] dataset [data1].

SDMX[3] [40, web12] also uses the OLAP cube as its data model. However, SDMX allows for just a single measurement column, which is labeled by *OBS_VALUE*. To allow for multiple measurements within one dataset there is a specialized dimension called *measure dimension* [134, Section 4.3.1.1]. The possible values of this dimension have to be defined in a concept scheme and specify the kind of measurement given in this record. Another addition to the basic data model is attributes. They provide the metadata for the given dataset like its title or the unit of measurement used. It is also possible to create so-called groups to attach attributes to just a subset of the data.

Another way to represent tabular data is in a relational model as described by Codd [130]. Data structures are captured in relations, which are defined as a subset of the Cartesian product over a number of sets. These sets are also called domains and correspond to the columns of the tabular model. They contain all possible values that could appear in the respective column. The set of tuples in a relation constitute the records or rows in a table. There is no inherent order for the tuples and due to the set nature of a relation, there can be no duplicates within a relation.

A subset of the domains of one relation that uniquely identify a record is called *primary keys*. In modern relational databases, there can only be one primary key per relation. However, in his initial publication, Codd states [130]: “ Whenever a relation has two or more nonredundant primary keys, one of them is arbitrarily selected and called the primary key of that relation ”. This implies that there might be multiple primary keys for a single relation in the generic model. An additional requirement to primary keys is that they do not contain any superfluous domain, which is not required for the unique identification. Even with this restriction it, is not guaranteed that there is only one primary key for a given relation. In SQL [81, Section 4.6.5.4] this notion has somewhat changed: Codd’s primary keys correspond to SQL’s *unique constraints*. One unique constraint can be chosen to act as a primary key for a table. Hence, there is at most one primary key per table.

Using these primary keys a relation is able to cross-reference entries of the same relation or other relations. Domains referencing the primary key of another tuple are called *foreign keys*. Using this technique the redundancy within a dataset can be reduced by moving repeating patterns in tuples to a separate relation with possibly a short artificial primary key. Instead of repeating the whole pattern multiple times in the original relation, one can then just reference the primary key of the newly created relation. A single dataset in general is now represented by a number of interconnected relations and not as a single monolithic table or cube.

In [50] Wilkinson omits the notion of a tabular data structure of some sort altogether. Furthermore, he states with a reference to Coombs [135] that this notion might even be preventing users from “noticing meaningful patterns in their data”. Instead, he uses a set of so-called *variable mapping functions*, which are required to return a valid value for each index. The returned value is not restricted to be one-dimensional, but can also be a vector of multiple values. In this case,

[3]Statistical Data and Metadata eXchange.

the respective variable is called p-dimensional, where p is the number of elements within the returned vector. As a consequence, the data model basically consists of a set of objects denoting the actual data and a number of variables given by their variable mapping functions to access individual data items.

The presented models themselves make almost no assumption with regard to the used data types. Only columns for which aggregated values are computed are usually assumed to be numeric. Systems implementing any of these models, however, have to restrict the data types they support. A detailed discussion of data types can be found in Chapter 5.

Tables and pivot tables are used by spreadsheets like Excel [web31], LibreOffice [web17], or Google Sheets [web33]. Examples for OLAP systems are Cubes [web80] and Oracle OLAP [web81]. SQL, as defined in ISO 9075 [81], is based on the relational model, but deviates in certain aspects. While, e.g., the relational model prohibits duplicates among the rows of one relation, in SQL the same row can appear multiple times within the same table. Based on SQL a multitude of implementations has been created including, e.g., MariaDB [web82] and IBM DB2 [web83]. Although there is a common standard, most database systems implement their own dialect of SQL. This results not only in a different syntax used most of the time but also in the supported subset of SQL.

6.2 Discussion

Before deciding for or against a specific data model one has to analyze the expected workload and rate the available options according to that. From the requirements presented in Section 1.1 one can deduce that there will just very few different kinds of operations on the actual data:

Aggregations Records can be grouped based on some equality criterion. For each group, a single aggregated record can be calculated (cf. Subsection 13.3.2). A simple example could be a two-column dataset with hourly measurements of temperature. One could now ask for the average temperature per day. This would group all records referencing the same day and then calculate the average over all temperatures within one group. The result would be a list of records consisting of two values: one stating the day and one the respective average temperature.

Additions or Updates Based on existing values and a given set of constants for each record a new value is computed, which is then either appended to the record or replacing an already existing value within the record. On a table-level this correlates with the creation of a new column or the update of an existing one. An example here would be the conversion of units of measurement like, e.g., from degree Celsius to degree Fahrenheit (cf. Chapter 9).

Removals Similarly to adding new properties, one can also remove those not needed anymore. If a certain property just adopts the same value for all records in the dataset, this property could be removed and the respective value-added as part of the metadata.

Joins Two datasets may be combined based on some equality criteria (cf. Subsection 13.3.3). Here, the task is twofold: First, matching records from both datasets have to be found. In a second step, for each pair of matching records, a new record is added to the result, which integrates the properties of both source records. Due to the equality of some properties, the number of properties in the resulting record is usually smaller than the sum of properties in both source records. Following the example, one could want to add precipitation measurements to the collection of measured temperatures. Both datasets identify their measurements by a timestamp, which can be used as the equality criterion for the combination. A resulting record would then use the timestamp from one dataset (which one does not matter, as both timestamps should be equal) as well as the temperature from the first datasets and the precipitation from the second one.

Filter The last operation, finally, is filtering the given dataset. Based on constraints for a subset of properties for each record will be decided whether to remove or keep it. A simple example would be to narrow down a given dataset to include just values from a specific week while dropping data for all other dates.

The average distribution over these operations can not be determined beforehand and may vary considerably between different tasks, so any weighting would be arbitrary and, hence, not constructive at this point. In consequence, the judgment here will only be qualitative and not quantitative.

Subsequently, pivot tables will be considered as OLAP cubes of lower dimensionality for the sake of simplicity. Furthermore, the set characteristics of the relational model increase the complexity of a possible implementation. This already leads to the situation, that most systems labeled “relational” in fact just implement the standard defined by SQL and not the actual relational model, which in turn basically uses the table model as present before. Wilkinson’s object-oriented approach will place an unnecessary overhead on rather homogeneous data. It may be an interesting conceptual model but seems unfit for direct implementation. Wrapping each tuple inside a separate object will consume substantially more memory and storage as opposed to structures that can exploit the data’s common structure. This leaves two options: the OLAP cube and a basic table.

The primary goal of OLAP cubes is data analysis [45], so the data structure is quite optimized for that use case. Even if aggregate values can not be precomputed, the multidimensional structure allows for easy grouping of values and hence easy computation of aggregates. In contrast, in a table to compute an aggregate function a system has to traverse the whole table and collect records per group[4].

When adding new columns to an OLAP cube, for each column it has to be determined whether the respective column is a dimension or a measurement. While adding measurements does not change the overall structure and thus is rather easy, adding a new dimension requires restructuring the whole cube. In general, this will also introduce many empty cells, which will unnecessarily consume memory or make more advanced techniques necessary. In a table on the other hand adding a new column is a rather simple task in both cases. A similar argument can be made for removing certain columns.

Joining datasets can be restated as adding new columns to one dataset based on the columns in the other dataset. Basically, the function to compute a new column is a mere lookup using the equality criterion in the other dataset. Hence, the argument made for adding columns before also applies here.

In OLAP cubes filtering a dataset corresponds to extracting a sub-cube by putting constraints on certain dimensions. In a table, this again requires a full table traversal. For each record, it has to be decided individually whether it will be part of the result. For filters, that do not only put constraints on individual columns, but use functions based on multiple columns, the problem can basically be restated into adding a new column, that includes the resulting value of the filter function, and then do simple filtering as described above. This approach results again in the same argument already made for adding columns.

6.3 Approach

As a result of the discussion in Section 6.2, it seems that for the task at hand a table is the data structure best suited. Although losing out on the performance of aggregation functions, it offers higher flexibility for most other operations, where OLAP cubes only perform well under certain conditions. Having decided on the general model in the remainder of this section the details of the employed model shall be discussed.

Following the approach taken in SQL, within a dataset, the metadata is separated from actual data. Hence, the actual table consists just of records and has no header row. The metadata includes besides information about the dataset as a whole also information about each column present[5]. For each column, the following aspects are represented within the metadata: the concept, the role, the data type, and the scale.

[4] Assuming at this point that there are no index structures that support clustering of values.
[5] For a comprehensive overview over collected metadata see Chapter 8.

Concept The *concept* of a column replaces the title as described before. The title is a mere string created by the dataset author to describe the contents of a column. This gives way to every kind of ambiguities within a single or between multiple datasets: Usage of synonyms, abbreviations, and mere typos make integration of datasets quite difficult. On the other hand, a concept is a link to an entity within the Linked Data Cloud [120]. This way ambiguities about the authors' intent are removed and datasets can easily be combined with one another. As those entities are often attributed with labels of different languages the possible translation of column titles according to users' preferences is a welcome side-effect.

Role The *role* of a column is used according to the distinction between dimension and measurement as used within the OLAP community. This distinction roughly correlates with dependent and independent variables used in statistics [125]. A *dimension* (independent variable) describes the circumstances of an observation and is usually stated by creators before the actual measurement. On the other hand, *measurements* (dependent variables) denote the actual observation. As an example take the continuous observation of temperatures at different locations. The produced dataset will have (at least) three columns: location, time, and temperature. As values for time and location are fixed by the observer beforehand, those two columns represent the dimensions of this dataset. In contrast, the temperature observed states the measurement.

Data Type For each column no matter whether being a dimension or a measurement a data type is specified. For the discussion of possible data types, refer to Chapter 5.

Scale To interpret numerical values a scale has to be given, which they are measured in. The same holds true for time values, which need a reference system like the Gregorian Calendar or the Islamic Calendar. On the other hand, categorical values do not require a specific reference system. They can use a code list, though, to map from abbreviations used in the data to actual entities.

Another part of the metadata is inter-column dependencies. A simple example is shown in Table 6.1. Here, the first two columns form a hierarchy, which associates countries with the respective continent they are a part of. Other possible relationships are functional transformations where, e.g., measurements are classified according to some scheme. A simple example would be a grading scheme where points reached in an exam are mapped to the final grade for that exam. Knowing such relationships can improve the selection of visualizations and the mapping of columns to artifacts which will be described in Chapter 11.

Continent	Country	Population
Africa	Burundi	11179000
Africa	Comoros	788000
...		
Europe	Belarus	9496000
Europe	Bulgaria	7150000
...		

Table 6.1: Example for a hierarchical relationship between columns.
Estimated population in selected countries as of 2015.
Data: [data2].

CHAPTER 7

VISUALIZATION DESCRIPTION

One of the main objectives of this thesis is to allow users to easily create visualizations. Instead of having them choose between myriads of possible visualizations, the system shall make valid suggestions which visualizations might be fitting best for the current dataset (cf. Chapter 11).

However, to do so, the system first needs a formal model of visualizations that is in line with the data model presented in Chapter 6. This model has to bridge the gap between the rows and columns of a given dataset and the marks and other visual variables as present in visualizations. Hence, there is a strong argument, that both models should share their vocabulary wherever possible. This especially includes the aforementioned general data structure and used data types.

In literature, there are two general approaches to this problem: A constructive one and a descriptive, monolithic one. The constructive approach starts out from a given dataset and successively maps its components to separate visual variables, which in conjunction form the visualization. In the process, several transformations are applied to make one fit the other. Representatives of this approach are, e.g., Bertin [49], Wilkinson [50], and Vega [136]. The monolithic approach takes a given visualization and then describes its various characteristics. These characteristics include functional properties like required data types as well as non-functional ones like the goal that is pursued. This second approach is used by a variety of sources, examples being VizAssist [127] or VizBoard [137].

A more hybrid approach is taken by Polaris [80]. While constructive at a first glance, it significantly reduces the number of options users can choose from. While this loses quite a lot of the power of arbitrary combinations of components, the risk of creating an inferior result also diminishes.

This chapter will present the aforementioned approaches in detail and discuss their applicability to the work at hand. Afterward, the approach used throughout this work will be explained. Finally, some examples will be given to illustrate the expressiveness of the proposed model and prepare for the selection process as presented in Chapter 11.

7.1 Related Work

One of the most influential contributions to formalizing statistical visualizations is ***Semiology of Graphics - Diagrams, Networks, Maps*** by Bertin [49]. Based on Bertin's experience as a cartographer and geographer, it provided the foundation to formalize the visualization of (geospatial) data. At some points, this includes precise details on the extent of some properties, that are not justified any further (cf., e.g., Table 7.2). Limiting the scope to static, planar graphs "on a sheet of white paper", six so-called *visual variables*[1] are identified (cf. Figure 7.1). The restriction to two dimensions especially excludes any kind of motion and a third spatial dimension like a relief. Together with the two planar dimensions, the following visual variables form the basis of a map[2] or statistical graph.

Size relates to the area a mark is occupying.

Value describes the lightness of the coloring.

Texture[3] denotes the filling of an area with repeating, geometrical shapes.

Color specifies the hue of an area's filling.

Orientation refers to the primary direction of a mark or its texture.

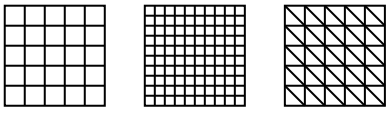
Shape corresponds to the geometric shape of a mark.

Each variable has one of three different kinds of *implantations*: point, line, or area. Although points and lines, in theory, cover no area, in practice they occupy a certain space in order to be visible. Some of the expressive power of a retinal variable differs according to the implantation

[1] The term *retinal variables* is used as a synonym.

[2] Bertin addressed cartographers in the first place, so most definitions and examples are geared towards that audience. However, he claims that the statements can be generalized to all graphical systems.

[3] The original French book emphasizes the distinction between texture (French: *grain*) and pattern (French: *texture*). Consider the following example.



While *A* and *B* share the same pattern, they differ in texture. *C* features a different pattern to *A* and *B*. The notion for pattern comes down to which basic shapes were used (here rectangles vs. triangles) and how they are arranged.

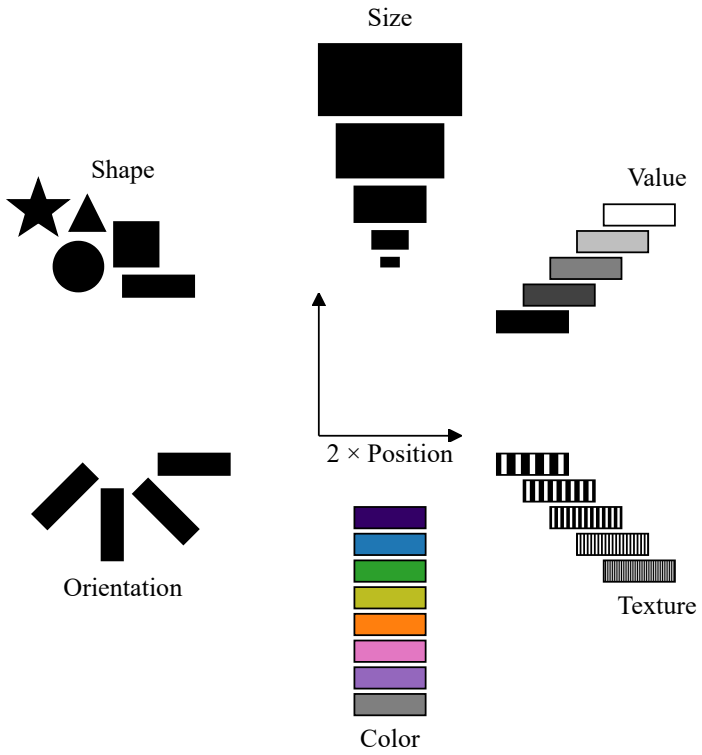


Figure 7.1: Semiology of Graphics: Retinal variables (after [49]).

used to represent it. This expressive power is called *level of organization* by Bertin. To classify the retinal variables accordingly, he first considers only variations within one variable and no combinations thereof.

His first level of variables is called *associative* (≡). This represents the ability of users to group different variations of a variable. An observer is able to identify all marks belonging to the same family. A particular property of an associative variable is, that variations within that variable alone do not cause the marks to appear with different degrees of "power". While all other visual variables are considered associative, size and value are not.

If a variable is *selective* (≠), it is part of Bertin's second level. An observer should be able to isolate all marks of the same category easily within a graph. The goal is to (virtually) remove all other categories and consider only the distribution of a single one. It is argued that for shape too much cognitive effort is required to differentiate between different variants. Hence, shape is not deemed selective, while all other variables are.

Ordered variables (O) form the third level. Their variations form a universally recognizable sequence. Given three variations A, B, and C, this sequence might either be A, B, C or C, B, A, but never, e.g., B, A, C. Size, value, and texture are considered ordered.

The fourth level consists of variables that are *quantitative* (Q). An observer may easily estimate the ratio between two variations of such a variable. Only size meets this requirement.

A summary of the membership for all retinal variables to the levels of organization is given in Table 7.1. Planar dimensions are the only variables that satisfy the requirements for all levels.

	Plan. Dim.	Size	Value	Texture	Color	Orient.	Shape
Selective	●	●	●	●	●	◐	○
Associative	●	○	○	●	●	●	●
Ordered	●	●	●	●	○	○	○
Quantitative	●	●	○	○	○	○	○

Table 7.1: Semiology of Graphics: Levels of organization for retinal variables (after [49]);
Sorted in descending order from left to right.
Requirements met: ● ... yes; ◐ ... partially; ○ ... no.

The difference according to implantation becomes relevant at two points: First, a variation in orientation for a variable represented by an area implantation has a greatly diminished selectivity and thus is discarded. This leaves orientation with only partial support for selective variables.

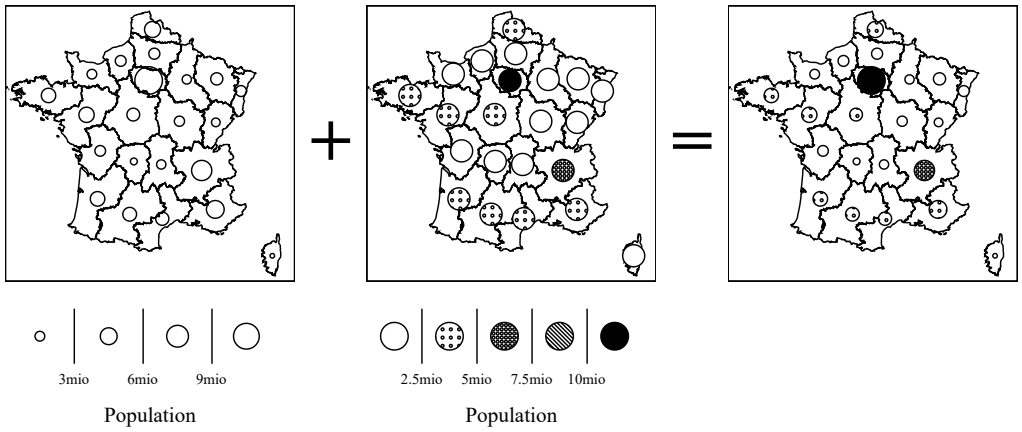
The second aspect is the *length* of a variable. This property is used to describe the number of variations for a variable that can be used in a graph. While this number is unlimited for associative, ordered, and quantitative perception, it is rather small for selective purposes. Here, it describes the maximum number of categories an observer will be able to distinguish. The length for each selective variable also depends on the implantation used. Table 7.2 lists the lengths of variables as suggested by Bertin. However, no reasoning for their specific values is given, so they have to be attributed to his personal experience as a cartographer.

	Size	Value	Texture	Color	Orientation
Point	4	3	2	7	4
Line	4	4	4	7	2
Area	5	5	5	8	

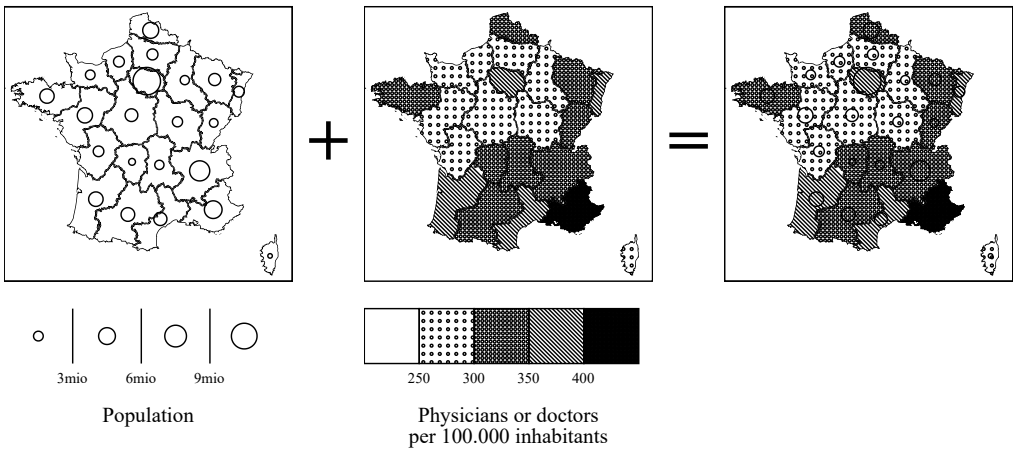
Table 7.2: Semiology of Graphics: Length of retinal variables by implantation (after [49]).

The combination of retinal variables within one mark is split into two groups: Redundant and meaningful combinations (cf. Figure 7.2). In a redundant combination, two or more retinal variables are used to encode the same value. This allows observers to distinguish different values more easily as opposed to the representation by a single retinal variable. The population of French regions in Figure 7.2(a), e.g., can be represented by both size and texture of the mark thus lowering the cognitive effort to understand the visualization. With respect to the level of organization, the combination of retinal variables retains the highest level of its components with the exception of the size-value combination. While size having higher level is labeled quantitative and value is not, their combination remains not quantitative.

A meaningful combination of retinal variables, on the other hand, encodes two or more different values within the same mark. It is noted, that this is used to highlight correlations within the underlying data. In this case, either size or value or both of them are part of the combination.



(a) Redundant combination of retinal variables: size and texture.



(b) Meaningful combination of retinal variables: size and texture.

Figure 7.2: Semiology of Graphics: Combination of retinal variables (after [49]).
Data: [data3, data4]; Map: [data5].

While Bertin [49] focuses on retinal variables and best practices in their application, Wilkinson in ***The Grammar of Graphics*** describes "grammatical rules for creating perceivable graphs" [50] which resemble a workflow from a given dataset to its visual representation. The process is divided into six steps:

DATA maps the input data to variables that can be used later on.

TRANS applies transformations to the variables.

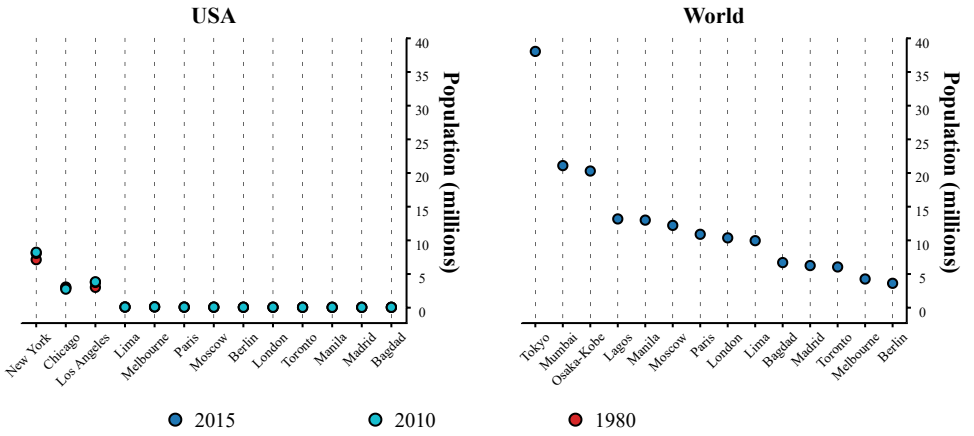
SCALE defines the structure of the axis.

COORD applies transformations to the geometric representations.

ELEMENT describes the mapping of variables to visual artifacts.

GUIDE allows adding other explanatory elements to the graph.

For each of these steps, a set of functions is defined that are used to describe the necessary operations. Furthermore, each step but the last has a default function to be applied. If this default function is used in a particular step, the respective step description can be omitted from the rule as in this case the default function is implicitly used. An example for a graph and the respective description is given in Figure 7.3.




```
ELEMENT: point( position( (city/group) * (pop1980+pop2000+pop2015) ) )
```

Figure 7.3: Grammar of Graphics: Example graph and description[4] (after [50]).
Data: [data6, data7, data8].

For *DATA* the default operation is the identity, which does not modify the data at all and can be used to map tabular data in a column-to-variable way. For other types like contents of an object-oriented database one has to specify how values for a certain variable are created. Possible

[4]The cities in the left visualization refer indeed to cities within the USA, which share their names with internationally more known counterparts. Both visualizations given here are part of an extended argument, that argues on the use of both Blend and Nest operators, which will be explained later.

functions include traversing a matrix in some way to generate values for a single variable. This mapping can, e.g., be a walk row by row, such that the index k is given by the row index i and the column index j as shown in Equation 7.1, where n denotes the total number of columns.

$$k = n * (i - 1) + j \tag{7.1}$$

In *TRANS* the default operation is again the identity. Besides this, one can use basically any mathematical or statistical function to transform a variable on a case-by-case basis. The transformation function might either use one or multiple other variables to compute its results. In this step, there is also the option to sort the entries according to one or multiple variables which comes in handy to highlight certain trends in the data.

The mapping of variables to axes is done in *SCALE*. For nominal and ordinal data this results in an equidistant mapping of the values on the respective axis. In the latter case, this also adheres to the given order. For numerical variables there are much more options: Values are split across the given axis proportionally to their size. In addition, one is able to specify the start and end of the scale, which default to the minimum and maximum of the values. This results in a linear scale which is used if no other scale is given. One can, however, apply a transformation function on the values before being distributed on the axis. Examples are given in Table 7.3. Using these transformations it is possible to spread the given measurement values more evenly across the axis or highlight a certain distribution.

	Transformation	Example
Ordinal	-	red orange yellow green blue violet
Linear	$x \longrightarrow x$	0 1 2 3 4 5
Logarithmic	$x \longrightarrow log(x)$	0 1 10 100 1000 10000
Power	$x \longrightarrow x^a$	0 1 4 9 16 25
Logit[5]	$x \longrightarrow log(\frac{x}{1-x})$	0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

Table 7.3: Grammar of Graphics: Scaling functions (after [50]).

Operations done in *COORD* define the geometrical properties of the graph. This is not limited to choosing a proper coordinate system like Cartesian or polar, but can also involve other transformations like reflection or stretching. The default coordinate system used is a simple rectangular Cartesian coordinate system with no other modifications applied. Also possible is the arrangement of multiple graphs in a table or matrix. This way, more than two variables can be encoded as position within a planar graph. While some of the variables are used to structure the matrix, the remaining positional variables form the actual graph. This is done using

[5] Only defined within the interval $(0, 1)$.

Form	Surface	Motion	Sound	Text
position	color	direction	tone	label
size	hue	speed	volume	
shape	brightness	acceleration	rhythm	
polygon	saturation		voice	
glyph	texture			
image	pattern			
rotation	granularity			
resolution	orientation			
	blur			
	transparency			

Table 7.4: Grammar of Graphics: Aesthetic Attributes (from [50]).

different operators, which shall be discussed later. An example partition is given in Figure 7.3. Here, the x-axis is split into two partitions depending on the world region (`group`). Within that partition, the respective graphs are drawn according to the remaining definition. Besides these planar transformations, Wilkinson also discusses the different options when projecting a higher dimensional (usually 3-dimensional) object onto the plane.

The transformations of *TRANS*, *SCALE* and *COORD* seem similar. However, it is argued that they work on different objects (variables, dimensions, and coordinates) and that this separation helps to distinguish between statistical and geometric operations.

In *ELEMENT* variables are mapped to visual artifacts, which are called *aesthetic attributes* (cf. retinal or visual variables in [49]). The aesthetic attributes used are partly derived from Bertin [49] but were reorganized and extended. A list of attributes discussed in [50] is given in Table 7.4.

Due to the limitations of a printed medium, only form, surface, and text are used throughout the book. The position attribute has a special role: While all other attributes can only be used to represent a single variable, position can visualize multiple variables. Besides mapping to the two planar dimensions, one can also partition the given area into a matrix as described before. To facilitate this partitioning an algebra is developed whose three operators describe how to merge variables. *Blend* (+) is used to combine the contents of two variables very much like a union-statement in databases. Unlike the other operators, *Blend* will display both variables within the same cell of the matrix. *Cross* ($*$) creates the Cartesian product of values from both variables. *Nest* (/) is similar to cross, but only retains value combinations that can actually be found within the underlying data. Using a cross operator one can, e.g., map two variables to the x and y-axis of a graph as shown in Figure 7.3.

GUIDE, finally, allows for adding legends to a graph and adapt the axes' labeling. Variables that are not encoded by position will by default create a legend to allow observers to decode the given graph. Legends include a list of all values or in case of continuous variables a subset thereof and a mapping to the visual artifact representing that particular value. Axes serve the

same purpose for variables encoded in position. One is able to change the formatting of the labels or apply geometric transformations like stretching and translation. The final guides mentioned are so-called *annotation guides*, which allow adding generic descriptions. Examples are a title or highlighting specific parts of the graph.

Tableau [web25] has already been described in Chapter 3. So at this point, only those aspects shall be repeated that are relevant to a visualization description. As later publications lack the necessary detail, the description will be based on [80, 138] mostly, which describe Tableau's predecessor **Polaris**. The specification later resulted in VizQL – Tableau's internal description language. Again, however, very little is publicly known as there seems to be only one very short publication on the topic [139]. Both systems, Polaris and Tableau, use an OLAP cube (cf. Section 6.1) as their underlying data model. In particular, some operations rely on a hierarchy over dimensional values to be present.

Polaris defines an algebra to split the given area of a visualization in a tabular fashion[6]. Possibly nested rows and columns are used to encode the values for some dimensions similar to a pivot table (see Figure 7.4; cf. Section 6.1). Users can either map a single dimension to one of the axes or give an expression using four operators (and parenthesis) to combine multiple dimensions. These operators include the ones defined by Wilkinson [50] and are defined as set operations. *Cross* ($\times$)[7] and *Nest* (/) remain unchanged. Wilkinson's *Blend* is replaced by *Concatenation* (+) to allow multiple expressions side by side not necessarily sharing the same data types. *Dot* (.) is added as a new operator. It is similar to *Nest* but is aware of existing hierarchies.

To illustrate the difference the following example is given: Assume a fact table that records revenue by `Quarter` and `Month`. This fact table has entries for all months but is missing one for `November`. *Nest* will only use values existing in the fact table resulting in 11 different entries.

$$\text{Quarter / Month} = \{ (1, Jan), (2, Feb), \ldots, (4, Oct), (4, Dec) \} \tag{7.2}$$

Dot, on the other hand, knows about the hierarchy between `Quarter` and `Month` and will add the missing entry totaling 12 entries.

$$\text{Quarter . Month} = \{ (1, Jan), (1, Feb), \ldots, (4, Oct), (4, Nov), (4, Dec) \} \tag{7.3}$$

In [138] it is claimed, that *Nest* requires a scan over the fact table, whereas *Dot* only has to traverse the much smaller dimension tables. Hence, for explicitly modeled hierarchies *Dot* will also have a performance advantage over *Nest*.

For all operators, the algebraic properties are explored. This leads to the following equations and properties:

[6] A table with just a single row and column results in a single visualization as defined by the other approaches.
[7] The representing symbol, however, has changed.

Associativity

$$(A + B) + C = A + (B + C) \tag{7.4}$$
$$(A \,.\, B) \,.\, C = A \,.\, (B \,.\, C)$$
$$(A \times B) \times C = A \times (B \times C)$$
$$(A / B) / C = A / (B / C)$$

Commutativity

$$A + B = B + A \tag{7.5}$$
$$A \times B = B \times A$$
$$A / B = B / A$$

Distributivity

$$A \times (B + C) = (A \times B) + (A \times C) \tag{7.6}$$
$$A / (B + C) = (A / B) + (A / C)$$

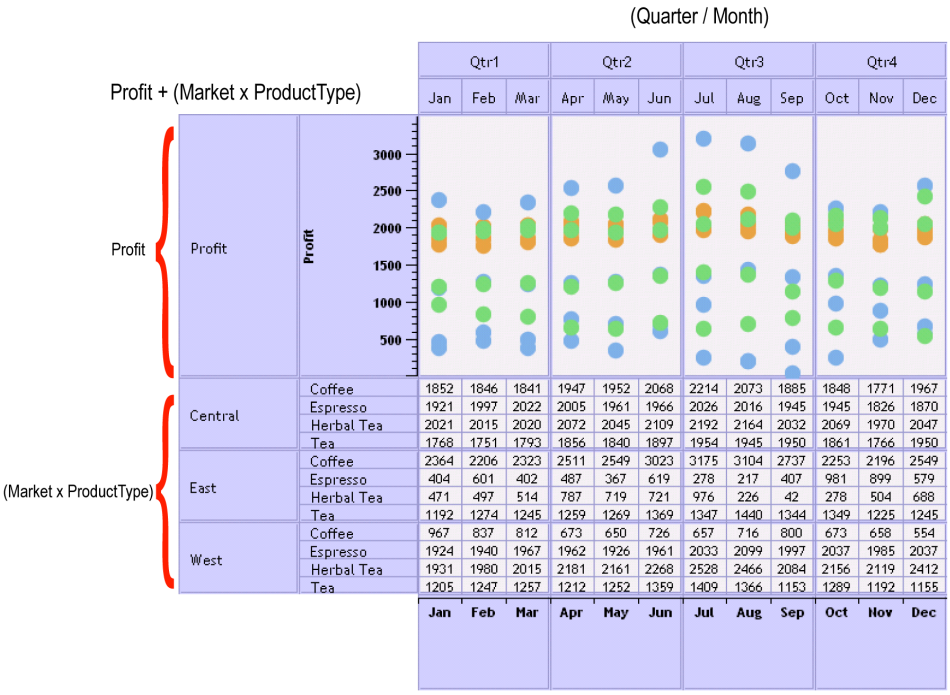


Figure 7.4: Polaris: Example of a visualization (from [138]).
Defining expressions for columns and rows as annotations.

Besides the planar partitioning, Polaris also allows for multiple layers[8]. Layers share the definition of their table structure as specified above but can refer to different data sources. All specified layers are then superimposed back-to-front to form the final visualization.

After this first partitioning step, the graphical elements within a cell (called "pane") have to be specified. Here users can choose between three mark types, which resemble Bertin's concept of implantation [49]. For each mark type also options to encode additional data are given.

Point marks represent single tuples. Additional data can be encoded in shape and size. Examples are text, shapes, and icons[9].

Line marks encode ordered sets of tuples. Points representing each individual tuple are connected in order. Additional data may be encoded in the fill pattern of the line. Line and Gantt charts are examples of this mark type.

Area marks are used for ordered sets of tuples. The points encoded by tuples are connected (including the first and last point) to form the boundary of an area. Additional data can be encoded in the fill pattern and the color of the area. Examples are area charts and polygons in maps.

Two other components of the specification are `Group` and `Sort order`. `Group` defines the level of detail with regard to the respective dimensions' hierarchies. In the case of line or area marks, this also joins tuples to be represented by the same mark. `Sort order`, on the other hand, defines the ordering of tuples as required by line and area marks.

In conjunction with the two spatial components, these – in total five – components (`X`, `Y`, `Mark`, `Group`, and `Sort order`) form the visual specification. The actual visualization used is then based on the type and number of fields assigned to the spatial components and the mark type. The fields' types (cf. Chapter 5[10]) hereby induce a classification of the possible visualizations as given in Figure 7.5. Within each class, the visualization is given by the mark type. In an ordinal-quantitative pane, e.g., an area mark type will lead to a bar chart, whereas a line mark type leads to a line chart.

Finally, additional variables can be encoded in visual properties. As already mentioned this encoding is dependent on the mark type. In general, Polaris uses five visual properties: Position, color, shape, size, and rotation (orientation). This constitutes a subset of the properties identified by Bertin [49] and Wilkinson [50] (cf. Table 7.8). The definition of each visual property is also mark-dependent. Shape, for example, refers to the external shape of the mark of a point mark,

[8] This corresponds to Wilkinson's *Blend* operator.

[9] Shapes are defined here as "points with a varying shape encoding", whereas icons are "fixed sized marks with shape encoded as images".

[10] In contrast to later works, categorical and ordinal types are subsumed as ordinal here. Categorical variables are sorted lexicographically to be displayed on the axes.

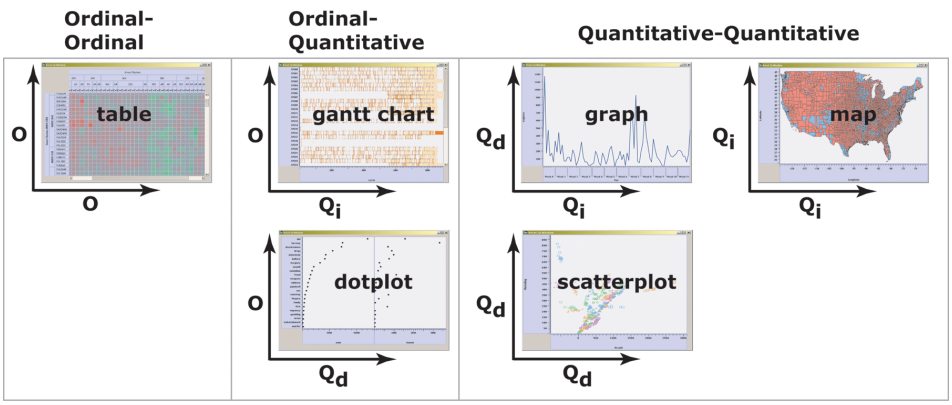


Figure 7.5: Polaris: Classes of visualizations (from [138]).
Ordinal (O), independent (Q_i), and dependent quantitative (Q_d) variable types.

whereas for line and area marks the fill pattern is described. Similarly, size and rotation have mark-dependent definitions. It is reasoned, that these definitions "simplify the encoding system" and prevent "nonsensical encodings".

In **VizAssist** [127, web84] each visualization is described as a collection of *visual attributes*. Each of those attributes has four properties:

Visual Type Roughly corresponding to Bertin's visual variables ([49], cf. Figure 7.1), although not explicitly referred to. See Tables 7.5 and 7.6 for examples.

Data Type Type of values that can be used for this attribute. Refer to Section 5.1 or Tables 7.5 and 7.6 for a complete list.

Maximum Number of Values Number of values that can efficiently be rendered by this attribute. It is stated that for the visual type *color hue* typically 10 distinct values can be represented.

Importance A value between zero and one hundred to describe whether this attribute is mandatory or optional to create the visualization.

Objective Weight For each of the predefined objectives a value between zero and one hundred is given. This value represents how well the visualization can achieve the respective objective.

The importance v_i of a visual attribute is computed by Equation 7.7. VA_i denotes the visual attribute, while vt_i and dt_i represent the visual type and data type accordingly. Mat_{GL} is the degree how well a visual type can render the given data type. This mapping is hard-coded using the values given in Tables 7.5 and 7.6. $Mandatory(\ VA_i\ )$ is a Boolean value that describes whether the respective attribute is mandatory or optional for the given visualization. This classification into mandatory and optional is continued in the overall value v_i: Values above 50 identify mandatory visual attributes, while values below 50 mark optional ones.

	Numeric	Ordinal	Nominal
Position	100	100	100
Length/Height	95	65	60
Angle	90	60	55
Slope	85	55	50
Area/Size	80	50	45
Volume	75	45	40
Density	70	95	75
Color Saturation	65	90	70
Color Hue	60	85	95
Texture	55	80	90
Connection	50	75	85
Containment	45	70	80
Shape	40	40	65

Table 7.5: VizAssist: Mapping visual type (rows) and data type (columns) to Mat_{GL} (from [127]).

Visual Type	Data Type	Mat_{GL}
Time Axis	Time	100
Image	ImageURL	100
URL	URL	100
Connection	Source/Target	100
Country	Country	100
Label	(any data type)	100

Table 7.6: VizAssist: Mat_{GL} values for special data types (from [127]; cf. Section 5.1).

$$v_i = 50 * Mandatory(\ VA_i\ ) + \frac{Mat_{GL}(vt_i, dt_i)}{2} \tag{7.7}$$

The objective weight has to be given by the visualization expert who adds the respective visualization. Objectives are drawn from a hierarchical list, which the authors created after literature review and personal experience (cf. Figure 7.6).

Table 7.7 shows the visualization model for a two-dimensional scatter plot as given in VizAssist. The marks within the plot are drawn as two-dimensional bars, whose length is given by the attribute L. The coordinates for each of those bars is given by X and Y respectively. The final visual attribute is the color of the bars given by C. One can see that only two attributes X and Y are mandatory, while L and C are marked optional. Furthermore; it is stated that at most 2000 different values can be mapped for X, Y, and L while the attribute C can only handle up to 10 distinct values. This example omits the associated objective weights.

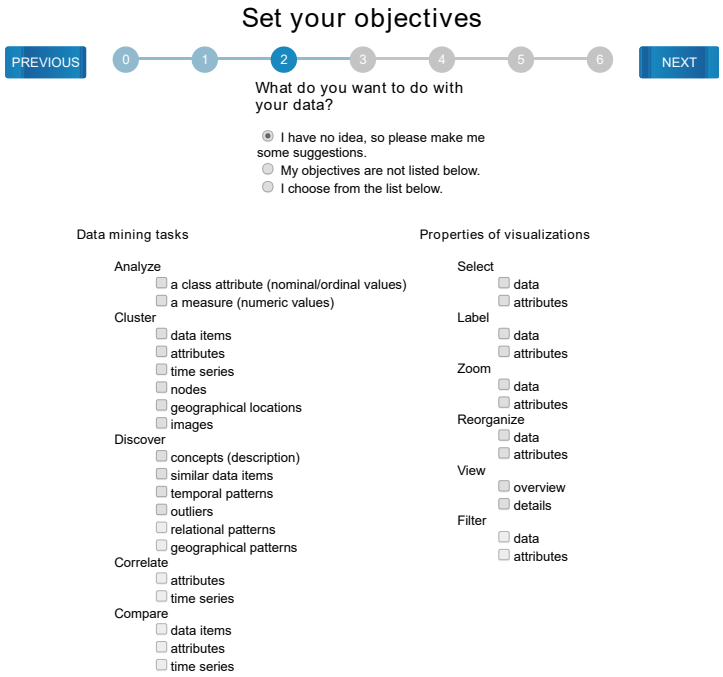


Figure 7.6: VizAssist: Selection of Objectives (from [127]).

Visual Attr.	Visual Type	Data Type	Importance	Max. Number of Values
X	position	numeric	100	2000
Y	position	numeric	100	2000
L	length	numeric	47.5	2000
C	color hue	nominal	47.5	10

Table 7.7: VizAssist: Visualization model for a two-dimensional scatter plot (from [127]).

7.2 Discussion

The common ground for all presented approaches is the mapping of input variables to the visual variables of the visualization. The subset of visual variables that may be used for that purpose, however, varies as shown in Table 7.8.

A mostly neglected aspect is the interplay of multiple visual variables within one visualization. While Bertin [49] at least mentions and briefly discusses the combination of variables, this context is lost in the other presented approaches. Consider the charts given in Figure 7.7. Both chart types share all properties, but the used coordinate system (polar coordinates for spider charts; Cartesian coordinates for parallel coordinates). Yet they differ in the number of tuples that can be displayed in a meaningful way.

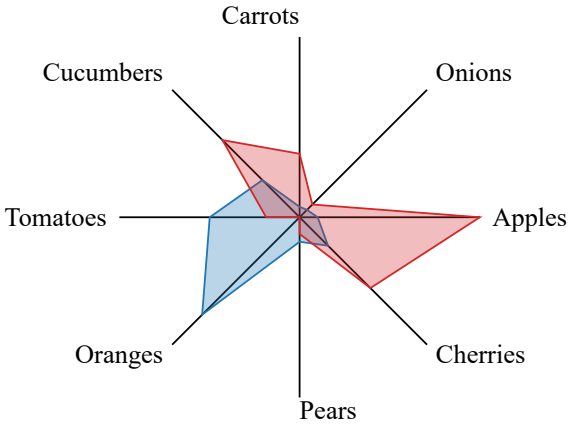
Bertin [49]	**Wilkinson** [50]	**Polaris** [80]	**VizAssist** [127]
Position	Position	Position	Position
Color Value	Hue Saturation Brightness	Color	Hue Saturation
Shape	Shape Pattern	Shape	Shape
Size Texture	Size Grain	Size	Size Texture
Orientation	Orientation Rotation	Rotation	Angle
	Blur Transparency Direction Speed ...		Slope Density Connection Containmemt Image URL Label ...

Table 7.8: Comparison of visual variables (after [80]).

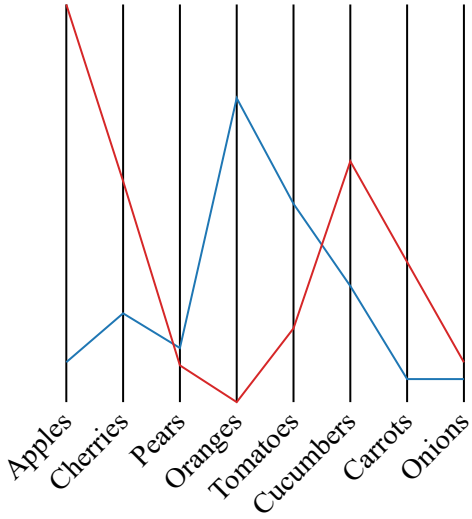
Bertin [49] gives maximum numbers of values that can be represented by a certain visual variable (and its implantation). These numbers, however, are just given per single visual variable. In general, this seems to be insufficient as can be seen in the given counter-example. Here, one has to consider not only the single visual variable but also its context and interplay with other used variables.

Furthermore, a singular fixed value is problematic even when given per visualization as done in VizAssist [127]. It induces a hard cut between “applicable” and “non-applicable”, where in reality it is most often a smooth transition. In other words, referring to Table 7.7: Why is a scatter plot with two thousand values possible, but two thousand and one values are too much?

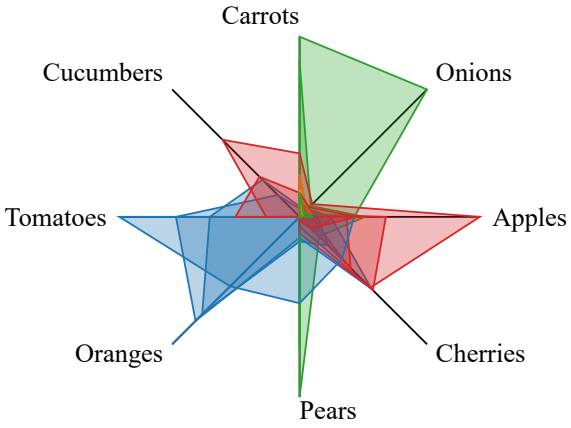
An option to extend the number of applicable variables to a visualization is to embed one visualization within another. The operators defined by Wilkinson [50] and within Polaris [80] allow splitting the given area into multiple panes. Polaris is more general at this point: Wilkinson only allows splitting into similar visualization types (i.e. all panes use the same type of visualization although applied to different datasets). Polaris, on the other hand, allows for each pane to have a different visualization (cf. Figure 7.4). Furthermore, Polaris introduced the *Dot* operator, which includes information that is not explicitly included in the data itself but has to be gathered elsewhere. As Polaris works on an OLAP cube, the information can be obtained from



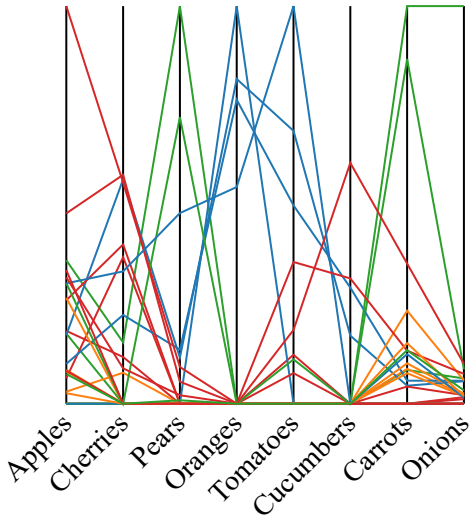
(a) Spider Chart: Few tuples (2).



(b) Parallel Coordinates: Few tuples (2).



(c) Spider Chart: Multiple tuples (26).



(d) Parallel Coordinates: Multiple tuples (26).

Eastern Europe Northern Europe Southern Europe Western Europe

Figure 7.7: Influence of chosen coordinate system on maximum number of tuples displayable. Number of tuples in (c) and (d) determined by availability of data. Data: [data9, data10, data11]; Classification: Eurovoc [data12].

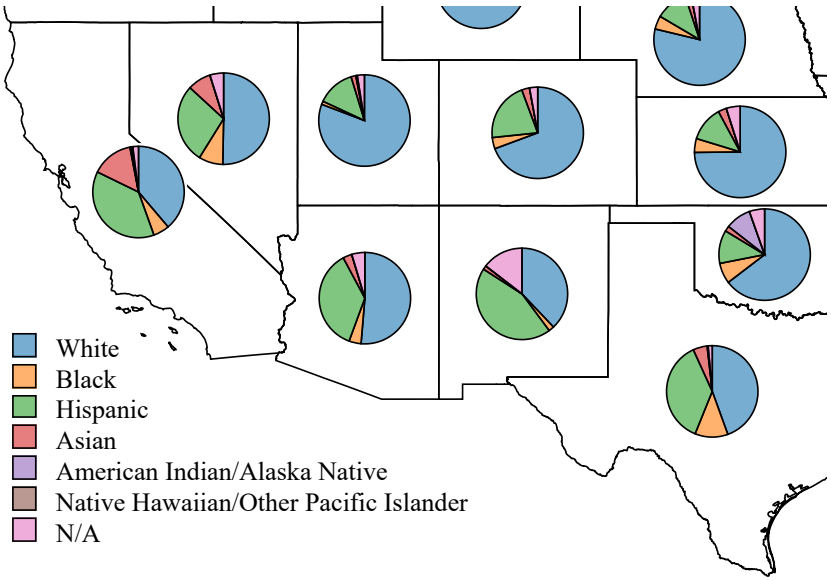


Figure 7.8: Nested visualization (Pie chart in map).
Data: [data13] via Kaiser Family Foundation [web85]; Map: [data14].

different hierarchy levels within the dimension tables. Both approaches, however, are limited to partitioning the given canvas in a tabular fashion. It is not possible to define other possible nested visualizations like the one given in Figure 7.8, where pie charts are nested within a map.

The limitation for rectangular layouts in Polaris extends beyond the partitioning of the given canvas. Within the Polaris' formalism, one can not create visualizations that are based on, e.g., polar coordinates like pie or sunburst charts. It is not clear if a coordinate system in addition to the pane visual attributes could remedy this situation. One would need to find suitable visualizations for every combination of mark type and class of visualization as given in Figure 7.5.

In contrast to the other approaches presented, VizAssist [127] includes an objective weight. Objectives describe possible goals or intentions of users. While this is a valuable addition to the description, it leaves room for subjective bias. The weights are assigned by human experts and are not derived by a systematic analysis of their usage in the wild. The latter might be better justified and could also reveal some contextual differences: Some visualization might only support a certain objective within a certain domain, while in others that visualization might be less appropriate if at all.

The final argument between the presented approaches is about their complexity and expressive power. Wilkinson's workflows [50] seem to be the most expressive. As argued before, however, in their current form they lack the means to describe nested visualizations in general. Furthermore, there is no measure which of the numerous possible combinations lead to meaningful visualizations. Polaris [80] fixes some of the issues here. It is able to describe at least rectangular layouts for nesting. On the other hand, it sacrifices some of the expressive power in favor of a

model, that prevents many meaningless visualizations and is better suited to create a direct user interface. Both these approaches try to describe a visualization by its components. In contrast, VizAssist [127] sees a visualization as one monolithic block. While this decreases the number of available visualizations initially, this model can cater better to specific characteristics of certain visualizations like the maximum number of possible values per visual variable.

7.3 Approach

The major decision in describing visualizations is between a monolithic description per visualization like VizAssist [127] or the component-oriented approach as given by Wilkinson [50] and to some extent Polaris [80]. While the latter requires fewer elements to describe a larger space of possible visualizations, this space also contains fewer meaningful instances. On the other hand, the monolithic approach can be seen as a selection of the most appropriate visualizations. As it also allows for easier handling of characteristics specific to certain visualizations, this approach is chosen for this work.

The most important part of the description is a list of possible bindings from dataset columns to components of the visualization. These components roughly correspond to the visual attributes mentioned before. However, a single visual attribute might be used by different components within one visualization. One could, e.g., separate the filling color of a mark from that of its border and bind both to different columns of the source data[11]. Using a visual attribute multiple times within the same visualization might confuse observers, though. While the proposed model allows for such combinations to not restrict the available space of representable visualizations, it is up to the author of a particular description to decide whether this is appropriate in the particular case.

Chapter 6 discussed the underlying data model for the source data. As each column should be mapped to one visualization component, some of the columns' properties reappear within the visualization description: For each component, both role and data type have to be specified following the definitions of Chapters 5 and 6. While oftentimes only one column will be bound to a component, there are instances where multiple ones can be used. This characteristic is captured by the *multiple*-flag within the model. In general, the order of bound columns, in this case, is important as it defines, e.g., the order of hierarchy levels. One example for the use of multiple columns within the same component is said hierarchy levels in a Sunburst chart (cf. Section 7.3.1). Here, multiple columns can be bound and will be interpreted as different levels of the hierarchy used for circles from inner to outer.

Per default, for each component, a binding has to be provided. Alternatively, a component can be labeled *optional*. In this case, the visualization can also produce meaningful results without having any data for this component. In a simple bar chart, the color of the individual bars is an

[11] Both color-components should use disjoint color scales to avoid ambiguity.

example: The chart is perfectly fine without having a column responsible for the coloring of each bar – all bars will share the same color. On the other hand, an additional value could be encoded in that color, making that component optional.

To accommodate for nested visualizations as shown in Figure 7.8, one additional data type `visualization` is needed. This allows for visualizations to define placeholders that can be filled with the descriptions of other visualizations. This technique can also model the tabular partitioning of the canvas used by Polaris (cf. Figure 7.4) with the exception of `Concatenation`. The parent visualization consists of three components: Two to represent the axes of the table and another one to represent the visualization used within the cells. The nested visualization can be any other visualization including another level of nesting. For an example of the description in Figure 7.8 refer to Subsection 7.3.1.

Most visualizations are agnostic with respect to the semantics of the data they show. However, some require a subset of their components to be of a specific type. If a map is modeled by just one component to represent regions of the world, columns bound to that component have to include some kind of location information. As already discussed in Section 6.3, the column description includes a concept for each column. The visualization component can similarly define a *semantic concept*[12]. In this case, only columns with those concepts can be bound to that component that is a child of the component's concept or the concept itself. So for the map example, the component's concept could be "geographic region". This limits the possible bound columns' concepts to things like "country" or "city". If no restricting concept is given, any column's concept can be matched. Applied to OWL ontologies [web86], this is equivalent to use `owl:Thing` as a restricting concept.

As discussed in Section 7.2, not every visualization component can represent an arbitrary number of distinct values[13]. The number of distinct values a component can accommodate will be called its *cardinality* according to the respective use in literature [140, 141].

However, fixed thresholds for minimum or maximum cardinality oversimplify the issue oftentimes. More realistic is a specification by a function that takes the number of distinct values as input and returns a value between zero and one. The returned value represents the suitability of that component for the given number of distinct values with zero signaling no match and one a perfect match. Reasonable choices for such a function are sigmoid functions, Heaviside functions, or the combination of multiple functions. Figure 7.9 shows an example of a combined lower and upper bound for the cardinality of a component. Here, a Heaviside step function is used for the lower boundary, while the upper boundary is given by a sigmoid function. The default for this property is a constant function with a value of one indicating that this component is applicable to any number of distinct values.

[12]This concept does not necessarily have to be a named concept, but can also be constructed by, e.g., the intersection or union of other concepts.

[13]This restriction only applies to components with a categorical data type. Those bound to numeric or time columns represent a continuous spectrum, which can not be described by "number of distinct values".

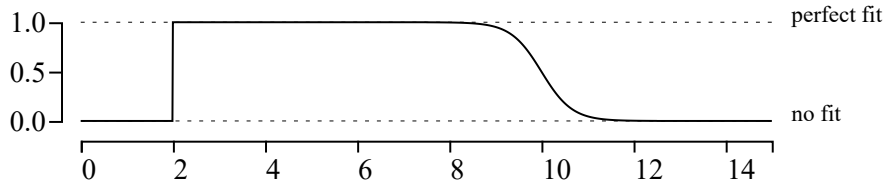


Figure 7.9: Example suitability function to assess the cardinality of a component.

Besides this component definition, a visualization description also contains user-facing properties like a human-readable name and a schematic example picture. As these properties are only used in the user interface, they will be omitted from most examples in the remainder of this work.

The above description does not enforce a one-to-one relationship between a visualization and its description. A single visualization might have multiple descriptions to accommodate for the input data to be modeled in different ways. Consider the table headers given in Table 7.9. Both datasets essentially describe the same data and could be represented using the same visualization. However, their column description is different and they can not be mapped to the same visualization description. To cater to these cases a single visualization can have multiple component bindings.

Year	Gender	Avg. Income
...	...	...

Year	Avg. Income (male)	Avg. Income (female)
...	...	...

Table 7.9: Different models of the same data.

In summary, each visualization description consists of a set of components. Each component, in turn, has the following properties and possible values[14]. In addition to the previous discussion, two more user-facing properties are included as well: a name and a natural language description of the property.

Name arbitrary human-readable name, e.g., "$x-axis$".

Description short text describing the component used, if the component is not sufficiently described by its name.

Data Type { categorical, time, quantitative, visualization, ... }

Role { dim, meas }

Multiple { true, false }; default: `false`

Optional { true, false }; default: `false`

[14]The actual descriptions will use shortened names in most cases.

Semantic Concept arbitrary concept from an ontology; default: `owl:Thing`

Cardinality (lower, upper); default: `unbound`

The property to describe the cardinality is a tuple. It consists of a `lower` and an `upper` definition to describe the upper and lower boundaries of the acceptable range. Both take values in form of function definition that can vary between the function types. Two examples for those functions are given in Listing 7.1.

If the default value for a property is to be used, it will be represented by an underscore _ at the property's position or the absence of that property altogether. To improve the readability of a component's description, a JSON notation [39] will be adopted as shown in Listing 7.1.

```
{
  dataType:       categorical,
  role:           dim,
  optional:       false,
  concept:        _,
  cardinality: {
    lower: {
      function:  heaviside,
      threshold: 2
    }
    upper {
      function:  sigmoid,
      threshold: 10,
      variance:  2
} } }
```

Listing 7.1: Notation example: Visualization component. For a visual display of `cardinality` see Figure 7.9. Name and human-readable description are omitted.

Listing 7.1 defines a component, that takes a categorical dimension column as an input. It places no restrictions on the concepts of bound columns. The cardinality has a hard lower boundary of two, meaning that it is not suitable for a column holding only a single value. The upper boundary uses a logistic function ($x \longrightarrow \frac{1}{1-e^x}$) with a center point of 10 and a variance of 2, which prohibits any value above 12 and uses a smooth transition between 8 and 12 (cf. Figure 7.9).

7.3.1 Examples

Subsequently, some example visualizations and their corresponding descriptions are shown. Only the column descriptions are given, while most of the user-facing model elements like component descriptions are omitted.

The examples shall highlight each aspect of the described model. Line charts are restricted in the number of distinct lines (and hence values) they can display. Sunburst diagrams show hierarchies and as such need multiple columns being bound to the same components. Maps, finally, can be used as a layout to nest other visualizations as already shown in Figure 7.8.

Sunburst Sunburst charts are used to visualize hierarchies, where the total value of the parent is given by the sum of values of its children. The measured values are mapped to the size of circular segments very much like in a traditional pie chart. Each hierarchy level adds another concentric circle, which is then partitioned according to the respective measured values. In this example, these hierarchy levels are represented by multiple dimension columns bound to the `Hierarchy` component.

Sunburst diagrams are usually part of interactive graphics. As there is oftentimes a large number of segments especially in the outer rings, some kind of dynamic tooltip is employed as a replacement of a static legend.

Figure 7.10: Example vis.: Sunburst
Data: [data15].

```
{
  title:       "sunburst",
  description: "A sunburst displays a multi-
                tiered hierarchy. ..."
  components: [{
    name:      "Circular Segment Size",
    dataType:  quantitative,
    role:      meas
  },{
    name:      "Hierarchy",
    dataType:  categorical,
    role:      dim,
    multiple:  true
  }]
}
```

Listing 7.2: Column Description: Sunburst.

Line Chart A line chart is used to visualize time-series. Different points in time are represented by the x-coordinate in a Cartesian coordinate system. The measured value then determines the respective y-coordinate. Values of the same series get connected to form a line. If multiple value series are to be displayed in the same chart, they can be distinguished by the color of their respective lines.

Figure 7.11: Example vis.: Line Chart. Data: [data1].

```
{
  title:       "line chart",
  description: "A line chart shows timeseries
                information that ..."
  components: [{
    name:      "X-Axis",
    dataType:  time,
    role:      dim
  },{
    name:      "Y-Axis",
    dataType:  quantitative,
    role:      meas
  },{
    name:      "Color",
    dataType:  categorical,
    role:      dim
    optional:  true,
    cardinality: {
      upper: {
        function: sigmoid,
        threshold: 8,
        variance: 2
      }
    }
  }]
}
```

Listing 7.3: Column Description: Linechart.

Map (Nesting) One example of a visualization that allows for nesting is a map as already shown in Figure 7.8. Here, the map itself has just one component to represent the geographical entities. The concept restricts possible bindings to `wd:Q82794`, which denotes the concept for "geographic region" in Wikidata [121, web69]. This prevents arbitrary columns from being bound to this component as it would not be able to display any non-geographical, categorical column like peoples' names. The other component is basically a placeholder, which can be filled with the definition of any other visualization.

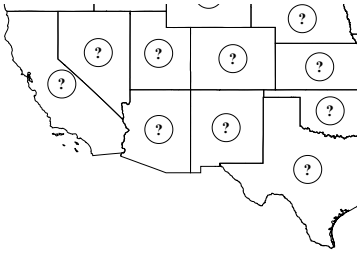


Figure 7.12: Example vis.: Map (nesting). Map: [data14].

```
{
  title:       "Map (nesting)",
  description: "A nest map can be used
                to distribute another
                visualization geographically. ..."
  components: [{
    name:      "Map",
    dataType: categorical,
    role:      dim,
    concept:  wd:Q82794 /* geographic region */
  },{
    name:      "Nested Visualization",
    dataType: visualization
  }]
}
```

Listing 7.4: Column Description: Map (nesting).

CHAPTER 8

DATASET DESCRIPTION

Metadata denotes “data about data” after the Greek prefix *μετά-* which translates to “after” and later adopted the meaning of “one level of abstraction higher” in the English language. Possible elements of metadata include information about the origin of the dataset like author and author affiliation or about the structure of the dataset like the number of columns and the respective headings. The importance of providing high-quality metadata has been emphasized on multiple occasions [web87, 142, 143, 144].

Despite the consensus that metadata is essential, opinions vary on the extent of such and the exact boundary between data and metadata. Assume a dataset that contains measurements at a specific point in time. Does that time value belong to data or metadata? This decision depends on both the opinion of the dataset creator as well as the intended application – both choices can be valid in certain contexts.

A prominent use case for metadata is data publication or exchange. While data creators may have extensive knowledge about their datasets, data consumers will need metadata in order to understand the data they are given. To ease the transfer of knowledge between creators and consumers in general a common format, a standard, is needed. Here, a common structure and possibly vocabulary of terms is defined to help different stakeholders understand datasets also from other creators. Without such an agreed-upon framework, metadata is bound to succumb to all kinds of perils in human communication, most notably in the form of misunderstandings [145]. Similarly in automated workflows and machine-to-machine interactions, common standards are vital to allow interpreting, interlinking, and finally exploiting data coming from different sources [143].

Beyond the mere understanding of metadata descriptions, standards also regulate their contents. They represent a community's consensus on which information is essential to summarize a dataset, which may be helpful, and which is better left to the primary data. The scope of metadata entries defined in such standards is determined by its intended audience and thus may vary widely. If metadata is used to estimate the trustworthiness of a dataset, information about the authors, their affiliations, and maybe a list of contributing data sources as well as the used methodology becomes more important. On the other hand, more information on the content is needed when the goal is to provide useful search capabilities across datasets. A system may use the title of the dataset as a whole and its columns in particular or the time and location of the collected data in order to meet a given search query.

In the last example, metadata can also be used to improve performance quite a lot, as scanning whole datasets or parts thereof can be avoided in favor of searching exclusively in their metadata descriptions. A search query asking for datasets from a specific period of time may be answered from metadata alone if the respective metadata includes such information. For other search queries, it might at least be possible to narrow down the search space quite substantially before having to scan the remaining datasets. As a consequence, one could argue to include as much information about the content in the metadata as possible. However, this can easily negate the biggest technical advantage of a metadata description, which is its small size in comparison to the actual dataset. Hence, it is important to find the right balance between keeping the metadata small enough and including all helpful information.

In the remainder of this chapter first, the requirements of this work will be stated. Afterward, existing approaches to metadata descriptions will be outlined and discussed, before the approach used throughout this work is presented.

8.1 Requirements

Before reviewing existing approaches to metadata descriptions, at this point, the requirements in the context of this work will be discussed.

General information The first set of requirements is motivated by having to display basic dataset properties to the user. This requires a minimum of descriptive information like a title, publishing entity, and publishing date. That way users can, e.g., make an initial guess on the trustworthiness of the data or use that as the starting point for further investigations.

Loading Different information is required to load a specific dataset. First of all, a location to load the data from is needed which most often will just be a simple URL. Sometimes this might also include a set of user credentials. Along with the location of the dataset, the system needs to know the type of the respective dataset whether it is a CSV file, a SPARQL [web88] endpoint, or any other supported type. For a given dataset, there might not only

be one way to access it, but several possible locations and types might be offered. Reasons include distributing traffic or providing files of different types so everybody may choose the one that is easiest to digest for their respective software. Hence, a dataset description should be capable to provide multiple locations and types for a single given dataset.

Further down that line, the system has to convert from a dataset's native model to the one used internally (cf. Chapter 6). This requires more details on the internal structure like the columns of the dataset. The most basic information at this point consists just of the data type of each column as discussed in Chapter 5 so the system can use appropriate parsers.

Search So far the user would need to specify a certain dataset to be loaded as no efficient search capabilities can be provided. As of now, the system would be limited to provide key-based search based on titles and publishers but has no information about the actual contents. For a more advanced approach, each column has to be described by a concept that describes its respective contents. For example, a column including values like `AT`, `BG`, etc. could be described to hold country names. To prevent fragmentation in the concepts used for description, either a central vocabulary is needed or provisions have to be made that different concepts can be related or translated to one another. To further narrow down the search space, the description should also be able to provide ranges for each column. That way, search for data might, e.g., be limited to a certain period of time or a range of measured values.

Integration Finally, when using multiple datasets some information on how to integrate them is required. One example here is units of measurement: Different authors may have chosen different units for their measurements. When combining multiple datasets those scales have to be aligned to get a meaningful result. Similarly, for categorical columns, code lists can be employed to map between abbreviations. Here, the same restrictions as for column concepts apply: Either a common vocabulary or means of translation are needed.

8.2 Related Work

As noted before, metadata standards are created by various communities and thus represent a wide variety of views. On the other hand, they represent the consensus within those communities and are the results of extensive efforts to distinguish essential from less crucial properties to describe a dataset. In the following, a selection of metadata standards will be reviewed to gain an overview of established approaches, identify commonalities among them, and see to which degree they may satisfy the requirements laid out before. The goal is to inform the decision on the details of metadata descriptions used throughout this work.

The **Dublin Core** forms the basis of many metadata standards. Maintained by the Dublin Core Metadata Initiative (DCMI) [web89], it provides a controlled vocabulary to annotate different kinds of resources. The vocabulary consists of two levels: The Dublin Core Metadata Element Set [web90], which contains 15 initially defined terms, and the DCMI Metadata Terms [web91], which includes all terms currently maintained by the DCMI.

The 15 terms in the Dublin Core Metadata Element Set are the following:

- contributor
- coverage
- creator
- date
- description
- format
- identifier
- language
- publisher
- relation
- rights
- source
- subject
- title
- type

They are described as "broad and generic, usable for describing a wide range of resources". The Dublin Core Metadata Element Set has also been adopted into a standard by several organizations [146, 147, 148].

The DCMI Metadata Terms include the 15 terms of the Dublin Core Metadata Element Set and 40 additional terms for a total of 55 terms. The additional terms provide a more precise definition of concepts. The general term `description`, e.g., now has two subproperties in `abstract` and `tableOfContents`. Similarly, `coverage` can now be given more precisely as `temporal` or `spatial`. Furthermore, it contains some new terms like `rightsHolder` or `subject`.

For the use of Dublin Core elements with RDF datasets, two namespaces have been established:

- `dcterms` - `http://purl.org/dc/terms/`
- `dc` - `http://purl.org/dc/elements/1.1/`

The distinction became necessary when the DMCI in January 2008 specified domains and ranges for the vocabulary. To not affect the conformance of already existing implementations, the latter namespace was not changed. Instead, the new `dc`-namespace was created for all entities including their respective domain and range definitions. Corresponding terms from the `dc`-namespace have been defined as subproperties of their counterparts in the `dcterms`-namespace. The standard definition does not demand to use the more comprehensive `dcterms`-namespace, but encourages new implementations to use it as they "more fully follow emerging notions of best practice for machine-processable metadata".

The **Data Catalog Vocabulary** (dcat) [149, web92] is an RDF vocabulary to describe governmental data catalogs or rather the datasets within. An overview of its model is given in Figure 8.1. The main three classes are `Catalog`, `DataSet`, and `Distribution`.

A catalog represents a data catalog containing the metadata of several datasets. Most of its properties refer to elements of the Dublin Core [150]. The additional properties include for one the listing of datasets contained and, on the other hand, a specification of the geographical

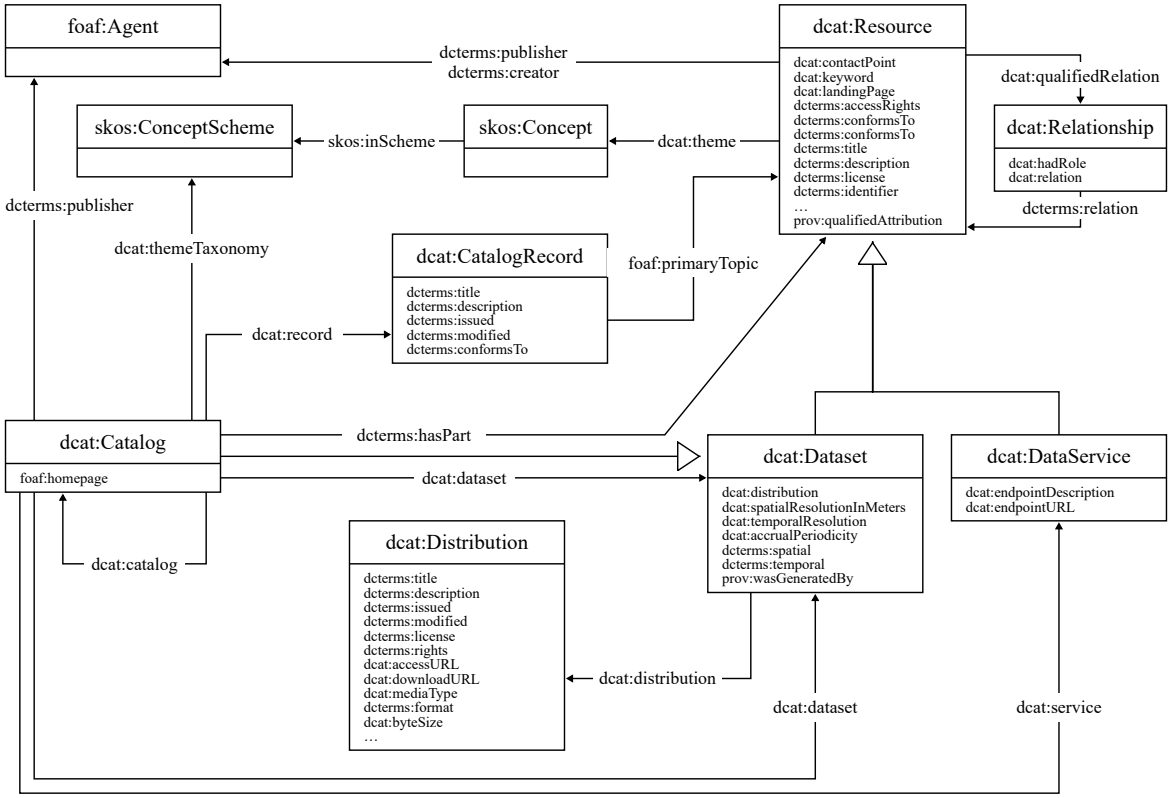


Figure 8.1: dcat: Summary of ontology (from [web93]).

coverage of the whole catalog. That coverage is determined by the union of coverages of the individual datasets. Additionally, it contains a list of keywords, so-called themes, which again is the aggregation of the individual datasets' themes.

The datasets themselves are also described by Dublin Core elements like publisher, license, date of last modification, and the publishing frequency of the dataset. The latter is not connected to the frequency which is used for taking the measurements. Additional properties are concerned with both spatial and temporal coverage of the dataset. Besides the aforementioned themes, a dataset can also contain associated keywords or tags to describe its content.

Finally, instances of the distribution class are tied to the respective datasets. They contain information about how to access the dataset, its size, and its format.

2020 saw the second version of dcat [web93], depicted in Figure 8.1. It remains backward-compatible with the first version, but includes new classes and relations and omits some of the constraints made earlier. The most notable additions and changes are as follows:

- Added `dcat:DataService` to describe webservices providing data.

- Introduced `dcat:Resource` as a superclass of `dcat:Dataset` and `dcat:DataService`.

- Added Dublin Core terms to describe the spatial and temporal coverage of a resource (`dcterms:Location` and `dcterms:PeriodOfTime`).
- Added `dcat:spatialResolutionInMeters` and `dcat:temporalResolution` to specify the spatial and temporal resolution.
- Relaxed the range of `dcat:themeTaxonomy` to include vocabularies that are not formalized as `skos:ConceptScheme`.
- Added terms from PROV-O ([web94]; cf. Chapter 12) to add the provenance of a dataset.
- Added `dcat:Relationship` to represent connections among resources.

The **Vocabulary of Interlinked Datasets** (voiD) [151, web95] is an RDF vocabulary for the description of RDF-based datasets. The goal is to allow humans and systems to get a rather quick overview what a specific dataset is about instead of having to run multiple queries in order to get an understanding of the dataset. One use case mentioned is search engines that can index RDF datasets much more efficiently using voiD than by analyzing the whole dataset. An overview of the major involved classes is given in Figure 8.2.

A voiD description covers several aspects of the respective datasets. At first, the general metadata about the dataset is given as specified in the Dublin Core [150]. This contains information about the creator and publisher of the dataset as well as the general subject. For the subject, it is recommended to use DBpedia [152] resources where possible. Another aspect is how the dataset is to be accessed. The information given here includes a possible SPARQL endpoint (`void:sparqlEndpoint`) or the location of RDF dumps (`void:dataDump`). If multiple data dumps are given for a dataset, the defined semantic is to see them as components of the whole dataset and not as possible alternatives.

There is also the concept of `void:Linkset` in voiD. A linkset can either be a dataset of its own or be contained within another dataset. These datasets form the connection between two datasets (referenced via `void:target`) using, for example, the `owl:sameAs` property to define equality between entities in different datasets. The link between two datasets might also be directed by, e.g., the `foaf:interest` property.

The structure of an RDF-graph is not restricted by a schema like in relational databases. So the structure of the dataset is given by so-called root resources using `void:rootResource` or representative example resources using `void:exampleResource`. Root resources follow the assumption that most datasets are hierarchical and thus root resources provide a good entry point to the RDF dataset. Starting from them, the whole dataset can be crawled by recursively following the given links. Other vocabularies used within a dataset can be defined using the `void:vocabulary` property. Within voiD a user is not required to specify all used vocabularies, but it suggests listing at least those that may be important when issuing queries.

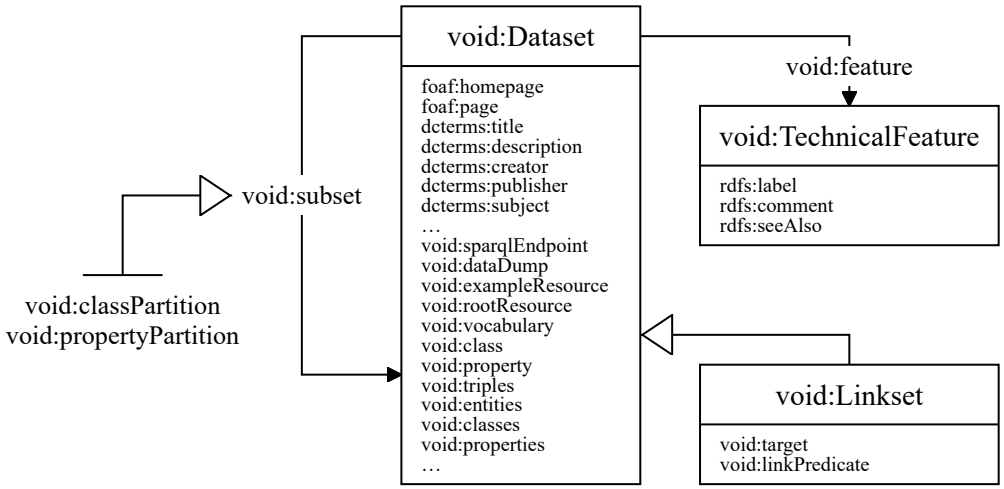


Figure 8.2: voiD: Ontology model (after [web95]).

If not all parts of a dataset share the same attributes, one is able to define partitions of a dataset using `void:subset`. The main dataset then contains all global properties and a list of its subsets. Each subset may have properties specific to it. One way of specifying subsets is to make restrictions on classes or properties using `void:classPartition` and `void:propertyPartition`, which are both defined as subproperties of `void:subset`. The partitions are required to have exactly one property of type `void:class` or `void:property` to describe the class or property all members of that partition share.

Finally to describe the size of a dataset, voiD provides the option to attach the counts of several components of the artifact like, e.g., `void:triples`, `void:entities`, `void:classes`, or `void:properties`.

Data Cubes [web9, 153] is an ontology trying to form a core vocabulary to express multidimensional datasets as RDF resources. The main goal was to unite the ideas formulated in the SDMX standard [40, web12] with the principles of linked open data [120]. An ontology overview is given in Figure 8.3.

The underlying data model is the OLAP cube (cf. Section 6.1). Each cell of the cube is modeled as a `qb:Observation` with the respective dimension and measurement values as attached properties. This introduces a lot of redundancies as dimension values are repeated over and over again. To tackle this problem a `qb:Slice` can define a subset of observations that share certain properties. Besides reducing redundancy, slices can also provide a separate URI for a subset of the data for annotation or external referencing. The standard definition allows for the option to repeat values defined for the slice at each observation, so queries do not have to traverse through

existing slices. As a generalization of `qb:Slice`, there is also `qb:ObervationGroup` which does not require a shared dimensional value for its member observations. The entirety of observations — possibly organized into slices or groups — form a `qb:DataSet`[1].

Besides representing the actual observations, Data Cubes also allows defining the dataset structure using `qb:DataStructureDefinition`. These definitions may be shared across different datasets. They follow the design of an OLAP cube in specifying dimensions and measurements as components. Another component type is attributes which are used to define constant values that apply to the whole dataset like the unit of measurement. For each component, one can specify the concept that is represented using `qb:concept`. If the values of a component are drawn from a finite set of options, one can mark that component as `qb:CodedProperty` and attach a code list of allowed values. For other components, the representation of possible values can be attached via `rdfs:range`. Examples are given by XSD Datatypes [web96] or "URIs drawn from a time reference service". The presentational order of components can also be encoded in the definition by using `qb:order`.

The authors give two primary objectives for data structure definitions: It enables users to get a quick overview over the given dataset without the need to traverse the actual observations. Furthermore, the definition can be utilized to verify the observation in terms of both structure and content.

The **Ecological Metadata Language** (EML) [154, web97] is an XML-based [web11] metadata standard with a primary focus on the ecology domain. As of EML version 2.2.0, it is comprised of 26 modules each concerned with a different aspect of the metadata description. If a module lacks certain features for a given domain, one can easily extend that module to cater to the specific requirements. That way, the standard can be adapted to more specific needs without an extensive approval process to change it as a whole.

EML is not restricted to tabular data, but also provides modules to deal with citations, software, or protocols. The following description will, however, focus on the modules involved in describing tabular data. Likewise, due to the extent of the specification, other aspects may be generalized or omitted.

A dataset in EML's notion is "all of the information describing a data collection event" which is captured in several data entities. Data entities supported are data tables, spatial raster images, and spatial vectors. If the dataset is stored in a DBMS, one is also able to describe stored procedures and views. A data table consists of a list of attributes and possibly constraints. An overview of the involved types and the discussed properties is given in Figure 8.4.

Some properties like `coverage` (cf. Figure 8.5) or `methods` may appear on multiple levels of the hierarchy. Here, the more specific property supersedes the more general one. So the methods specified on a `dataTable` level supersede the methods given at the `dataset` level for that particular data entity. Other entities will not be affected. Most of the properties relate to

[1] Contrary to other ontologies like voiD or dcat, Data Cubes capitalizes the "S" here. The reason is the compatibility with the SDMX element `DataSet`.

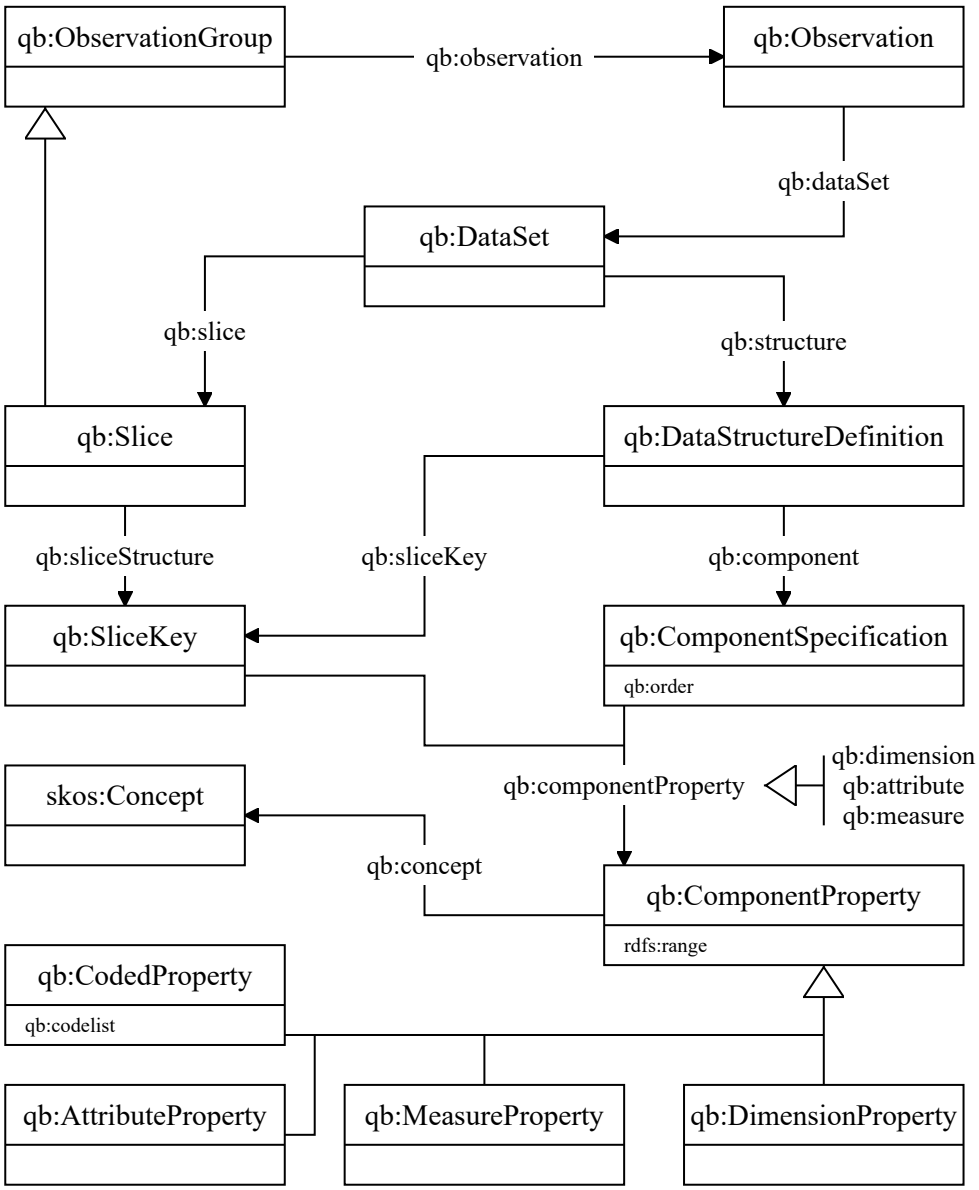


Figure 8.3: Data Cube: Ontology model (from [web9]).

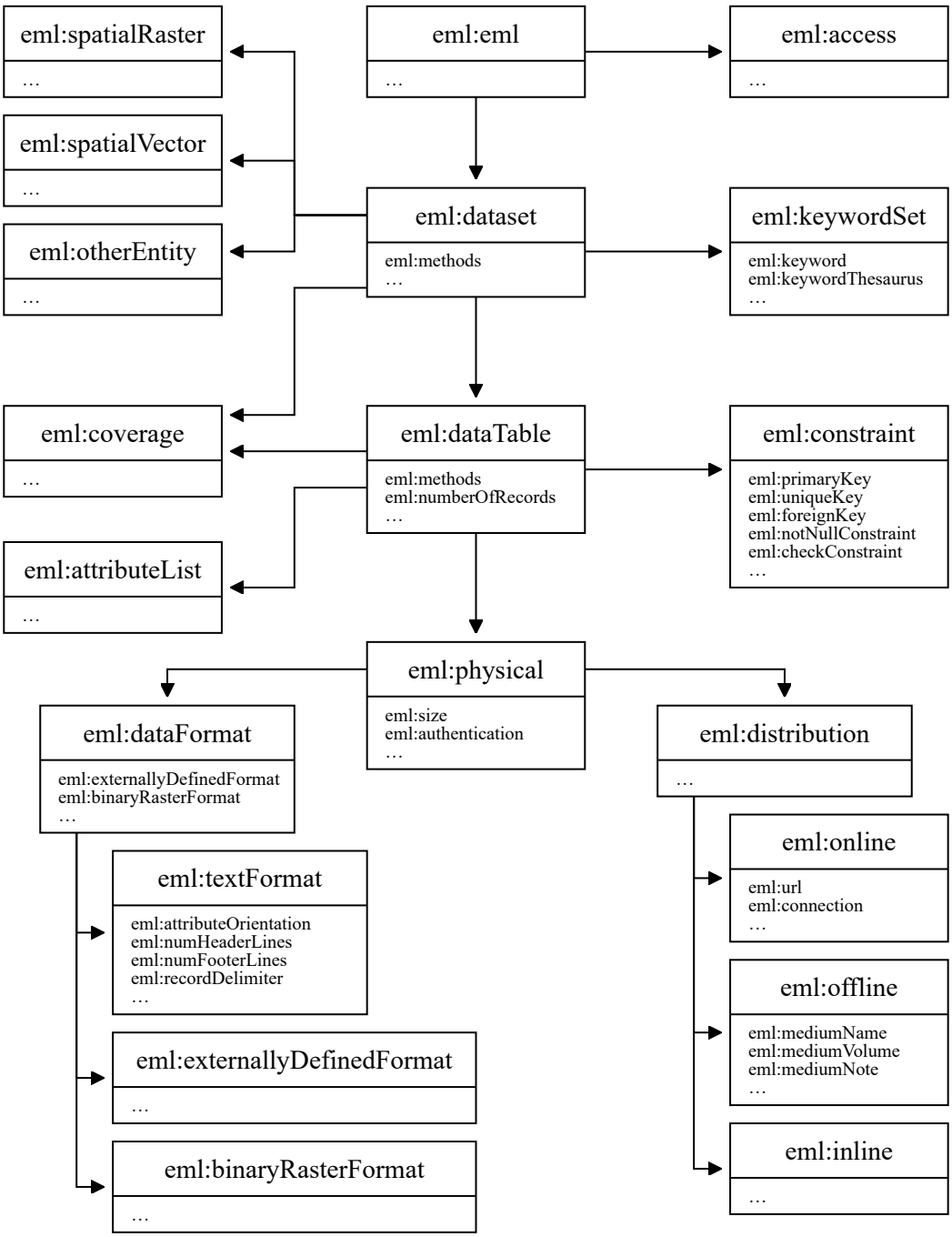


Figure 8.4: EML: Schematic overview (after [web97]).

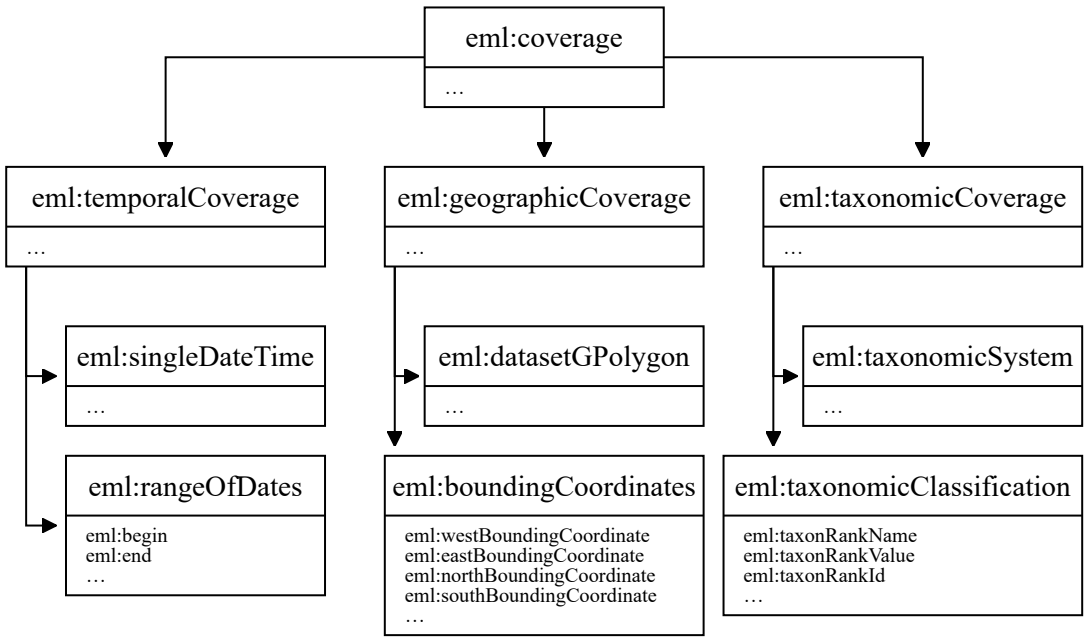


Figure 8.5: EML: Schematic overview: Coverage (after [web97]).

counterparts of the DCMI Metadata Terms [web91]. However, there are some differences: At the dataset level, one is able to specify multiple sets of keywords to describe the dataset. Each set can be linked to a thesaurus from which the keywords were taken. Furthermore, the coverage can be given in terms of spatial, temporal, and taxonomic coverage. The latter highlighting the main use case for EML.

For a specific data table, EML includes both physical distribution and logical structure. The physical distribution contains information about the size of the data entity and authentication data like the MD5 hash [155] of the entity. The distribution of a data entity can be either online, offline, or inline. Online data is identified by a URL and further connection information. If the dataset is given in a textual format like CSV files, one is able to attach quite detailed parsing instructions including the number of header or footer lines, field lengths, or delimiter characters. Data included in the metadata specification itself is called inline data. In this case, EML expects mostly text-based data but has no further provisions. Finally, offline data is characterized by the distribution medium. Here, no formal point of contact on the distribution level is given, but a user has to refer to the general contact information on the dataset level. However, there is a `mediumNote` property to hold any information not fitting for one of the other properties.

The logical structure is given in form of an attribute list (cf. Figure 8.6). Here, EML follows the classification of Stevens [124] (cf. Section 5.1), but adds a separate `dateTime` class. Ordinal and nominal attributes can either be of type `textDomain` or `enumeratedDomain`. While the latter allows to restrict possible values to a given code list, the former may be restricted by the use of regular expressions. A unit of measurement is required for interval and ratio attributes. This

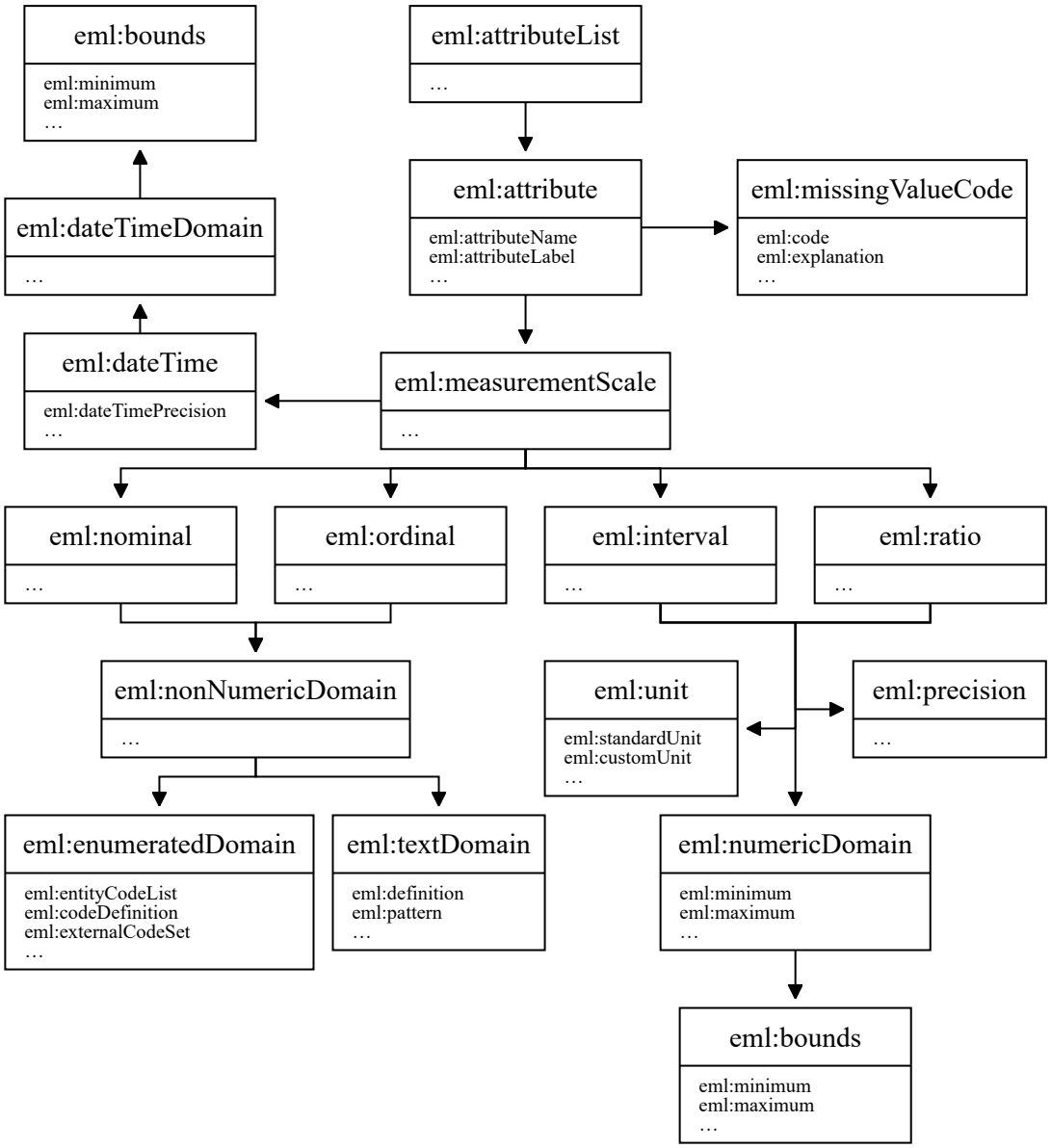


Figure 8.6: EML: Schematic overview: AttributeList (after [web97]).

unit can either be drawn from the unit list provided by STMML [156] or be provided as a string following the STMML syntax. The range of values can be given by minimum and maximum values. Furthermore, one is able to specify the precision of values given as a fraction of the base unit. Attributes of type `dateTime` share the option to attach minimum and maximum value as well as precision. Additionally, a string has to be given to describe the format of values. This string has to follow the widespread pattern of using predefined symbols like Y, m, or h to indicate the position and extent of year, month, and hour values within the value string.

For each attribute, one can also list codes used to indicate missing values. Each such code consists of the actual code as used within the table and a free text explanation like “removed because of a calibration error” or “measurement not taken”.

On a datatable level, constraints can be added. These include key constraints like defining primary and foreign key relationships or a `notNullConstraint`. Furthermore, there is the option to define arithmetic constraints like “`site>10`”. This constraint has just the form of a string, though. The authors suggest using SQL [148] statements, but also allow for any other language.

8.3 Discussion

The first requirement, identified in Section 8.1, is concerned with general metadata. As all presented approaches support the DCMI Metadata Terms [web91] as part of their dataset description, no major differences are apparent. Dublin Core is only concerned with a high-level description of resources. In consequence, it does not comply with any of the other requirements and, hence, will be excluded from further discussion.

Due to their different purposes, the approaches differ in their capability to specify a dataset’s location(s). Data Cubes is intended to hold the actual data and not just metadata. Hence, it has no provisions to specify the location of a dataset. Similarly, voiD is only concerned with RDF datasets and thus is restricted to either list SPARQL-endpoints or the location of an RDF data dump. For the latter, it is not possible to include alternative locations or the type of serialization like Turtle [web63] or RDF/XML [web64]. Within dcat, on the other hand, one is able to attach multiple distributions to a single dataset. Each of these distributions also has a media-type property. Both voiD and dcat provide no distinct properties to specify user credentials. If the credentials consist just of username and password, they can, however, be embedded in the URL location entry [web98]. Finally, EML has the most extensive description of distributions. Here, not only multiple locations but also a very detailed data structure — at least for text-based files or binary raster data — can be attached.

The next requirement is concerned with the specification of a dataset’s structure. Neither voiD nor dcat provides the means for detailed information in that regard. In Data Cubes the structure is given as a collection of components where each component is augmented with the covered concept, the included range, and the distinction in dimension and measurement. On the

other hand, EML classifies its attributes by content type following Stevens [124]. Similar to Data Cubes for each attribute the range can be given as well as a pattern to validate or parse the values for that attribute.

With regard to the last requirement of information needed for integration of datasets both Data Cubes and EML are on par: Both allow for adding units of measurement or referencing of (standardized) code lists.

A summary is given in Table 8.1. One can easily recognize, that only EML natively supports all requirements. On the other hand, ontologies, in general, can easily be combined to exploit the advantages of different modeling approaches and compensate for any missing components. As previously mentioned it is also possible to extent EML modules, but here the embedding of other standards requires more effort.

	dcat	voiD	Data Cubes	EML
General Information	●	●	●	●
Loading – Location	●	◐	○	●
Loading – Structure	○	○	●	●
Search	○	○	●	●
Integration	○	○	●	●

Table 8.1: Summary of support for requirements in existing metadata standards.
(●. . . fully supported; ◐. . . partially supported; ○. . . no support)

The final argument is concerned with vocabularies used for identifiers outside of the core metadata model like the concepts used to describe units. At this point EML is less flexible than the ontologies: For concepts of columns no code list can be given. But even if code lists can be specified like for the values within a column, translation between different code lists is completely out of scope. For ontologies — especially those, that are part of the Linked Data Cloud [120] — it is a sign of good design to link to other ontologies [web99]. This provides a way to decide for concepts of different code lists whether they describe the same real-world entity or not.

8.4 Approach

While EML seems as the single best choice with regard to the listed requirements, a combination of ontologies can yield the same result. Talking about issues of data integration, though, ontologies outperform the XML-based solution. Using semantic concepts, it becomes much easier to determine the relationship between different entities. One could by convention adopt semantic entities for EML as well, but then each resolving task would involve an ontology anyways. So having the description in an ontology format ab initio avoids another media break.

From the presented ontologies, Data Cubes covers the given requirements best. Hence, it seems the best choice to base the dataset description upon. Although not explicitly stated as a use case in the standard description [web9], there is also no statement preventing the use of

an “empty” Data Cube dataset to describe other datasets. An overview of the final structure of the dataset description is given in Figure 8.7. As previously noted, elements of multiple other ontologies — namely DCMI Metadata Terms and dcat — have been integrated to account for missing parts. Where no existing ontology provided suitable elements, new ones were created under the namespace prefix `yavaa`.

At the center of the model is, of course, the dataset. A dataset is actually a subclass of both `qb:DataSet` and `dcat:Dataset`. The former is used for attaching an instance of `qb:DataStructureDefinition`, while the latter is required by `dcat:distribution` to add instances of `dcat:Distribution`. A distribution as of now just consists of a download URL and a media type. In the future, one could add access credentials or more detailed parsing instructions following the ideas of EML. Besides structure and distribution, the dataset is also described by several general metadata properties like `dcterms:publisher` and `dcterms:title`.

The structural definition follows the Data Cube standard. The `qb:order` property is used to indicate the index of the column in the internal data model (cf. Chapter 6). If there is an order of columns given in the dataset serialization, `qb:order` is equal to the respective index, such that the index of the column in the source and in the internal model are identical. The contents of a particular column are then given by an instance of one subclass of `qb:ComponentProperty`. Here, the distinction between dimensions, measurements, and attributes is kept. Following the Data Cube standard, a component property is also a subclass of `qb:CodedProperty` if the values contained stem from a given code list. A common attribute for all columns no matter the role is the concept covered. This is represented by a property of type `qb:concept` linking to a specific instance of `skos:Concept`[2]. If the column is numerical, a unit of measurement is given by `yavaa:hasUnit`. Details on the handling of units are given in Chapter 9.

Another important requirement is the description of the range of values. This is implemented using `rdfs:range` pointing to either an instance of `skos:ConceptScheme` or `rdfs:Datatype`. The former is used, when the values in the respective column consist of codes that can be mapped to instances of `skos:Concept`. If the values are numerical measurements or time values, an instance of `rdfs:Datatype` will define the minimum and maximum value using OWL-restrictions on the respective XSD types [web96].

As the values for date and times have a wide range of possible representations, the respective columns also include a link to parsing information via `yavaa:hasFormat`. The definition here is rather simple and is shown in Figure 8.8. The definition is again based on an OWL-restriction, this time put on the `xsd:string` type. Using the `xsd:pattern` property a regular expression is used to parse a given string into a date and/or time. To specify which component contributes to

[2]Defining a common concept for a column is far from easy. Oftentimes, multiple naming schemes are implemented by different data providers leading to issues of interoperability among them. The author of this thesis is part of a working group, *InteroperAble Descriptions of Observable Property Terminology WG* [157, 158, web100] as part of the Research Data Alliance [web101], to address these issues and provide guidelines to alleviate or even overcome the impact of such issues. Unfortunately, the results of these efforts did not become available in time and thus have to be deferred to future work.

which part of the date or time value, each capture group of the regular expression is referenced by one `yavaa:TimeComponent` instance. This instance consists of a number referencing the index of the respective capture group and an interpretation (like minutes or years) of the results matched by the capture group.

Finally, instances of `qb:AttributeProperty` are used to encode invariants pertaining to all entries in a dataset. They are interpreted as an additional virtual columns that are appended to the existent ones and that hold the same value in their all rows. While such a column could be made explicit in the dataset itself, it is more storage-efficient to summarize it into an attribute within the metadata description. Each instance of `qb:AttributeProperty` is also an instance of either `qb:DimensionProperty` or `qb:MeasureProperty`. It thus assumes all properties regarding the general description for the virtual column like a code list or the time format depending on their type. In addition, `yavaa:hasValue` provides the link to the actual value in the corresponding virtual column.

The proposed metadata schema fulfills all requirements posed before in Section 8.1. General information like title or author is given by the properties of `qb:DataSet` / `dcat:Dataset` leveraging the vocabulary of DCMI Metadata Terms. Data necessary to load and parse a dataset is provided by two elements: First, a `dcat:Distribution` instance provides the URL and media type of a possible download location. Second, the structures reused from Data Cubes outline the content of the dataset by describing its columns in instances of `qb:ComponentSpecification`. Besides keyword-based search over title and description properties, the model allows retrieving datasets based on their actual contents. This is facilitated by having a `skos:Concept` associated with each column describing the kind of value included and an instance of `yavaa:Range` to summarize the values contained in it. Finally, data integration is supported by information that allows to further interpret the values inside a dataset. For categorical columns, a `skos:ConceptScheme` describes a mapping from used abbreviations to `skos:Concepts` allowing to map different representations to the same concept. In quantitative columns, the unit of measurement is included to facilitate conversions if necessary. Finally, for date/time columns a `yavaa:TimeFormat` allows to parse and identify individual components of the provided values. With all requirements covered, this model provides the basis for both search among and integration of dataset described in the subsequent chapters.

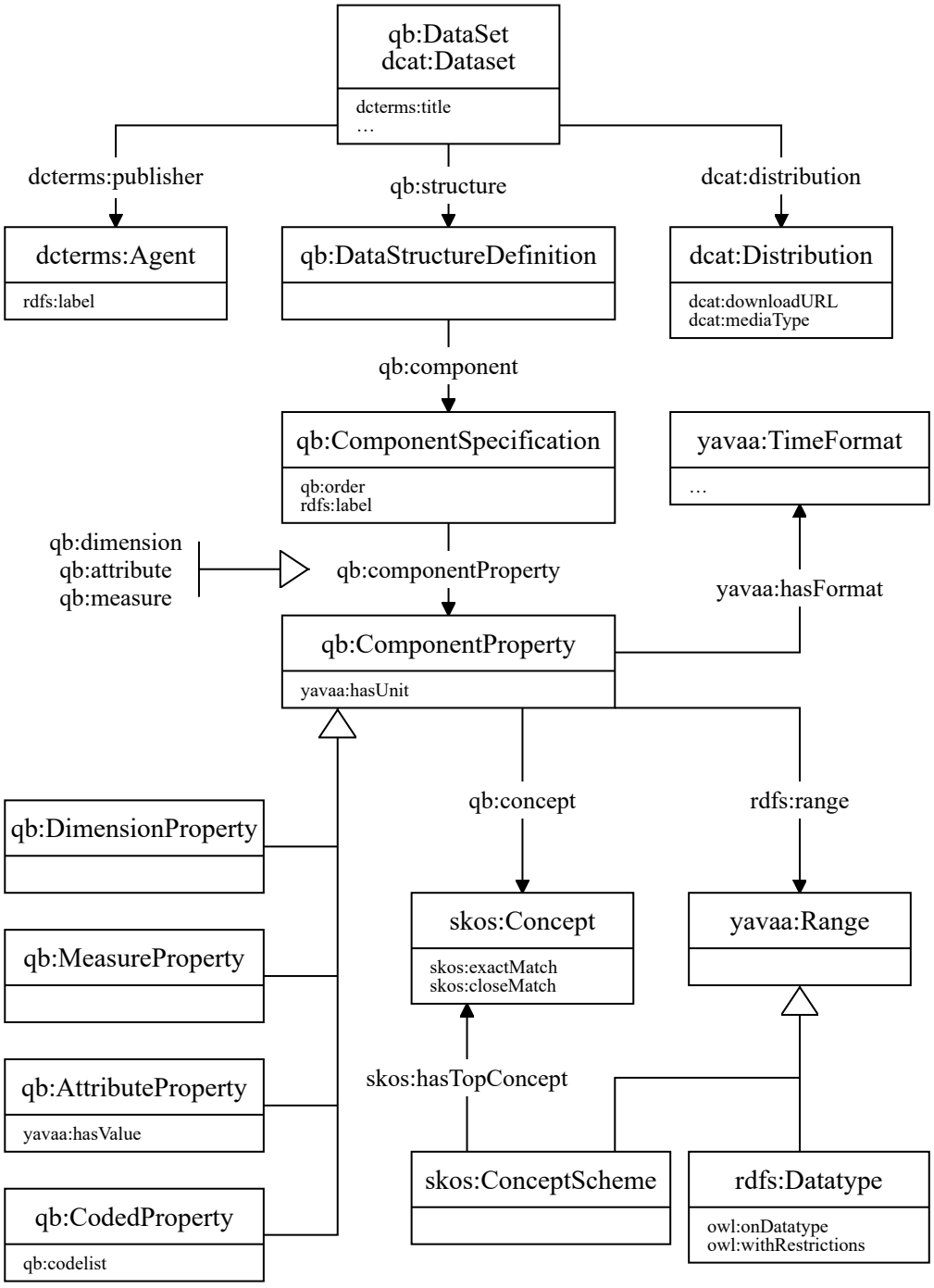


Figure 8.7: Yavaa: Ontology model.

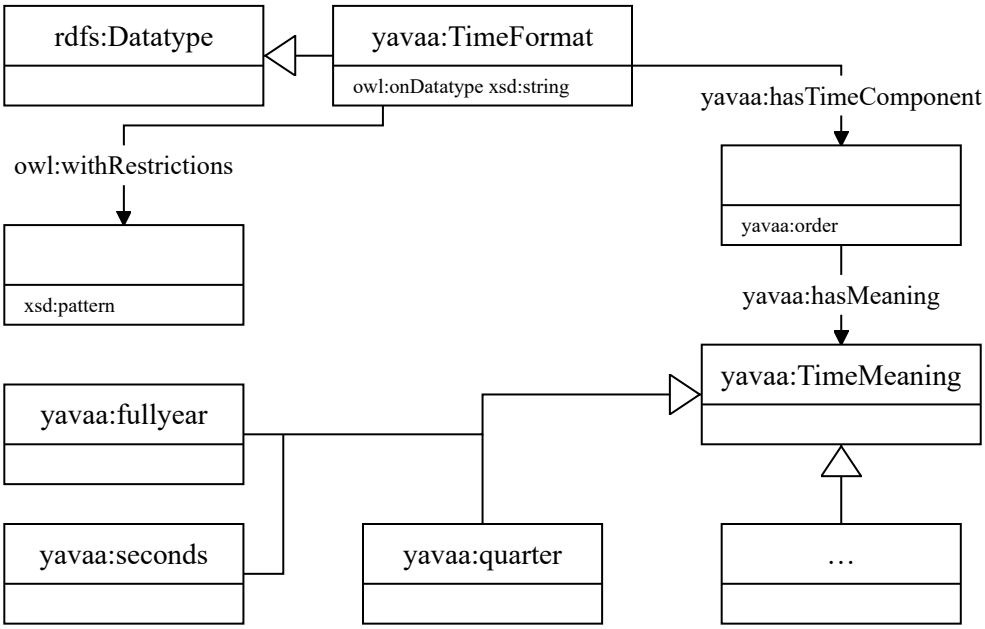


Figure 8.8: Yavaa: Ontology model - `yavaa:TimeFormat`.

CHAPTER 9

HANDLING OF UNITS

While working with measurements, the annotation of values with their respective units is essential. Among the most prominent examples of failing to do so are the loss of the Mars Climate Orbiter in 1999 [48] or the failed laser-beam missile defense experiment onboard the space-shuttle Discovery in 1985 [47]. On both occasions, the unit expected by a subsystem was different than the one provided, which obviously led to wrong results. While these incidents arose from an error in the design of the system or its implementation, the same problems persist when human users are working with datasets. Missing unit annotations leave it to the diligence of users to make sure units are used consistently throughout the workflow. Sometimes this even includes guessing the actual unit, when the metadata provided lacks the respective information.

Proper annotation and sufficient support of units throughout the workflow can prevent most of these errors. The most basic approach is to ask users for the expected result of the entered formula and alert them to any violations of unit consistency. Depending on the kind of violation users may then either add the required conversions or — after determining the root cause — rephrase the workflow in a proper way.

Then again, why burden users with tasks that can be automated? If the units for all operands of a formula are known, the system should be able to determine the result or at least a list of possible results. Furthermore, if the addition of conversions can fix an otherwise faulty formula, the system can add those implicitly. An automated approach can also limit the number of errors introduced by other mishaps like typos or plain out wrong conversions.

While the above argument has been made for unit consistency, a weaker form — dimensional consistency [159] — is less computationally expensive. Each unit usually measures a certain set of quantity kinds. Compatible kinds of quantities can then be subsumed under a single dimension. An example is the dimension *length* that includes among others subclasses like *width*

and *diameter*. Each dimension can be expressed as a vector of base quantities[1]. Using these dimension vectors, a formula can be validated rather quickly as compared to validating the unit consistency. However, certain errors can not be detected using checks for dimensional consistency only. For example, the sum of a value given in foot and a value given in meter while being dimensionally consistent is a violation of unit consistency unless the proper conversion is added. To achieve the described level of automation a few prerequisites are needed, which should be stated now.

Unique identifiers Each unit needs a unique identifier to annotate datasets (cf. Chapter 8). While this may sound trivial at first, there are differences in the definitions of units oftentimes as a result of historic processes. One example is the unit *inch*, which has multiple valid definitions as shown in Equations 9.1 [web102].

$$\begin{aligned} 1\ (international)\ inch &= 0.025\ 4 \quad meter \\ 1\ (US\ survey)\ inch &\approx 0.025\ 400\ 050\ 8 \quad meter \\ 1\ (British\ Imperial)\ inch &= 0.025\ 399\ 98 \quad meter \end{aligned} \tag{9.1}$$

If the prefix is omitted, the system can not detect which particular unit was intended. Another argument for unique identifiers is the different spelling of some units. An example would be the writing of the unit *meter* which can also be spelled *metre*. Not having a controlled vocabulary here may result in a computational overhead and possible errors while mapping between different labels of units. These and similar issues were also encountered and documented during a preceding study described at the end of this introduction [161].

Conversion of units This requirement is actually twofold: For one, users have to be able to explicitly convert a given column from one unit to another. For usability purposes, the list of possible conversion target units should be limited to compatible ones. So the system needs to be aware of the dimensions of units and the respective conversion formulae.

Another variation is implicit conversion. If users try to apply a formula that is dimensional consistent but not unit consistent, the system should be able to automatically insert the necessary conversions without requiring further user interaction.

Determination of result unit The system should be able to deduce the final unit of a given formulae automatically. This is quite easy if there are just summations and subtractions involved. If formulae go beyond that, though, this does also include the substitution with compound units. An example is the expression in Equation 9.2, which replaces a combination of three units with a single one for simplicity's sake. To facilitate such replacements the system needs to know about the decomposition of compound units.

$$1\ Kilogram * \frac{Meter}{Second^2} = 1\ Newton \tag{9.2}$$

[1] Usually the International System of Units (SI) [160] is used as a basis.

Minimizing the number of applied conversions For summations a naïve approach to inserting conversions could be to always convert the right-hand side operator to the unit of the left-hand side operator. However, the example given in Equation 9.3 shows that this might lead to suboptimal results in terms of the number of conversions applied.

$$x\ [m] + y\ [ft] + z\ [ft] \tag{9.3}$$

Evaluating the formula from left to right using the above rule results in two conversions to be applied: Converting both y and z to m. In this example, however, it is only necessary to convert x to ft to make the formula unit consistent. Fewer conversions lead to a smaller margin of error in the result, which is caused by inaccuracies in the supplied conversions or shortcomings of floating-point arithmetics (cf. Section 5.2).

The above list of requirements can roughly be divided into two groups: One group is concerned with a data source to supply all unit-related information like unique names, conversions, and other relations. The second group is more of an algorithmic nature. It represents the problem of finding the resulting unit of a formula and minimizing the number of required conversions and will be the main focus of this chapter.

With regard to finding a proper data source for unit related information, an extensive study was conducted in collaboration with Jan Martin Keil and involving Markus Steinberg, a MSc student at the time [161, 162, 163]. In the context of this work, the data source has to be compliant with the dataset descriptions discussed in Chapter 8 that are represented in an RDF-based format. So, the study focused on finding appropriate ontologies and compare them with one another to determine the currently most suitable ontology for this thesis and similar projects. With various ontologies been developed over the years, the study was split into two parts: First, generic use cases for unit ontologies were defined [162]. In a second step, the extent of ontologies with respect to provided individuals was determined [161]. The latter part also revealed a number of issues in all examined ontologies ranging from typos to erroneous conversion factors. These issues were reported to the ontology authors and have been fixed in some of the ontologies. The experiences during the study also led to general conclusions for the development of such and similar ontologies [163]. As a result of this study, the choice for this thesis fell on *Ontology of units of Measure and related concepts* (OM) [164]. Of the examined ontologies, it contained the smallest number of errors, includes the second largest number of units (only surpassed by Wikidata [121]), and as the sole ontology provided information about the unit composition as described before. In particular, that last aspect is a crucial component for the implementation of the approach presented in the remainder of this chapter.

9.1 Related Work

Support for units of measurements in programming languages has been the subject of research for many years now. Extensions to several languages (e.g., B [165], C [166, 167], GLISP [168]) have been proposed, but as of now, these efforts have not led to widespread adoption. Similarly, various software products have included solutions for the handling of units, examples of which are Mathematica [web103], Matlab [web104], R [web105], Octave [web106], SPSS [web107], or Stata [web108]. According to [166] current approaches can be categorized into three groups:

Library-based solutions add functionality to manipulate units by providing new classes or functions. By these means, programmers can ensure unit safety within their code.

Language or type system extensions extend the typing system or possibly the language syntax. This allows for validation of unit usage by the compiler or interpreter.

Annotation-based solutions require the user to add annotations to variables and functions. Tools can then analyze a program's source code to find possible unit errors.

The approach presented in [166], **CPF[UNITS]**, is itself an annotation-based solution for C. An example for annotated code is given in Listing 9.1. The annotations are embedded into C comments starting with either `//@` or `/*@` for line or block comments. They describe the pre- and postconditions of a function. The token `@result` is used to refer to the return value of a function. Furthermore, single variable assignments can be annotated using `//@ assume(UNITS)`. The set of used units has to be given by the software developers. They can specify base units like `meter` or `foot` and combinations thereof using algebraic operators like multiplication and exponentiation. One can also provide canonical forms of units, so that, e.g., `meter` can be an alias to m.

If within a conditional code block units of variables are changed, CPF[UNITS] will maintain multiple so-called "environments", where each environment represents a mapping between variables and units. Each statement afterward is evaluated once per environment, thus traversing all possible code paths.

The actual analysis is performed in two steps: In a first step, all direct annotations and assignments[2] are exploited to derive an initial environment. Afterward, expressions are evaluated according to a set of rules, which for each binary relation check certain conditions and return the resulting unit. For multiplication, this is just the concatenation of both operands' units as there are no further restrictions. For summation, the units of both operands have to be identical or a unit `fail` is returned to indicate the error. Comparisons — like summations — first validate the equality of both operands, before returning a unit `noUnit` as a placeholder for a dimensionless value. There are also similar rules to handle operations on structures and pointers. However, there

[2] `int x = y;` will also assign the unit of y to x.

```
typedef struct {
  double atomicWeight;
  double atomicNumber;
} Element;

//@ pre(UNITS): unit(material->atomicWeight) = kg
//@ pre(UNITS): unit(material->atomicNumber) = noUnit
//@ post(UNITS): unit(@result) = m ^ 2 kg ^ -1
double radiationLength(Element * material) {
  double A = material->atomicWeight;
  double Z = material->atomicNumber;
  double L = log( 184.15 / pow(Z, 1.0/3.0) );
  double Lp = log( 1194.0 / pow(Z, 2.0/3.0) );
  return ( 4.0 * alpha * re * re) * ( NA / A ) *
         ( Z * Z * L + Z * Lp );
}

//@ pre(UNITS): unit(material->atomicWeight) = kg
//@ pre(UNITS): unit(material->atomicNumber) = noUnit
//@ pre(UNITS): unit(density) = kg m ^ -3
//@ pre(UNITS): unit(thick) = m
//@ pre(UNITS): unit(initEnergy) = kg m ^ 2 s ^ -2
double finalEnergy(Element * material, double density,
  double thick, double initEnergy) {
  double X0 = radiationLength(material);
  return initEnergy / exp ( thick / X0 );
}
```

Listing 9.1: Sample C program with CPF[UNITS] annotations (from [166])

is no automatic conversion at any point – the annotated units are just propagated throughout the program and possible contradictions will be highlighted. In the example of Listing 9.1 this will result in an error message[3]:

```
Function finalEnergy: ERROR on line 26(1): Assert failed!
```

In [168] a **unit integration into GLISP** ("Generic LISP" [169]) is presented. According to the above classification, it represents a typing extension. Before using units, users have to define them as shown in Listing 9.2. Unit definitions are split into two groups: Simple units are given by name, conversion factor, and a list of aliases. The conversion factor is always relative to the respective SI [160] unit of the same dimension. So, e.g., `meter` has a conversion factor of 1, whereas `foot` uses 0.3048, as both are relative to the respective SI base unit, which in this

[3] The result of `radiationLength()` is given by $m^2 * kg^{-1}$ and `thick` is annotated to use m. In consequence, the exponent in line 26 is evaluated to $kg^1 * m^{-1}$, which violates a rule that requires the exponent to be unitless.

```
(defsimpleunits 'length
  '((meter 1.0 (m meters))
    (foot 0.3048 (ft feet))
    (angstrom 1.0e-10 (a angstroms))
    (parsec 3.083e16 (parsecs)) ) )

(defderivedunits 'force
  '((newton
      (/ (* kilogram meter) (* second second))
      (nt newtons))
    (pound-force
      (/ (* slug foot) (* second second))
      (lbf)) ) )
    (ounce-force
      (/ pound-force 16)
      ()) ) )
```

Listing 9.2: GLISP unit definitions (from [168])

case is `meter`. Compound units on the other hand are given by a composition of simple units. Their conversion factor is automatically calculated as the product or quotient of the conversion factors of their components. Units with conversions that include an offset different from zero are explicitly excluded from this work.

Units are also grouped by their dimension – `length` and `force` in the example of Listing 9.2. For each dimension, a dimension vector is defined according to the SI base dimensions with the addition of `money`[4] (cf. Table 9.1).

Index	Kind of Quantity	Unit
0	length	meter
1	time	second
2	temperature	kelvin
3	mass	kilogram
4	current	ampere
5	substance	mole
6	luminosity	candela
7	money	dollar

Table 9.1: Dimension vector definition of [168]. In the source "quantity" instead of "kind of quantity" has been used. This contradicts the definitions used throughout this work and has been replaced accordingly.

[4]The unit `dollar` is chosen as a base unit for `money`. No reason for this particular choice is given in [168].

The index assigned to each base dimension is used to serialize the dimension vector into a single integer. As the exponents within the dimension vector are usually quite small (±4 oftentimes less), the whole dimension vector fits into a single 32-bit integer. For this purpose two vectors are defined:

```
dimsizes = [ 20, 20, 20, 10, 10, 10, 10, 10 ]
dimvals  = [ 1, 20, 400, 8000, 80000, 800000, 8000000, 80000000 ]
```

The entries in `dimsizes` allocate a certain range of values for each component of the dimension vector. A value of 20 represents a range of ±9 and a value of 10 a range of ±4 respectively. So together with the mapping of Table 9.1, this defines the allowed range of exponents for `length` to ±9 while for `amount of substance` only ±4 is valid.

The second vector `dimvals` is derived from `dimsizes` by the means of Equation 9.4. It basically represents offsets, so integer representations can be computed by mere multiplication.

$$dimvals_i = \begin{cases} 1 & if\ i = 0 \\ dimvals_{i-1} \times dimsize_{i-1} & otherwise \end{cases} \tag{9.4}$$

The integer representation *dimint* of a dimension vector v can be computed using Equation 9.5. This approach does not recognize yet alone handle cases when the exponent for a base dimension is outside of the valid range. Equation 9.6 shows an example for computing *dimint* for the dimension `force` ($force = length^1 \times time^{-2} \times mass^1$). Having the integer representation of the vectors, dimensional analysis can be performed on (integer) numbers instead of vectors. The dimension vector for compound units can, e.g., be computed as a sum or difference of the dimension vectors of all compounds.

$$dimint(v) = \sum_{i=0}^{7} dimvals_i \times v_i \tag{9.5}$$

$$dimint([1,-2,0,1,0,0,0,0]) = (1 \times 1) + (-2 \times 20) + (1 \times 8000) = 7961 \tag{9.6}$$

Within the actual program, users can define their variables to use a unit in addition to the data type using

```
x: (units real meters)
```

While this example uses `meters`, users can insert any defined alias for a unit or a unit expression following the same rules as the unit definition.

For calculations, the compiler will check for dimensionally consistency of the given formula. If the formula is dimensional consistent, the resulting unit is determined. There are two major areas: Summations and subtractions are evaluated in a pairwise fashion. Here, the result of a single operation always uses the unit of the operand on the left-hand side. If the right-hand side operand uses a different unit, it will automatically be converted according to the defined rules.

```
1. The goal system of units is determined by the following priority:
   – user input
   – dominant system; estimated by counting unit occurences
   – SI system
2. Units are expanded to the equivalents in the base dimensions.
   Dimensionless units are converted to numbers.
3. Base units not present in the goal system are converted.
   Conversion factors are accumulated.
4. Numerator and denominator are sorted alphabetically.
5. Corresponding duplicate units are removed in a linear pass.
   This cancels units present both in numerator and denominator.
6. Derived units of the goal system (and their inverse)
   are tested against the result.
   The largest derived unit found replaces its compounds.
   This process is repeated until no further replacements are possible.
```

Listing 9.3: Simplification of compound units (from [168])

```
#define millimeter milli-meter
#define inch meter 39.370079
```

Listing 9.4: Osprey: Additional unit definitions (from [167]).

In the case of products and quotients, on the other hand, units are derived by multiplying or dividing the source units. Afterward, the system tries to simplify the result according to the algorithm given in Listing 9.3.

A typing extension for C is **Osprey** [167]. Listing 9.5 illustrates the typing system of Osprey using basically the same sample code as Listing 9.1. The types of variables are given in form of predefined units like `$meter` or `$unity`. Not all variables, however, have to include a unit as can be seen from lines 10-17. Osprey comes with a list of "aliases and abbreviations for commonly used units", but users are also able to add custom units in an additional configuration file. The definitions follow the syntax of C macro statements. Listing 9.4 defines `millimeter` as an alias for `millimeter` and the new unit `inch` including the conversion factor relative to meter. Units that include an offset in their conversions are not included yet but are referred to as future work.

At compile time, Osprey will traverse the abstract syntax tree of the program in general and the variable assignments in particular to create a set of constraints. Each binary operation will result in a separate constraint – most of them belonging two one of two forms: $u_a = u_b$ and $u_a = u_b \times u_c$, where u_a, u_b, and u_c are unit variables or constants. The former is used to indicate equality like used in variable assignments or summations or subtraction. The latter represents the results of multiplications and divisions.

```
1  double pow(double, $unity double);
2  $unity double log( $unity double);
3  $unity double exp( $unity double);

4  extern $unity double alpha, NA;
5  extern $meter double re;

6  typedef struct {
7    $kilogram double atomicWeight;
8    $unity double atomicNumber;
9  } Element;

10 double radiationLength(Element * material) {
11   double A = material->atomicWeight;
12   double Z = material->atomicNumber;
13   double L = log( 184.15 / pow(Z, 1.0/3) );
14   double Lp = log( 1194.0 / pow(Z, 2.0/3) );
15   return ( 4.0*alpha*re*re ) * ( NA/A )
16        * ( Z*Z*L + Z*Lp );
17 }

18 double finalEnergy(Element * material,
19   $kilogram*meter^-3 double density,
20   $meter double thick,
21   $kilogram*meter^2*second^-2 double initEnergy)
22 { double X0 = radiationLength(material);
23   return initEnergy / exp( thick / X0 );
24 }
```

Listing 9.5: Osprey: Sample C program using Osprey types (from [167]).

Each time a new variable with a type is allocated, the respective unit variable is bound to that unit. In formulae, a unit variable is created for the result of each binary sub-operation. In the unit variables' names, Osprey encodes the line number of the corresponding code fragment, the variable name, and other information that later allows pinpointing errors. For assignments and operations the respective constraints[5] are also created. Given the example of Listing 9.5, some examples for these constraints are given in Table 9.2.

These constraints are passed through a constraint resolution engine and fed into a Gaussian Elimination engine. In the process contradictions found are reported as errors. In the example of Listing 9.5, this leads to the following error message[6]:

[5] Both operands of a summation or subtraction need to have the same unit. The unit of a product or quotient will be the product or quotient of its operands' units respectively.

[6] The reason has been given in Footnote 3 before.

Line#	Source Code	Unit Environment	Constraints
4	`$unity double alpha;`	u_4_alpha: *unity*	-
11	`A=...->atomicWeight`	-	u_11_A = u_5_unamed@atomicWeight
16	Z * Lp	u_16_Z_MUL_Lp: δ	u_16_Z_MUL_Lp = u_12_Z * u_14_Lp
16	(Z... + ...)	-	u_16_Z_MUL_Z_MUL_L = u_16_Z_MUL_Lp

Table 9.2: Osprey: Example-constraints for code of Listing 9.5 (from [167]).

```
ERROR: The constraint:
   u_20_thick = u_23_thick_DIV_X0 * u_22_X0
is reduced to:
   $meter^1$ = $meter^2 kilogram^{-1}$
```

One further feature of Osprey is the ability to annotate constants within formulae as conversion factors. Listing 9.6 shows such an annotation – in this example for conversion between millimeter and inch. Osprey will include those conversions into the set of constraints to be validated as well.

```
$millimeter double mm;
$inch double inch;
mm = inch*($f)25.4;
```

Listing 9.6: Annotating a constant as conversion factor in Osprey (from [167])

An **annotation-based validation for B** is described in [165]. The annotation itself follows the already discussed ways. An example is shown in Listing 9.7. Variables can be annotated with units drawn from a pool of predefined units or combinations thereof. Furthermore, users are able to define their own aliases to known units using `unit_alias` or define new unit with `new_unit`. Defining a new unit, however, does not allow defining conversions to other, already defined units. They act as separate base units which form a base dimension of their own. Predefined units are restricted to the ones from the SI-system [160].

As shown in Listing 9.7 users are also able to annotate conversions as such to distinguish them from other operations. The system will in these cases validate the conversion factor.

Units that are no aliases are defined in terms of the SI base units. If prefixes are attached, they are replaced by the respective powers of ten[7]. This leads to the canonical representation of units where each one is represented by an (ordered) set of factors of the form $u = 10^p \times u^e$. By definition, each base unit can only appear once in such a set. The decision has been made to

[7] Conversions using other factors or requiring offsets are stored as exceptions.

```
MACHINE ConversionExample
VARIABLES
  /*@ unit 10**-2 * m */ x,
  /*@ unit 10**-3 * m */ y
INVARIANT x:NAT & y:NAT
INITIALISATION x, y := 0, 0
OPERATIONS
  mmToCm = x := /*@ conversion */ (10*y)
END
```

Listing 9.7: Annotating a B machine as given in [165]

preserve the prefixes for each component instead of having a single prefix for the whole compound unit. Improved traceability of errors is given as justification for this decision, although it is noted that this has a certain impact on other calculations.

This approach leads to a rather easy verification process of formulae: In the case of multiplication and division, the exponents of shared base units are added or subtracted, while the others are just added (possibly using the inverse). Existing prefixes are also multiplied or divided separately per base unit. The check for equivalence as needed for summation or subtraction boils down to comparing the exponents for each base unit as well as the products of all prefixes per operand.

Besides the approaches documented in literature, there are several **software packages** dealing with similar challenges. Systems like Mathematica [web103], Matlab [web104], R [web105], Octave [web106], SPSS [web107], or Stata [web108] allow users to perform analysis on different datasets. They vary in their main focus and support for units of measurements. Details of the respective approaches are rarely available. So to estimate the extent of their unit support a small empirical survey was compiled, whose results can be found in Appendix B.

SPSS and Stata were found to have no unit support at all. Octave provides a function to explicitly convert from one unit to another. This, however, places the burden of validating unit consistency solely on users. For R there are several packages to provide unit support. Besides those providing explicit conversions like in Octave, there is at least one package (`units` [web109]) that also deals with automatic conversions. Here, however, the naïve approach is used, which determines the result of a binary operation by just taking the unit of the left-hand side operand.

Mathematica and Matlab in their current releases both have native support for units of measurement. Mathematica includes composition and decomposition of compound units. Regarding unit conversions the survey suggests the system chooses the unit, which will result in fewer decimals – actual magnitudes of values omitted. Matlab on the other hand also supports decomposition of units, but support for the composition seems to be missing as of now. Regarding unit conversions no general strategy became apparent from the samples taken.

9.2 Discussion

Approaches found in the literature mostly focus on the technical details of adding unit support to specific programming languages. With units added, dimensional and unit consistency is validated for the given source code. Dimensional consistency is a weaker property compared to unit consistency. So most systems will only check explicitly for unit consistency, which includes the validation of dimensional consistency.

While language extensions require users to annotate all variables, annotation approaches are more flexible. When users only annotate some of their variables, some systems try to infer the missing annotations. This inference, however, seems somewhat limited. Some systems like [165] restrict the units to the SI system [160], which makes the task significantly easier: Only the prefix can vary for a given dimension, so there is no need to decide between units of different systems. Others like Osprey [167] are not restricted to a single system of units. However, cases like the summation of variables using different units for the same dimension are reported as errors here. Implicit conversions are not (yet) supported.

The current situation in common software packages seems similar. Only some systems support units at all. In these systems one can observe an evolution of unit support: From providing functions to explicitly convert between units [web106, web110], over adding distinct objects to encapsulate unit annotations for values [web109] to "real" support as part of the language used [web103, web104]. The latter then differs in the extent and sophistication of support.

At the time of writing, the support for units as found in Mathematica [web103] seems to be the most advanced. Neither is it limited to SI units nor does it seem to be overly biased towards them. The empiric results of Appendix B show a preference for the unit that results in the least amount of decimals no matter which system it belongs to. Mathematica is also able to recognize compound units as well as decompose a compound unit when necessary.

One point of discussion in Mathematica's strategy is the lack to recognize (or deliberately disregard) the dominant unit in a formula. If the majority of operands use a particular system, chances are, that the result is expected to have a unit of the same system.

There are also two drawbacks to Mathematica's approach when trying to apply it to the system described in this work. When applied to a rather small number of values (like usually the case in Mathematica) the number of conversions applied can be disregarded in terms of execution time. If, however, the amount of data grows, unnecessary conversion can have a serious impact. So from a performance point of view alone, the number of conversions should be kept as low as possible.

```
1. Convert the given formula to an AST
2. Optional: Apply pre-processing rules (post-order)
3. Walk the the AST in post-order and apply postProcess() to each node
4. Optional: Apply post-processing rules (pre-order)
             Recalculate conversion count
5. Simplify units for variants of the root node
6. Return unit list for the root node of the AST
```

Listing 9.8: Pseudocode to determine the result unit for a given formula.

Another disadvantage of a higher number of conversions is the impact on the numerical precision of the result[8]. As most conversions aside from prefixes are multiplications with non-integer factors and the precision of floating-point arithmetics is limited, each additional conversion will degrade the result to a certain degree. So again, a lower number of conversions will improve the results – this time with respect to quality and not just performance.

9.3 Approach

To reduce the overall number of conversions the approach has to shift from only considering local, binary relations to a more global one, which takes the whole formula into account. All approaches discussed before focus on local relations. They repeatedly replace relations by the respective result, thus successively working their way up to the result of the whole formula. While this is valid for the magnitudes of the values, it may increase the number of conversions applied unnecessarily.

A natural way to represent a formula is an Abstract Syntax Tree (AST) like the one shown in Figure 9.1(a). All of the approaches discussed before can be translated to a post-order walk over an AST, while choosing one of its children's units for each inner node. This general approach is kept with two modifications: Instead of propagating just one resulting unit from leaves to root, a list of reasonable units is used. A reasonable unit in this context is a unit that was present in at least one of the child nodes. Furthermore, for each node and unit, a counter is added to keep track of the conversions needed. This is used as a criterion to choose between different units for a single node. It can also be used to rank resulting units when presented to users. In addition, pre- and post-processing phases are added to enable further optimizations to reduce the conversion count. As these optimizations are optional, they will be discussed after the actual algorithm in Subsection 9.3.3.

The general workflow of the algorithm is given in Listing 9.8, while Listing 9.9 shows the processing of each node. A variant in the sense of the algorithm is a tuple of an operation (for inner nodes), a unit, a conversion count, and possibly child variants. It represents one possible

[8]For Mathematica this is of no concern, as all computations are performed symbolically.

instantiation of a node including all conversions needed in the subtree rooted at the respective node. These variants and the connections between them form ASTs themselves. They shadow the original AST and preserve its general structure, but may add some additional conversion nodes if necessary.

To enable a more meaningful way to do arithmetic with units the definitions of a unit U and its decomposition given in Equations 9.7 are used. A base unit[9] is a unit that can not be decomposed further. A compound unit consists of two products of base units – one for the numerator N and one for the denominator D. The list for numerator or denominator may be empty, which would yield a 1 as a result. For the unit *unity*, both lists are empty.

(9.7)
$$U = \begin{cases} U & \text{base unit} \\ \frac{N}{D} & \text{compound unit} \end{cases}$$
$$N = \prod_i N_i$$
$$D = \prod_j D_j$$

While addition and subtraction still require both operands to have the same unit, multiplication and division combine the units of both their operands as given in Equation 9.8 and Equation 9.9. Division uses the fact that it can be defined as reciprocal multiplication. After merging the respective numerators and denominators the unit should also be simplified. This process is similar to reducing a fraction but uses base units instead of prime factors.

(9.8)
$$U = U_{\text{left}} \times U_{\text{right}}$$
$$N = \prod_i N_i = \left(\prod_{j \in \text{left}} N_j \right) \times \left(\prod_{k \in \text{right}} N_k \right)$$
$$D = \prod_i D_i = \left(\prod_{j \in \text{left}} D_j \right) \times \left(\prod_{k \in \text{right}} D_k \right)$$

(9.9)
$$U = U_{\text{left}} \,/\, U_{\text{right}}$$
$$N = \prod_i N_i = \left(\prod_{j \in \text{left}} N_j \right) \times \left(\prod_{k \in \text{right}} D_k \right)$$
$$D = \prod_i D_i = \left(\prod_{j \in \text{left}} D_j \right) \times \left(\prod_{k \in \text{right}} N_k \right)$$

The definition of `processNode()` given in Listing 9.9 uses both definitions. The first block in lines 3 to 10 is basically an initialization phase. For each leaf of the AST a variant is created that contains the source unit as well as a conversion count of zero. The lines afterward deal with inner

[9]This is a different definition compared to a base unit for a system of units.

nodes, which represent the operations of the formula. The block from lines 12 to 22 represents multiplication and division as stated before. As the operands may have multiple variants — i.e., they can be represented using different units — a new result variant is added for each possible combination. In case multiple variants are created with the same unit, only the variant with the lowest conversion count is maintained. Finally, the last block starting in line 25 deals with sums and differences. A list `units` of all units present in the operands' variant sets is created (line 26). The algorithm then needs to validate dimensional consistency (lines 27 and 28), which a minor modification of the approach presented in Listing 9.3 is used for. For each of these units, a new variant for the current node is created in lines 29 to 41. As the children of the variant need to share the same unit the respective variants from the respective child nodes are picked in lines 32 to 35. Furthermore, in this pass, the total number of conversions needed is accumulated. If no matching variant exists, a new one is created that is based on the variant of that node that currently has the lowest conversion count (cf. Listing 9.10).

In a final step, the units of variants for the top node are simplified. Albeit technically not required, this step will improve the readability of the results. Otherwise, results like $[\frac{ft^2}{m \times s}]$ would be possible, which are unnecessarily hard to recognize as a unit of velocity. So in this case, the unit should get reduced to something like $[\frac{ft}{s}]$. To achieve this, first, all base units of the same dimension have to be unified, i.e. be converted to the same unit. The decision is made analogous to the choice given in [168] (cf. Listing 9.3, step 1): The dominant system of units is preferred. If this criterion results in a tie, then the SI unit will be chosen, as user input might not be available at this point. If there is a user selection available, it overrides any decisions made by the algorithm. Finally, the algorithm will try to substitute for compound units where possible again using the approach given in [168] (cf. Listing 9.3, step 6).

After processing the AST this way, a number of shadow trees have been generated – one for each possible result unit. They can all be accessed through the variants of the root AST node. As the number of required conversions was tracked throughout the process, the system can now order the resulting units by conversion effort needed. It can automatically select the most efficient one or present users with a more detailed list of options. If users request a valid unit that is currently not in the list, the system can again apply `getVariant()` to the root node and retrieve a new shadow tree for that.

The algorithm delays conversions as much as possible. This allows to take advantage of terms, that cancel out each other during the course of execution. If those values would be converted at the beginning (i.e. conversions are applied directly to all leaf nodes when necessary), superfluous conversions would have been introduced. This characteristic also allows certain optimizations, which will be discussed in Subsection 9.3.3.

The algorithm presented considers all conversions to be equal with respect to numerical stability. However, there are differences: Using standard floating-point arithmetics [128] the conversion from kilometer to meter will merely result in a change of the exponent by three. As long

```
FUNCTION processNode( node )

  IF node is a leaf
    Add a variant to node: (
      operator:  value,
      unit:      node.unit,
      convCount: 0,
      children:  null
    )
    RETURN

  IF node is a product or quotient
    left = left-hand child
    right = right-hand child
    FOREACH variant leftChild of left
      FOREACH variant rightChild of right
        Add a variant to node: (
          operator:  node.operator,
          unit:      leftChild.unit node.operator rightChild.unit,
          convCount: leftChild.convCount + rightChild.convCount,
          children:  [ leftChild, rightChild ]
        )
    RETURN

  IF node is a sum or difference
    units = all units of variants present in child nodes
    IF units contains units of more than one dimension
      THROW ERROR "dimensional inconsistency"
    FOREACH unit of units
      childVariants = []
      convCount = 0
      FOREACH child of child nodes
        variant = getVariant( child, unit )
        childVariants += variant
        convCount += variant.convCount
      Add a variant to node: (
        operator:  node.operator,
        unit:      unit,
        convCount: convCount,
        children:  childVariants
      )
    RETURN

END
```

Listing 9.9: Pseudocode to process an AST node.

```
FUNCTION getVariant( node, unit )

  IF node has variant using unit
    variant = pick variant using unit of node
    RETURN variant

  base = variant of node with lowest conversion count
  add variant newVariant to node: (
      operator:  conversion,
      unit:      unit,
      convCount: base.convCount + 1,
      children:  [ base ]
  )
  RETURN newVariant

END
```

Listing 9.10: Pseudocode to retrieve a specific variant for a node.

as the result is within the range of the exponent, this has no impact on precision. The conversion from inch to feet on the other hand involves a factor of $\frac{1}{12}$ [web102]. Here, the conversion factor itself can not be exactly represented using the double-precision standard. Hence, all conversions will include a small error[10]. In contrast, the inverse conversion from feet to inch will use a factor of 12, which most often will not cause any issues.

The examples show, that some conversions are more preferable than others. The algorithm can account for this by introducing additional weights for the conversions. Where the default execution uses a weight of 1 when counting conversions, a more sophisticated approach could use different weights based on numeric stability. That way the final ranking would favor more stable results. The exact values of these weights, however, require further considerations, which are out of the scope of this work.

9.3.1 Example

In Figure 9.1 the algorithm is successively applied to the formula $\frac{[m]+[ft]}{[s]} + \frac{[ft]}{[s]}$. First, the formula is transformed to the AST shown in Figure 9.1(b). For simplicity's sake, the magnitudes of values are omitted as will, later on, be the connections between variants including the respective added conversions. As of now, no rules for pre- and post-processing have been defined, so steps 2 and 4 of Listing 9.8 can be skipped and the processing is reduced to the repeated application of `processNode()` (cf. Listing 9.9) to all nodes in a post-order walk.

[10]The error most often is negligible, but on occasion, it might cause problems.

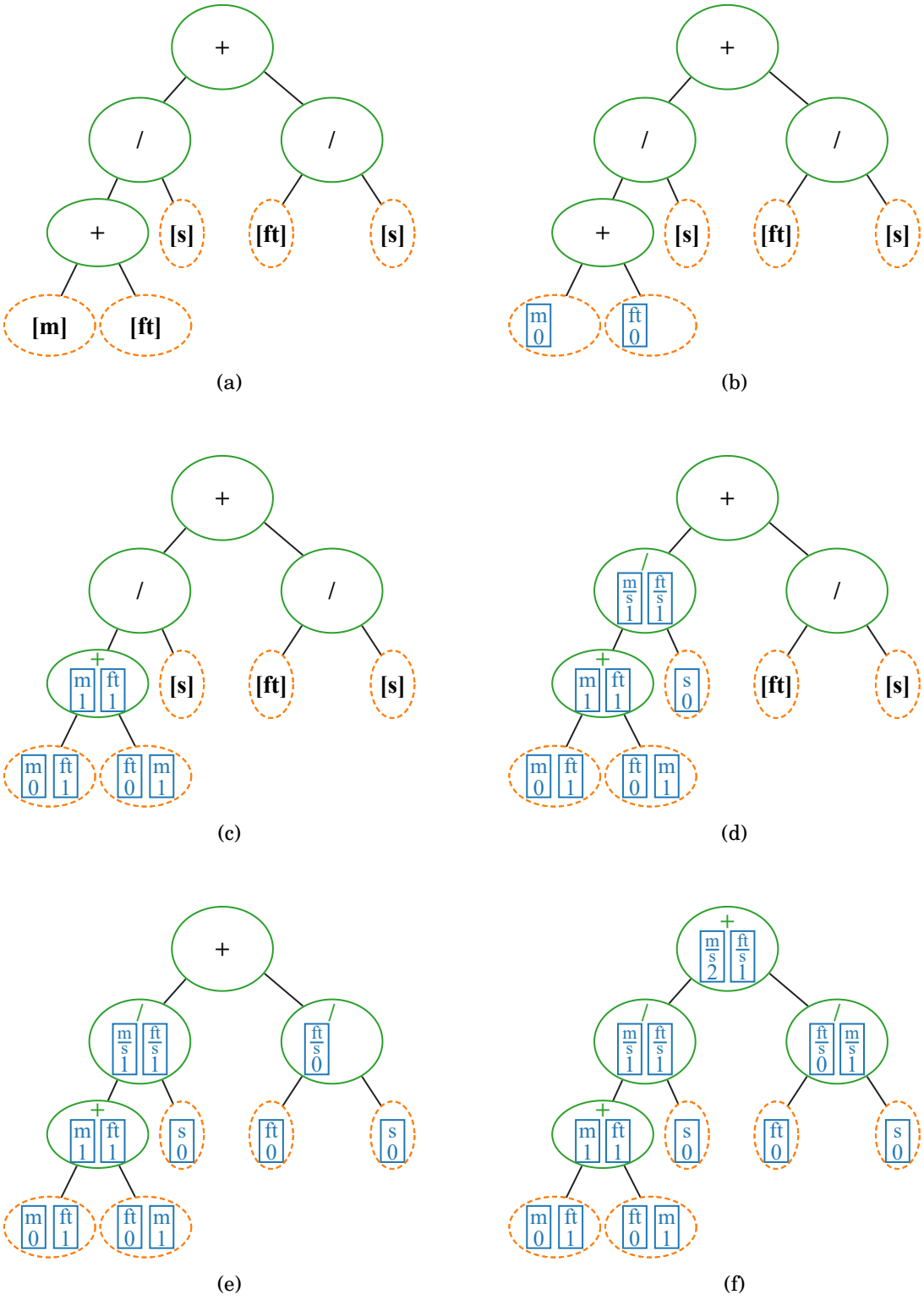
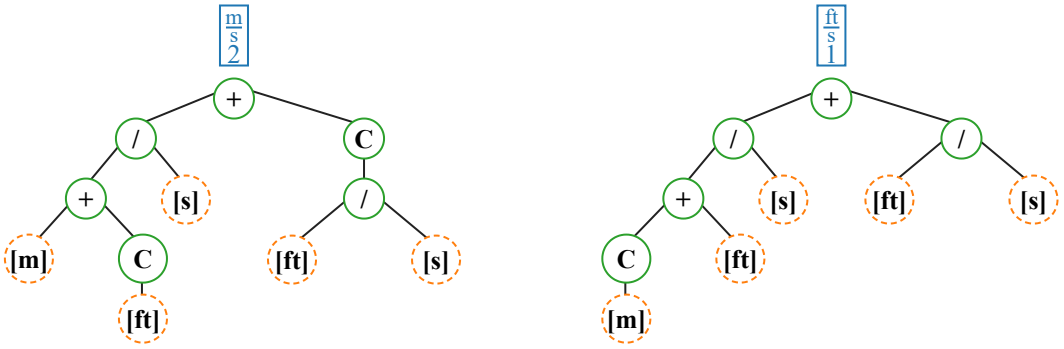


Figure 9.1: Successive application of unit resolving to formula: $\frac{[m]+[ft]}{[s]} + \frac{[ft]}{[s]}$.
Inner nodes (ellipsis, solid), leaf nodes (ellipsis, dashed), variants (rectangles, solid).
Variants are given by unit and conversion count. Their connections are omitted.



(a) (b)

Figure 9.2: Resulting ASTs after applying unit resolving for formula: $\frac{[m]+[ft]}{[s]} + \frac{[ft]}{[s]}$. Conversion nodes are marked by a "C". Conversion details have been omitted.

The algorithm will first traverse the AST until it reaches leaf nodes. The first two leaf nodes encountered are the ones on the lower left containing the units $[m]$ and $[ft]$. The respective variants with a conversion count of zero are created and added to the respective AST nodes. The state after processing both nodes is shown in Figure 9.1(b).

Afterward, the lower-left summation node is visited. The first step is to pull all units from the child nodes variants. In this case, the result is a list with two elements: $[m]$ and $[ft]$. For both these units, a variant to the summation has to be created. In order to have matching operands, the algorithm has to add new variants to the child nodes. So the left-hand side operand has to be converted to $[ft]$, while for the right-hand side a variant with unit $[m]$ is added. Both these new variants have a conversion count of one as they had to be derived. Now the respective variants including links to the operands can be added to the summation node as shown in Figure 9.1(c).

The next visited node is the left leaf node using $[s]$ as a unit. It is processed in the same way as the other leaf nodes before, adding a single variant with conversion count zero. In the left division node, all combinations of units from the left-hand side and right-hand side operators have to be combined. The left-hand side operator currently has two units ($[m]$ and $[ft]$), while the right-hand side one has just one ($[s]$). Hence, two variants are added to the division node representing the respective results. This results in the state shown in Figure 9.1(d), where there is one variant using $\frac{[m]}{[s]}$ and another one using $\frac{[ft]}{[s]}$.

The right-hand side subtree node is processed similarly. Here again, one variant is added to each leaf node. The division node, however, has just one variant as both its children have just one variant each. The state of the AST after the processing of the right division node is shown in Figure 9.1(e).

The last node processed is the root node. Again all units of the child nodes' variants are collected, which this times yields $\frac{[m]}{[s]}$ and $\frac{[ft]}{[s]}$. The left-hand side child already has variants for both units. The right-hand side child, however, just has a variant for $\frac{[ft]}{[s]}$. So to represent all units in the root node, a new variant has to be created in the right child node. The algorithm uses the

variant with the lowest conversion count as a bases, which is the variant using $\frac{[ft]}{[s]}$ as it is the only variant present. After that variant is added, all units can be represented in the root node and the AST takes the form of Figure 9.1(f).

All variants of inner nodes have connections to their operands or the source variant they were derived from. The respective ASTs for specific result units are already given by that. In Figure 9.2 the final ASTs for both resulting units of the example are shown.

9.3.2 Limitations

While the described approach works fine for most units, there is one group of compound units that necessitates some modifications. If one component of a compound unit is a scaled unit, conversion is not defined unambiguously. The general formula for a unit conversion is given in Equation 9.10, where $f_{U \to A}$ is the factor and $o_{U \to A}$ the offset used for a conversion from unit U to unit A.

$$xU = (x \times f_{U \to A} + o_{U \to A})A = yA \tag{9.10}$$

For most units in use, the conversion offset is equal to zero. There is, however, at least one group of units for which the offset has a value different than zero: temperatures. The conversion from degree Celsius to degree Fahrenheit, for example, uses an offset of 32 and a factor of 1.8. If one component of a compound unit is such a scaled unit, the order of conversions matters for the result as illustrated in Equation 9.11 and Equation 9.12.

$$\begin{aligned} xUV &= (x \times f_{U \to A} + o_{U \to A})AV \\ &= ((x \times f_{U \to A} + o_{U \to A})A \times f_{V \to B})B \\ &= (x \times f_{U \to A} \times f_{V \to B} + o_{U \to A} \times f_{V \to B})AB \end{aligned} \tag{9.11}$$

$$\begin{aligned} xUV &= (x \times f_{V \to B})UB \\ &= (x \times f_{V \to B} \times A \times f_{U \to A} + o_{U \to A})B \\ &= (x \times f_{V \to B} \times f_{U \to A} + o_{U \to A})AB \end{aligned} \tag{9.12}$$

Computing the difference between both orderings yields a value different than zero in contrast to unit conversions without offsets.

$$\begin{aligned} &(x \times f_{U \to A} \times f_{V \to B} + o_{U \to A} \times f_{V \to B}) - (x \times f_{V \to B} \times f_{U \to A} + o_{U \to A}) \\ &= o_{U \to A} \times f_{V \to B} - o_{U \to A} \\ &= o_{U \to A} \times (f_{V \to B} - 1) \end{aligned} \tag{9.13}$$

In these cases, an unambiguous conversion seems not possible. In an attempt to partly rectify the situation, the algorithm is adjusted as follows. Before traversing the AST, the system will scan for occurrences of scaled units in the input and group them by dimension. Depending on the characteristics of the dimension group, different actions will be taken for each one:

- Dimension groups containing no more than one scaled unit.
  None of the scaled units will have to be converted and, hence, no adjustments are required.

- Dimension groups containing more than one scaled unit, but none of these are part of any compound unit – i.e. they appear just as single base units.
  Additional variants are added in the initialization step of the algorithm (cf. lines 3 to 10 of Listing 9.9). In each node that uses one of those scaled units, variants are added for all other units of the same dimension group.

- Dimension groups containing more than one scaled unit and some of these are also part of compound units.
  Here, two sub-cases have to be distinguished:

  - Just one of the scaled units is present in compound units.
    This can be handled like dimensions groups with just scaled units that do not appear in compound units. The difference, however, is, that only variants for the one unit appearing in compound units have to be added.
  - Multiple units of this dimension group appear in compound units.
    There is a chance, the algorithm will have to convert between two scaled compound units at some point. For the previously mentioned reasons, it can not do this unambiguously. So in this case an error is thrown and the algorithm will abort.

Using the above measures, some of the ambiguous situations can be avoided. For all scaled units that might need to be converted the respective variants are added to nodes, where they appear as base units and conversions are clearly defined. As a consequence, the last block of Listing 9.9 might add two variants with the same (compound) unit to a node. In these cases, only one variant is maintained, namely the one with the lower conversion count.

Some formulae that after the discussed algorithm extension are rejected with an error, could be processed using a different technique. Consider, e.g., the formula ([m°C]) / ([m°F])[11], which will have a dimensionless result. This formula could be reduced to something like [°C] / [°F], where again the scaled units appear on their own and can be converted. To cater for such cases, one could still apply the above changes, but do not throw an error on the last condition. Instead, the algorithm would try to process all nodes as given in Listing 9.8. Variants that would require

[11] Meter times degree Celsius and meter times degree Fahrenheit, respectively.

a conversion of a compound scaled unit would be dropped. If after executing `processNode()` a node does not include a single variant attached to it, an error will be raised. In that case, the algorithm was not able to mitigate the ambiguity problem.

Another solution approach at this point would be an attempt to restructure or reorder the formula. This, however, would require other techniques, which are for the most part outside of the scope of this work.

9.3.3 Optimizations

As already indicated before, certain restructurings of the AST can yield additional benefits towards reducing the number of conversions needed. The algorithm as given in Listing 9.8 already includes two optional steps, in which rules to change the AST structure may be applied. These rules should only reorder the AST to reduce the number of conversions, but not change the result in any way. While more rules are possible, in the following a few examples will be described to illustrate the potential impact.

As a motivational example for the first rule, assume the following formula which sums up a list of length values:

$$[m] + [ft] + [m] + [ft] + [m] + [ft] + [m] \tag{9.14}$$

In the default algorithm in total three conversions would be applied – one for each occurrence of a value measured using [*ft*]. If the formula, however, is restructured as follows, this number could be reduced to one without changing the core algorithm.

$$([m] + [m] + [m] + [m]) + ([ft] + [ft] + [ft]) \tag{9.15}$$

The basic idea is to sort sums by the unit of the input provided. As a first step, the rule given in Listing 9.11 is added to the pre-processing phase. The rule only affects nodes that represent summations (line 3). If any child nodes themselves are summations, it will move their children to the parent node's child list. Repeated application of this rule will collect all operands of the sum at a single node. A visual illustration of this rule is given in Figure 9.3.

In contrast to multiplication and division, the algorithm does not limit the number of operands of summations to two (cf. lines 25 to 41 of Listing 9.9). In consequence, it is able to process summation nodes with more child nodes attached.

After the main processing step, a second rule (cf. Listing 9.12) is applied to convert the AST back to a state that uses only binary summations. In general, it groups the used variants by unit – however, for variants that represent conversions (i.e. the input node did not supply the required unit) the respective source variant's unit is used instead (lines 4 to 6). Afterward, each group is converted to a binary tree. If necessary, a conversion is added to the root node of that subtree (lines 10 and 11). Finally, the list of children of the original node is replaced by the list of subtrees' roots.

```
FUNCTION collectSums( node )
2
  IF node.operator == +
    FOREACH child of child nodes
5     IF child has operator +
        Remove child from node's child nodes
        Add child nodes of child to node's child nodes
```

Listing 9.11: Pseudocode for rule to collect operands of summations in pre-processing.

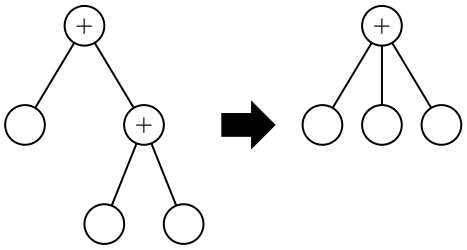


Figure 9.3: Visual illustration of summation collection rule.

```
FUNCTION splitSums( node )
2
  IF node.operator == +
    children = child nodes of node
5   Replace converted variants in children by their source
    groups = Group children by unit
    childList = []
8   FOREACH group of groups
      head = create binary tree from group members
      IF head.unit != node
11      head = getVariant( head, node.unit )
      Add head to childList
    node.children = childList
```

Listing 9.12: Pseudocode for rule to split up summations in post-processing.

In order to be able to apply the aforementioned rules to differences as well, another rule can be applied earlier in the pre-processing to convert differences to summations. This is simply done by transforming the binary difference-relation to a unary negation followed by a sum. This rule is given in Listing 9.13 and by the following formula:

$$a - b = a + (-b) \tag{9.16}$$

```
FUNCTION transformDifference( node )

  IF node.operator == -
    right = right-hand child
    newNode = unary negation of right
    Replace right-hand side child with newNode
    node.operator = +
```

Listing 9.13: Pseudocode for rule to split up summations in post-processing.

Other restructuring rules are possible. One could, e.g., restructure formulae of the form $\frac{[a]+[b]}{[c]}$ to $\frac{[a]}{[c]} + \frac{[b]}{[c]}$. Although this would further increase the applicability of the summation rule mentioned before, it comes at the cost of an additional division. This might outweigh the benefits of saving on the amount of conversion, which is why it is not listed as a default optimization rule here. The same holds true for the respective multiplication rule.

As some of the transformations may change the total conversion count, it has to be recalculated as part of the post-processing step.

CHAPTER 10

DATASET COMBINATIONS

The visualization process requires a single, consolidated dataset including all data necessary for the specific task. The effort needed to create such a consolidated dataset varies substantially. If the visualization task is in line with the original intent of the data producer, all data necessary might be contained within a dataset. However, the intentions of data producers and consumers can also differ quite a lot. In these situations, required data might be scattered over multiple datasets even created by different data producers [64].

Data consumers have to accomplish two tasks to retrieve a consolidated dataset. First, they have to identify and retrieve datasets that can contribute to the result. Having found matching datasets, these will need to be integrated in a second step. Traditional search implementations are oftentimes restricted to metadata descriptions and will only search in fields like title or at best column headers. Primary data is usually neglected, though, forcing users to check each dataset individually to verify whether the requested data is actually contained within a dataset. When the required data is spread across multiple datasets and join- or union-operations need to be applied, those need again to be specified by users in a mostly manual manner.

For non-tech-savvy users, these tasks may prove too big a barrier to entry. Here, a query-by-example approach similar to the one presented in [122] seems promising. Instead of manual search, users specify a table structure that represents their current search goal. The system subsequently looks through the available data sources and attempts to fulfill the search request.

For this thesis, data integration challenges like duplicate detection or schema mapping are mostly considered out of scope. Furthermore, in order to distinguish the given approach from previous work on data integration and data fusion, the general problem is labeled dataset combination. The main goal in dataset combination is to find a workflow that transforms and combines multiple data sources to fulfill users' search requests as given by an example table.

10.1 Related Work

The **Mannheim Search Join Engine** (MSJ Engine) [61, 170, 171, web111] is a system to augment a given (relational) table with additional information gathered from websites. There are two types of input tables to the system: *constrained* and *unconstrained* queries. While constrained queries just ask for the addition of a single specified attribute, for unconstrained queries the system will add all matching attributes to the input. Both kinds of queries are answered using the same three-step process: Table Indexing, Table Search, and Data Consolidation.

In the *Table Indexing* step, the given corpus is preprocessed and indexed within Lucene [web112]. Apart from tabular datasets like the WikiTables Dataset [172] and the WebDataCommons HTML Tables Dataset [173], the approach is also applied on RDF-based datasets like the WebDataCommons Microdata Dataset [174] and the Billion Triples Challenge 2014 Dataset [web113]. The former two can immediately be converted to an internal subject-attribute table. However, the RDF-based datasets have to be converted to a tabular structure first: For each class, a separate table is constructed and all RDF-predicates are added as attributes accordingly. In the case of multi-valued objects, only the first value is maintained.

In order to be indexed, a table has to fulfill two criteria: It has to contain at least three attributes (columns) and it must contain a subject attribute. The subject attribute is identified by the use of the following heuristic: If the table contains an attribute `rdfs:label` or the respective table header contains the string `name`, it is selected as a subject attribute. If this condition fails, the attribute with the highest count of distinct values is selected[1]. Ties are broken by choosing the left-most attribute.

Furthermore, by the use of manually defined rules and regular expressions, the data types number, timestamp, and geo-coordinate are identified. Another set of rules enables the detection of units of measurement and converts those to the respective base unit. As the evaluation is using English queries, tables containing data in other languages are also filtered as is adult content. These transformation and filter rules result in a usable corpus of 36,337,000 tables total. This represents about 25 % of the input datasets' tables. Before being indexed in Lucene, all cells' values are normalized, i.e. they are tokenized, lowercased, and values in brackets as well as stop words are removed.

In the *Table Search* step, the actual query is posed. The input table is processed in the same manner as the tables of the corpus before, i.e. in particular a subject attribute is identified. A search operator will then try to find matching subject values across all indexed tables. Two approaches are used to find such matches: an exact matching as well as a FastJoin matcher [175], which allows for fuzzy matching of tokens. Tables, where at least one subject value could be matched, are ranked according to Equation 10.1, which describes an average similarity between input and corpus table regarding the subject values contained. Here, $T(t)$ is a table within the

[1] Only attributes with at least 60 % unique values are considered here.

corpus, $T(q)$ is the query table, $|T(q)|$ denotes the cardinality of the query table, and $l(q.k, t.k)$ the Lucene similarity score of the matched subject values. Only the top-k[2] results are considered further on.

$$r_{T(t)} = \frac{1}{|T(q)|} \sum l(q.k, t.k) \tag{10.1}$$

All found attributes are appended to the query table using a series of left outer joins called a multi-join operator. Afterward, the third step *Data Consolidation* is applied which depends on the query type.

For constrained queries only those attributes are kept, whose attribute header contains the specified header. So for a queried header “GDP”, attributes with headers like “Total GDP” or “GDP (US$)” are retained. The remaining attributes are consolidated in a row-wise fashion. All attribute values belonging to a single subject value are now clustered using the similarity measures proposed in [176]. The final value is selected by choosing the cluster with the highest number of elements and then picking the most frequent element of that cluster. It is argued, that this better accounts for different spellings present in the corpus. With a majority vote alone, “Ipoh” might be picked as the capital of Malaysia as the correct answer is present with multiple different spellings each of a lower frequency: “kuala lumpu”, “kuala lumpua”, and “kuala lumpur”.

The results for unconstrained queries are processed in a similar way. Here, attributes with the same semantic intention have to be matched. This is done by the use of two different approaches: The first approach matches attributes based on their values for each subject using the same techniques as in the clustering before. The second approach relies on matching attribute headers. First, a matching to background knowledge bases like DBpedia [152], YAGO [177], and Wordnet [178] is attempted, after which is-a relations of those knowledge bases are exploited [179]. If no matching concepts can be found this approach falls back to string similarity using Levenshtein distance [180]. The results of both approaches are combined afterward. Regarding the attribute values, users may choose between different conflict resolution strategies [181]. For the evaluation, voting is used for nominal attributes, while the median is selected for numeric ones.

In an evaluation, the authors indexed tables of the four aforementioned corpora and ran “several constrained and unconstrained queries covering different topical domains”. For constrained queries, they report a coverage between 88 % and 100 %, if exact subject matching was used, and a coverage between 95 % and 100 % for similar subject matching. The manually evaluated precision ranges between 67 % and 100 % for exact matching and 100 % for similar matching For unconstrained queries, between 1700 and 2700 attributes were added which covered at least 50 % of the subject values. Here, no precision evaluation is given. However, they also report that running a table extension query using their example corpus and a single machine takes “several minutes” to complete [170].

[2]For the evaluation $k = 1000$ is chosen.

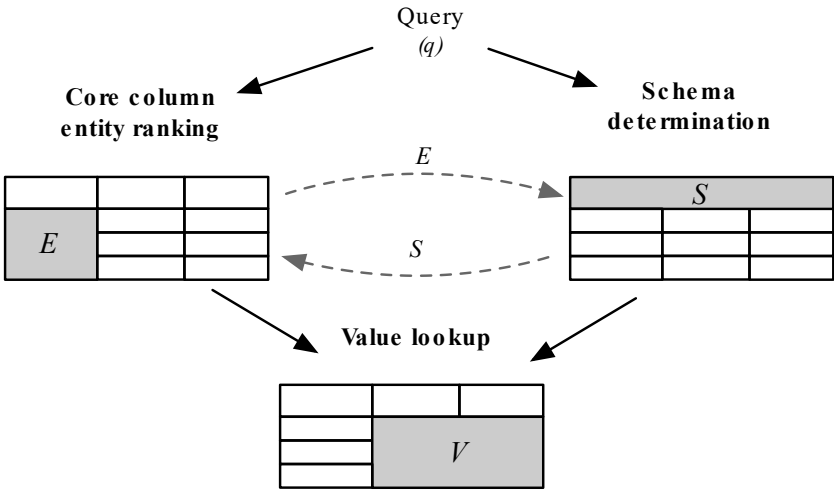


Figure 10.1: “On-the-fly Table Generation”: Approach (from [182]).

A different take on the problem is presented in **“On-the-fly Table Generation”** [182]. Here, the assumption is that many keyword-based queries can be answered by a summarizing table instead of a ranked list of entities. Subsequently, the overall task is split into three different components as shown in Figure 10.1. The individual components do not work in isolation but inform each other, resulting in an iterative process to generate the final table. The approach is based not only on the information in a table corpus but also relies on a knowledge graph. Consequently, many of the ranking signals in the following are present in two variations, once for the table corpus and once for the knowledge graph.

Core column entity ranking identifies the concepts in the query to be used in the core column.

Schema determination determines the header labels used to construct the table.

Value lookup retrieves values for the individual cells of the table.

The approach is based on the assumption that the requested table describes only one entity per row, the so-called “core entity”. During core column entity ranking, several different techniques are exploited to match the input query to entities that may be included. Initially, the approach relies on term-based matching using a Language Modeling approach [183] and a deep relevance matching model (DRMM) [184]. While the former relies on statistical measures like TF-IDF [185], the latter employs a deep learning model to match queries to documents (or entity descriptions in this case). Starting with the second iteration, candidates for the schema information are available as well. For the DRMM model, the concatenation of all schema candidates is appended to the query in order to improve the matching accuracy. A final ranking signal is given by the so-called “entity-schema compatibility”. This compatibility is determined by the candidate entity having

properties corresponding to the identified schema labels. In the knowledge graph this directly translates to properties, whereas in the table corpus it is evidenced by at least one table that includes the respective entity and a corresponding header field.

Similarly, schema determination depends on multiple ranking signals. A first signal, column population, matches the query to tables of the corpus. Relying on prior work [186], BM25 is used to determine a set of tables τ from the corpus that is relevant to the query. Candidate headers are then matches to the headers of the tables from τ using the maximum edit distance [61]. The second signal applies the same DRMM-based approach as the core column entity ranking. Once initial core column entity candidates are available, three more signals can be exploited: The column population is extended by a factor that captures how many of the core column entities are covered by a given table. A second signal defines the task as an attribute search for the current set of core column entities and follows the method proposed by Kopliku et al. [187]. The final signal is again given by the entity-schema compatibility in the same manner as before.

From each round, the top-k ($k = 10$ has been reported as the best choice) candidates of both core column entity ranking and schema determination serve as input to the next iteration. Once the set of candidates has stabilized across iterations, the final component, the value lookup is executed. This lookup is done for each cell independently. First, a list of candidate values is retrieved from the table corpus by using a fuzzy string matching of header and entity labels. This list is subsequently ranked by a so-called confidence score that depends on the relevancy of a given table to the query (cf. the selection process for τ during the schema determination). However, results from the knowledge graph, if existent, are always prioritized over the ones found in the table corpus. The authors reason that the data contained within the knowledge graph is manually curated and thus of higher quality compared to the table corpus. In the initial publication [182] only the top-ranked value is included in the result. Subsequent work also proposed user interactions to select among other high-ranking candidates [188].

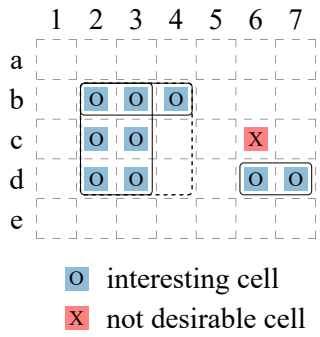
A related problem is posed within the OLAP community under the name of **Minimum description length (MDL) summarization** [189, 190]. The problem can be stated as follows: Given a k-dimensional OLAP cube and a user query, all cells of said cube are marked as either blue (“interesting”), red (“not desirable”), or white (“don’t care”) [190]. The task is to find a minimal set of rectangular regions within the cube, such that

(a) all blue cells are covered by a region,

(b) no red cell is covered by a region, and

(c) a minimal number of white cells is covered by a region.

An example is given in Figure 10.2. Here, the OLAP-cube has two dimensions whose values are denoted by numbers and letters. A covering set of regions is, e.g., given by regions $R1$, $R2$, and $R3$. For this set, the number of included white cells, also-called budget, is zero. Another set of regions



$R1 = (\, b-d,\, 2-3\,)$
$R2 = (\, b,\, 2-4\,)$
$R3 = (\, d,\, 6-7\,)$

$R4 = (\, b-d,\, 2-4\,)$

Figure 10.2: MDL summarization: Example.

consisting of $R3$ and $R4$ has a smaller cardinality but will include two white cells as well. In this example, this second set is minimal in terms of regions required. A region like $(\, b-d,\, 2-7\,)$ is not possible due to the not desirable cell at $(\, c,\, 6\,)$.

In general, this problem is NP-complete [191], but faster heuristic algorithms have been proposed. One of these is called *CAS-Interior* [190] whose pseudo-code is given in Listing 10.1.

```
1  Build indices I_A, I_B for all the red and blue cells respectively.
   Construct a single Region R containing all the blue cells.
   Initialize the covering C to contain R only.
   Initialize curr—consumption to be the number of white cells in R.

2  WHILE ( there exists R∈C containing a red cell )

   2.1  Grow the red cell in R to a larger region not containing any
        blue cell using the blue index I_B.

   2.2  Split R into (at most) 2d regions, where d is the dimensionality
        of the space, excluding the entire red region.

   2.3  Remove R from C, but add the split regions into C.
        Update cur_consumption.

3  WHILE ( cur_consumption > white budget )

        Do splitting similar to step (2), this time growing based on white cells.

4  Return all the regions in C as the MDL covering.
```

Listing 10.1: MDL-summarization: CAS-Interior (from [190]).

The algorithm works in a top-down manner and successively removes sub-regions that contain red or white cells. For the actual splitting, a reference to R-tree node splitting [192] is given. Each split operation may result in $2 \times d$ regions as illustrated in Figure 10.3. A red cell is located at $(\, c,\, 5\,)$. The region around this red cell is expanded as long as it does not contain any blue cell

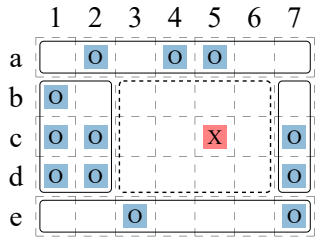


Figure 10.3: CAS-Interior: Worst case splitting (after [190]).

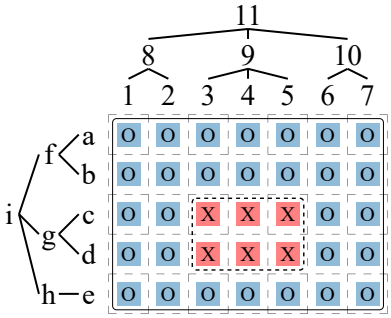


Figure 10.4: MDLH: Example.

leading to a region ($b-d,\ 3-6$) (dashed border) to be removed finally. As all regions have to be rectangular in shape, this limits the size of the region to remove and forces four new regions after the split.

It is noted that the regions are non-overlapping. The main reason given is that this reduces the necessary efforts to keep track of the number of white cells. With non-overlapping regions, each white cell is part of at most one region and thus possible multiple counting of a single white cell is prevented.

As an extension of this algorithm, *CAS-Corner* is mentioned. Instead of using a random red and white cell seed for the splitting regions, corners are examined for such seeds first. If a matching cell is found and the respective region is removed, at most two remaining regions will be created, thus lowering the number of regions created.

Another approach is given in [193]: *MDL summarization with holes* (MDLH). The budget for white cells used before is omitted and replaced by a definition of regions with holes. Assume the example given in Figure 10.4. In contrast to the examples before, hierarchies over the dimensions have been added. No matter the approach, this results in a solution using four regions along the borders. If, however, holes are allowed, the solution can also be given as ($i,\ 11$) $\ominus$ ($g,\ 9$), reducing the cardinality of the solution from four (regions) to two. To this purpose, the algorithms presented in [193] define the notion of benefit:

$$\text{Ben}(\ S\ \ominus\ H\ )\ =\ |D|\ -\ (\ |S|\ +\ |H|\ ) \tag{10.2}$$

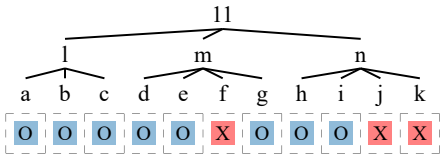


Figure 10.5: MDLH: One-dimensional example.

Here, S denotes the set of regions covering all blue cells, H the set of non-blue cells, and D is a collection of blue cells. The benefit in essence describes the gain of describing D by use of S and H instead of D alone. In other words, how much shorter is the description using regions and holes as compared to a listing of cells?

This benefit can also be applied to elements of the dimensional hierarchy as they themselves describe a certain region. Consider the one-dimensional example of Figure 10.5. The benefit for (m) and (n) is calculated as follows:

$$\begin{aligned} \text{Ben}(m) &= 3 - (\,|\{(m)\}| - |\{(f)\}|\,) &&= 3 - (\,1 + 1\,) = 1 \\ \text{Ben}(n) &= 2 - (\,|\{(n)\}| - |\{(j),(k)\}|\,) &&= 2 - (\,1 + 2\,) = -1 \end{aligned} \tag{10.3}$$

So while the use of (m) as a region yields a positive benefit, using (n) results in a negative benefit as the description for holes can not be shortened.

Using this benefit definition, the algorithm MDLH-Greedy is presented. The pseudo-code is given in Listing 10.2. In the first step, S, the set of solution regions, is initialized to contain a list of all blue cells. Furthermore, the list of holes H is set to an empty set, while two auxiliary sets S' and H' are also set to empty sets. Afterward, the benefit is calculated for all so-called "parent-regions". A parent region is a region, that uses one parent node from one dimension while using leaves for all other dimensions. So in the example of Figure 10.4, $(\,g,\,2\,)$ constitutes a parent region, while $(\,g,\,8\,)$ does not. The list of parent-regions and respective benefits is subsequently sorted by benefit (descending) and the number of holes (ascending).

This sorted list is processed from start to end in steps three and four. Step three simulates the addition of region x to the solution by removing the contained blue cells from the solution and adding non-blue cells to the list of holes. Step four checks whether this solution is advantageous over the current solution and adopts the new solution in case of improvements. Step six marks all selected holes blue and uses another algorithm[3] to simplify the solution S. After marking all holes blue, selected regions so far are rectangular and without holes. So using the default algorithm with a budget of zero will return a reduced solution for the set of selected regions S. Finally, the reduced set of regions along with the accumulated set of holes is returned as the output of the algorithm.

[3]The authors at this point chose the MDL-Tree algorithm [190], but any other MDL-algorithm would suffice.

```
Input: a k–d data cube and the set of blue cells D
Output: an MDLH description for D

1  /* H and H' are sets of holes. S ⊖ H is a description for D.
   Each time we select a region x, S' contains the blue cells in S but not in x.
   H' contains holes in H and x. S' ⊖ H' is the new description by selecting x.*/
   Initialize S = D and H = S0 = H0 = ∅.

2  Compute the benefit of each parent region and sort those regions with non–negative benefits to
   form a sorted list in descending order of benefits but in ascending order of the number of holes.

3  Obtain the next best unprocessed region x from the sorted list.
   Set S' = S − BlueCell(x), and H' = H + NonBlueCell(x).

4  If (|S'| + |H'| < |S| + |H|) then { S = S' and H = H' }.

5  If there is any unprocessed region in the sorted list, then go to step 3.

6  Mark each cell in H as blue. Apply MDL–Tree algorithm on all the blue cells and the description
   generated is stored in S

7  Return S ⊖ H as the MDLH description for D.
```

Listing 10.2: MDL-summarization: MDLH-Greedy (from [193]).

There are a few things to note regarding this algorithm: First, the selection of only parent regions seems to restrict the solution space at first glance. The authors argue that this approach reduces the number of regions to check in steps three and four. On the other hand, possible improvements of selecting a parent node instead of its children to form a region are gathered in step six.

Another important aspect is that the benefit is not recalculated after selecting a specific region. The authors state that step four includes an implicit calculation of the benefit. Their experiments show that adding a recalculation step offers only minor benefits in terms of solution quality, but had significant impacts on the overall runtime.

In the form of Listing 10.2, the algorithm will return a list of cells as the set of holes. The algorithm can, however, be applied recursively to create a minimal description length for this list as well. In a first iteration, this would result in a description of the form $S \ominus S_H \oplus H_H$ with S_H and H_H being the results of the recursive call using H as the input set.

In the database community, a related problem is studied under the name of query answering or **query answering using views** [194]. It has been studied in different contexts ranging from query optimization to data integration. The general assumption is that there is a global (database) schema and a set of materialized views over that schema. As the views are materialized, replacing parts of the original query can yield significant performance improvements over using the database relations themselves. This is the main motivation in the context of query optimization.

In data integration, the problem is phrased in a different manner. Again there is a global, mediated schema to represent the domain dealt with. However, the views refer to different, possibly autonomous, datasets. They are described as views of the global schema. Users are able

to pose queries against the global schema and the system will determine which datasets can contribute to the answer. Here, the result is not an execution plan as with query optimization, but a query referring to the individual datasets instead of the global schema.

Both aspects deal with the same problem: Given a set of views, which of these can contribute to answering a given user query? Before discussing possible solutions, the problem itself has to be formalized. A first definition is concerned with the relation of two queries like the input query and the result query [194].

Query containment and equivalence. A query Q_1 is said to be contained in a query Q_2, denoted by $Q_1 \sqsubseteq Q_2$, if for all database instances D, the set of tuples computed for Q_1 is a subset of those computed for Q_2, i.e. $Q_1(D) \subseteq Q_2(D)$. The two queries are said to be equivalent if $Q1 \sqsubseteq Q2$ and $Q2 \sqsubseteq Q1$.

The determination of this relation itself has attracted quite some research over time. It has been studied in different contexts [194] like select-project-join queries and unions thereof, queries with arithmetic comparison predicates, recursive queries, and queries with bag semantics. For conjunctive queries, the decision of whether one query is contained within another is known to be NP-complete [195].

When rewriting a given query using views, two different kinds of results are possible: *Equivalent rewritings* and *contained rewritings*. While the former's result is equivalent to the input query, the latter only represents a subset. A formal definition can be given as follows [194].

Equivalent rewritings. Let Q be a query and $V = V_1, \ldots, V_m$ be a set of view definitions. The query Q' is an equivalent rewriting of Q using V if:

- Q' refers only to the views in V, and
- Q' is equivalent to Q.

Maximally-contained rewritings. Let Q be a query, $V = V_1, \ldots, V_m$ be a set of view definitions, and $\mathscr{L}$ be a query language. The query Q' is a maximally-contained rewriting of Q using V w.r.t. $\mathscr{L}$ if:

- Q' is a query in $\mathscr{L}$ that refers only to the views in V,
- Q' is contained in Q, and
- there is no rewriting $Q_1 \in \mathscr{L}$, such that $Q' \sqsubseteq Q_1 \sqsubseteq Q$ and Q_1 is not equivalent to Q'.

A contained rewriting is defined in the same way as a maximally contained rewriting but omits the third requirement. Approaches to determine query containment only validate a given solution, but do not create one by themselves. Finding such rewritings is a problem in its own right.

Class	Subclass of	Attributes	Disjoint from
Product		Model	Person
Automobile	Product	Model, Year, Category	Stereo
Motorcycle	Automobile	Model, Year	Car
Car	Automobile	Model, Year, Category	Motorcycle
NewCar	Car	Model, Year, Category	UsedCar
UsedCar	Car	Model, Year, Category	NewCar
CarForSale	Car	Model, Year, Category, Price, SellerContact	

Table 10.1: Information Manifold: Class hierarchy (from [196]).

Source 1: Used cars for sale. **Contents:** $V_1(c) \subseteq CarForSale(c),\ UsedCar(c)$
Source 2: Luxury cars for sale. All cars in this database are priced above $20,000. **Contents:** $V_2(c) \subseteq CarForSale(c),\ Price(c,\ p),\ p \geq 20000$
Source 3: Vintage cars for sale (cars manufactured before 1950). **Contents:** $V_3(c) \subseteq CarForSale(c),\ Year(c,\ y),\ y \leq 1950$
Source 4: Motorcycles for sale. **Contents:** $V_4(c) \subseteq Motorcycle(c)$
Source 5: Car reviews database. Contains reviews for cars manufactured after 1990. **Contents:** $V_5(c,\ y,\ r) \subseteq Car(c),\ Model(c,\ m),\ Year(c,\ y),\ ProductReview(m,\ y,\ r)$

Table 10.2: Information Manifold: Information sources (from [196]).

One approach to create query rewritings, the so-called bucket algorithm, is given in [196]. It was implemented in the context of the *Information Manifold System* that provided users with a unified query interface to "more than 100 information sources, many of them on the WWW". The global schema is called world view and consists of several (virtual) classes as well as (virtual) relations which can be of any arity.

Table 10.1 lists examples for classes used in [196]. The classes and their instances are mapped to relations for unified access in the following way: Each class is represented by one unary relation. Properties of the classes are mapped to one binary relation each. Disjointness and inheritance are maintained in the relational view. So if C and D are disjoint classes, for the respective relations $C \cap D = \emptyset$ holds true. Similarly, if C is a subclass of D, then $C \subseteq D$ applies to the relations.

The information sources are described in terms of views over the world view as shown in Table 10.2. Like the world view relations and classes, these descriptions are created manually. No single data source will contain all instances of a certain relation. To highlight this fact the views are defined using a subset relation $\subseteq$ instead of a defining property $\leftarrow$ as used within the queries later on.

Users may pose queries against this global view. An example is given in Listing 10.3. The query asks for a list of models, their prices, and reviews for sportscars manufactured after 1992.

$$q(m_1, p_1, r_1) \leftarrow CarForSale(c_1), Category(c_1, \textbf{sportscar}), Year(c_1, y_1), Price(c_1, p_1), Model(c_1, m_1), ProductReview(m_1, y_1, r_1), y_1 \geq \textbf{1992}$$

Listing 10.3: Information Manifold: Example query (from [196]).

The actual algorithm works in two steps. First, a bucket for each subgoal is created. A subgoal denotes a single requirement in the query. So in the example of Listing 10.3, the first subgoal is given by $CarForSale(c)$. Each bucket will contain a list of views that can contribute tuples to the respective subgoal. In a seconds step, all combinations of views are considered where one view is drawn from each bucket. For each combination, it is then tested whether it is contained within the original query using the algorithm provided in [197]. If this is the case, the respective combination is added to the set of correct solutions. The final result is given by the union of all these solutions.

The algorithm for the creation of buckets is given in Listing 10.4. The input is given as a conjunctive query (cf. Listing 10.3). For each component of that query a bucket is created (line 4). A relation can appear multiple times within such a query and for each occurrence, a separate bucket is created. Afterward, these buckets are populated by comparing the respective component with each component present in the views (lines 6 to 16). If the relation of the component matches one present in the view (line 9), a mapping of variables is created (lines 11 to 13). This mapping also applies to the other components of the view. The view definition with the applied mapping is appended to the original query to form a new query Q' (lines 14 and 15). If this new query is satisfiable, i.e. it does not contain any contradictions or results in an empty result set, the view including that mapping is added to the respective bucket (line 16). A single view can be added to one bucket multiple times, if multiple relations can be matched and, hence, multiple mappings exist.

Consider the example views of Table 10.2 and the query given in Listing 10.3. The first bucket is created for the component $CarForSale(c)$. The respective relation $CarForSale$ is also found in the first component of the first view. Hence, a mapping $c \rightarrow c1$ is created, which results in the query Q' given in Listing 10.5. As $CarForSale$ and $UsedCar$ are not disjoint, the respective query is satisfiable and Source 1 alongside the mapping is added to the first bucket. Source 2 will be processed in a similar way. For Source 3 the resulting query is not satisfiable as $y \leq 1950$ and $y \geq 1996$ will return an empty result. Source 4 is not added as $Motorcycle$ and $CarForSale$ are disjoint classes. Finally, Source 5 is also added as Car and $CarForSale$ can be mapped and the resulting formula is satisfiable. The remaining components of the input query are processed in a similar fashion. The bucket for the final component $ProductReview(m_1, y_1, r_1)$ will only contain a single view and mapping, since only Source 5 can be matched here.

$CarForSale(c_1), Category(c_1, \textbf{sportscar}), Year(c_1, y_1),$
$Price(c_1, p_1), Model(c_1, m_1), ProductReview(m_1, y_1, r_1), y_1 \geq \textbf{1992},$

```
Input: V is a set of content descriptions, and Q is a conjunctive query of the form
       Q: Q(X̄) ← R1(X̄1), ..., Rm(X̄m), CQ:

Set Bucket_i to ∅ for 1 ≤ i ≤ m.
For i = 1, ..., m do
  For each V ∈ V
    Let V be of the form: V(Ȳ) ⊆ S1(Ȳ1), ..., Sn(Ȳn), CV
    For j = 1, ..., n do
      If Ri = Sj or Ri and Sj are non–disjoint classes
        Let ψ be the mapping defined on the variables of V as follows:
          If y is the jth variable in Ȳj and y ∈ Ȳ
            then ψ(y) = xj, where xj is the jth variable in X̄i
            else ψ(y) is a new variable that does not appear in Q or V
        Let Q' be the 0–ary query:
          Q' ← R1(X̄1), ..., Rm(X̄m), CQ, S1(ψ(Ȳ1)), ..., Sn(ψ(Ȳn)), ψ(CV)
        If Satisfiable(Q') then add ψ(V) to Bucket_i

End.
```

Listing 10.4: Information Manifold: CreateBuckets (from [196]).

$CarForSale(c_1)$, $UsedCar(c_1)$

Listing 10.5: Information Manifold: Intermediate query Q' (after [196]).

In the second step, the contents of the individual buckets are combined, such that each candidate solution contains one element of each bucket. The candidate solutions' components are then minimized, i.e. all redundant components are removed. If the resulting candidate is contained in the input query or can be made to be contained by adding further restrictions, it is added to the list of correct solutions. The returned result is then given by the union of all correct solutions. Further steps within the Information Manifold system will create executable plans from these solutions by the means presented in [198], which shall not be discussed here.

The MiniCon algorithm presented in [199, 200] uses the same basic approach but improves the algorithm by reducing the search space even further early in the process. Similar to the bucket algorithm, in the first step mappings between components of the input query and the view descriptions are detected. While the bucket algorithm will insert any matching to the respective bucket, the MiniCon algorithm will apply more checks.

$Q1(x) \leftarrow cites(x, y), cites(y, x), sameTopic(x, y)$
$V4(a) \leftarrow cites(a, b), cites(b, a)$
$V5(c, d) \leftarrow sameTopic(c, d)$
$V6(f, h) \leftarrow cites(f, g), cites(g, h), sameTopic(f, g)$

Listing 10.6: MiniCon: Example setup (from [200]).

Consider the example of Listing 10.6 where $Q1$ is the input query and $V4$, $V5$, and $V6$ are the data source views. The first occurrence of $cites$ in both $Q1$ and $V4$ can be matched with a mapping like $x \rightarrow a$, $y \rightarrow b$. The bucket algorithm will stop here and add $V4$ to the bucket for $cites(x, y)$. MiniCon, however, considers further implications: As b is existential, i.e. it does not appear in the head of the view, and there is a mapping $y \rightarrow b$, all other occurrences of y also have to be present in $V4$. Informally speaking, y is needed in the join condition for all relations containing it. So these joins either have to be performed within the respective view definitions or the variable y has to be exposed in the head to be available for further joins outside the view. In the example, y is mapped to the variable b. The head of the view definition for $V4$ does not contain b, so all other relations of y in the input query also have to be present in $V4$, also respecting the mapping created so far. The second $cites(y, x)$ can be mapped to the second occurrence of $cites(b, a)$ in $V4$. However, the last relation $sameTopic(x, y)$ has no equivalent in $V4$, so MiniCon will not include $V4$ in the solution. As a consequence, MiniCon maintains sets of subgoals alongside the mapping in contrast to the bucket algorithm, which only considers isolated subgoals. Within the MiniCon algorithm, sets of subgoals alongside the found mapping are maintained within so-called MiniCon-Descriptions (MCD).

In the second step, MCDs found in step one are combined to form the final solution. Here, only MCD with non-overlapping sets of subgoals is considered. A proof that this still results in accurate solutions is given in [200]. This proof also justifies the omission of the query containment check present in the bucket algorithm. Both of these aspects, reducing the search space in step one and omission of query containment checks in step two, lead to substantial performance gains of MiniCon as compared to the bucket algorithm [199, 200].

The complexity of query answering using views has been studied in [201, 202]. The problem has been examined under both the Open World Assumption (OWA) as well as the Closed World Assumption (CWA). Under CWA view instances include all the tuples that satisfy the view definition. OWA, on the other hand, allows for incomplete instances that might only store some of those tuples. One of the results stated is, that even under the OWA queries using inequalities are co-NP hard. Restricting both query and views to conjunctive queries, on the other hand, results in a PTIME complexity.

10.2 Discussion

All approaches deal with the same situation: A dataset that is the combination of multiple parts. While MDL descriptions are concerned with describing each part in relation to the whole dataset, the others are concerned with finding suitable pieces to form the dataset.

MDL summarization could be employed as part of a dataset combination as follows. The user query for a dataset can be interpreted as a virtual OLAP cube. The next step would now find the source dataset that covers most cells of this cube. For the MDL summarization, the cells not

covered by the source datasets are now marked as blue cells. This results in a new query that covers the remaining area, which can in turn be fed to another recursive call of the algorithm. This process is repeated until no area remains or no matching source dataset can be found.

While this resembles the general structure of the approach presented later on, the use of MDL summarization in this scenario is rather limited. The main advantage of summarization algorithms is the generation of short query result descriptions targeted towards human users. Within reasonable boundaries algorithms, however, are agnostic of this. Finding the source dataset to match the current query might even reverse this process to estimate the coverage of a certain source. Assume, e.g., the summarization detects that data for Africa is missing from the current result, the candidate source datasets, however, contain data on the country level. Here, the second step of the algorithm would need to reverse the summarization in order to rank those datasets.

Query answering using views addresses a similar problem to the one posed. It defines a global mediated schema and describes all data sources as views over that schema. User queries are also posed as queries over the global schema and are then rewritten by algorithms in terms of the data sources' views.

This global schema, however, also represents a potential weak point. A fixed schema might need frequent adjustments when new datasets shall be added to the pool. The authors of [196] state that their process of describing datasets was "leaving the world-view relatively stable". This highlights, however, that even for small numbers – -only 100 datasets were used in their experiments — such adjustments were necessary.

Another disadvantage lies in the description of the user queries. Users have to be familiar with the global schema to pose valid queries. This especially includes the need to include the join conditions needed to combine multiple relations. If the domains grow beyond a certain threshold, this will impede the accessibility for users that are no experts in the respective area. Similarly, queries spanning multiple overlapping domains will pose problems if these domains have been modeled separately. In this case, most certainly redundant relations will have to be mapped to one another and possibly conflicting models need to be reconciled.

A further problem of this query answering is hidden in the solution being defined as a union of multiple queries. Under both bag semantics [203, 204] and set semantics [130], the result might contain tuples users may regard as duplicates. The general query answering approach does not distinguish between dimensions and measurements. As a result, conflicting measurements might not be resolved. Instead of resulting in the equivalent of one coherent OLAP cube, the result might contain multiple tuples referring to the same cell. The system itself is not capable of resolving these issues but needs further information or user interaction.

Finally, the MSJ engine and "On-the-fly Table Generation" come closest to solving the posed problem. In MSJ, users provide lists containing objects of interest and possibly concepts they wish the table to be extended with. The system will search through a web corpus and try to add the

respective attributes and present the respective result table. In contrast to the query answering algorithms, it is not reliant on a predefined global schema. Similarly, the approach of "On-the-fly Table Generation" attempts to extract a list of objects of interest from a keyword-based query first. Subsequently, the corresponding candidates are used to generate a table answering the query.

Both approaches are focused on single identifying labels. While this might be suitable to retrieve information about single objects, it becomes problematic if the required data is multidimensional. Assume a query for the annual budget of certain countries. A user might pose a query with a list of countries and requesting columns relating to year and budget. The MSJ engine will acquire all relevant datasets and create a combined table. However, after the consolidation phase only two values per row will be present. The proposed algorithm at this point does not even retain the connection between those two cells. As users might not be aware of such behavior, they might keep on working using this false data as to them it appears valid.

The consolidation phase of MSJ and the overall approach of The focus on individual cells in "On-the-fly Table Generation" suffers from might yield another possible issue. Values are retrieved individually for each returned row and cell, losing the connection of values from the same dataset. So while the value for one cell might be taken from dataset A, another cell in the same tuple might be filled with a value from dataset B. This may introduce incoherences in the result. In case of "On-the-fly Table Generation", the situation gets even worse. As values are fetched independently for each cell, values for year and budget of a country might be fetched from completely unrelated sources (i.e., either tables from the corpus or the knowledge graph). Again, no hint would be given to a user that the provided data can not be relied upon.

The quality of the result might be increased by choosing values from preferred datasets and only resorting to others if the preferred one can not provide a value for a given cell. That is, of course, under the assumption that there are no or at least fewer incoherences within the source datasets.

10.3 Approach

Before discussing an actual approach, the format of the input has to be specified. The expected result is a new dataset, so it is only natural to reuse the dataset description of Chapter 8 at this point. A user query may hence be given as a collection of at least two columns. Two columns are needed so the final dataset will consist of at least one dimension and one measurement. A mere list of individuals is considered out of scope for this approach.

Each column can consist of a concept and a value range. Omitting a value range labels a particular column as *infinite*, i.e. there are no restrictions placed on the values that might be included. In contrast to that, columns that include a range of values will be called *finite*. For quantitative or time-based[4] columns there might also be *semi-finite* specifications that only state an upper or lower bound for the range of values.

The concept definition for a column is optional if that column is of a categorical type. Here, the set of specified values implicitly defines the column's concept. The respective user interface has to ensure that all values of a dataset can be attributed to the same concept. This way of implicitly defining a column's concept can not be extended to quantitative columns. A given value range can be attributed to basically any concept. Even the addition of a unit can only narrow down the list of concepts, but can not uniquely identify a single one.

The general approach is given in Listing 10.7. It extends the idea stated towards the end of the previous discussion to a general principle for a data combination approach: Satisfy the user query as much as possible with a single dataset, then apply the same principle to the parts not covered yet. Adhering to that principle yields two kinds of benefits. On the one hand, the risk of incoherences introduced by using multiple sources is reduced[5]. On the other hand, using fewer data sources will also increase the performance of such an approach, as fewer sources have to be accessed and fewer resources have to be spent on the resolution of possible conflicts.

```
INPUT: Target dataset description as list of column definitions
       (concept and/or range of values)
OUTPUT: Workflow to create said dataset as closely as possible

FUNCTION createWorkflow( user query )

  LET descriptions = List of matching dataset descriptions from the repository

  LET bestDs = Dataset description best fitting to the user query

  LET regions = Remainders after applying bestDs to the user query

  LET subWorkflows = FOREACH region OF regions: CALL createWorkflow( region )

  LET result = Combine subWorkflows and bestDs using JOIN and UNION operators

  RETURN result

END
```

Listing 10.7: Pseudocode for dataset combination approach.

[4]The definition of "time-based" is in line with the restrictions stated in Section 5.3. This denotes a certain point in time and not the measured length of some event.

[5]This labels the given approach as conflict-avoiding according to the classification of [205].

The individual steps in the approach will be discussed in detail in the following sections. At this point, just a brief overview is given. The first step (line 7) is to gather a list of dataset descriptions that might satisfy at least a part of the user query. This step will reduce the search space by a large margin as most dataset descriptions will not be relevant for the given query. The next step (line 9) will rank all remaining datasets according to their usability for the current user query and select the highest-scoring one. This follows the aforementioned principle by satisfying the user query as far as possible by using just a single source. Now, the user query is split with respect to the selected dataset into regions (line 11). Subsequently, for each of these regions the algorithm is called recursively (line 13). In a way this resembles the approach of recursive MDLH-descriptions, but, instead of specifying holes in the solution, those holes are used as new user queries to the recursive calls. The final step (line 15) combines the results of the recursive calls with the selected dataset for the current user query to create the overall solution.

10.3.1 Searching dataset descriptions

This step assumes that there is a repository that contains the descriptions for datasets in the format outlined in Chapter 8. Besides information to access a particular dataset, this especially includes the respective column definitions. As the user query is given in a similar format, both descriptions can easily be matched.

Some categorical columns in the user query might have no concept assigned to them but are merely defined by a collection of values. In a preprocessing step, the concepts for these columns are determined by retrieving a common concept that includes all given values as possible instances. After this preprocessing, each column is described by a concept. Some columns might also be further restricted by a given value range.

A description has to meet the following criteria to be considered a candidate for the remainder of the algorithm. The concepts of at least two columns in the description have to match to counterparts within the user query. If the user query poses restrictions on the value ranges for the matched columns, columns in the description need to specify a non-empty intersection with those ranges. Furthermore, within the set of matched columns, there has to be at least one dimension and one measurement.

The rationale for the first requirement is rather obvious. Without any overlap in terms of columns, the described datasets will not contribute to the answering of the user query and, hence, can be removed from the search space. The second requirement is caused by applying the same reasoning to the value ranges: The dataset will not be part of the solution if none of the matched columns' value ranges overlaps with the restrictions posed by the user query.

The remaining requirement calls for at least two matching columns, in particular at least one dimension and one measurement. On the one hand, this matches the restriction on user queries which prohibits requests for simple lists of values. On the other hand, this removes datasets that only match dimensions or only match measurements. Both of them are of little use here. The

datasets will be transformed in a later step of the algorithm to remove columns that are not part of the query, so only matched columns need to be considered at this point. Values in columns are connected if there is a measurement that is described by one or multiple dimensions. As a consequence, datasets that consist of only dimensions or only measurements can only contribute unconnected values to the result. The query, however, asks for connected values of some sort, so these datasets will not provide any meaningful benefit to answering that query.

The above discussion can be applied to all levels of recursion in the algorithm. However, all but the first level leave room for optimization. All recursive calls will concern subsets of the original user query. So each dataset that fits one of these subqueries has also to be applicable to the parent query. Instead of retrieving dataset descriptions again and again from the repository, the system can reuse the candidate list from the parent call and apply the restrictions of the subquery on this list. Depending on the size of the repository, the number of candidates, and the recursion level of the current call, this can yield substantial performance benefits as the size of the search space will be strictly monotonic decreasing with each new level of recursion[6].

10.3.2 Ranking Datasets

The ranking of datasets follows the same general notion like OLAP cubes and MDL-descriptions before (cf. Figures 10.3 to 10.5). Both, user query and candidate datasets, are seen as multidimensional OLAP cubes. The ranking is determined by the overlap between the cubes of a dataset and the cube defined by the user query: Candidates that cover a larger share of the user query cube will be ranked above those of lesser coverage.

As outlined during the discussion of MDL-approaches, a precise description of such cubes can get arbitrarily complex. The dataset descriptions described in Chapter 8 already introduced a simplification to avoid this complexity: Datasets are assumed to have no holes in their coverage. If the description of a dataset claims a certain coverage for a specific dimension, each value of that dimension has a corresponding measurement. The same notion is extended to combinations of dimensions, so there are no cells lacking a measurement in the corresponding OLAP-cube. While this assumption will likely only hold up for very few real-world datasets, affected datasets could be split into multiple (virtual) ones following the ideas of MDLH. Anyhow, its effects will only be noticeable once there is a substantial share of missing measurements. So for the purpose of this thesis, potential missing entries in a dataset will be disregarded.

The coverage of any particular dataset with respect to the given user query can be computed as a product of the coverage for each mapped dimension individually. The mapping of columns has already been determined during the previous process of finding candidate datasets by matching the columns' concepts to one another. So, for a given source dataset s containing dimensions

[6] At least one dataset is picked to satisfy parts of the parent query. All remainder regions have no intersection with this dataset, so the number of descriptions within the dataset will decrease by at least one. Further, no new dataset descriptions can be added as there can be no dataset meeting the requirements of a subquery but failing to do so for the parent query.

c_i^s mapped to their counterparts c_i^q in a user query q, the coverage can be calculated using Equation 10.4. Dimensions that could not be mapped are omitted from this calculation but will be accounted for in another metric later on.

$$\text{Coverage}(\ s,\ q\ ) = \prod_i \text{Coverage}(\ c_i^s,\ c_i^q\ ) \tag{10.4}$$

To calculate the coverage of a particular dimension, first, the intersection of values in the dataset and query has to be computed. The dataset's dimension is always finite, so this intersection is finite as well. However, it might be empty if there is no overlap between both columns. The size of this intersection is then scaled according to the size of the query dimension if it is finite. If the query dimension is infinite or semi-finite, a similar scaling can be achieved by using the intersection's reciprocal size. The size of the intersection is calculated differently for categorical columns on the one side and quantitative and time columns on the other side. For categorical columns, it is given by the number of distinct values, whereas the other two types use the difference between the maximum and minimum value. If *Intersection()* denotes the intersection of two columns and *Size()* its size, Equation 10.5 calculates the coverage for an individual column.

(10.5)

$$\text{Coverage}(\ c_i^s,\ c_i^q\ ) = \begin{cases} 0 & \text{if Intersection}(\ c_i^s,\ c_i^q\ ) = \emptyset \\ \dfrac{\text{Size}(\ \text{Intersection}(\ c_i^s,\ c_i^q\ )\ )}{\text{Size}(\ c_i^q\ )} & \text{if Intersection}(\ c_i^s,\ c_i^q\ ) \neq \emptyset \text{ and } c_i^q \text{ is finite} \\ 1 - \dfrac{1}{\text{Size}(\ \text{Intersection}(\ c_i^s,\ c_i^q\ )\ )} & \text{otherwise} \end{cases}$$

The discussions so far assume that source datasets and query share the same dimensions, which does not hold in all cases. Source datasets may only cover a subset of the query's columns or include other columns the query does not request[7]. To address these cases in the ranking, two more metrics are defined. Together they describe the deviation from meeting the schema of the query exactly.

Support describes the fraction of columns in a query q that can be satisfied by a source dataset s.

$$\text{Support}(\ s,\ q\ ) = \frac{\#(\text{ shared columns between } s \text{ and } q\ )}{\#(\text{ columns in } q\ )} \tag{10.6}$$

Excess is the fraction of dimensions in a source dataset s that are not requested by a query q.

$$\text{Excess}(\ s,\ q\ ) = \frac{\#(\text{ dimensions present in } s \text{ but not in } q\ )}{\#(\text{ dimensions in } s\ )} \tag{10.7}$$

[7] If these datasets are chosen to be part of the result, additional actions have to be taken to adjust them to the target schema. The respective efforts will be discussed later.

Excess only measures the number of superfluous dimensions in the dataset but ignores superfluous measurements. The rationale behind this is, that additional measurements might just be dropped from the dataset without affecting the usable result. Dimensions, on the other hand, need to be eliminated by the use of aggregation operations. These operations require additional effort and, hence, the respective datasets are penalized here.

Combining the three described aspects yields a final ranking for each pair of dataset and query. Following the goal to use as few datasets as possible, high values for both coverage and support will yield better results. While both seem to capture similar properties, they differ in their reference parameters. Coverage measures the overlap on the cell level, whereas support measures it on a schema level. Coverage is directly derived from the notion of an OLAP-cube that is to be filled with values. The rationale behind the support metric is to favor those datasets that are closer to the overall intention of the query or even share it. On the other hand, the datasets picked should contain only few to none superfluous columns, so the excess rating should be low.

The final ranking is now given by a vector of the metrics discussed before as shown in Equation 10.8. The excess metric is inverted, such that higher values signal better suitability. This is only done to simplify later comparisons and has no further effect. To derive the best candidate, the ranking vector is computed for each dataset individually. Those with a zero value in the first element of the vector are discarded as they do not match the current query. All others are sorted by using the elements of their vectors successively: Initially, they are ordered by comparison of the first components. If this results in ties, the second elements are used. This results in a ranking of datasets, from which the best candidate can now be picked.

$$\text{Score}(\, s,\, q\,) = \begin{pmatrix} \text{Coverage}(\, s,\, q\,) \times \text{Support}(\, s,\, q\,) \\ 1 - \text{Excess}(\, s,\, q\,) \end{pmatrix} \tag{10.8}$$

10.3.3 Splitting Queries

Determining the remainder after applying a certain dataset has to consider two scenarios. In the first scenario, the dataset and the user query use equivalent schemata. Here, the areas already covered by the dataset can be removed and the remaining regions are identified. The dataset might also cover the whole user query, thus resulting in an empty remaining region. In another scenario, the dataset differs from the query's schema. An example for this scenario is oftentimes queries containing multiple measurements. As each measurement might be provided by a different data source, the system will have to cope with several deviating schemata in the process. In the following, the scenario of equivalent schemata will be discussed first, before the approach is extended to the more general situation of deviating schemata.

The splitting process for the first scenario can be visualized as shown in the example of Figure 10.6. The user query is represented by a two-dimensional OLAP cube (cf. Figure 10.6(a)). Similarly, datasets are also given as two-dimensional subcubes. The distinction between dimen-

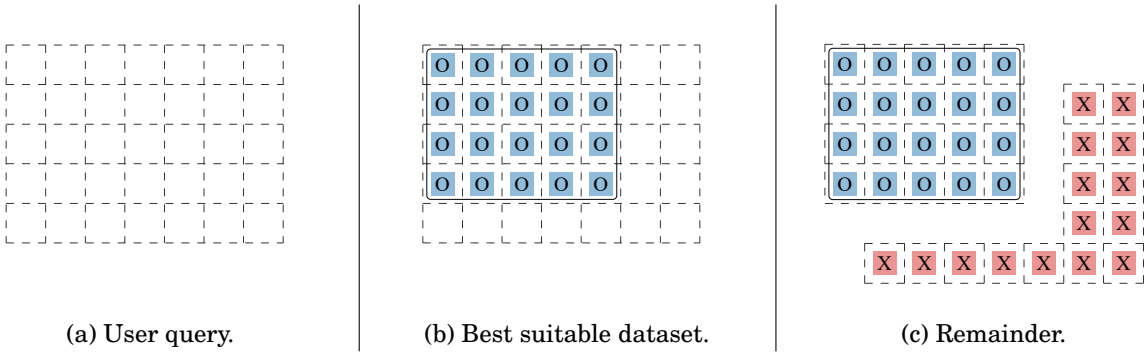


Figure 10.6: Division of a user query by a source dataset.

sions and measurements is not given by the user query in the first place, but by the definition of the dataset. However, there might be restrictions imposed on some columns as a result of splitting a query by a dataset with deviating schema. This will be discussed later in the context of the second scenario. The best-fit dataset as determined by the ranking process will cover a subcube within the query's cube (cf. Figure 10.6(b)). The goal of the splitting process is to identify the remaining region (cf. Figure 10.6(c)). The algorithm itself is then run again on the remainder to fill in the gaps in the user query. It will terminate if either the user query has been satisfied entirely or for all remainders, no overlapping dataset can be found.

To form a rectangular OLAP cube, the remainder itself has to be cut again. Possible strategies are illustrated in Figure 10.7. The remainder can either be split into overlapping subcubes (cf. Figure 10.7(a)) or into disjoint ones (cf. Figures 10.7(b) and 10.7(c)). The advantage of a split into overlapping subcubes is, that the number of subcubes only grows linear in the number of dimensions of the user query. However, this comes at the cost of possibly requiring conflict resolution techniques when putting the pieces back together.

On the other hand, both disjoint strategies do not require such a reconciliation as they create no overlaps. In the asymmetric strategy (cf. Figure 10.7(b)) the dimensions are ordered by some kind of preference. Dimensions of a higher preference are less likely to be split along than those of a lower preference. In the example of Figure 10.7(b) the vertical dimension precedes the horizontal one and thus is not split here. In the symmetric strategy (cf. Figure 10.7(c)) there is no such ordering of dimensions and all of them are treated equally. This, however, results in more subcubes being created, so while the two other strategies grow just linear with regard to the dimensions, the symmetric strategy grows by $O(\ 3^d - 1\ )$ in each level of the recursion, where d is the number of dimensions.

While this exponential growth in complexity seems excessive at first, it is important to note that the other strategies may deteriorate similarly if not worse. Consider the example of Figure 10.8, which continues the splitting process of Figure 10.6. Figure 10.8(a) shows the

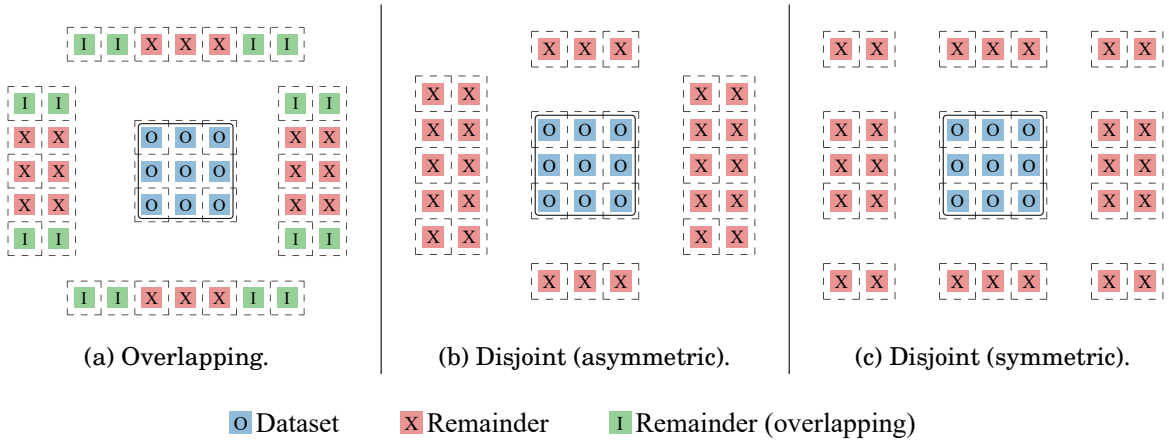


Figure 10.7: Strategies when splitting a user query.

process for splitting into overlapping subcubes. While only two subcubes are created in the initial splitting, the next recursion level has to create two more subcubes for each call. Hence, the number of subcubes that have to be checked is higher compared to the disjoint, symmetric splitting strategy (cf. Figure 10.8(b)). Furthermore, the additional recursion step includes another iteration of actually finding the best match and splitting the query accordingly. As a consequence, the disjoint symmetric splitting strategy will outperform the overlapping strategy here, as it saves this intermediate step. The argument can be extended in a similar fashion for the disjoint, asymmetric strategy. While the number of subcubes to check is equal to the symmetric strategy, it involves another recursion level as well. As a consequence, it is similarly outperformed by the disjoint, symmetric approach.

Figure 10.8(a) also highlights another potential drawback in the overlapping splitting strategy. Under certain circumstances, identical subcubes are processed multiple times. This issue can be tackled by use of a lookup to memorize already processed subcubes, but this comes at the cost of additional computational resources within the actual lookups. Furthermore, the subcubes do not necessarily have to be equivalent as shown in the example, but can also manifest in other “subcube”-relations, where a superset of the currently considered subcube has already been processed. This further increases the computational cost of said lookup and thus decreases this strategy’s performance.

The second scenario involves deviating schemata between user query and chosen dataset. In particular, the dataset schema will be a subset of the query’s schema. The reverse situation, where the query represents a subset of the dataset, will be discussed later in Subsection 10.3.4. For the purpose of splitting the query, only the matched columns of the dataset are relevant. If the query schema is a subset of the dataset’s, this results in an instance of the first scenario presented before and no additional efforts are needed.

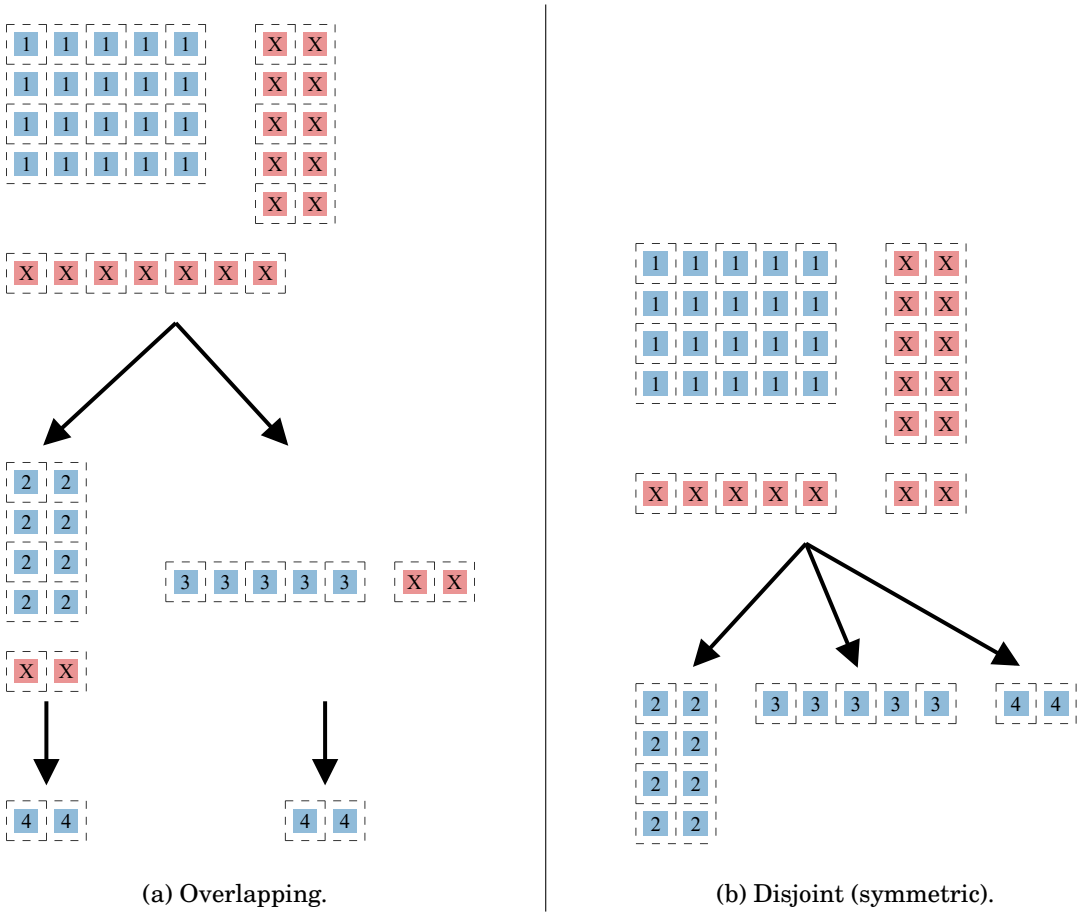


Figure 10.8: Worst case deterioration of splitting strategies.

If the dataset covers only a subset of the columns, the metaphor of an OLAP-cube used so far falls short. While the remainder-subcubes can be treated as before, the dataset itself does not cover the entirety of its respective subcube. Instead, it satisfies only a lower-dimensional subset of it, while leaving columns related to the higher dimensions of the subcube valueless. So in order to contribute a complete subcube, these remaining columns have to be filled as well.

In addition to the previous remainder-subqueries, another subquery for the subcube pertinent to the found dataset is needed. In this subquery, all columns already satisfied by the current dataset can not be matched to measurement columns in other datasets. As a consequence, they are labeled dimension for the sake of the next recursion call. This causes datasets found in later recursion steps to fill up the remaining columns, but they can not contradict values in the already satisfied ones. No recursive calls are necessary if all columns are labeled dimensions. As per requirement before, each candidate dataset has to feature at least one matching measurement, so no candidate dataset can satisfy a query with only dimensions remaining.

10.3.4 Assembling Workflows

In the final step of the algorithm given in Listing 10.7, the results of the individual recursive calls have to be merged together into an executable workflow. The workflow itself will follow the structure imposed by the recursive approach and can, hence, be represented in a tree-like fashion. Leaves in that tree represent source datasets to be loaded, whereas internal nodes stand for operations combining them. Finally, the result for the query is given in the root node.

Subsection 10.3.3 introduced different scenarios that might occur when combining subresults. One scenario was left open, as it did not affect the necessary splitting of queries during the recursive calls: candidate datasets that include additional columns beyond what is requested in the respective query. Up to now, those were only considered regarding the columns that match the queries' schema, disregarding the additional columns. However, before using such datasets within the result, their schema has to be adjusted to remove those additional columns. Per definition, additional measurements do not affect other columns. So measurement columns that could not be matched to the query can just be dropped while loading the dataset.

Additional dimensions, on the other side, can not be dealt with in the same way. Simply removing them would remove a crucial piece of information from the dataset and, hence, alter its content. To remove such columns, the dataset has to be aggregated along the remaining dimensions that were mapped to the user query. This aggregation raises the issue, which aggregation function to use for the remaining measurements. In general, the system will not be able to decide on a suitable aggregation function, as this relies on the particular user intention which is unknown to the system.

As an example, consider sales data over some time for a certain number of branches of a company. Dropping the time dimension can have two reasons: On the one hand, users may want to have the overall sales numbers for each branch, which would result in a summation aggregation function. On the other hand, the goal might be to get an average number of sales. In this case, the aggregation function to chose would be averaging. This example highlights the need for user interaction in the dataset combination process. Hence, the workflow will include nodes that prompt for user input on these occasions.

Within these interactions, users are presented with several options. The choice of the aggregation function depends on the type of the column in question, but will include the standard aggregation functions like `sum`, `average`, or `max`. Besides choosing an aggregation function for the measurement columns, users also have two other options. The first one offers to drop the superfluous dimension without any aggregation done. While this is most often not a suitable solution, some datasets might store redundant information in different dimension columns. In these cases, users may choose to remove that redundancy and just drop that column.

Another option is to include said dimension in the result. By selecting this option users will discard the current query results, and issue a new query that includes the column(s) in question. This is necessary as the addition of a new column to the query will invalidate any ranking made

before and the result will, hence, not represent the optimal solution with respect to the stated criteria. If users choose to stick to the current query and select any of the other options, a new node will be added as the parent of the leaf-loader node for the respective dataset. That node will contain all processing instructions necessary to harmonize the dataset and query schemata.

After the schema is adjusted, the values within the dataset are filtered with respect to the requirements of the current query. This may introduce a second filter node as the parent of either the dataset-loader node directly, if no schema adjustments are required or of the previously created filter node that adjusts the schema. This step may also be executed before adjusting the dataset schema. If aggregations are necessary, this reduces the amount of data needing to be processed.

After datasets are brought in line with the respective query, missing columns have to be replaced. If necessary, the previous splitting process created one subquery to bridge the gap between dataset and query schema. So in this step, both these results will be joined into an intermediate result conforming to the queries schema. Within the formalism of relational algebra this represents a `LEFT OUTER JOIN` of the dataset found in the current recursion (left table) with the results found for the remainder-query (right table). The result for the subquery may be empty if no matching dataset had been found in the recursive call. In this case, all missing columns are filled using null values. This way it is guaranteed that the schema of intermediate result and query are identical.

On all other subresults, the splitting process placed no additional restrictions. Hence, by the recursion invariant, they already adhere to the user query's schema and can be added to the intermediate result without the need for any adaptations. The respective pendant in relational algebra for this step is the union-operator.

As already mentioned, the result of one recursion step is a tree-like structure. Datasets form the leaf nodes in this tree. If their schema differs from the query, there might be an additional filter node as their direct parent to remove superfluous columns. Similarly, an additional parent node might be added to request user intervention in determining a suitable aggregation function, if this is required. The remaining inner nodes represent either join (for deviating schemata) or union (for conforming schemata) operations on some subresults. The root node itself will also be an instance of one of these node types. Furthermore, it serves as the entry point for executing the resulting workflow, which executes all operations in this tree in a post-order walk.

10.3.5 Optimizations

The presented workflow generation approach does not include any optimizations. However, there is some room for improvement here. First of all, the order of join-operations can be changed to reduce the overall computational effort. This is studied in the context of relational databases for quite some time now [206] and is, hence, omitted here.

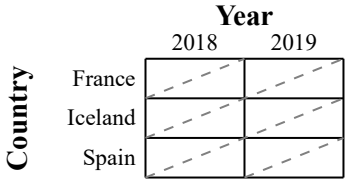
Another opportunity to reduce the complexity of the workflow is a direct result of the splitting strategies presented in Subsection 10.3.3. The query-remainders in any particular step might be split in such a way that a single source dataset may satisfy multiple subqueries. In the workflow as given so far, these datasets would be loaded multiple times and filtered multiple times before being added to the result. This situation may be remedied by an additional step before applying the union-operator. If the root of several subresults are filter-nodes working on the same dataset, these filters can be combined into a single node. The filter conditions are connected using an or-operator and may subsequently be simplified. The subresults relating to this dataset are removed and replaced by a single subresult using the combined filter. This removes multiple different passes over the respective dataset. A single dataset might still occur on different subtrees and, hence, not get optimized. In such cases all filter-nodes can point towards the same loader-node for that dataset, at least preventing the dataset to be loaded multiple times. However, this does not prevent multiple passes over the same dataset caused by the filter-nodes.

10.3.6 Example

The goal of this section is to illustrate the presented approach based on the example query outlined in Figure 10.9. The requested dataset contains four columns: *country*, *time*, *population*, and *sheep*. It further constraints the values for two of the columns. Data is requested only for France, Iceland, and Spain as well as the years 2018 to 2019. All categorical values as well as the column headers have been resolved to concepts in a prior step.

- **Country**
  France, Iceland, Spain
- **Year**
  2018 - 2019
- **Population**
- **Sheep**

(a) Textual representation.



(b) OLAP-cube representation. Cells diagonally split between *population* and *sheep* measurements.

Figure 10.9: Example: Initial query.

In a first step, the system has to look for proper candidates that cover at least one dimension and at least one measurement of the query (cf. Subsection 10.3.1). Further, candidate datasets are required to have some overlap with the values requested by the query. Given the previous example, this could identify the four datasets, Ⓘ to Ⓘⅴ, shown in Figure 10.10 as possible candidates.

Ⅰ

Country	Year	Pop.
France	2019	67,177,636
France	2018	67,026,224
Iceland	2019	356,991
Iceland	2018	348,450
Spain	2019	46,937,060
Spain	2018	46,658,447

Ⅱ

Country	Year	Sheep
France	2019	7,105,000
France	2018	7,166,000
Iceland	2019	416,000
Iceland	2018	432,000

Ⅲ

Country	Year	Color	Sheep
Spain	2019	black	3,095,724
Spain	2018	black	3,170,506
Spain	2019	white	12,382,896
Spain	2018	white	12,682,024

Ⅳ

Country	Year	Pop.
France	2019	67,177,636
Iceland	2019	356,991
Spain	2019	46,937,060
Turkey	2019	82,003,882

Figure 10.10: Example: Source datasets (partially fictitious, data after [data3, data16]).

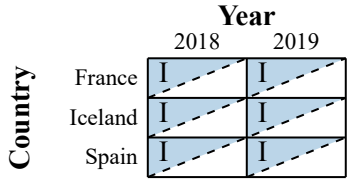
Next, these candidates have to be ranked according to their overlap with the initial query (cf. Subsection 10.3.2). The results for the example query and the identified candidate datasets are given in Table 10.3. The first dimension, *country*, is categorical, so the coverage is calculated by comparing the number of distinct, matching values in datasets with those requested in the query. For the second dimension, *year*, the coverage is calculated using the difference between the minimum and maximum value. Here, the coverage of dataset Ⅳ highlights a particular aspect of time-typed columns: A time-value actually represents a period of time given by its respective precision (cf. Chapter 5). For example, the value `2019` represents the period between the first and last moment of that year[8] and as such covers all 365 days in between. Consequently, the coverage of the column *year*, in this case, is $\frac{1}{2}$ and not 0. The coverage from both dimension columns subsequently yields the total coverage for all datasets of the example. The support for all datasets is equal, as all of them cover the two dimensions, *country* and *year*, and one of the two measurements. Dataset Ⅲ has an additional dimension, *color*, which results in an excess of $\frac{1}{4}$ contrary to all other datasets whose value is 0, as they do not contain any such additional dimension. Finally, the score vector is computed according to Equation 10.8.

[8]The first and last moments are given by the start of Jan. 1st 2019 and the end of Dec. 31st 2019 respectively.

	Coverage per Dimension					
	Country	Year	Coverage	Support	Excess	Score
Ⓘ	$\frac{3}{3}$	$\frac{2}{2}$	$\frac{3}{3} \times \frac{2}{2} = 1$	$\frac{3}{4}$	$\frac{0}{3}$	$\begin{pmatrix} 1 \times \frac{3}{4} \\ 1 - \frac{0}{3} \end{pmatrix} = \begin{pmatrix} \frac{3}{4} \\ 1 \end{pmatrix}$
Ⓘ	$\frac{2}{3}$	$\frac{2}{2}$	$\frac{2}{3} \times \frac{2}{2} = \frac{2}{3}$	$\frac{3}{4}$	$\frac{0}{3}$	$\begin{pmatrix} \frac{2}{3} \times \frac{3}{4} \\ 1 - \frac{0}{3} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix}$
Ⓘ	$\frac{1}{3}$	$\frac{2}{2}$	$\frac{1}{3} \times \frac{2}{2} = \frac{1}{3}$	$\frac{3}{4}$	$\frac{1}{4}$	$\begin{pmatrix} \frac{1}{3} \times \frac{3}{4} \\ 1 - \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ \frac{3}{4} \end{pmatrix}$
Ⓘ	$\frac{3}{3}$	$\frac{1}{2}$	$\frac{3}{3} \times \frac{1}{2} = \frac{1}{2}$	$\frac{3}{4}$	$\frac{0}{3}$	$\begin{pmatrix} \frac{1}{2} \times \frac{3}{4} \\ 1 - \frac{0}{3} \end{pmatrix} = \begin{pmatrix} \frac{3}{8} \\ 1 \end{pmatrix}$

Table 10.3: Example: Coverage and scores for initial query.

After rating each candidate dataset, the top-scoring one, here dataset Ⓘ, is selected as an initial data source[9]. Using that dataset the measurement *population* can be provided for all dimensions as illustrated in Figure 10.11(a). However, as the query is not yet covered in its entirety, the query has to be split with respect to the already covered portions. As noted before, there is no need to keep looking for *population* measurements and, hence, this is dropped from the query as outlined in Figure 10.11(b). The input to the next iteration of the algorithm is now given by this query and the remaining candidates, i.e. the ones listed in Figure 10.10 minus dataset Ⓘ.



(a) Current state of the OLAP-cube. Dataset Ⓘ providing all requested *population* measurements.

- **Country**
  France, Iceland, Spain
- **Year**
  2018 - 2019
- ~~**Population**~~
- **Sheep**

(b) Remaining query. Measurement *population* removed.

Figure 10.11: Example: State of processing after first iteration.

The search step in the second iteration basically degrades to a filter operation on the remaining datasets. In the example, dataset Ⓘⅴ will be removed, as it does not contain both a dimension and a measurement of the remaining query. This leaves Ⓘⅱ and Ⓘⅲ as viable candidates. Again, for both datasets, a score is computed. However, this time according to the new query of Figure 10.11(b). The results of this second iteration of scores are given in Table 10.4.

[9]The ranking is done according to the first element of the score vector. The second element is only used to break ties.

	Coverage per Dimension					
	Country	Year	Coverage	Support	Excess	Score
Ⅱ	$\frac{2}{3}$	$\frac{2}{2}$	$\frac{2}{3} \times \frac{2}{2} = \frac{2}{3}$	$\frac{3}{3}$	$\frac{0}{3}$	$\begin{pmatrix} \frac{2}{3} \times \frac{3}{3} \\ 1 - \frac{0}{3} \end{pmatrix} = \begin{pmatrix} \frac{2}{3} \\ 1 \end{pmatrix}$
Ⅲ	$\frac{1}{3}$	$\frac{2}{2}$	$\frac{1}{3} \times \frac{2}{2} = \frac{1}{3}$	$\frac{3}{3}$	$\frac{1}{4}$	$\begin{pmatrix} \frac{1}{3} \times \frac{3}{3} \\ 1 - \frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{1}{3} \\ \frac{3}{4} \end{pmatrix}$

Table 10.4: Example: Coverage and scores for second iteration's query.

The choice in the second iteration is dataset Ⅱ covering a larger share of the remaining query. This dataset is able to cover two-thirds of the remaining dimensions but provides no data for *Spain* in the requested years. Figure 10.12(a) shows the updated response after this second round. As still not all values for the *sheep* measurement could be provided, the query is split again reducing it to the form of Figure 10.12(b).

(a) Current state of the OLAP-cube. Dataset Ⅱ providing some *sheep* measurements.

(b) Remaining query. *France* and *Iceland* removed from the *country* dimension.

Figure 10.12: Example: State of processing after second iteration.

For the third iteration only one dataset, Ⅲ, is left. It covers part of the remaining query and is thus chosen to contribute to the final result. At this point, the list of candidate datasets is exhausted and the initial query has been covered in its entirety, so the recursion in the algorithm ends. The final contributions of each source datasets are outlined in Figure 10.13.

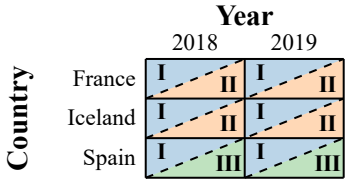


Figure 10.13: Example: Final composition of result.

In a final step the selected candidate datasets have to be assembled into a single workflow. The schema of the last selected candidate dataset, Ⅲ, contains a column, *color*, that is not requested by the example query of Figure 10.9. As discussed before, an aggregation necessary to remove that column requires additional user interaction. The particular aggregation function chosen is out of

scope for this algorithm and will be represented by a placeholder in the workflow graph (cf. the node “user” in Figure 10.14). After applying that aggregation function, the schema of the results conforms to the one of Ⓘ and both can be combined via a union-operation. The combination of both datasets accounts for all *sheep* measurements but does not cover the corresponding data for *population*. The latter is provided by dataset Ⓘ. The schemata of both intermediate results are already compatible, so both (sub)results can be merged using a join-operation. This concludes the creation of a workflow as all selected datasets have been included. The final workflow is illustrated in Figure 10.14. After replacing the placeholder operation with a suitable aggregation derived through user interaction(s), the workflow can be executed to materialize the requested dataset.

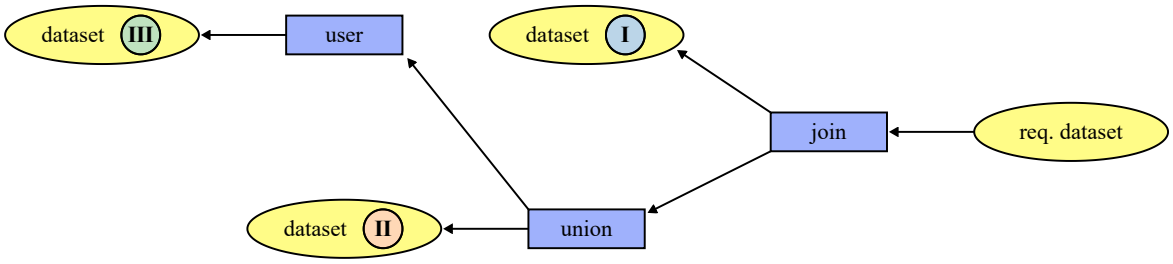


Figure 10.14: Example: Workflow to fulfill the posed request. Styling and layout according to guidelines for PROV-documents, which will be described in Subsection 12.1.3 and Section 13.5. In particular, arrows point back in time to a activity that generated a result or an entity that was used as part of an activity.

CHAPTER 11

SELECTION OF VISUALIZATION

After the necessary data is gathered and prepared, the next step is finding a suitable visualization. Over time, a plethora of variants has been developed [207]. People came up with more and more sophisticated ways to embed complex data and relationships into easy-to-access visual metaphors.

With the wide range of visualizations came the agony of choice for willing users. Which factors determine the visualization that is best suited for the current situation? Some factors are easier to recognize like the structure of data at hand or the message to convey. Others are less obvious to the uninitiated like targeting specific audiences with a visualization. One example of the latter is the use of colors. Here, certain guidelines [web114, 208, 209] have to be followed to not exclude parts of the population by, e.g., their level of vision [210] or their sex [211, 212]. Recently, efforts to collect and formalize these and best practices have gained traction [213, 214].

While visualization experts have the experience to take many of those factors into account, many other users will fail to do so. But even for experts, automating parts of the process can free mental capacity for other aspects of their work.

The goal of this chapter is to provide users with a meaningful selection of visualization options instead of confronting them with an excessive amount of visualizations to choose from. For this purpose, both the description of the current dataset (cf. Chapter 6) as well as of visualization types (cf. Chapter 7) are needed. The subsequent process is separated into three steps: select visualizations that fit the input data, find suitable mappings, and, finally, rank the results in order to be presented to the user.

11.1 Related Work

APT [1] [215] is one of the early works on automatic visualization selection. It operates under the assumption that there is a second system providing APT with the necessary input dataset in form of relational tables as well as certain other information like the importance ordering among the input columns. It models visualizations as a graphical language. That language consists of a basic set of primitive languages, which either represent visual attributes as defined by Bertin [49] or more advanced components like Venn diagrams (cf. Table 11.1), and a set of composition operators to combine them. There are also two criteria defined – expressiveness and effectiveness – to judge the suitability of a particular language for a given dataset or parts thereof.

The expressiveness criterion refers to the underlying data which is seen as a collection of facts. If the language – and hence visualization – encodes all the facts of the underlying data and no additional facts, the data is *expressible* in that language.

$$\begin{aligned} Expressible(\, facts, lang\,) \Leftrightarrow \exists\, s\, [\, lang(s) \wedge \forall f\, [\\ f \in facts \Rightarrow Encodes(\, s, f, lang\,) \wedge \\ f \notin facts \Rightarrow \neg Encodes(\, s, f, lang\,)\,]\,] \end{aligned} \tag{11.1}$$

The expressiveness criterion results in a number of `Encodes`-relations which describe certain characteristics of a given visualization. The fact, e.g., that in a bar chart the length of a bar encodes some kind of ordering over values is formalized using the following relation, where b_x are values of the dataset and $Length(bar_x)$ are the lengths of the associated bars.

$$Encodes(\, Length(bar_i) > Length(bar_j),\, b_i > b_j,\, BarChart\,) \tag{11.2}$$

The second criterion, effectiveness, is dependent on the observer. It is noted that a comprehensive theory of human perceptual capabilities is missing and, hence, only a conjectural theory is used. This theory is based on earlier works of Cleveland et al. [216], who examined the human perception of statistical graphs but focused only on quantitative data. In APT this was extended to the ranking shown in Figure 11.1 where visual attributes are ordered with respect to the data types they are to represent. In the shown diagram, each column represents a particular data type. Visual attributes are ordered vertically by their effectiveness in decreasing order. Connecting lines highlight the differences in the rank of visual attributes as the data types get more general from left to right[2].

[1] "A Presentation Tool"

[2] Each quantitative column can be regarded as ordinal. Similarly, each ordinal column can be interpreted as nominal. Although this loses some inherent information about the column, it is technically valid.

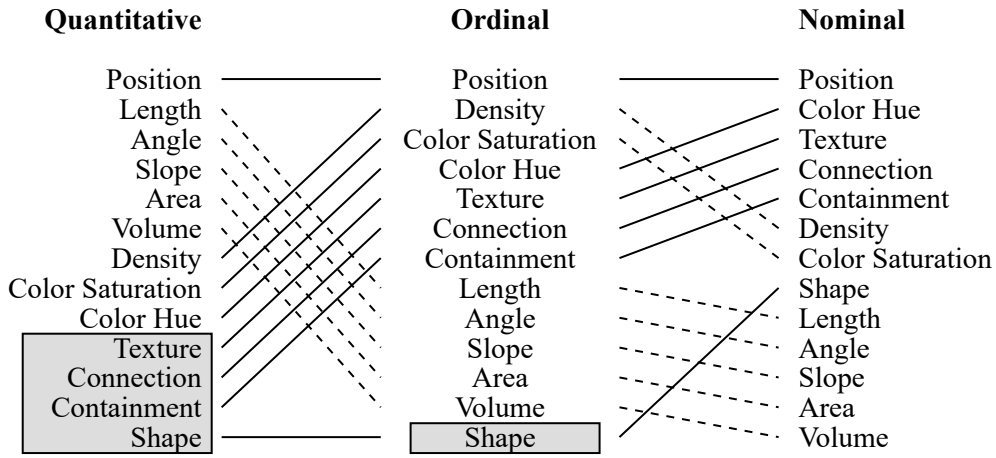


Figure 11.1: APT: Ranking of visual attributes depending on data type (from [215]). Attributes in gray boxes are not relevant for the particular type. Lines linking visual attributes across data types with solid lines showing an increase in effectiveness, whereas dashed lines represent a decrease.

In a dataset, multiple columns can share the same data type, so there is a need to decide which column gets mapped to which visual attribute. For this purpose, the following principle is formulated. The importance ordering of columns is given by the system providing the input data.

Principle of Importance Ordering:
Encode more important information more effectively.

The primitive languages mostly follow the classification of Bertin [49] (cf. Chapter 7). This origin results in the visual property "position" exclusively being represented in Cartesian coordinates. Other coordinate systems, however, are used in primitive languages APT classifies as "Misc." (cf. Table 11.1). Some visualizations like bar charts or plot charts are also considered primitive languages here, although they might be composed of the other components as well.

The composition of these primitives is led by another principle:

Principle of Composition:
Compose two designs by merging parts that encode the same information.

This results in three composition operators: double-axes composition, single-axis composition, and mark composition. The former two can be used if both operands share one or two common axes. The latter is used if multiple properties of the dataset are encoded within the mark. However, both operands have to be compatible, i.e. they do not use the same retinal variables or, if they do, those retinal variables are mapped to the same properties of the data.

Encoding Technique	**Primitive Graphical Language**	**Expressiveness Criteria**
Single-position	Horizontal axis, vertical axis	$X \rightarrow Y$ (X is nominal)
Apposed-position	Line chart, bar chart, plot chart	$X \times Y$ (X, Y are not nominal)
Retinal-list	Color, shape, size, saturation, texture, orientation	X, or $X \rightarrow Y$ (X is not quantitative)
Map	Road map, topographic map	$L \rightarrow X$ (L is a location)
Connection	Tree, acyclic graph, network	$X \times X$ (X is nominal)
Misc. (angle, contain, ...)	Pie chart, Venn diagram	Generally, $X \times Y$

Table 11.1: APT: Base set of primitive graphical languages (from [215]).

The actual selection algorithm consists of three steps: Partitioning, Selection, and Composition. The first partitioning step takes the given input data and partitions its properties in such a way that each partition can be mapped to a graphical primitive (i.e. it fulfills the respective expressiveness criterion). Here, the principle of importance ordering is used to give precedence to partitions of higher importance.

In the selection step candidate designs for each partition of the previous step are chosen. The effectiveness criterion is then used to order these candidate designs.

In the final composition step, the composition operators combine the chosen designs to form a visualization for the whole dataset. The primitives chosen for two partitions might conflict with each other as, e.g., both are mapped to the x-axis. This triggers a backtracking algorithm and will pick the next most effective candidate design for the right-hand side operator[3]. The resulting design is then presented to the user, who does not get to choose between multiple alternatives.

The **Show Me** [126] features of **Tableau** [web25] are a set of user interface commands that allow users to easily generate visualizations from multi-dimensional data. The "Automatic Marks" feature chooses a default visualization for each pane[4]. "Add to Sheet" allows to include another column in the current visualization. "Show Me Alternatives", finally, provides a list of visualizations to chose from when starting from scratch.

Automatic Marks relies on a set of rules that will choose the mark type based on the data type of the selected columns. Due to the tabular nesting of columns, only the innermost nesting level is relevant for this choice. In the Tableau user interface (cf. Figure 3.14), this correlates to the rightmost fields in the row and column shelves. Based on the data type of these two fields the mark type is chosen according to Table 11.2.

[3] Partitions are ordered according to their importance from left to right. This results in the less important property to use a less effective design.

[4] At this point be kindly reminded, that Tableau uses a tabular scheme of panes to include more dimensions into a visualization. See the respective parts of Section 7.1 for more details.

Field 1 Type	Field 2 Type	Mark Type	View Type
C	C	Text	Cross-tab
Qd	C	Bar	Bar view
Qd	Cdate	Line	Line view
Qd	Qd	Shape	Scatter plot
Qi	C	Gantt	Gantt view
Qi	Qd	Line	Line view
Qi	Qi	Shape	Scatter plot

Table 11.2: Tableau: Mark type selection (from [126]).
Categorical (C); Categorical date (Cdate); Quantitative dependent (Qd); Quantitative independent or Quantitative date (Qi)

Using the Add to Sheet functionality, users are able to add additional columns to the current visualization. These columns might either be added to the row and column shelves or to shelves that lead to a representation via size, shape, or color. To determine in which shelve to place a particular column, Show Me uses a set of heuristics.

The first heuristics tries to place new fields of the same or a similar type within the row and column shelves. This results in another level of nesting within the resulting visualization table. If the system detects that the added column belongs to the same hierarchy as an already selected one, both columns are placed next to each other. An example is the hierarchy of country, region, and state within the USA.

Another heuristic matches the column against the encoding requirements for visual variables. Tableau as of [126] includes a palette of 20 colors and 10 shapes, which places restrictions on fields assigned to those variables.

For quantitative fields, there is also the option to use the special fields “Measure Names” and “Measure Values”. The functionality of these is similar to Wilkinson’s blend operator [50] (cf. Section 7.1) in that it allows to combine an arbitrary number of fields. So for example, Measure Names could be assigned to the color value which then classifies the marks according to the measurement they represent.

Finally, Show Me Alternatives presents the user with a stable grid of visualization choices. Selectable visualizations are shown with saturated colors, whereas inactive ones are shown using a lesser opacity. To determine which visualizations are selectable, Tableau uses the data types of the given fields and respective conditions for each visualization. A line chart, e.g., requires at least one Cdate field and one quantitative field.

In addition to this user selection of a visualization, Tableau also offers a direct Show Me function. Visualizations are ranked to “default Show Me designs that embody best practices”. If the user selects the Show Me option, the system will select the highest-ranking visualization whose conditions are met. As the conditions generally get more complex with higher rank, this is

supposed to select the best visualization for the current task. Not all visualizations are included in the ranking. Those that are not, are only selectable directly by the user and will not be chosen by automatic selection.

Articulate [217, 218] tries to provide meaningful visualizations by use of a conversational interface. Users can select a dataset which is then analyzed regarding data types and attribute names. In the next step, users pose natural language questions to the system like "How does conductivity relate to depth when depth equals 7.5" to describe their information need. This query is analyzed using standard NLP techniques[5] and a feature vector is derived from it.

This feature vector includes Boolean elements like the presence of "relationship keywords" or "time series keywords" and one element that counts the number of attributes mentioned within the query. The Boolean attributes are determined by the use of dictionaries for each attribute. Words like "associate", "correlate", "link", "relate", or "relevant" are, e.g., part of the dictionary for "relationship keywords". The number of attributes in the query is calculated by comparing nouns appearing in the query with the attribute names of the underlying dataset. It is categorized into four groups: 0, 1, and 2 directly represent the number of appearing attributes, whereas 3 just signals, that more than two attributes were found.

The vector was later extended by using four more Boolean and one ordinal element. The new ordinal attribute describes the type of sentence according to the Penn treebank [219] which distinguishes, e.g., simple declarative clauses like "I want a pie chart" and direct questions like "What is the correlation between depth and temperature". The added Boolean attributes describe the presence of a certain construct. However, this time this is not done using a dictionary, but by applying part-of-speech tagging. They include, whether a comparative or superlative adjective or adverb was used, whether any cardinal numbers appear, whether quantifier like "all" or "both" are present, and whether a filter was used.

In the next step, three different classifiers (Decision Tree, Bayesian Network, and Support Vector Machine) map the feature vector to one of seven classes: comparison, relationship, composition, distribution, statistics, manipulation, and filter. These classes are not identical to the Boolean elements of the vectors although there is some overlap. The answers for some of these classes will just result in the change of the currently shown visualization, while others require a new visualization. The results of the query analysis are formalized in the "Simplified Visualization Language" (SimVL)[6] and passed on to the graph reasoner[7]. The final visualization is then selected through the decision tree shown in Figure 11.2.

[5] In particular part-of-speech tagging and stemming.

[6] This language seems to have no further adoption beyond Articulate and even here seems to be non-essential for the visualization selection process. Hence, a further description is omitted.

[7] The term "graph" is used here with the same meaning as visualization.

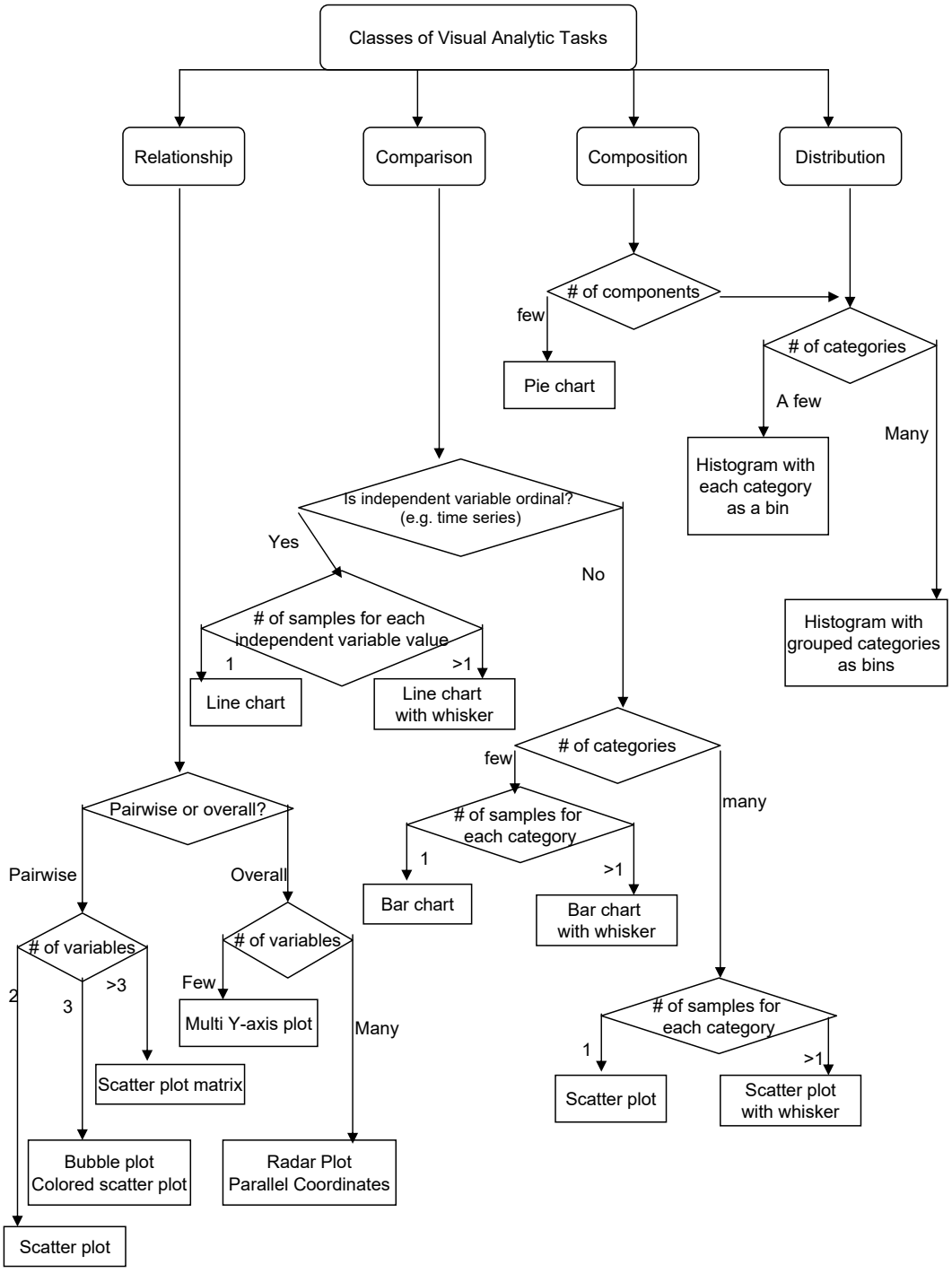
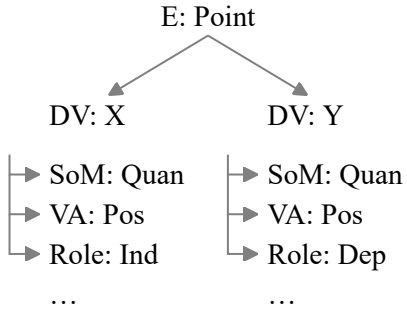
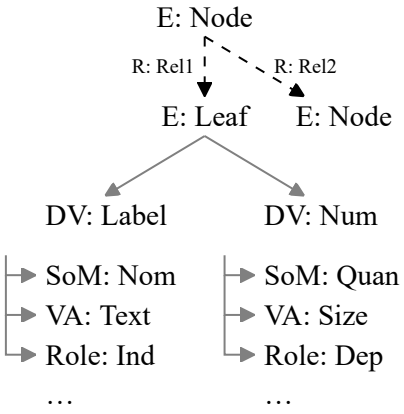


Figure 11.2: Articulate: Selection process for visualization (from [217]).

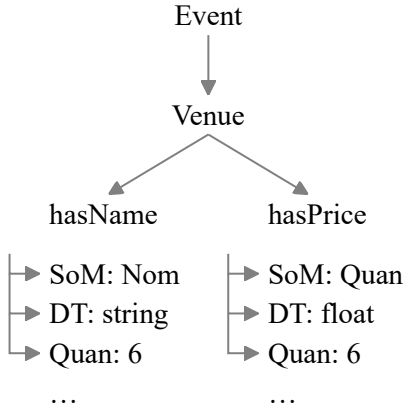
VizBoard [140, 220, web115] uses an ontology called VISO[8] [221] to recommend visualizations. The ontology models visualizations in a monolithic way using standard properties like data type, role, visual attribute, and cardinality. Example models are shown in Figure 11.3. In addition, domain ontology concepts can be linked to both input data and visualizations or parts thereof. The annotations for the input data are assumed to be created automatically and, hence, also have a probability value attached to specify the assumed accuracy. Within the model, an entity can have links to several other entities as shown in Figure 11.3(b). This allows VISO to also represent graph-based visualizations.



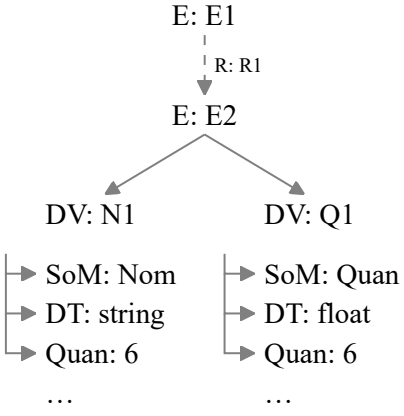
(a) Tabular Schema of a scatter plot.



(b) Graph schema of a treemap.



(c) Selected data (example).



(d) Generic schema for selected data (example).

Figure 11.3: VizBoard and VISO: Visualization and data model (from [140]).
Entity (E); Data Variable (DV); Relation (R); Scale of Measurement (SoM); Visual Attribute (VA); Data Type (DT); Quantity (Quan)

[8]The ontology is claimed to be published under `http://purl.org/viso/`. At the time of writing this URL is not accessible, though.

The visualization selection process is split into two major parts: finding possible mappings and then ranking them. The search for potential mappings consists of four tasks. In the first phase ("pre-selection"), visualizations are filtered according to three criteria:

Compatibility with the client device . The client device has to be able to create and display the visualization. As an example, the availability of certain plugins is mentioned.

Support of the number of data properties . Each column of the input dataset will have to be represented within the visualization. At this point, no check for data types is performed.

Support of tasks . If the user has specified certain tasks like "overview", the visualization has to support those.

The second phase ("gathering semantics") analyzes the input data further. First, the data is converted to RDF triples [web10] and afterward annotated using a lexicographical analyzer [220] with concepts from DBpedia [152] and WordNet [178]. Similarly, all the attributes, that also describe visualizations, are extracted from the data. This includes the "scale of measurement", which is roughly equal to the data type discussed in Chapter 5, the (technical) "data type" like `xsd:float`, and the "quantity" that equals the concept of cardinality of Chapter 7.

In the third phase ("generating generic data schemas"), a model is created to represent the input data in the same way as the visualizations (cf. Figures 11.3(c) and 11.3(d)). Here, VizBoard distinguishes between tabular and graph-based schemas. Tabular schemata consist of a single root node called "entity" with all columns attached as so-called "data variables" (cf. Figure 11.3(a)). The data variables in turn hold the aforementioned properties to describe each column. Graph-based schemata can contain multiple entities each with a number of data variables attached (cf. Figure 11.3(b)). If there is just a single class present in the input data, only a tabular schema is created. If there are multiple classes present, both tabular as well as graph-based schemata are generated.

In the final phase, the schemata of the input data are mapped against the previously selected visualization schemata. Here, the properties attached to the data variables are also considered. So in the example of Figure 11.3, the schema of Figure 11.3(c) can only be mapped to the schema of Figure 11.3(b) and not to the one of Figure 11.3(a) as the scales of measurement only fit to the former. Multiple mappings between input data and the same visualization can be created due to permutations of the data variables.

The list of mappings created is ranked in a second step. Up to four different kinds of ranking are used depending on the availability of data. The final ranking is subsequently calculated by the average of the individual rankings.

Factual Visualization Knowledge . This is a collection of rules to describe a specific visualization. It includes scores for each visual encoding following the results of Cleveland et al. [216]. The quantitative data variable of Figure 11.3(b) (treemap), e.g., is rated with 0.5 as it only uses size and not position to represent values.

Domain Assignments . This compares the semantic similarity of all concepts attached to both the visualization and the input data. To calculate that similarity, [222] is given as a reference method.

User and Device information . Similar to the factual visualization knowledge, this includes a number of rules that are evaluated at runtime to describe the user preferences. The example given is past usage of visualizations which results in a rating between 0 and 1 for each visualization.

User shared Knowledge . This uses collaborative filtering to rank the visualizations, similar to earlier works by Gilson et al. [223]. User interest in a certain visualization is recognized by three different actions: If a user uses a particular visualization more than three times, the visualization is put on a white list. If a user discards a certain visualization, it is added to a black list. The user may also explicitly rate the visualization in the context of another dataset. Depending on the particular rating, the visualization is inserted into either a black or white list.

VizRec [137, 141] is a two-phase visualization recommender system. It uses a monolithic visualization description (cf. Chapter 7) called "visual pattern" that includes two properties for each visualization component: (primitive) data type and "occurrence"[9] (cardinality). If a visualization component can have multiple data types, a separate description is given for each possible combination of data types. Listing 11.1 lists the descriptions for bar charts as an example.

```
{x-axis: string, y-axis: number}
{x-axis: date,   y-axis: number}
{x-axis: string, y-axis: number, color: string}
{x-axis: date,   y-axis: number, color: string}
```

Listing 11.1: VizRec: Bar chart descriptions (from [137]).

In a pre-processing step, the input data is analyzed with respect to data types, cardinalities, and semantic types. The extraction of semantic types is done using gazetteer lists. Using this information and the given visual patterns of the first phase[10] creates a list of all possible mappings. Example mappings from datasets in the movie domain to the bar chart descriptions of Listing 11.1 are given in Table 11.3. This also allows for redundant mappings as defined by Bertin [49]. Further, not all columns of the input data need to be represented in the mapping.

[9] All example definitions of [137] seem to omit occurrence. The examples given here will follow that convention.
[10] This part of the approach is based on earlier work [224], which resulted in a publicly available prototype [web116].

Visual Pattern	Mappings
{x-axis: string, y-axis: number}	{x-axis: movie name, y-axis: budget} {x-axis: movie name, y-axis: gross} {x-axis: genre, y-axis: gross} {x-axis: genre, y-axis: budget}
{x-axis: date, y-axis: number}	{x-axis: creation year, y-axis: budget} {x-axis: creation year, y-axis: gross}
{x-axis: string, y-axis: number, color: string}	{x-axis: movie name, y-axis: budget, color: genre} {x-axis: movie name, y-axis: gross, color: genre} {x-axis: movie name, y-axis: budget, color: movie name} {x-axis: movie name, y-axis: gross, color: movie name} {x-axis: genre, y-axis: gross, color: genre} {x-axis: genre, y-axis: budget, color: genre} {x-axis: genre, y-axis: gross, color: movie name} {x-axis: genre, y-axis: budget, color: movie name}
...	...

Table 11.3: VizRec: Possible mappings (after [137]).

The second phase ranks the created mappings according to the principles of collaborative and content-based recommender systems. Users are able to rate specific mappings on a scale from one to seven. These ratings are fed to a collaborative recommender system. When a user wants to visualize a new dataset, the previous ratings are used to select a group of similar users using the Pearson correlation coefficient [225]. From the ratings of those similar users a predicted rating for the current user is calculated which is then used to rank the given mappings.

However, new users have no previous ratings given, so the collaborative recommender system will fail due to a lack of data (cf. the "cold start problem" [226]). For these cases, a content-based recommendation is calculated. Each mapping is considered a separate item which is then tagged using the concepts for each component. So the first mapping from Table 11.3 will get two tags: `movie name` and `budget`. Similarly, the user will be assigned tags for which all concepts from the input dataset are used. Users may also manually add more tags which will be added to the existing ones. For each item (mappings and the current user) a vector is calculated that holds the TF-IDF [185] values for each tag[11]. Using a conventional vector space model and cosine similarity, the system ranks all mappings by their similarity to the current user.

Finally, VizRec will combine the scores of both recommendation strategies. With rec_j denoting the recommendation score of the jth strategy, w_j the respective weight, u the current, user and i a mapping item, the final, hybrid ranking is calculated as follows:

$$\text{pred}_{hyb}(u,i) = \sum_{j=1}^{n} w_j \times \text{rec}_j(u,i) \tag{11.3}$$

[11]The set of all candidate mappings and their tags is used as the corpus for the IDF computation.

VisRec currently employs two recommender systems ($n = 2$) and a uniform weighting scheme ($w_j = 0.5$). Ranking these scores will result in the final recommendation as presented to the user.

The underlying data model of **VizAssist** [127, web84] has already been described in Section 7.1. Given an input dataset as well as user-assigned importance values for each column and a list of objectives, VizAssist will compute two different scores for each visualization: an objective score and a match score.

The objective score will be calculated using Equation 11.4. $Ouser_j$ represents the user-defined objectives as a Boolean variable. $Ovisu_{ij}$ are the supported objectives of the ith visualization. Higher scores indicate better suitability for the objectives.

$$Score_{objectives}(V_i) = \sum_{j=1}^{l} Ouser_j \times Ovisu_{ij} \tag{11.4}$$

The match score, on the other hand, uses the suitability matrix Mat_{GL} which assigns a suitability value to each combination of a visual attribute $type(VA_{ij})$ and data type $type(A_l)$ (cf. Section 7.1). This value is multiplied with the given importance value u_l for each attribute A_l, before calculating the overall sum. If a visualization can not support all the attributes of the given dataset, it will be discarded.

$$MatchScore(V_i) = \sum_{\substack{(VA_{ij},A_l) \\ \in\, Mapped\,Attributes}} Mat_{GL}\left(\, type(VA_{ij}),\, type(A_l)\,\right) \times u_l \tag{11.5}$$

Users are presented with a list of all feasible visualizations which can be ordered using both scores individually. The algorithm is deterministic as it always will result in the same output for the same input data. However, it does not allow to explore different attribute mappings. In a second recommendation step, users may explore different mappings using an interactive genetic algorithm.

Genetic algorithms are modeled after the evolution process in biology and provide approximate solutions to combinatorial problems [227]. A solution instance to the given problem is described by a number of genes. Each gene represents a particular aspect of the solution and is usually coded as a nominal value. In order to determine the quality of a given instance, a so-called fitness function has to be provided. This function calculates a single value for each instance to determine the quality of the solution.

The actual algorithm starts with a number of randomly or heuristically generated instances – the population. The fitness value is computed for each of these instances. The instances performing worst are discarded from the current population. They are replaced by new instances that are generated by combining the remaining instances. With a certain probability some genes are mutated, i.e. their values are replaced with randomly chosen ones. This newly generated population is the input to another iteration of the algorithm. Genetic algorithms terminate if either a certain threshold regarding the best fitness value within a population is reached or the fitness value is stagnant over a predefined number of generations.

The rationale behind an *interactive* genetic algorithm is that it is not always possible to find a mathematically precise formulation for the required fitness function. In the case of visualization where user perception is an important factor, the feedback of the current user can be used instead.

VizAssist uses the vector of user-assigned importance weights as a genome. It is reasoned that this can be explained to a user more easily and allows for reuse within VizAssist or other tools. Alongside, three operators are defined: a creation operator, a mutation operator, and a crossover operator. The *creation operator* is used to create the initial population and just applies the mutation operator to the initially given importance weights. The *mutation operator* adds some random noise $[-\delta_b, +\delta_b]$ to each weight individually with a probability $p_{mutation}$. The *crossover operator*, finally, takes two genomes as input and returns a single genome where each gene is chosen with equal probability from either of its parents.

During the interaction, users are presented with nine different mappings in form of the resulting visualization. They are asked to select those visualizations that currently fit their needs best. In the next generation, all other visualizations will be replaced using the above operators. They are also provided with a slider for "diversity" which changes the parameters of the mutation operator. Less diversity is associated with an exploitation approach, whereas more diversity results in an exploration approach. By using multiple generations of this phase, users can refine their information need even without having to state particular importance values for each data attribute.

Draco [228, web117] is a rather recent visualization recommender. In contrast to other tools, it models the recommendation process by a set of hard and soft constraints on the visualization task. Valid recommendations have to fulfill all hard constraints and are ranked subsequently by the soft constraints they fulfill. The actual materialization of the recommended visualization is relayed to Vega-Lite [229], a grammar and corresponding compiler in the tradition of *The Grammar of Graphics* by Wilkinson [50] (cf. Section 7.1) and the works it inspired like Polaris/Tableau [web25, 80] (cf. Sections 3.4 and 7.1). As the focus is on visualization recommendation here, details of Vega-Lite will largely be omitted at this point.

Constraints are constructed following the requirements of Answer Set Programming (ASP). They are built from using so-called atoms and their negations that describe certain aspects of the problem at hand. Atoms can further make use of variables to be instantiated. So the general definition of an atom $encoding(E)$ can, e.g., be instantiated as $encoding(e1)$[12]. Each constraint consists of a head and a body, such that the single atom in the head is `true` ("can be derived") if all atoms in the body are `true`. Both head and body, can also be empty. An empty head represents an integrity constraint that derives `false` from its body, i.e. all atoms in the body being `true` results in a contradiction. Similarly, an empty body declares the respective atom to be `true`.

[12] In the notion of [228], upper case letters represent variables, whereas lower case letters are used for instances.

Syntactically, soft and hard constraints are distinguished by the operator that separates head from body. Hard constraints use :− as an operator, whereas soft constraints resort to :~. In addition to head and body, soft constraints are further augmented with a weight w at the end of the constraint. This weight represents the cost of a solution when it violates the respective constraint. The authors note that a single constraint can be violated multiple times when the variables are instantiated by different values. Examples are shown in Listing 11.2. Here, the upper soft constraint states that continuous fields are preferred to include a zero baseline and penalizes a solution by 5 if this does not hold. This formalizes the intuition that non-zero-based scales are generally misleading but might be reasonable under certain conditions. The hard constraint is a direct consequence of Vega-Lite implementing only 8 different shapes. It states that any variable bound to the channel "shape" has to have a cardinality less than or equal to 8.

```
:~ continuous(E), not zero(E) . [5]

:- channel(E,shape), cardinality(E,C), C > 8
```

Listing 11.2: Draco: Soft (upper) and hard constraint (lower) examples (after [228]).

In addition, ASP allows to define aggregations in the form of $lA_0,\ldots,A_nk$. This defines that out of the set of n atoms, at least l and at most k atoms need to be true. Such aggregates can appear in both the head and the body of a constraint.

Using this formalism, Draco defines the Vega-Lite specification, two user tasks[13], and the data schema as a set of atoms. Visualizations are defined by a number of atoms. Examples include `mark(X)` to define a specific mark type (point, line, area, or bar), `encoding(E)` to declare an encoding E, or `channel(E, Y)` to state that an encoding E uses an encoding channel Y. Draco currently supports two tasks, summary and value, that are declared by `task(X)` which can also be associated with individual fields using `interesting(X, Y)`. Finally, the data schema is described by five atoms:

- `num_rows(X)` to state the number of rows,
- `fieldtype(X, Y)` to declare data types (string, number, date, ...),
- `cardinality(X, Y)` provides the number of distinct values within a column,
- `entropy(X, Y)` defines a column's entropy, and
- `extent(X, Y, Z)` describes the minimum and maximum value of a column.

The total cost of a particular visualization is given by the sum of costs of the soft constraints it violates. While the individual costs can be manually defined by experts, [228] also suggests a learning-to-rank approach [230] to determine weights and, hence, costs. The training is accom-

[13] A wider collection of tasks is announced as part of future work.

plished by using pairs of visualizations. For each such pair, the preference for either visualization is given. Using these preferences and a RankSVM model [231], the cost for all soft constraints can be determined.

The input to the actual system is the knowledge base described above including a description of the dataset at hand and potentially a partial specification of the result. Such a partial specification can already include certain binding as shown in Listing 11.3. Here for a fictitious dataset about cars, already one encoding $e1$ for the field $horsepower$ is specified which after a binning operation is bound to the channel x.

```
encoding(e1).
:- not channel(e1,x).
:- not field(e1,horsepower).
:- not bin(e1,_).
```

Listing 11.3: Draco: Example for Partial Specification (from [228]).

At this point, three sets of constraints are available: dataset constraints describing the data to be visualized, query constraints to capture the user's needs including a potential partial specification, and knowledge base constraints formalizing the background knowledge about visualization components and their interactions. All these constraints are forwarded to an ASP solver, Clingo [232]. Following the example of a car dataset, the facts provided in Listing 11.4 might be returned and be interpreted as an additional encoding $e2$ that binds a field $count$ to the y-axis. This is translated back to a Vega-Lite specification which is subsequently returned as a result to the user.

```
encoding(e2).
channel(e2,y).
aggregate(e2,count).
```

Listing 11.4: Draco: Sample response (from [228]).

11.2 Discussion

While Section 7.1 presented two approaches to visualization descriptions, in practice there seems to be an overwhelming preference towards the monolithic model. Only APT, Draco, and to a lesser extent Tableau use a somewhat constructive approach. But even here some approaches have monolithic components: APT also uses monolithic visualizations as components besides the constructive ones. In Tableau, the Show Me features are also based on predefined visualizations instead of a truly dynamic composition. Only Draco appears as a truly constructive recommender.

Besides Draco which uses a constraint solver for the whole task at once, the general process of selecting a visualization consists of three phases: In the first phase, the list of all visualization will be filtered down to those that can hold the given input data or support the selected tasks. A second phase tries to establish bindings between input data and visualization. The last phase takes the list of bindings and ranks them according to some criteria.

Sometimes the filter phase is skipped, though. In VizRec, for example, it is possible that only a subset of columns appears in a given binding. As a consequence, there is less opportunity to discard certain visualizations early in the process.

Afterward, the systems generate bindings from input data to the visualizations. Finding such bindings is usually based on comparisons of data types and sometimes cardinality. VizBoard and the Automatic Mark selection of Tableau also use the role of a certain column. APT uses an importance ordering of the input columns: The most important column will be bound to the most prominent visual attribute and so on.

The candidate bindings will afterward be ranked in order to be presented to the user. The exceptions are APT and Articulate. APT will return only the first suitable binding found and not explore other possible bindings. Articulate uses a decision tree to determine the appropriate visualization, so there is no need for further ranking.

In most systems, the results will be ranked, though, so users get an idea of which visualizations are deemed more suitable options by the system. For this ranking, different criteria are used and/or combined: A data-centric approach uses an importance ordering of the input columns to assign more important columns to more prominent components (APT, VizBoard, Draco). A second criterion applies collaborative or content-based filtering techniques to the domain (VizBoard, VizRec). If semantic information is available on both the input data as well as the visualization, semantic similarity measures are used (VizBoard). Best practices may contribute to the ranking as well (Draco). Finally, the pursued tasks can be used for ranking (VizBoard, VizAssist, Articulate, and Draco to some extent).

The use of some kind of importance ranking for the columns requires a similar property on side of the visualization components. Already APT mentioned the lack of a holistic theory of human perception and there seems to be little progress regarding this in the visualization recommendation area. As a consequence, systems using the importance criterion only refer to the works of Cleveland et al. [216], (APT, VizBoard) or to some rather vague “best practices” (Tableau). On the other hand, Draco proposes a way to learn best practices including the importance of components from provided example ratings. However, as other with machine learning approaches, this depends on a rather large training collection.

Some recommender techniques from the field of information retrieval have also been applied. Collaborative filtering sees visualizations as items. The rating is either explicitly given (VizBoard, VizRec) or deduced by past usage (VizBoard). The content-based approach taken by VizRec uses the different bindings as items that are tagged by the used columns. These are compared

with tags provided by the user – either explicitly or by means of the input data. However, the use of recommender systems also subjects a system to problems like scarcity of data and the aforementioned cold start problem. They also push users towards visualizations that they have been using in the past. This limits the option to explore other types of visualization that might be better suited for the task at hand.

VizBoard also uses the concept of semantic similarity to compare the input data to possible bindings. While this, in general, seems a sound approach, it requires the annotation of visualizations with (domain-specific) concepts. At this point, it remains doubtful whether these annotations can be made sufficiently general to match all concepts attached to the input data.

For the task criterion, one major challenge is the transfer of implicit knowledge about this from the user side to an explicit representation within the system. The naïve approach is to present users with a list of options and have them select one or more entries. This strategy is used by VizAssist, but this begs the question if the user really needs to be bothered with such a task. Some users may also be lost when confronted with a list of possible tasks if they do not understand the technical terms presented or are unsure to which categories their current work belongs. Articulate, on the other hand, extracts a task vector from the user query. This further automates the process and, in doing so, releases users from one input step. The actual description of tasks seems more sophisticated within VizAssist as here also hierarchical dependencies between tasks are modeled.

Systems that result in only a single visualization instead of a (ranked) list of options suffer from one problem: They do not allow users to explore new visualizations. This will prevent them from discovering new visualizations they did not know before. These new visualizations might be better suited in their current situation, though. So by not giving them a list to choose from, the respective systems do also not foster an improved visualization literacy among their users.

One disadvantage almost all presented systems share is that they do not allow the user to manually change the binding. Most systems present users with a list of ranked bindings without any option to change just parts of them. The two exceptions are Tableau and VizAssist. Tableau itself offers users to manually assign columns to specific properties. However, the general selection of visualizations is rather restricted (cf. Section 7.1). VizAssist does not allow users to change the binding directly, but at least provides means to alter the binding once a visualization is selected. If advanced users have a specific visualization and binding already in mind, this approach will reduce their productivity, though.

11.3 Approach

Each visualization recommendation process starts with the generation of candidate bindings. Here, the first decisions have to be made, though: Should only one final result be presented or should users be able to select between multiple options? This first decision also extends to the variations of a single visualization: Should the system show only the best binding for each visualization — whatever "best" means in this context — or display multiple variations?

Within this work, multiple visualizations will be shown, but for each one, only a single binding will be provided. Users will be able to adjust this binding if they feel a different binding is more suitable. The rationale behind this is that showing multiple variations of a single visualization clutters the result list, but provides little additional value. In some cases, other visualization types might even be completely overshadowed by the numerous variations for a single one. Assume, for example, a high-dimensional dataset for which parallel coordinates are deemed to be best suited. Due to the high number of possible combinations (which input column gets bound to which position in the order of coordinates), other visualizations like spider charts might be way down in the list of recommendations. As users tend to highly favor content and items that are immediately visible[14], those visualizations will hardly be noticed thus limiting diversity in actual results.

Another decision is whether to use all columns of the input dataset or to allow for subsets to be mapped to a visualization. Here, the decision has been made to favor inserting all available columns into the visualization over including only subsets. However, instead of preventing the latter altogether, it is only penalized during the ranking step. Requiring all columns to be used in the visualization would imply that all superfluous columns have been removed during data preparation. In general, this condition will not hold and, hence, the system also has to account for non-reduced input datasets. While the system will still attempt to include as many columns as possible, it may drop individual columns to meet a visualization's requirements.

Having made these decisions, next up is finding suitable bindings. To reduce the overall set of visualizations to consider, the system will first remove those visualizations from the candidate pool whose requirements can be met neither by the input dataset as a whole nor by any of its subsets. Any valid subset of the input will still need to include at least one measurement, though. Subsequently, for each of the remaining visualizations, one suitable binding will be calculated.

11.3.1 Weighted Bipartite Matching

At this point, the problem at hand can be phrased like this: Given a set of column descriptions (cf. Chapter 6) and one visualization description consisting of multiple components (cf. Chapter 7), find the best possible binding with respect to the stated criteria. The criteria can be formalized

[14]The content "above the fold"; see, e.g., [233].

into a scoring function that given a particular column and component returns a score to estimate the suitability of that component to represent the given column. Details of this scoring function will be discussed later.

This problem statement itself resembles the one of *Maximum Bipartite Matching*, which is discussed in graph theory [234]:

> Given an undirected Graph $G = (V, E)$, a *matching* is a subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of M is incident on v.
> A *maximum matching* is a matching of maximum cardinality, that is, a matching m such that for any matching M', we have $|M| \geq |M'|$.
> [...] *bipartite graphs*: graphs in which the vertex set can be partitioned into $V = L \cup R$, where L and R are disjoint and all edges in E go between L and R.

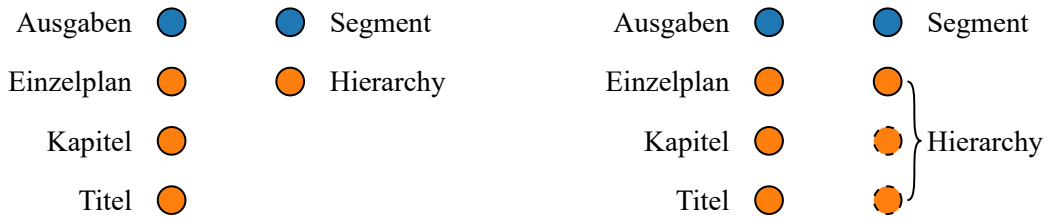
The definition can be extended by assigning weights to the edges resulting in a *Weighted Bipartite Matching*[15] [235]. This problem has long been discussed and several algorithms have been presented to solve it [235, 236, 237].

Before actually applying the algorithm, the binding problem has to be stated as an input instance to the Weighted Bipartite Matching. The partitions of the bipartite graph can easily be associated with the input data and the visualization components respectively. So for each column one vertex will be created in the L partition and for each visualization component, one will be added to R.

However, this will fail for components that allow for multiple bindings. As per definition, within a matching every vertex is linked to only one other vertex. So to accommodate for multiple bindings, multiple additional copies of the vertex representing the respective component will have to be inserted (cf. Figure 11.4). There will be one original vertex and possibly multiple copy vertices for one component. This distinction will become relevant when discussing the scoring function. As for the number of copies, a naïve approach inserts as many copies as there are vertices in the other partition. To reduce the instance size, this can be reduced to the number of vertices with a compatible data type[16] as no other can be matched anyhow. For the sake of brevity in further discussion, there is no distinction made between a vertex and the column or component it represents except where noted explicitly.

The other ingredient to the problem instance are the edges. Here, all vertices of L will be connected to all vertices of R, i.e. $E = \{ (l, r) \mid (l \in L) \wedge (r \in R) \}$. Again, some of these edges could be omitted due to filtering by data type or other properties. Some of the algorithms applied to the problem require a complete bipartite graph as input which is usually represented by an adjacency matrix. In order to allow for an easy substitution of the solving algorithm, this requirement shall be fulfilled in all instances.

[15] Also-called *Assignment Problem*.
[16] That is a vertex representing an input data column with a compatible data type.



(a) Before Duplication.

(b) After Duplication.

Figure 11.4: Yavaa: Preparation of mapping graph - duplication of vertices; duplicate nodes shown with dashed border, data types represented by color; using description of Listing 7.2 and Figure 7.10.

11.3.2 Scoring Function

For all edges, a weight has to be calculated by the use of a scoring function $score()$. This scoring function has to fulfill at least two criteria:

1. A higher score indicates better suitability for a given binding.
2. Edges representing infeasible bindings, like a data type mismatch, have a score of zero.

These criteria alone would allow for at least two cases of infeasible bindings: Multiple columns might be bound to a single component that allows for multiple bindings while other required components have no binding. Similarly, optional components might have a binding attached while required ones were omitted. To prevent this behavior, vertices representing visualization components will have to be distinguished into two groups: A set of primary vertices V_p will contain all the single-binding components as well as the original vertices of multi-binding components. Another set of secondary vertices V_s is comprised of the copied vertices for multi-binding components and optional components. Using this distinction, a third criterion for the scoring function can be formulated:

3. Let V_i be the set of all vertices representing input data columns. For any combination of input vertex $v_i \in V_i$ and primary vertex $v_p \in V_p$ the following property has to hold up:

$$score(v_i, v_p) = 0 \quad \vee \quad \underset{v_s \in V_s}{\forall} score(v_i, v_p) > score(v_i, v_s) \tag{11.6}$$

Adding this criterion ensures that bindings will favor primary vertices first, before resorting to the secondary ones. Without loss of generality, scores for edges including secondary vertices can be assigned a range of $[0, 0.5)$ and edges including primary vertices a range of $[0.5, 1]$.

As for the scoring function used within this work, a product will be used whose factors each represent one binding criterion. This approach allows to easily add further aspects in the future without the need to change the core approach. The calculations for the factors themselves

are ordered by their computational cost. This allows short-circuiting computational expensive calculations if earlier, less expensive aspects already resulted in a factor of zero. While this might not be overly relevant with the criteria provided within this work, this simple property might improve the performance in the future.

In the following, the individual factors used within this work will be listed and described. They are ordered by computational complexity in ascending order, putting computational more expensive ones towards the end.

Data Type A Boolean factor indicating whether the data types of the component matches that of the column.

Role A Boolean factor indicating whether the role of the component matches that of the column.

Primary vs. Secondary Component This factor takes a value of 1 for primary components and a value of 0.5 for secondary ones.

Cardinality If the component description gives a function description for cardinality, this function is evaluated for the given column. Otherwise, a value of 1 is returned.

Semantic Concept If a semantic concept restriction is given for the component, it is evaluated whether the column's concept is a descendant of that concept. A value of 1 is returned if such a connection can be found or the component places no restriction. If the column's concept can not be established as a descendant of the component's concept, a value of 0 is returned.

Not included here is any ranking of the visual attributes as, e.g., suggested by APT [215]. As already mentioned previously, this could easily be added in the future, but the current lack of research in this area does not allow for assigning specific numeric values.

If the scoring function has not been short-circuited by one factor resulting in a zero value, the final result is computed by Equation 11.7.

$$score_{binding}(\, component,\, column\,) = \prod_{factors\ f_i} f_i(\, component,\, column\,) \tag{11.7}$$

11.3.3 Ranking

Having constructed a complete graph and assigned weights to all edges using the aforementioned function, the resulting graph can serve as an input instance to the problem of Weighted Bipartite Matching. At this point, any of the existing algorithms can be used to determine a maximum weight matching. The resulting matching has a total weight (the sum of all weights from the used edges). So, the result consists of a list of visualizations, each having one representative binding and an associated weight or score.

When presented in the user interface, the visualizations are arranged in a particular way. This arrangement suggests an order, which might or might not be intended, though. In general, items towards the top, left, or appearing first are presumed more important. So instead of placing visualizations next to one another without considering this implication, the system has to rank the alternatives and use that ranking as a basis for display.

The first step towards such a ranking is already defined by the aforementioned weight or score for each visualization mapping. As the scoring mechanism is uniform for all visualizations, the results are comparable as well. The presented criteria for bindings, however, mostly consist of Boolean factors which will skew the results to both extremes (0 and 1). This provides little differentiation between different visualizations, though, and thus other ranking factors have to be included as well.

An often used factor to rank visualizations is the support for certain tasks (cf. Section 11.1). Here, the main challenge is to derive a list of perceived user tasks. As already discussed before, simply asking users for their intended goals is less preferable as it requires explicit user interaction and knowledge. Other approaches (cf. Articulate [217, 218]) are beyond the scope of this work. So at this point, it is just assumed that user tasks have somehow been specified.

With a predefined, fixed list of tasks, the system is able to define a task vector such that every element of that vector represents the presence of a task in either the user request or the visualization description. For the ranking factor $score_{tasks}$, the request vector $\vec{r}$ can be compared to the visualization vector $\vec{v}$ using standard cosine similarity (cf. Equation 11.8).

$$score_{tasks} = similarity_{cosine}(\vec{r}, \vec{v}) = \frac{\vec{r} \cdot \vec{v}}{\|\vec{r}\|_2 \|\vec{v}\|_2} = \frac{\sqrt{\sum_{i=1}^{n} r_i v_i}}{\sqrt{\sum_{i=1}^{n} r_i^2} \sqrt{\sum_{i=1}^{n} v_i^2}} \tag{11.8}$$

A third factor is the past usage of a visualization in certain application domains. By harvesting user-created workflows (cf. Chapter 12), the system over time accumulates knowledge about which visualizations are used in which contexts. These workflows contain not only the selected visualizations, but also which columns and which concepts were used in creating a particular visualization.

The process of creating a ranking out of this past usage is related to the problem of classification: In general, given an input instance, the classifier has to decide to which category or class this instance belongs. This decision is based on a training set of instances for which the respective categories are known. An extension to this are probabilistic classifiers. Instead of returning just the “best” category for a given instance, they provide a probability distribution over all categories.

Another related field is the mining of association rules [238]. Here, problem instances are databases of transactions $D = \{ t_1, t_2, \ldots, t_n \}$ which in turn consist of a collection of items from a fixed set $I = \{I_1, I_2, \ldots, I_m\}$. The task is to find rules of the form $X \Rightarrow Y$ with $X, Y \subseteq I$. The quality of said rules is estimated by different measures:

Support describes the fraction of transactions that include all elements of a set Z that usually consists of the union of X and Y [238].

$$supp(Z) = \frac{| \{ t \in D \mid (Z) \subseteq t \} |}{|D|} \tag{11.9}$$

Confidence is the fraction of transactions containing X that also contain Y [238].

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{11.10}$$

Lift estimates the significance of a rule by comparing it to the expected confidence [239].

$$lift(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X) \times supp(Y)} \tag{11.11}$$

In theory, the number of rules can grow exponentially in the number of items. To limit this growth, algorithms to find rules are oftentimes given values for minimum support and confidence. That way, the most insignificant rules are immediately pruned from the results.

A combination of both approaches can be found in associative classification. Here, the right-hand side of rules found is limited to items representing a category, whereas the left-hand side represents the features of the input instance. One major advantage over other classification approaches is that the decision is based on simple if-then-rules which are easily accessible to human users [240]. The set of rules can also easily be updated as soon as new training data becomes available.

The transition from visualization workflows to instances for associative classification is straightforward: The columns' concepts are represented by items, while the respective visualizations form the categories. To solve the associative classification problem, a multitude of algorithms have been proposed (for a survey see, e.g, [240]).

Within this work, the SBA[17] approach presented in [241] shall be used which similar to probabilistic classification approaches returns not just a final category, but assigns scores to each category. It is intended to separate between two classes ("positive" vs. "negative"). This, however, can be overcome by sequentially computing the score for each category individually, subsuming all other categories under "negative". For the basic rule generation, any algorithm may be used (e.g., Apriori [242]). SBA uses a different minimum support and confidence for each category

[17] Scoring Based on Associations.

C_i to compensate for unbalanced training data D. The function $f(\ C_i\ )$ defines the number of transactions labeled with the category C_i and t_minsup is a configuration parameter.

$$minsup(\ C_i\ ) = t\_minsup \times \frac{f(C_i)}{|D|} \tag{11.12}$$

$$minconf(\ C_i\ ) = \frac{f(C_i)}{|D|} \tag{11.13}$$

The scoring itself uses all rules applicable to the input instance and the category in question, i.e. both positive as well as negative rules contribute to the result. Other approaches will only use the rule with the highest confidence and in doing so ignore a vast proportion of the knowledge base. The score S is calculated by the means of Equation 11.14, where

- POS and NEG are the sets of positive and negative rules,
- $W^i_{positive}$ and $W^j_{negative}$ are weights assigned to the respective rules,
- $conf^i$ is the original confidence of a positive rule, and
- $conf^j_{positive}$ equals $1 - conf^j$.

$$score_{concept} = \frac{\sum_{i \in POS} (W^i_{positive} \times conf^i) + \sum_{j \in NEG} (W^i_{negative} \times conf^j_{positive})}{\sum_{i \in POS} W^i_{positive} + \sum_{j \in NEG} W^j_{negative}} \tag{11.14}$$

The weights are defined differently for positive and negative rules in Equations 11.15 and 11.16. Here, k is a constant to reduce the influence of negative rules.

$$W^i_{positive} = conf^i \times sup^i \tag{11.15}$$

$$W^j_{negative} = \frac{conf^j \times sup^j}{k} \tag{11.16}$$

The calculations necessary for the concept-based score are all just dependent on a number of counts: The total number of workflows (transactions), the frequency of selection for visualization (category), and the frequency of concept sets (items). Over time, when more workflows become available, all these counts can easily be updated without the need to redo a modeling process like required by other classification algorithms.

Having computed three different scores for each visualization, finally, all of them have to be merged together. As per definition all of them return a value in the range of $[0, 1]$, this can easily be done by computing the product:

$$score = score_{binding} \times score_{tasks} \times score_{concept} \tag{11.17}$$

11.3.4 Special Case: Nested Visualizations

One type of visualization has been omitted so far from the discussion: visualizations allowing for nesting. Here, the number of possible components also depends on the nested visualization used and thus is not fixed as with other visualizations. The aforementioned process, however, is not capable to handle such dynamic descriptions. So before inserting nested visualizations into the process, feasible combinations have to be found.

Nesting visualizations have one helpful property: It always reduces the number of remaining columns, but never increases it. Consequently, one can first apply a nesting visualization and then try to map the remaining columns against the set of visualizations again. As the number of columns is strictly decreasing in each iteration, this approach will terminate when either all columns are accounted for or no more feasible visualization can be found for the remaining columns. While the former represents a feasible combination, the latter does not. By enumerating all feasible combinations, a list of *virtual visualizations* is created, which can be fed into the mapping/ranking algorithm. These visualizations are called virtual as they are not part of the visualization repository consisting of the descriptions defined in Chapter 7. They are, in fact, just created on the fly, if the combinations they represent are applicable to the current input data.

```
FUNCTION findVirtualViz( inputCols )

  virtViz = <empty list>

  FOREACH nesting visualization nViz DO

    if ( nviz is applicable to the input data )

      cols = remove columns needed for nviz from inputCols

      FOREACH( for cols feasiable visualization viz )
        add ( nViz, viz ) to virtViz

  RETURN virtViz

END
```

Listing 11.5: Pseudocode: Creation of virtual visualizations.

Listing 11.5 shows the pseudocode for this algorithm. At this point, the feasibility of a visualization is not estimated by a complete binding, but rather by comparing the number of columns per data type and possibly attached concepts. The actual binding is created later by passing the virtual visualization into the general process as described before.

There are cases where the search for feasible nested visualizations can be short-circuited. If the number of remaining columns is less than two, no more visualization can be found as all visualizations consist of at least two required components. Furthermore, the set of remaining columns has to contain at least one measurement. This again is caused by the fact, that each visualization will require at least one component with the role measurement.

11.3.5 User Interface

Finally, the computed recommendations are presented to users. The visualizations and virtual visualizations that passed the previous selection process are ordered by the score defined in Subsection 11.3.3. Users are able to choose their preferred visualization by selecting a generic preview image.

As the system only suggests one binding, users are given the possibility to change it. The changes, however, are also subject to the restrictions as given by the visualization descriptions. So for example, data types and concepts have to line up. The restriction on the cardinality of a column, on the other hand, may be overridden by users. In this case, only a warning is issued which can be dismissed by users.

11.3.6 Summary

Figure 11.5 summarizes the steps necessary for the visualization selection approach presented. Two lists of visualization descriptions are retrieved from a repository of visualization descriptions: nesting and non-nesting visualizations ①. Both lists are used to create a list of virtual visualizations ② (cf. Subsection 11.3.4). The non-nesting visualizations are also initially filtered by comparing the number of columns to the number of columns of the input dataset ③.

These two candidate sets are then fed into the binding search ④ (cf. Subsection 11.3.1). Here, using a scoring function (cf. Subsection 11.3.2) and weighted bipartite graph for each candidate a binding is generated which is passed on alongside the respective its total score.

The final ranking algorithm ⑤ (cf. Subsection 11.3.3) uses these scores as well as an association classification algorithm and the task similarities between visualization and user input to create the final ranking. Subsequently, this ranking is passed on to the user interface (cf. Subsection 11.3.5).

Table 11.4 compares the approaches presented in Section 11.1 and Yavaa along three main categories: (i) general properties of the approaches, (ii) data characteristics used in the recommendation, and (iii) other relevant features. In the remainder, the focus will be on Yavaa as the other approaches have already been discussed previously in Section 11.2.

Yavaa's visualization selection approach is envisioned as a recommender. So in particular, it provides a ranked list of visualizations instead of a "single best" choice. Unlike many of the discussed systems, the bindings for specific visualizations can be adjusted by users. Similarly, the dataset may contain columns that should not be visualized. While Yavaa is not able to determine

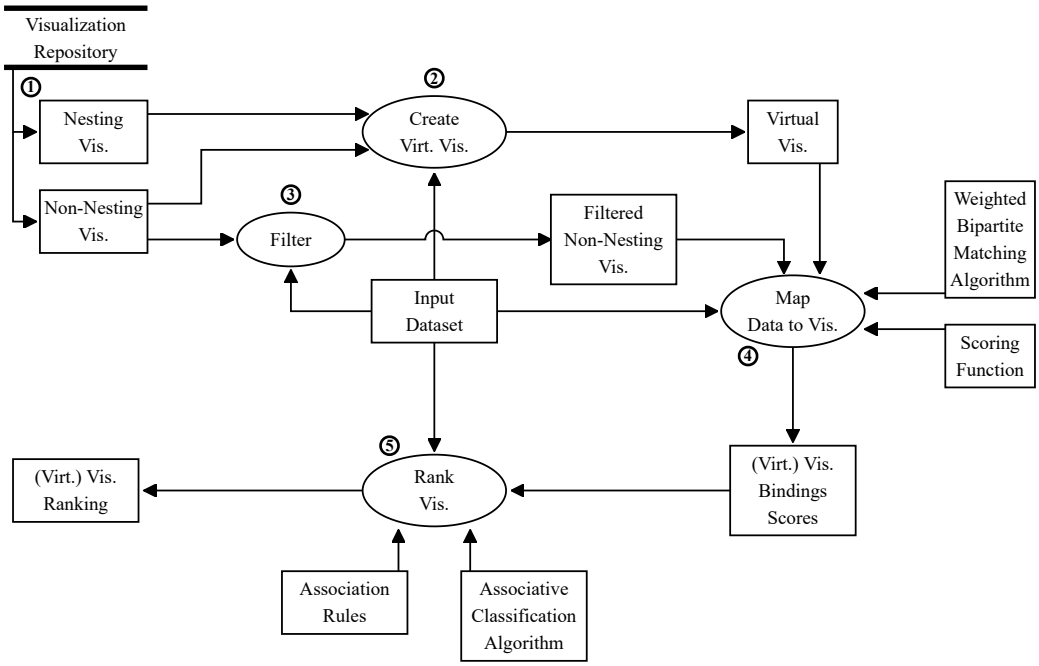


Figure 11.5: Yavaa: Schematic overview for visualization selection.

which columns can be omitted, it also provides recommendations for subsets of the data if this adds new options to the result list. This way, Yavaa allows users to adjust the visualization to their particular needs without necessarily having to restrict the dataset beforehand – contrary to most other approaches discussed here.

With regard to data characteristics used in the recommendation, Yavaa exploits almost all the criteria appearing in the discussed approaches. One exception are potential links across different columns which is only exploited by VizBoard to support graph-based visualizations. Yavaa's data descriptions (cf Chapter 8) would allow extracting possible relationships across columns. However, these relationships would subsequently need to be translated into weights of the bipartite graph that is at the core of Yavaa's recommendation. Determining possible connections and translating them into weights is assumed non-trivial and thus left for future work. Contrary to any of the discussed systems, Yavaa does not use fixed cardinality thresholds but relies on functions to represent the transition from suitable value ranges to those that are not.

For now, Yavaa does not include a user model to represent, e.g., the visualization preferences of a specific user. Such a model has been omitted in favor of a collaborative recommender. On the one hand, this bypasses the cold start problem [226] for new users when using a model based on past user interactions. On the other hand, user models explicitly provided by users themselves would constitute an additional step in the process and possibly another source of complexity – both aspects that Yavaa in general tries to avoid.

	APT [215]	ShowMe (Tableau) [126]	Articulate [217, 218]	VizBoard [140, 220, web115]	VizRec [137, 141]	VizAssist [127, web84]	Draco [228]	Yavaa
Single (s) or Multiple (m) Recom.	s	s	s	m	m	m	s	m
Ranking of Recom.	-	◐[a]	-	●	●	●	●	●
Adjustable Mappings	○	●	○	○	○	●	◐[b]	●
Subsets of Data	○	◐[c]	○	○	●	○	○	●
Data Types	●	●	●	●	●	●	●	●
Role	○	●	○	●	○	○	○	●
Cardinality	●	●	○	●	●	●	●	●
Links across Columns	○	○	○	●	○	○	○	○
Tasks / Objectives	○	○	●	●	○	●	●	●
Semantic Constraints	○	○	○	●	●	○	○	●
Collaborative Recom.	○	○	○	●	●	○	○	●
User Model	○	○	○	●	●	○	○	○
Optional Visual Artifacts	○	-	○	○	○	●	●	●
Nested Visualizations	○	○	○	○	○	○	○	●
Suitability Functions for Card.	○	○	-	○	○	○	○	●

[a] Only available to users in “Best Practice Mode”.
[b] Via partial specifications.
[c] Via user selection of a subset of the data.

Table 11.4: Comparison of visualization recommending systems.
●... supported; ◐... partially supported; ○... not supported; – ... not applicable

A final addition by Yavaa is the support for nested visualizations like arranging multiple scatter plots in a matrix. While in general supported by visualization tools like Tableau, it is not part of any recommender strategy including Tableau’s ShowMe. Here, Yavaa provides a generic mechanism to include more data dimensions into a single visualization and thus widens the range of possible suggestions.

CHAPTER 12

PROVENANCE MANAGEMENT

For any given data product — no matter whether it is a table or a visualization — questions may arise regarding its origins: Who created it? How was it produced? Which sources were used? What processing was applied? When was it created? The answers to all these questions require detailed documentation of the process that finally led to the given data product. This documentation is usually called data provenance and is defined by [243] as follows:

> The provenance of a computational result [...] refers to a complete record of the source of any raw data used, the computer programs or software packages employed, etc. The concept of provenance generally includes a record of changes that the dataset or software has undergone.

Having such provenance data readily available enables a wide range of applications [110, 244, 245, 246]. A prime use case is the ability to trace the origins of a particular data product. Propagating the trust [247] placed in the involved actors, data sources, and possibly operations to the final outcome allows making a statement about the believability of the latter [245]. Similarly, questions of quality or attribution can be answered [244]. On a different account, detailed documentation will also allow repeating the workflow to either validate past results or update the content based on more recent data [244, 246]. Finally, large collections of provenance data are a valuable resource for analysis. They document common usage of datasets and operations including the resources spent. This information can be used to assist other users [110, 246] or even predict the runtime characteristics of similar workflows before their actual execution [246].

This chapter discusses the efforts made to capture the provenance of datasets or visualizations created using the presented architecture. As data provenance has been studied in many different areas, first different approaches to and aspects of provenance will be described. After a discussion of this related work, the approach used in this thesis will be presented.

12.1 Related Work

In literature, provenance is oftentimes categorized along different dimensions. One such categorization is derived from the granularity of provenance information collected [248, 249]. In *data provenance*, the focus is on particular data items within larger datasets oftentimes within a database (cf. Subsection 12.1.1). This can either be done by propagating annotations during individual operations or by analyzing the operations themselves alongside their input and output data. In contrast, *process provenance*, also called *workflow provenance*, deals with the transformation of datasets as a whole during a workflow (cf. Subsection 12.1.2). This is oftentimes used in business applications [248] or scientific workflows [249] and is usually captured automatically by dedicated software(-components) [248].

A different classification of provenance is induced by the intended use. Here, one can distinguish between prospective and retrospective provenance [250, 251, 252]. *Prospective provenance* can be seen as a "recipe" that includes all necessary steps to create a certain data product. On the other hand, *retrospective provenance* collects data on how a specific data product was generated including factors like execution times and software or environment used.

For a better understanding, the different views of provenance, its collection, and preservation are separated in the following sections. Provenance in the context of existing workflow systems has already been discussed in Chapter 3 and will not be repeated here.

12.1.1 Database Provenance

Data provenance is oftentimes treated as provenance inside (relational) database systems. Data transformations are modeled by SQL-queries [253] or relational calculus. This leads to the following scenario as a basis for most considerations: At one point in time, the database consists of a number of tables. These tables have provenance records attached to them or some kind of description from which provenance records can computationally be deducted. Now an SQL-query is executed resulting in a new table. The problem statement for database provenance can now be phrased as follows: "What records have to be attached to the new table in order to sufficiently document its provenance?". This includes the issue of what information constitutes the provenance of the respective data, at which granularity it shall be documented, and how this provenance has to be stored.

There are three major approaches to the kind of data which has to be stored: why-provenance [254], where-provenance [254], and how-provenance [255]. **Why-provenance** represents all the input tuples which in some kind have contributed to the output. This can include the whole input table, though. Consider the query given in Listing 12.1. Here, through the `AVERAGE` function, one can argue that every record in the employee table contributed to the result of the query.

```
SELECT name, telephone
FROM   employee
WHERE  salary > (SELECT AVERAGE salary FROM employee)
```

Listing 12.1: Why-Provenance: Example SQL-query (from [254]).

Why-provenance can also be seen as witnesses or proofs for the inclusion of a specific record in the result set [256]. Depending on the query structure, otherwise equivalent queries can result in different why-provenance records. Especially when operators like `DISTINCT` or `UNION` come into play, the structure of the query plays an important role. Here, multiple identical records may be included in an intermediate result and combined into a single record in the final result. This final record could be witnessed by any one of these intermediate records and their respective sources which do not have to be identical or exclusive. An example is shown in Figure 12.1. Two identical Queries `Q` and `Q'` produce the same output. The provenance records shown to the right of each result record, however, differ substantially.

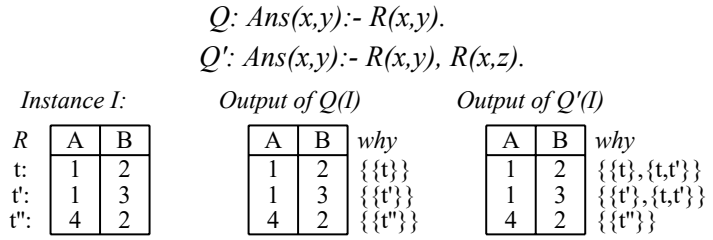


Figure 12.1: Why-Provenance: Different witnesses for same query result (from [256]).

To address this, [254] introduces the notion of a *minimal witness basis*. This denotes the set of all minimal witnesses. A witness is minimal if none of its proper subinstances is a witness itself. Furthermore, the paper showed that this minimal witness basis is invariant under equivalent queries with only equality conditions.

Another notion of provenance is **where-provenance**. Unlike why-provenance, the focus is on where each part of a result tuple came from. It provides a connection between individual cells in input and output tables. It relates to the study of the view update problem [257]: Upon an update on a database view, a system has to determine which parts of the raw data underlying have to be changed. This is especially hampered by the fact that there might be multiple such changes possible - all resulting in the same result on the view. Where-provenance, like why-provenance, depends on the structure of the query.

In [255] commutative semirings are used to represent several extensions to relational algebra. One of the considered extensions is why-provenance. It is shown that their model generalizes the notion of why-provenance and extends it to what the authors call **how-provenance**. They distinguish two distinct ways tuples can contribute to the result. They are either joined with other tuples or merged using a projection or union-operator. These two transformations are

associated with the two operands of the semiring, namely + for joins and · for merges. So, a term like $t1 + (t2 \cdot t3)$ indicates that first $t2$ and $t3$ were merged and afterward that result was joined with the tuple $t1$. That shows not only that the tuples $t1$ through $t3$ were involved in generating the resulting tuple, but also provides some insight into how they contributed.

Orthogonal to the problem of what data should be stored is the question of how it should be stored or extracted. In [256], the distinction is made between **eager** and **lazy** approaches. The difference between both of these approaches is the point in time when the provenance is computed. Eager approaches consume more storage space as they compute all the provenance entries no matter what. Lazy approaches, however, just restore the provenance of a specific entry upon request.

An eager approach computes the provenance of a tuple while it is created. This can be done by annotation propagation as demonstrated in DBNotes [258]. It is based on an extension to a subset of SQL, which is called pSQL [259]. Each part of every tuple can have annotations associated with it. When a query is run using these tuples, their annotations are propagated to the resulting tuples. To specify which annotations to propagate, a user can use an additional *PROPAGATE* clause as shown in Listing 12.2.

```
SELECT DISTINCT selectlist
FROM fromlist
WHERE wherelist
PROPAGATE DEFAULT | DEFAULT-ALL | r1.A1 TO B1, . . . , rn.An TO Bn
```

Listing 12.2: pSQL: Query syntax (from [259]).

There are three modes of annotation propagation. In *default* mode, only those annotations are propagated that are associated with the very field the data is copied from. Under this mode, both $Q1$ and $Q2$ of Listing 12.3 result in possibly different annotations being propagated. The result in $Q1$ yields the annotations of table R, while in $Q2$ table S propagates its annotations. When using *default-all*, however, both sets of annotations appear in the result. For that purpose. the system constructs a finite subset of all possible, equivalent queries and merges all the results. The last mode is called *custom propagation scheme* and allows users to specify which annotation gets propagated. This comes in handy when users only want to preserve annotations from one trusted source and neglect all other.

```
Q1: SELECT r.B FROM R r, S s WHERE r.B = s.B

Q2: SELECT s.B FROM R r, S s WHERE r.B = s.B
```

Listing 12.3: pSQL: Sample queries (from [259]).

Provenance is preserved in this system as soon as every field in every tuple is annotated with its own location. When a transformation occurs, these annotations get propagated along and it is easy to deduce which tuples contributed to a specific value in the result. Furthermore, using this information the data flow that leads to a specific result can be restored. Another use case mentioned in [259] is the preservation of error reports. If a specific entry in a curated database is marked incorrect or imprecise, this information is automatically assigned to all other tuples that get created using this entry.

The lazy approach is characterized by the fact, that the provenance is computed at the point in time when it is needed. In large datasets, the overhead is thus dramatically reduced. An example of lazy provenance computing is given in [260]. Here, the authors define the *weak inversion* of a function f as the function f^{-w} that attempts to map the results of f back to its inputs but is not guaranteed to be accurate. For some functions, an inverse function is defined, but in general, it may be hard or impossible to determine the correct inverse. For example, the square function ($f(x) = x^2$) for the result 9 yields two possible inputs $\{3, -3\}$. Extending the notion of a weak inversion function, two properties are defined. A complete, weak inversion contains no false negatives meaning, that all possible solutions are included. On the other hand, a pure, weak inversion contains no false positives in the sense that it generates only solutions that can result in the given output data[1]. A weak inversion, that is both pure and complete, is a regular inverse function. While weak inversion functions have no reference to the input tuples, their result may be refined using a *verification function*. These functions use the result of f^{-w} and the input tuples to further adjust the results.

The authors describe a database system where each function has at least one (weak) inverse function attached to it. There may, however, be multiple weak inverse functions attached to a single function. Using multiple inverse functions, the system is able to increase the accuracy of the presented result. If both inverse functions are complete, the intersection of the individual solutions yields a smaller, but still complete set. On the other hand, if both functions are pure, then the union of both results in a larger, pure set.

The premise in this paper is that the process flow resulting in a specific set of tuples is still available and attached to the result. Using this information and the defined inverse and verification functions, their system is able to compute the where-provenance of a data product in retrospect.

12.1.2 Script Provenance

Although plenty of scientific workflow management systems are available (cf. Sections 3.6 and 3.7), many scientists resort to the use of scripting languages like R [web105], Matlab [web104], or Python [web39]. Reasons may include a steep learning curve or the time-consuming need to wrap

[1]The definition is equivalent to saying that complete indicates that no less than all solutions are included, whereas pure declares that there are no more solutions included.

tools for use in a workflow management system if they are not geared towards it [261]. Others also argue that big "on-size-fits-all" solutions are not appropriate for use in highly dynamic and heterogeneous scientific domains [262].

Nevertheless, the need for documentation of provenance prevails in these environments as well. Multiple attempts have been made to target this situation. Examples can be found at [263, 264, 265, 266, 267]. Two relatively recent efforts to capture provenance information for the execution of scripts are noWorkflow[2] [261, 268, web118] and YesWorkflow [269, web119]. While noWorkflow monitors the execution of a script including the respective environment, YesWorkflow relies on annotations within the scripts. Both approaches complement each other as they capture different characteristics of script executions, so attempts have been made to combine them [270]. At the time of writing, these efforts were discontinued, though[3].

noWorkflow monitors the actual execution of a Python script in a way similar to a debugger. It requires users to execute their scripts not directly using Python, but by the use of the noWorkflow executable. By default, noWorkflow collects data about function activations, global variables, parameters, and return values. If a finer granularity level is required, noWorkflow can also track "variable attributions, loop definitions and other variable dependencies" [271].

When files are read from or written to, copies of these files are also retained. For this purpose, Python's `open` function is replaced, so that each time it is called two copies of the accessed file are made: one before executing the original `open` call and one after. That way, all changes to files are documented within the system. To reduce the amount of redundancy created by this approach, files are identified by a SHA1 hash [272] which is stored in a local database and is used to avoid duplicates within the stored files. This part of the provenance collection is called *execution provenance*.

Besides this, noWorkflow also captures *deployment provenance* and *definition provenance*. The former includes the execution environment and possibly existing module dependencies. Here, noWorkflow uses existing Python packages to capture information like the operating system, the hostname, the machine architecture, and the used Python version. Furthermore, using `modulefinder` it also detects the transitive closure of libraries used as well as their version and source code, if available.

Finally, by the name *definition provenance* noWorkflow stores information about function definitions and calls that are referred to within the script. This is done by analyzing the abstract syntax tree (AST) of the script and limits itself to user-defined functions. Functions of other packages, which also includes user created packages, are seen as atomic and are not analyzed in further detail. Calls to `open` are still intercepted no matter which function triggers the call.

The provenance information collected from all three parts is stored within a subfolder called `.noworkflow` within the script directory. To distinguish separate runs of the script, called *trials*, each trial is given a sequential identification number. By the use of the `now restore` command,

[2] not-only workflow.
[3] Statement by J.F. Pimentel and V. Braganholo in a meeting on 23rd February 2021.

users are able to revisit past trials and make changes similar to common version control systems. Making changes creates a new trial, which is linked to its parent. This way, dependencies of trials and their data products can also be established (cf. Figure 12.2).

For the analysis of the captured provenance, noWorkflow currently offers a text-based diff-functionality, a static dataflow view, and a web-based visualization component. The text-based diff provides basic comparisons like parameters used and duration of the execution. An example of the dataflow view is shown in Figure 12.3. Nodes with round corners represent data, white nodes represent files and dark blue rectangles represent function calls. Figure 12.4 shows an activation graph as collected by noWorkflow. The coloring of a node represents the duration of the respective function call from red (slow activation) to white (fast activation). Blue edges represent a sequence of calls whereas dashed arrows signal return values. Numbers at the edges show the number of activations.

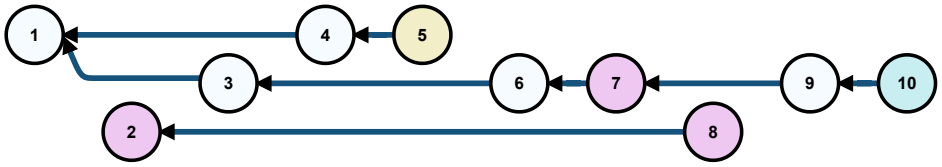


Figure 12.2: noWorkflow: trial history (from noWorkflow demo project [web118]).

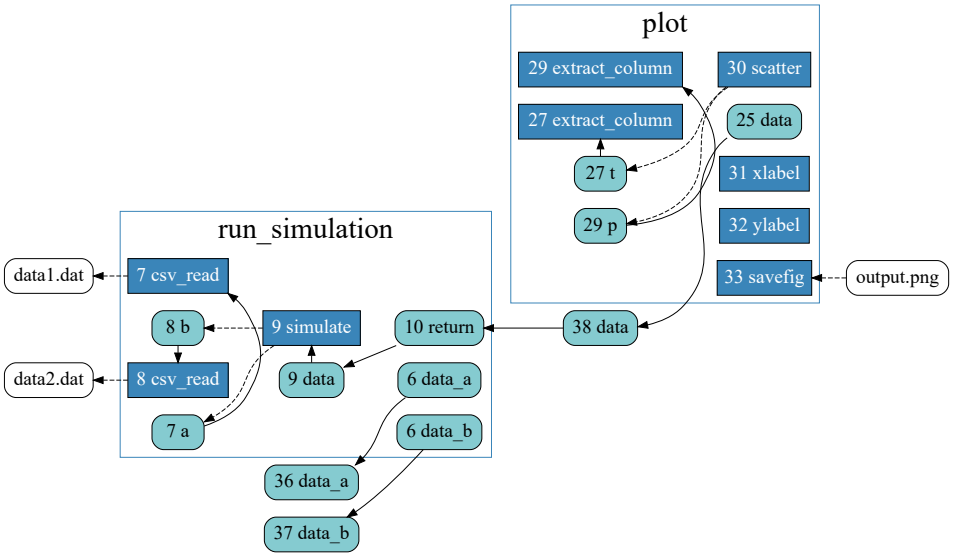


Figure 12.3: noWorkflow: dataflow (from noWorkflow demo project [web118]).

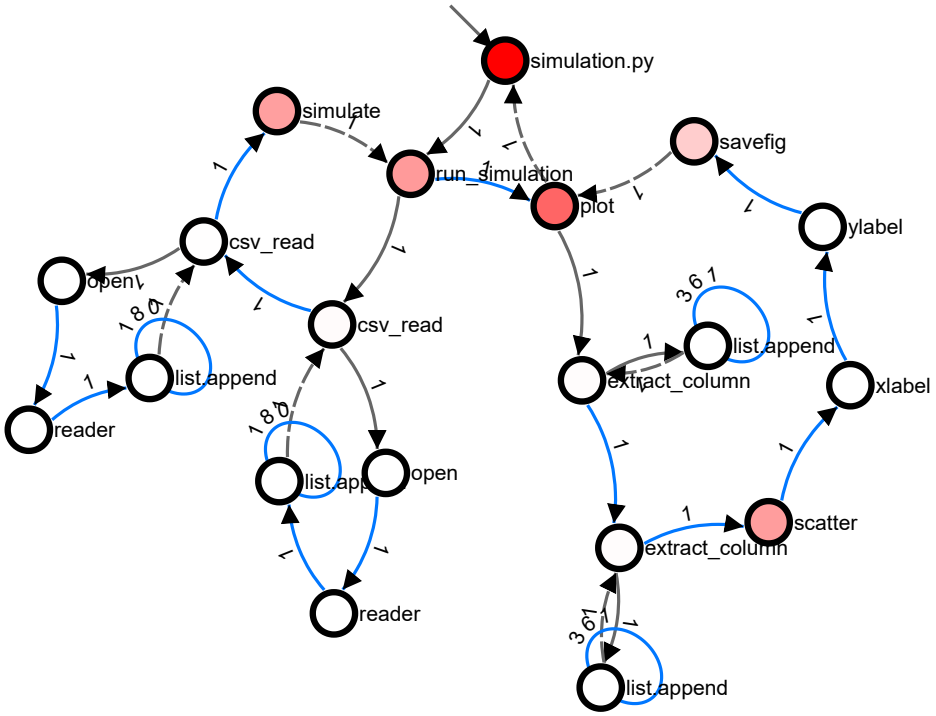


Figure 12.4: noWorkflow: details of trial execution (from noWorkflow demo project [web118]).

YesWorkflow, on the other hand, requires users to annotate their scripts similar to existing JavaDoc [web120] or JSDoc [web121] descriptions. Using these annotations, users are able to mark program blocks[4], ports, and channels. Ports describe the data flow in and out of the program block. Channels are the connections between the in and out ports of different program blocks. They are inferred by YesWorkflow using the names of ports within the same workflow.

An example of an annotated script can be found in Listing 12.4. In the actual analysis, YesWorkflow first extracts the relevant annotations from the comments. As of now, YesWorkflow supports eight languages including C, C++, and Java. The extracted data flow can be exported to a visual representation as shown in Figure 12.5. Parameters are represented using white boxes while data elements have yellow ones and program blocks are shown using green boxes. Figure 12.5 shows a combined data and process-oriented view. There is also the option to visualize just the process or just the data flow. Further querying of the provenance graph is currently not available but was announced as part of future work.

12.1.3 Documenting Provenance

PROV is a W3C [web122] effort to "achieve the vision of inter-operable interchange of provenance information in heterogeneous environments such as the Web" [web22, 273]. It consists of several documents [web22, web94, web123, web124, web125, web126, web127, web128, web129, web130,

[4]Program blocks denote any arbitrary computational step, possibly coinciding with function definitions in the script.

```
import netCDF4
import numpy as np
from netCDF4 import ma
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages

# @BEGIN main
# @PARAM db_pth
# @PARAM fmodel
# @IN input_mask_file  @URI file:{db_pth}/land_water_mask/LandWaterMask_Global_CRUNCEP.nc
# @IN input_data_file  @URI file:{db_pth}/NEE_first_year.nc
# @OUT result_NEE_pdf  @URI file:result_NEE.pdf

def main(db_pth = '.', fmodel = 'clm'):

    # @BEGIN fetch_mask
    # @PARAM db_pth
    # @IN g  @AS input_mask_file  @URI file:{db_pth}/land_water_mask/LandWaterMask_Global_CRUNCEP.nc
    # @OUT mask  @AS land_water_mask
    g = netCDF4.Dataset(db_pth+'/land_water_mask/LandWaterMask_Global_CRUNCEP.nc', 'r')
    mask = g.variables['land_water_mask']
    mask = mask[:].swapaxes(0,1)
    # @END fetch_mask

    # @BEGIN load_data
    # @PARAM db_pth
    # @IN input_data_file  @URI file:{db_pth}/NEE_first_year.nc
    # @OUT data  @AS NEE_data
    f = netCDF4.Dataset(db_pth+'/NEE_first_year.nc', 'r')
    data = f.variables['NEE']
    data = data[:]
    data = data.swapaxes(0,2)
    adj = 60*60*24*(365/12)*1000
    data = data*adj
    # @END load_data

    # @BEGIN standardize_with_mask
    # @IN data @AS NEE_data
    # @IN mask @AS land_water_mask
    # @OUT data @AS standardized_NEE_data
    native = data.mean(2)
    latShape = mask.shape[0]
    logShape = mask.shape[1]
    for x in range(latShape):
        for y in range(logShape):
            if mask[x,y] == 1 and ma.getmask(native[x,y]) == 1:
                for index in range(data.shape[2]):
                    data[x,y,index] = 0
    # @END standardize_with_mask

    # @BEGIN simple_diagnose
    # @PARAM fmodel
    # @IN data @AS standardized_NEE_data
    # @OUT pp  @AS result_NEE_pdf  @URI file:result_NEE.pdf
    plt.imshow(np.mean(data,2))
    plt.xlabel("Mean 1982-2010 NEE [gC/m2/mon]")
    plt.title(fmodel + ":BG1")
    pp = PdfPages('result_NEE.pdf')
    pp.savefig()
    pp.close()
    # @END simple_diagnose

# @END main
```

Listing 12.4: YesWorkflow: example script (from [web119]).

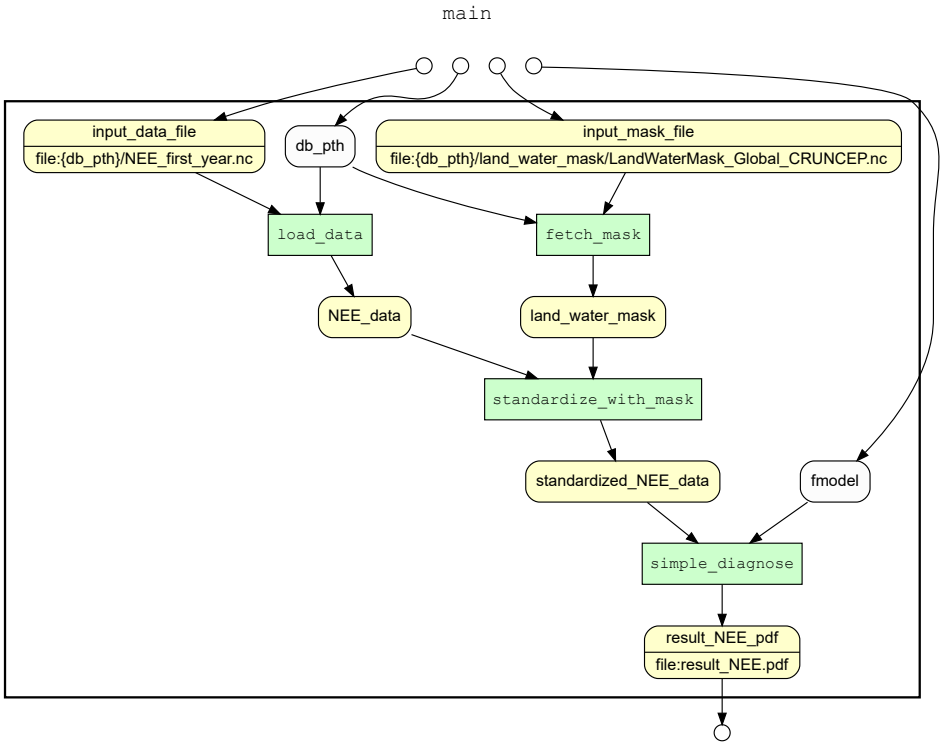


Figure 12.5: YesWorkflow: dataflow of the YesWorkflow example script [web119].

web131, web132, web133] at various levels of the W3C standards track [web134][5]. The effort is based on earlier activities that emerged from the provenance challenge series [275, web135, 276, web136] and that had resulted in the Open Provenance Model (OPM) [106].

The PROV data model [web129] itself is agnostic of a specific implementation. It uses a human-readable representation given by PROV-N [web133]. Furthermore, the standard defines an OWL2 [web86] serialization (PROV-O [web94]) as well as an XML one (PROV-XML [web137]). Finally, there is a member submission for a JSON serialization [web131] which as of now is not part of the official family of documents [web22]. It is stated that the conversion between several serializations is not subject of the PROV working group [274]. Accordingly, a round trip of conversions between different serializations does not necessarily result in an equivalent representation. Other parts of the PROV document family are concerned with setting constraints on the validity of PROV instances [web125], mechanisms to access provenance information [web126], a particular dictionary data structure [web128], a mapping to Dublin Core [web89] terms [web127], a specification of PROV in terms of first-order-logic [web124], and the option to link between different parts of a provenance graph [web132].

[5]Some of the documents are deliberately maintained as “group notes” for reasons of less strict rules to the editing process [274].

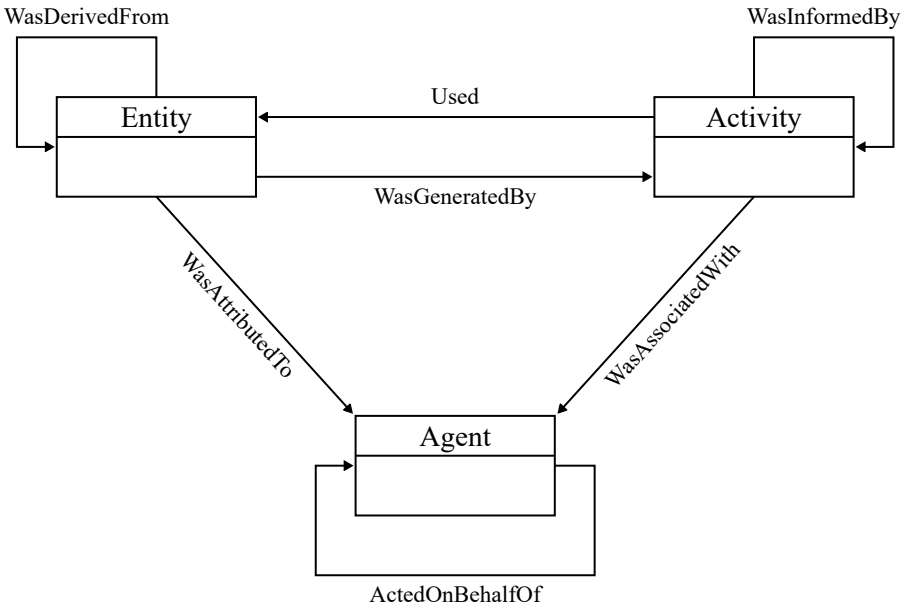


Figure 12.6: PROV: Concepts and relations (after [web129]).

PROV models provenance as a directed graph with three types of nodes and seven types of edges. The nodes can be of the following types [web129]:

Entity . "A physical, digital, conceptual, or other kind of thing with some fixed aspects; entities may be real or imaginary."

Activity . "Something that occurs over a period of time and acts upon or with entities; it may include consuming, processing, transforming, modifying, relocating, using, or generating entities."

Agent . "Something that bears some form of responsibility for an activity taking place, for the existence of an entity, or for another agent's activity."

These node types are not necessarily mutually exclusive. In particular, PROV allows modeling the provenance of agents which in turn makes them the entity of their own provenance graph. Similarly, agents may adopt the role of an activity. The use of an activity as an entity and vice verse, on the other hand, is prohibited by [web125]. While PROV adopts a fairly flexible approach here, applications are allowed to make an explicit distinction between all three classes.

Similar to the nodes, their connections are also classified into distinct types. All relations defined in the data model point towards the past. While this is a requirement for the data model, it is just a convention and recommendation within PROV-O. An overview of the defined properties as well as their domains and ranges is given in Figure 12.6.

Each component of a PROV model has the option to include a number of attributes. PROV provides five built-in attributes. All of these attributes are optional in their use, but some of them might be restricted to specific classes or relations. In general, however, an application can define its own attributes, which are not bound to the predefined ones.

prov:label a human readable representation.

prov:location either a geographical place or a non-geographic location like a specific row.

prov:role the role of an entity or agent within an activity.

prov:type further typing information available for all classes and relations.

prov:value the actual value of an entity.

Activities occur over a specific period of time; they have a start and end time which can be attached to the respective object. All relations as well as the life-cycle events for entities and activities are, on the other hand, regarded instantaneous. While the activity that creates an entity can take some time, the actual generation[6] of the entity happens in an instant. The notion, in that case, is that the timestamp represents the completion of the respective activity with regard to that entity[7]. In consequence, all relations have just a single timestamp associated with them.

The only relation for which PROV does require a strict ordering is derivation. This, however, might result in cycles for other relations. The only consensus reached by the working group [274] was to consider all events along such a cycle as simultaneous, but allow applications to impose stricter rules.

To adapt PROV to domain-specific use cases, three different ways are suggested [web129]. Applications can make use of sub-types and sub-relations of their predefined counterparts. Within the data model, this is represented by the use of `prov:type` within the attribute set of the respective individual or relation. Similarly, `prov:role` can be used to assign specific roles to entities, agents, or activities within a relation. Finally, applications are allowed to add arbitrary information to the attribute lists. The implementation of those extension points is different between the serialization options. While PROV-O [web94], e.g., uses `rdf:type` to represent `prov:type` attributes, PROV-XML [web137] and PROV-JSON [web131] choose to follow the data model more closely and include `prov:type` itself.

[6] The event associated with `prov:wasGeneratedBy`.
[7] A single activity can generate multiple entities which might be completed at different points in time.

Figure 12.7 shows a visual representation of a provenance graph as defined in [web129]. The visual elements follow the layout conventions[8] of [web138]. These conventions suggest the use of yellow ovals for entities, blue rectangles for activities, and orange pentagons for agents. Furthermore, preceding events should be positioned towards the top (vertical layout) or to the left (horizontal layouts). As already stated, relations point towards past events.

The example itself represents the provenance of a document `WD-prov-dm-20111215`. The document was created by an activity `edit1` of type `editing`, which led to version two of the document. Two agents took part in the editing activity: `Simon` was associated as a `contributor`, whereas `Paolo` was an `editor`. No further information is given, so in particular, any timestamps are missing here.

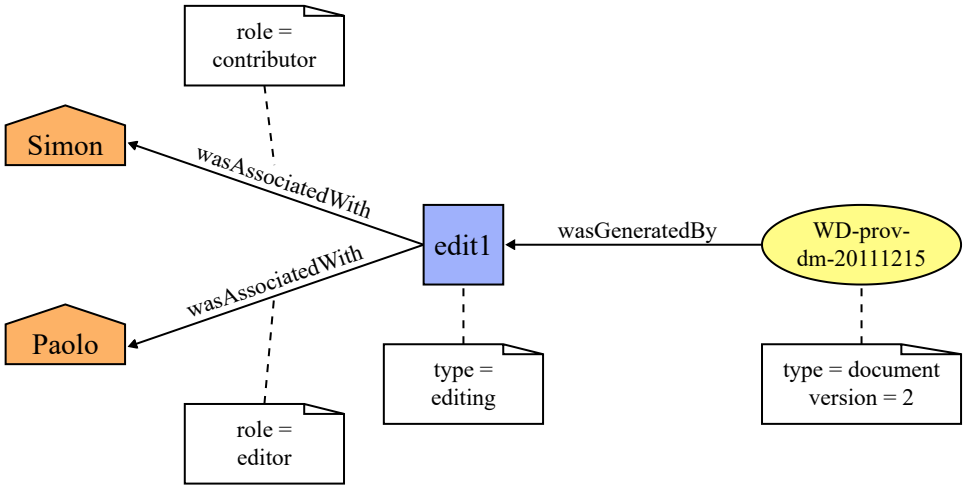


Figure 12.7: PROV: Example provenance graph (from [web129]).

Besides the mentioned basic components, there are others defined within the standard. They are usually defined as subtyping of the base components and account for more specific use cases. One example are plans which allow users to represent a set of actions to achieve some goal. As this set can change over time, a plan is an entity as per aforementioned definition and can itself be subject to a provenance record. Another extension are collections. They allow the grouping of different entities to maintain common provenance information for all of its constituents. One particular type of collection that attracted special attention are dictionaries [web128]. They represent a list of key-entity pairs also known as maps. The document on dictionaries also notes that the full content of a dictionary is regarded unknown unless its history can be traced back to an empty dictionary via a series of insertion and removal activities.

[8]These are recommendations and are not normative [274, web138].

As noted before, PROV encourages domain-specific extensions. **ProvONE** [277, web139] is one such extension that is targeted specifically towards scientific workflows. It aims to be the unifying provenance standard across different scientific workflow management systems like the aforementioned Taverna and VisTrails (cf. Sections 3.6 and 3.7). An overview of the ProvONE data model is given in Figure 12.8.

The model is separated into three parts: workflow representation (blue), trace representation (brown), and data structure representation (purple). Further, ProvONE mentions a workflow evolution representation that does not have distinct classes but is given by the relation `prov:wasDerivedFrom`. This relation links different versions of the same workflow, thus forming a derivation tree for its history.

The workflow representation captures the prospective part of the definition. At its heart is the `Program` that captures a computational task and may represent a `Workflow` of its own. The inputs and outputs of such a `Program` are given by different `Ports`. For input `Ports` that are not connected to the outputs of other `Programs`, there is the option to define default values using `hasDefaultParam` and a corresponding entity, usually of type `Data`. The actual data flow across the workflow is modeled by instances of `Channel` that allows connecting the ports of involved `Programs`. Finally, the `Controller` class allows specifying that the execution of one `Program` is controlled by another, thus allowing to define different models of computation (e.g., synchronous vs asynchronous execution models).

The data structure representation extends PROV's `Entity` with three classes: `Visualization`, `Data`, and `Document`. These types are supposed to represent the different entities exchanged within a workflow. Here, `Visualization` seems to represent all kinds of media files, as the standard lists MP4 as an example among multiple image formats. `Document` is defined as any kind of "article or report that was created as a result" of a workflow execution. The `Data` type is a general placeholder for all kinds of data (apart from the aforementioned media files) covering XML structures, tables, and individual values as, e.g., used as default parameters to `Programs`. Multiple instances of different types can further be subsumed by a `Collection` that may "represent a set, bag, list or another variant of a group of items".

Finally, the trace representation covers the retrospective provenance of particular `Executions`. It links to a specific `Program` or `Workflow` and may involve a responsible `User`. The artifacts used and created throughout this execution are attached via instances of `Usage` and `Generation`. Each involved `Entity` is linked to the input or output port of the respective `Program`.

12.2 Discussion

The three presented areas of provenance research cover different parts of the provenance life cycle. Subsection 12.1.1 is concerned with the usage of provenance information, Subsection 12.1.2 covers its collection, and, finally, Subsection 12.1.3 deals with its storage.

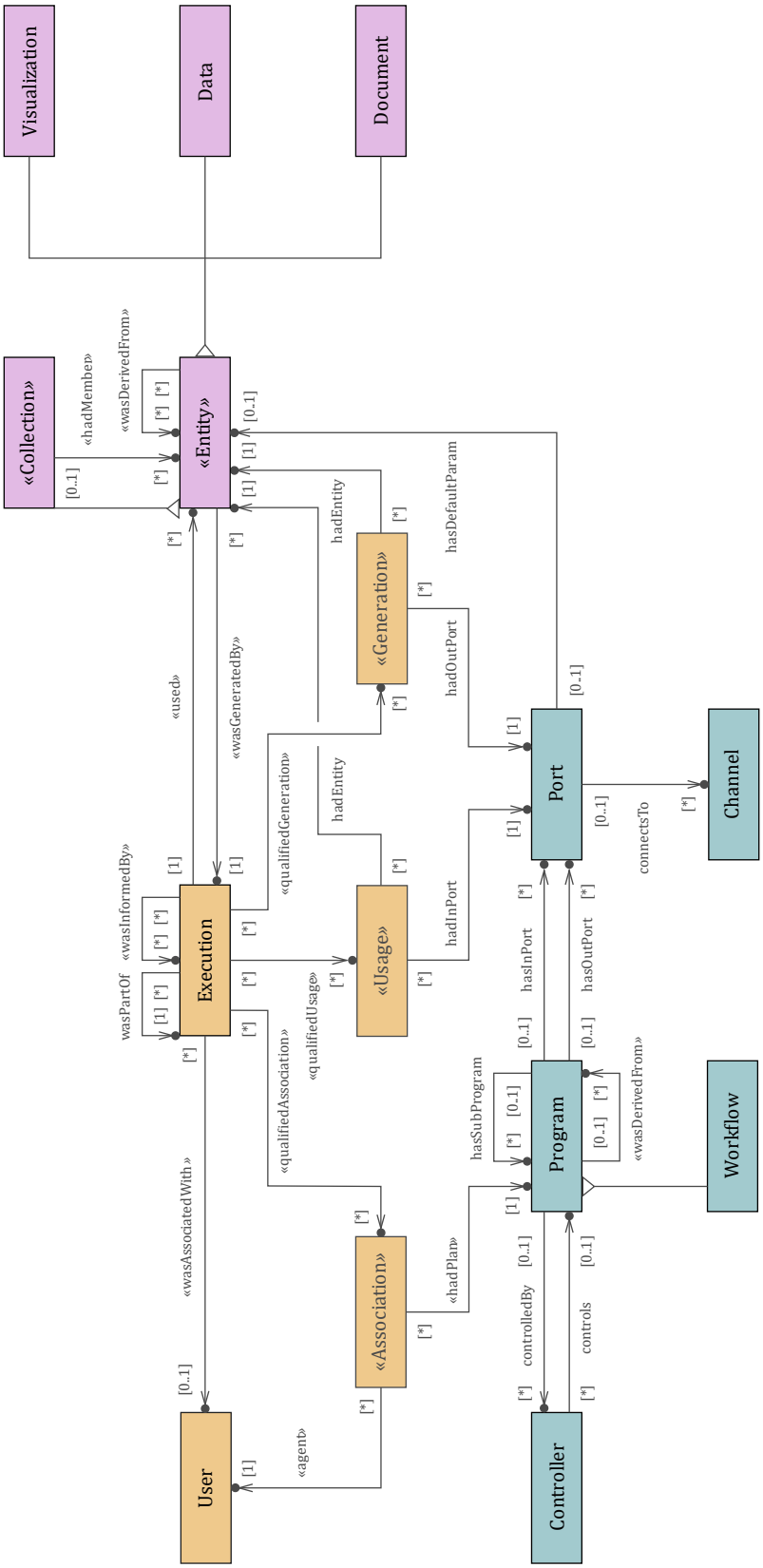


Figure 12.8: ProvONE datamodel (from [web139]).

The collection of provenance information becomes relevant when using custom scripts instead of workflow systems. Taverna (cf. Section 3.6) and VisTrails (cf. Section 3.7) collect provenance information in their execution phase. One difference between the presented collection methods is the handling of different runs. The question is whether provenance data is to be stored only for the final run or for all runs before as well. Taverna chooses to store provenance information only for the current run, whereas VisTrails and noWorkflow maintain it for all executions of the workflow. The latter also include any intermediate (persisted) results as opposed to just saving the final outcome. This allows these systems to also answer questions related to the workflow evolution but comes at the cost of additional storage requirements.

It remains doubtful whether gains outweigh the costs in data-intensive and, thus, storage-demanding scenarios. Workflows are often created by a trial and error approach including both a variation in parameters as well as changes in the actual workflow. Keeping all (intermediate) results of all unsuccessful attempts will just occupy storage without much benefit for analysis. If such an analysis of the workflow evolution is to be made, versioning of the workflow and an exhaustive description of the execution environment should suffice to answer most provenance-related questions. As the input datasets oftentimes remain the same, preserving them would allow executing past workflows again and regain access to those past results. Especially when the datasets grow in size, the storage of workflow descriptions rather than all workflow results is less demanding. These considerations only apply to workflows where the amount of data excels the amount of time to create it. If, however, the time it takes to generate a specific dataset exceeds a certain threshold or the results of different runs are expected to all be used, storing intermediate as well as final results for all runs may be useful and in fact save time and space overall.

Another aspect that differs between the presented approaches is the granularity in which provenance data is kept. The two extremes are keeping the information for each tuple within a table separately as opposed to just tracking the dataset as a whole. Workflow systems and script-based provenance tracking as presented has decided for the latter, whereas database provenance has a more detailed view. The decision within workflow systems can be argued for by the fact that here not only tabular information can be processed, but any data format can be used as long as the respective implementation allows for it. So the concept of a tuple can not be maintained for all datasets and, hence, is dropped.

PROV as a means to store provenance information is in general able to handle both kinds of granularity. While it is possible to store provenance information on a tuple level and aggregate them using a collection entity, it is also possible to treat complete datasets as atomic entities. However, considerations for gains versus costs are again relevant. If provenance is to be stored with very fine granularity, lots of provenance information is generated. In fact, all intermediate results will be stored along with the general overhead as introduced by PROV. If there is no need to retrace every tuple's detailed history or if the workflow leading to it can easily be executed again, storing provenance information on a tuple basis seems overly precise.

12.3 Approach

A first decision when tracking provenance is that of granularity of objects whose provenance is to be collected. Two obvious options are to follow the concept of database provenance and describe individual tuples or use the approach of other workflow systems and track datasets as a whole. The former leads to an extensive amount of provenance data likely to exceed the size of the final result. The latter, however, provides less insight into the process overall.

The other components of the proposed architecture work on a column level. Visualization, e.g., will map columns to visual components (cf. Chapter 11). As a consequence, it seems natural to also use columns as the central entity for which provenance is tracked. This allows users to determine the sources which contributed to a specific property of the final visualization like the color of artifacts but also limits the amount of information to store.

The next question deals with the representation of the collected provenance data. Here, W3C's PROV [web123] seems to have gained a lot of attention. Many workflow systems adopted PROV at least as one way to persist their provenance information (cf. Sections 3.6 and 3.7). Due to its extendible nature, PROV can also serve as the basis to store all workflow-related information. Hence, for now, the architecture presented will use only PROV as the main representation of workflows. Besides preserving provenance information about past runs, this also includes the option to reenact a workflow using new data.

A particular workflow-extension like ProvONE seems excessive, though. While it can cater to the workflows of pretty much any workflow system, its level of detail is not required here. Contrary to generic workflow systems (cf. Sections 3.6 and 3.7), the workflows created within Yavaa are rather simple and well structured. At this point in time, it seems doubtful that using the overhead of a generic representation like ProvOne would yield any specific benefit for the system. With ProvOne and the chosen approach both being based on PROV, a future mapping seems possible and comparatively low effort now. So for now, ProvONE will be foregone in favor of a custom model tailored to the provenance of workflows for tabular data.

Having decided upon the provenance representation, the next step is to identify and map all involved components of the workflow to their respective counterparts within the PROV formalism. Each (non-empty) workflow within the proposed architecture will start with the loading of a dataset. This process will result in more than three objects within the provenance documentation: The source location of the dataset is an entity, the loading procedure is an activity, and the resulting in-memory dataset is again an entity. As the provenance is tracked on a column level, the in-memory dataset is of type `prov:Collection` and consists of one entity for each column.

Additional information to uniquely identify the source dataset is attached by the use of attributes to both source entity and load activity. While the source entity contains attributes that specify the location and publisher of the dataset, the time the dataset was loaded is stored in an

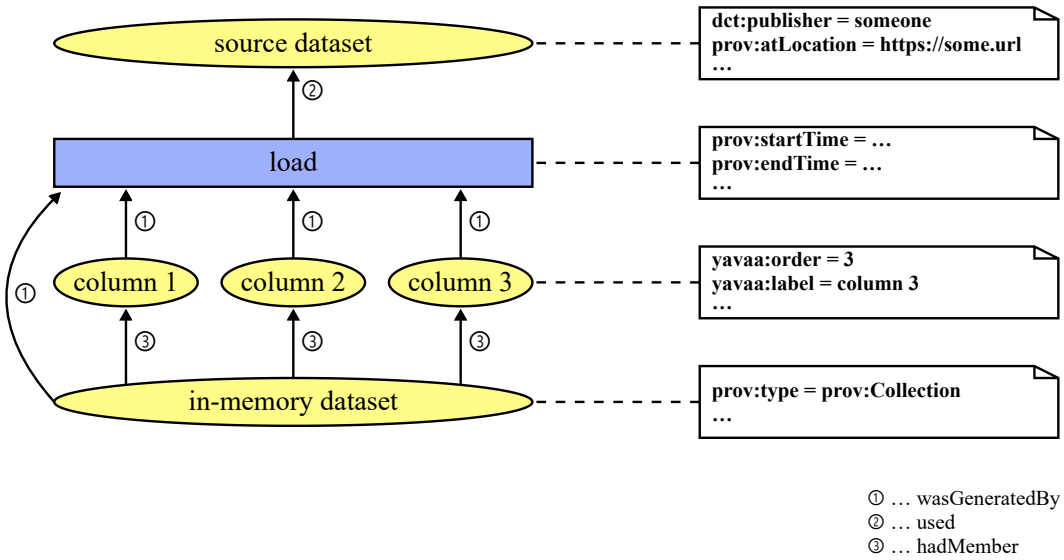


Figure 12.9: Yavaa: Schematic example for provenance graph of a load activity. Attribute lists for column 1 and column 2 omitted.

attribute of the load activity. This split of identifying information seems counterintuitive at first but is more consistent with later activities, which also store their respective execution times. To reduce the redundancy, the time information of the source entity is hence omitted.

Figure 12.9 shows a schematic example for a load activity and the involved entities. The in-memory dataset has no attributes but its type. Similarly, the columns' attribute lists contain only a few items. Attributes in the `yavaa` namespace are used for workflow reenactment. The `yavaa:order`, e.g., describes the position of the respective column within the array of all columns of that dataset and is used as an identifier within the system.

After a dataset is loaded, a series of operations can be applied. Each operation is represented by an activity within the provenance graph. Similar to the load activity, each other activity will result in a set of new columns. The resulting dataset does not necessarily consist only of such new columns, but may also include a subset of the input columns for that activity. Those columns have not been affected by the activity in question.

Figure 12.10 shows a schematic example for the provenance graph of such a computational activity. Here, `column 3a` was transformed in a computation that resulted in `column 3b` and used `column 2` as an additional input. Besides the relations used in the load activity, the changed column has references to other columns. The column which was the direct source for the computation is referenced using `wasDerivedFrom` ④, whereas other columns that influenced the computation are linked by `wasInfluencedBy` ⑤. The attribute lists have been omitted. In this example, they contain the start and end time for the activity as well as the order number for all columns in the same way as shown in Figure 12.9.

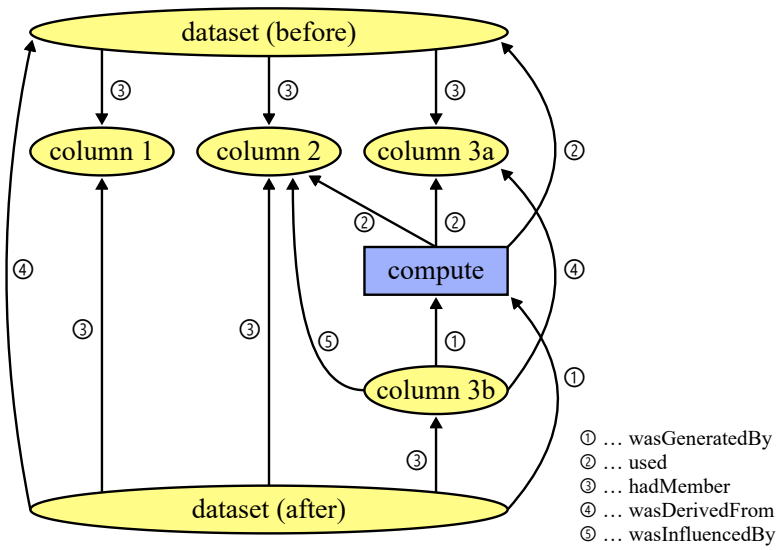


Figure 12.10: Yavaa: Schematic example for provenance graph of a computational activity. Attribute lists omitted.

The final visualization step is treated like other computational steps with just minor differences: First and foremost, the resulting visualization is considered final, i.e. it will not be used by another activity within the proposed architecture. The visualization consists of components instead of columns. There is, however, a close relationship between both (cf. Chapter 11). This is represented by a `wasDerivedFrom` edge ④ that links each component to the column(s)[9] which provided the respective values. A schematic example for a visualization activity is given in Figure 12.11.

In addition to the already described attributes, for each activity, some technical information is stored. In particular, this includes the command and parameters that triggered the respective activity. Commands used within the prototype and their description are given in Appendix D. This information is primarily used to reenact the workflow, but also includes more details on the activities performed. These attributes have no equivalent within the PROV standard and, hence, reside within a `yavaa` namespace.

For the display within the prototype, some adaptations have been made which are described in Section 13.5. These adaptations remove redundant information from the displayed graph. They are in line with the inference rules and constraints as given in [web125].

[9] One component can be bound to multiple columns as per definitions of Chapters 7 and 11.

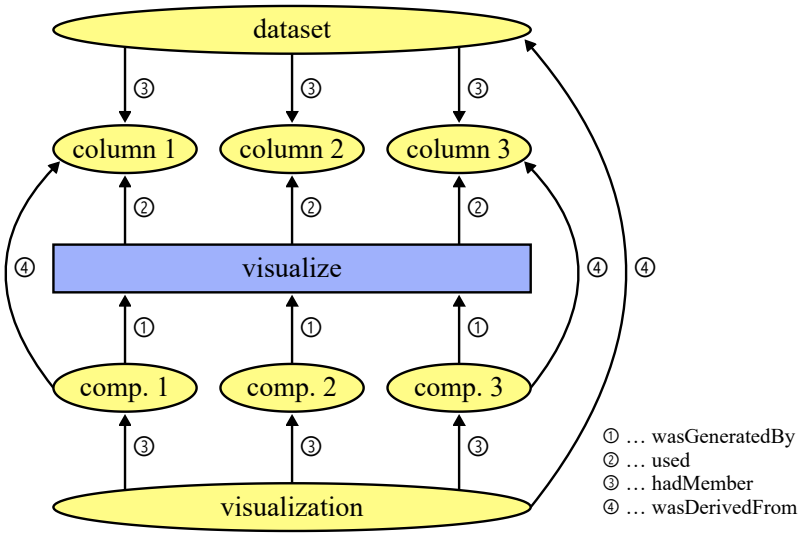


Figure 12.11: Yavaa: Schematic example for provenance graph of a visualization activity. Attribute lists omitted. A `wasDerivedFrom` edge from component to column 2 is also hidden for clearer presentation.

Part III

Genesis & Analysis

CHAPTER 13

IMPLEMENTATION

Previously, concepts necessary for an integrated data workflow were presented. To evaluate their validity and compare against other approaches, they were implemented in a proof-of-concept prototype called Yavaa. This chapter will focus on the technical aspects of said implementation. It will justify the choices made and describe its overall structure.

All concepts presented before have been implemented as part of the prototype. However, there is one restriction: The data types as described in Chapter 5 are only used within their first level (categorical vs. time vs. quantitative). Second-order classifications have so far been omitted and are left open for future implementation. The reason for this lies in the automatic creation of dataset descriptions which will be discussed subsequently in Section 14.1. Distinguishing between the second level types is non-trivial and considered out of scope for this work.

13.1 Architecture

An overview of Yavaa's architecture is shown in Figure 13.1. It consists of two separate components, user interface, and worker, connected by a communication layer[1]. The sole purpose of the user interface is to provide human access to the worker and present its results. All other tasks, including data processing and repository lookups, are performed from within the worker. Datasets accessible via Yavaa are not part of the application itself. Instead, they remain exclusively at the respective provider and are only accessed on demand by using proper wrappers.

The two components may either be run in a single environment, i.e. a browser, or be split up to run at different locations. The latter allows for thin clients where the browser only serves as a user interface, whereas the computations are actually run on dedicated servers. The user

[1] Details on the messages exchanged are given in Section 13.6.

interface is implemented as a single page application using HTML5 [web140], CSS [web141], and JavaScript [web142]. The worker is also implemented using JavaScript. This allows reusing the very same code in both modes of execution either in users' browsers using a WebWorker [web143] or on a server using a Node.js [web144] environment. In both cases, the communication between the user interface and the worker is abstracted into a separate layer. This layer hides environment details and exposes a common interface.

Inside the worker module, there are three kinds of components. *Repositories* hold dynamic data. This data is considered dynamic as new additions do not affect the overall architecture and can be made without understanding the concepts underlying other parts. This includes the description of datasets and visualizations, the datasets that are currently used, and code fragments, e.g., aggregation functions or the serializations of datasets. According to their contents, the repositories use different techniques to keep their data: While the code repositories basically just organize code fragments and their JSON-descriptions, dataset descriptions and unit data are stored in an RDF triple store. Finally, the data store keeps already downloaded data using in-memory structures which will be discussed in detail in Section 13.2. As mentioned before, Yavaa does not include any means to maintain datasets long term but reuses existing capabilities by external dataset providers.

Stores provide the access to the repositories. For RDF-based repositories, this means issuing the respective SPARQL-queries [web88] and converting the result to a standardized format. Other descriptions like the ones for visualizations are parsed into in-memory structures that can easily be accessed and queried. Code-fragments are interpreted or compiled[2] and returned to the caller as executable objects. The data and workflow store use the same repository but expose different operations based upon the datasets contained. The data store provides access to the primary data of a dataset and thus forms the basis for all computations applied. On the other hand, the workflow store allows to access the provenance record of a dataset. Each version of a dataset is annotated with the operation and source(s) that lead to its creation. By traversing these links, the workflow store provides access to the full history that resulted in a specific dataset version. Unless explicitly requested by users, all data is kept in memory. As such, it will be lost once the session ends. More details on the internal data handling will be given in Section 13.2.

Finally, *Functional Blocks* include most of the application logic. Their execution is triggered via the user interface. During their operation, they mostly rely on the underlying stores for fetching datasets and storing them after modification. The only exception is the access of remote data sources which does not rely on a separate store object but accesses the remote datasets directly. However, both, the necessary wrapper as well as the resulting (internal) dataset, are managed via the respective stores again.

[2]Details depend on the execution environment which might use different techniques to execute the JavaScript code.

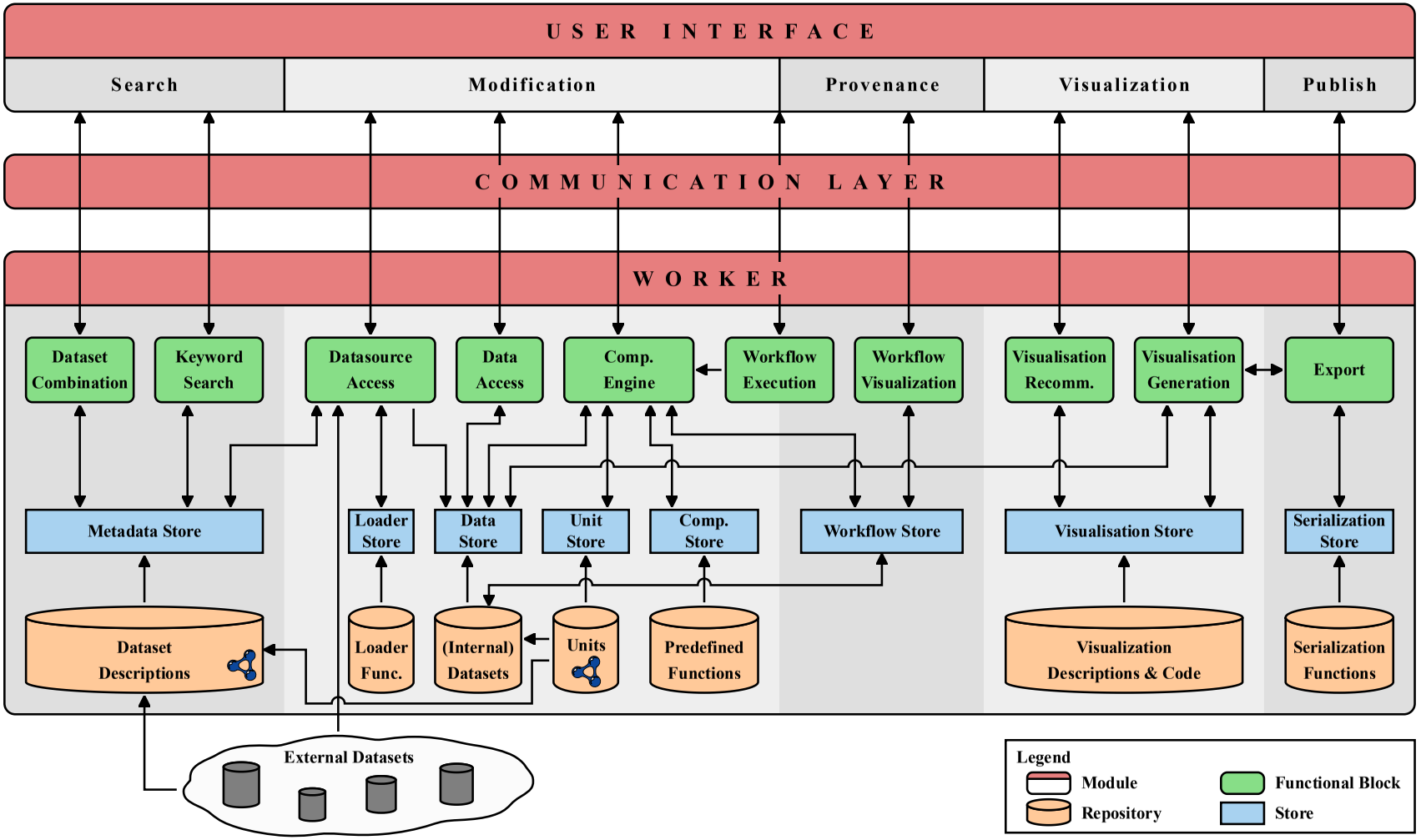


Figure 13.1: Yavaa: Architecture Overview.
Horizontal assignment by main purpose, individual parts might be reused for other tasks as well, though. Connections from `Export` to `Data Store` and `Workflow Store` omitted.

In addition to this separation into layers, the individual components can be classified by the tasks they refer to. These are highlighted by the vertical sections of Figure 13.1 and follow the steps identified in Chapter 1. In general, each task is represented in both the worker as well as in the user interface. In the remainder of this section, a rather high-level description of the provided functionalities will be provided. The following sections elaborate on individual components as well as their counterparts in the user interface.

Search allows users to identify the datasets they need to accomplish their goals. One option is a keyword-based search that relies on the datasets' titles. The other option is to describe a dataset structure for which the system will attempt to create a corresponding workflow following the concepts described in Chapter 10. In both cases, there is not yet direct interaction with the datasets themselves. They only rely on the dataset descriptions outlined in Chapter 8 and are stored within the *Dataset Descriptions Repository*. The worker will return a dataset identifier or a workflow that can be further refined by users. Once users agree with the choice made, they trigger the workflow execution and/or retrieval of datasets.

The *Modification* section allows users to load a dataset and transform it according to their needs. The most important functional block is the *Computation Engine*. After a dataset has been loaded, all other operations will be performed through this engine. It will apply the given function in a row-wise fashion to all rows of the dataset and store the result as a new dataset. As noted before, this dataset includes an annotation that identifies the operation and sources used. To ensure consistency with regard to units, the engine relies on the *Units Store* to provide the required conversion functions and may modify the submitted operations to ensure the results stay consistent.

The *Datasource Access* component, upon request, fetches the dataset's description to determine its location and the corresponding loader to use. It then retrieves the dataset from said location, parses it using the loader, and stores the result in the *Data Store*. The *Data Access* component allows other components or the user interface to access the primary data of a dataset or parts thereof. Finally, *Workflow Execution* can parse a given workflow into executable commands which are then passed to the computation engine to materialize the described dataset.

The main objective of the *Provenance* section is to provide other sections and users access to the provenance of a given dataset. Users are able to view a graph representation of their workflows. Details of this representation will be provided in Section 13.5. As previously mentioned, all operations on datasets are performed through the *Computation Engine*, which will ensure that the respective annotations are present for each (intermediate) dataset.

The *Visualization* section features two functionalities. On the one hand, it implements the *Visualization Recommender* presented in Chapter 11. On the other hand, it facilitates the creation of a selected visualization. The bindings as provided by the recommender and adapted by users

are passed alongside the primary data to a code fragment that creates the respective visualization. The result is then passed back to the caller to either be downloaded or rendered via the user interface.

The last section is *Publish*. It provides download capabilities for the results of a given workflow. This includes the resulting dataset, the visualization, and the respective workflow. Here, different serialization-formats for each aspect are possible and are stored in the *Serialization Store*. The overview in Figure 13.1 omits the connections from the *Export* component to the *Data* and *Workflow Store* for the sake of clarity in the display. Within the implementation, however, the export component uses both these stores as well as the *Visualization Generation* component to retrieve the requested data.

13.2 Data Store

Each operation of the Computation Engine and the Datasource Access component creates a new internal dataset. It is referenced by a numeric identifier that is unique throughout a particular user's session and is used to identify the dataset an issued operation is supposed to work on. This way, multiple concurrent datasets can be handled within a single session which especially becomes important in case multiple datasets are to be integrated (cf. Subsection 13.3.3).

Besides the identifier, these datasets are comprised of three parts: the dataset's label or title, the workflow entry associated with this dataset, and the actual primary data. The title is used to reference the dataset within the user interface as a header and in other circumstances that require a reference to a specific dataset to be shown to users. Beyond that, the label is not used within the system.

As already mentioned, each dataset is annotated with a workflow entry describing its creation process. In particular, this includes the command send to trigger that operation on the worker including all parameters (cf. Section 13.6 on the messages used). Part of these parameters are the dataset-ids to use as sources of the operation. They are maintained in the workflow record as a pointer to the source datasets to establish a chain of entries and thus the complete provenance for a given dataset. One exception is loading datasets from external providers. Here, no pointer to another dataset within the system is available. However, the description of the operation itself will contain an identifier from Yavaa's dataset descriptions, the URL the dataset was fetched from, and the specific format that was used.

For more detailed provenance tracking, a similar process is applied to each column separately. Each column in a dataset falls into one of three categories: It remained unchanged from the previous dataset version, it was adapted from a column of the previous dataset version, or it was newly-created, e.g., as part of loading a dataset or the result of applying a formula. Unless it was adopted unchanged, other columns might have contributed to the change or creation. For example, if a new column represents a population density it might have been computed of two

other columns present, namely population and area. In total, this leads to three properties for each column: a flag to signal whether it was changed or not, a nullable pointer to its equivalent in a previous dataset version, and a nullable list of pointers to other columns that contributed to its modification or creation. Using these three properties, all three provenance options can be modeled as shown in Table 13.1. Furthermore, this contains all information required to create and visualize the provenance graph as discussed in Chapter 12 and Section 13.5.

Column status	Changed Flag	Previous Representation	Contributing Columns
unchanged	false	set	null
changed	true	set	set
new	true	null	set

Table 13.1: Yavaa: Workflow annotation properties for a single column.

The final component of the internal dataset representation is the actual primary data. Here, two approaches are available in principle: row-oriented or column-oriented. Column-oriented databases have been found to outperform row-oriented ones in analytic workloads [278]. However, this advantage is attributed to a large extent to differences in I/O where column-oriented databases have to read less from disk. As all datasets within Yavaa will be held in memory, this particular aspect can not inform the decision.

On the other hand, memory allocation is a major factor in Yavaa. At least when run exclusively in a browser environment, there is no way of storing (parts of) the primary data on disk. Another important factor in the decision is the observation that most operations will result in the change of a single column's values and leave the other columns unaffected. In a row-oriented system, this would nevertheless require a copy of the whole tuple to be stored with just that single value changed or added. Then again, in a column-oriented system, both dataset versions can share a certain set of columns and just differ in the column changed as shown in Figure 13.2. Yavaa uses this approach as it reduces the overall memory footprint of the system considerably[3]. It considers columns immutable and, hence, can reuse them in an arbitrary number of datasets. Consequently, any change in the values of a column will result in the creation of a new column object. Inside column objects, data values are stored in form of an array. The indices of these arrays are aligned for column objects pertaining to the same dataset. Hence, the nth-tuple of a dataset can be reconstructed from the nth-elements of each of its column objects.

With the columns immutable, their metadata can be attached directly instead of being maintained on a dataset level. So, column properties mirroring the ones listed in Chapter 8 like title, unit, role, or type are maintained as part of the column object. Optionally, the column object

[3] In a worst-case scenario, all columns will change. In this case, the column-oriented approach will only deteriorate to a similar level as a row-oriented approach. Depending on the implementation overhead of linking values to a tuple vs. linking them in a column-array, the substantially higher number of tuples vs. columns will probably even yield a slight benefit towards the column-oriented approach.

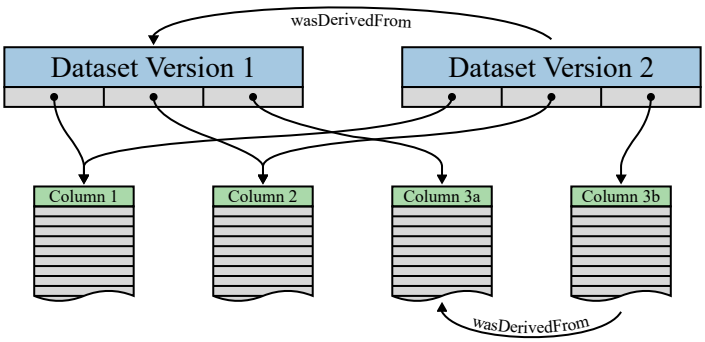


Figure 13.2: Yavaa: Reuse of columns across dataset versions.
Columns 1 and 2 remain unchanged between both dataset versions and are reused. Column 3a was modified and is replaced by Column 3b in Dataset Version 2.

might also contain the range of values for the respective column if it has been established either by the loading process, a specific filter, or any other operation that has requested this kind of information.

A common operation, which would change the values in all columns, is ordering the dataset according to some criteria. In order to prevent a full duplicate of the actual data, Yavaa uses a separate `SortedDataset` object. It mimics the behavior of the standard dataset in all aspects but the primary data. Instead of holding the column-objects it only stores a single array to map entries from the previous dataset version to their new positions. The indices of this array are identical to the ones in the previous version's columns. The values represent the new ordering of the respective entries. This sorting will be obsolete as soon as another operation takes place that modifies the dataset. Here, the operation will basically work on the previous dataset version and thus invalidate the ordering.

13.3 Computation Engine

The Computation Engine is the central part of the worker component. At one point or another, most operations will resort to the functionalities provided here. The supported modifications of primary data can be distinguished into three classes: *Simple operations* add one or multiple columns to a dataset, leaving the remaining columns unaffected. These operations can be modeled to work on the dataset in a row-wise fashion. *Aggregations and expansions* combine multiple input rows to a single output row or the other way around. In consequence, they can not just be executed separately on a row-by-row basis. Finally, *joins* combine two datasets. Again, an execution for each line is not possible and other challenges have to be addressed.

The details presented in the following are modeled after the MapReduce paradigm [279] where needed. Simple operations can be represented by a single map function without the need for an additional reduce step. Aggregations use a full MapReduce cycle. The mapping function

will collect rows based on the aggregation condition(s), whereas the reduce function will then apply the actual aggregation function. Expansions only require a single map function that will possibly emit multiple generated rows. Finally, joins can be calculated by unifying both involved datasets into a single one and then applying an aggregation. However, the aggregation function this time merges the matched rows from both datasets into one or multiple rows of the resulting dataset. Rows of the source datasets are annotated with their respective dataset-id, so they can be distinguished from one another in the aggregation step. In this case, two MapReduce cycles — one for labeling, one for the actual join — will be required.

Although not implemented yet, this compatible modeling will allow Yavaa workflows to be run on Hadoop [web145] clusters or any other infrastructure following the MapReduce paradigm and supporting JavaScript. The rationale is to allow users to work on a sample of the actual data if the size of the datasets exceeds the capacities of their local environment. They can specify the workflow on this subset of the data and tentatively evaluate its results. After completing the workflow, it can then immediately be deployed to an infrastructure capable of handling the actual dataset. In particular, the same code will be executed in both iterations, thus reducing the risk of code-induced deviations in the result[4]. A corresponding implementation is postponed for future work, though.

Changing a single cell's value can also be modeled using this technique. The function applied to the dataset is equal to the identity function for all rows but the one changed. In this row, a changed value is returned, thus modifying the targeted cell without affecting any other row.

13.3.1 Simple Operations

As mentioned before, simple operations append new columns to an existing dataset. They do not change the overall number of rows, i.e. no rows are added or removed in the dataset. Changing a column in a dataset is represented by the creation of a new column, which is followed by a subsequent replacement of the original column as described in Section 13.2.

The input for these kinds of operations is a dataset and a function whose return value generates the new column. There can be two sources for said functions: Either users explicitly specify the function to be applied or trigger it indirectly by selecting one of the predefined operations like changing the unit of measurement. In both cases, the function is given as an AST. This allows applying structural optimization techniques at a single point which prevents repeated implementations at the point of generation of the ASTs in the respective modules. For now, only constant folding is implemented, but an extension point for other techniques is given.

The optimized AST is subsequently converted to executable code in form of a JavaScript function. This function is subsequently called for each row of the input dataset and its return values are collected in a new column. For some operations, a generic function could simplify

[4]Depending on the sampling algorithm used to create the subset using in preparing the workflow, the results might differ though.

this generation process. However, a compiled, custom-tailored function will yield considerable performance benefits, as it does not need to include any other command than those needed for the task at hand. Furthermore, it can make use of the built-in optimizations of modern JavaScript engines like Chakra [web146], Spidermonkey [web147], or V8 [web148]. This, of course, is based on the assumption, that the execution time of applying the function to the dataset is considerably larger than the time needed to prepare it. This assumption seems reasonable in the current context as the function has only to be prepared once, whereas it is executed for thousands of rows at a time.

Functions triggered by other internal worker-operations are immediately available as ASTs. Functions supplied by users have to be parsed into such an AST first. They also need to be validated against a given schema or grammar to prevent erroneous inputs and possible code injections by malicious users. To this end, the grammar provided in Appendix E was created. It is a Parsing Expression Grammar (PEG) [280], which can be compiled into JavaScript code using PEG.js [web149].

PEGs differ from Context-Free Grammars (CFGs) in that they replace the (unordered) choice operator | in CFGs with a prioritized choice operator /. In a CFG, the following two rules are equivalent: $A \rightarrow a|ab$ and $A \rightarrow ab|a$. However, in a PEG the two rules $A \rightarrow ab/a$ and $A \rightarrow a/ab$ are different. In particular, in the second rule, the latter alternative will never be matched as the first alternative will precede it for every input string starting with an a. A well-formed PEG is a grammar that includes no left recursive rules like $A \rightarrow Aa/a$. If applied, such rules would lead to endless loops[5] in parsing and are thus provide no useful applications in practice. All aspects combined allow for the creation of efficient, i.e. linear time, parsers using memoization like the packrat parser [281].

The grammar used in Yavaa is well-formed and provides support for the four basic mathematical operations (Addition, Subtraction, Multiplication, Division), a reference to the current value (`value`), and references to other cells of the same row (`col0`, `col1`, ...). Similar to the HTML-specification [web150], an arbitrary number of whitespace-characters[6] can be inserted between the tokens. PEG.js also allows to label parts of an expression and to use that label in code snippets directly attached to the grammar. These snippets are used to transform matched expressions for further use. In the case of Yavaa's formula parsing, they are used to return an AST comparable to the ones created by other worker-operations.

In the prototypical implementation described here, user-defined operations are only supported on quantitative columns. Compared to the rather well-defined and commonly used operations on quantitative values, defining meaningful and useful counterparts on, e.g., semantic entities is non-trivial. So while the developed architecture allows for adding this in the future, it is considered out of scope in this thesis.

[5] A parser would try to match rule A to an input string. This, in turn, starts by applying a recursive matching to rule A, which does the same ad infinitum.

[6] TAB (U+0009), LF (U+000A), CR U+000D, and SPACE (U+0020).

13.3.2 Aggregations and Expansions

Both aggregation and expansion, modify the number of rows within a dataset and can be seen as inverse functions to one another. While within an aggregation one or multiple input rows contribute to a single output row, an expansion function takes a single input row and returns one or multiple output rows.

Expansions work similarly to simple operations. The dataset is processed line by line and a function is applied to one column. However, instead of returning a single value for each row, the function applied might return an array of values. Subsequently, for each value within that array, a new row is inserted into the result dataset. The values for all other columns are copied from the source row. At the time of writing, the only expansion function to be applied is unbagging, which is the reverse function to bagging as discussed later on.

On the other hand, aggregations require more effort. Here, input rows first have to be grouped according to the values in a subset of columns. This is implemented by a tree-structure where each level represents one column to be grouped by. Each child node spans the subtree for a given value represented by its hash-value. Leaf nodes contain pointers to each matching row where the path from the root represents the values within that row. After building this tree, all leaf nodes are traversed in order. Each value on the path towards that leaf is inserted at the respective position of the result row(s). Next, for each remaining position, all values from the rows collected in the leaf nodes are gathered into a list. The resulting lists are passed to aggregation functions to retrieve the final values to be used in the generated row. Aggregation functions currently supported are listed below.

avg computes the average of a given collection of numbers.

bag collects all values in a multiset. This is the default aggregation function.

sum returns the sum of a given collection of numbers.

set collects the values in a set.

takeOne returns the first value of the collection.

New aggregation functions can easily be added: All that is needed is a function that takes a list of values as an input and returns a single value. Alongside the actual implementation, a description has to be provided to define the required properties. This includes, besides the name and a description, a list of data types the respective aggregation function can be applied upon or the keyword `all` if there are no restrictions[7].

At this point, the difference between `bag` and `set` shall be emphasized. While `bag` returns a multiset in which duplicate entries will be retained, using `set` will eliminate said duplicates and return only a list of unique values. Depending on the use case, one or the other might be

[7] From the provided aggregation functions, only `sum` and `avg` are restricted to the numeric data type. All other functions can be applied without restriction.

chosen. Identifying all entities satisfying a certain condition, set semantics might be preferred. However, if another aggregation function should be applied subsequently, bag semantics retain more information. In particular, for mathematical functions like summation or average, the correct result can most often only be achieved using bag semantics, whereas set semantics will skew the results here.

13.3.3 Joins

Join algorithms are generally implemented using one of three techniques [282]: Nested loop joins, sort-merge joins, and hash joins. In the following discussions, `L` will denote the left-hand side dataset in the join, whereas `R` will stand for the right-hand side one.

In their most basic form, nested loop joins traverse all rows of `L`. For each row, `R` is scanned for matching rows and a resulting row is returned for each match. An optimization is to scan `R` in blocks ("nested block join") reducing necessary I/O-operations depending on available memory.

Sort-merge joins are executed in two steps. In the first step, both datasets are sorted according to the columns present in the join condition. Afterward, both datasets can be traversed in order, evaluating the join condition for each pair and adding matching rows to the result. Using this approach, each row has only to be read once from disk saving a lot of I/O-operations compared to a nested loop approach.

Hash joins exist in many variations. Common to all of them is, that for each row in `L` the values present in the join condition are hashed. The hashes along with a pointer to the source row are stored in a hash table. Now, the rows of `R` are traversed. For each row, again corresponding hashes are computed. If there is a matching entry in the hash table, both rows are joined and added to the result.

Yavaa's implementation uses a hash join as well, as this can be transferred easily to a MapReduce setting. In particular, a left outer join is implemented using the same technique already described for aggregations. `R` is added to a tree structure using the join conditions to identify the respective leaf node. Then `L` is traversed and for each row, a match within the tree is determined. If no matching leaf node can be found, the respective result row is filled with `null` values for the missing columns.

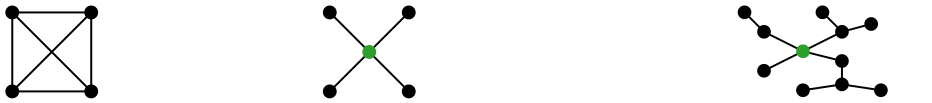
13.4 Unit Store

The Unit Store is the primary source for information related to units of measurement. As discussed in Chapter 9, OM [164] is used as a knowledge base for this purpose. However, to extend the number of available datasets in the evaluation (cf. Chapter 14) several units and dimensions had to be added. In particular, this includes units related to counting entities that are missing in OM. A full list of added individuals is given in Appendix C. They follow the schema given by the T-Box definition of OM and can thus be treated in the same way as the original ones.

As hinted in Figure 13.1 before, the Unit Store is backed by an RDF triple store. This separates between maintaining unit data in the triple store and using it within Yavaa's Unit Store. In particular, should new units become necessary or issues with the existing ones need to be fixed, no code changes are necessary. Instead, any change made to the triple store will immediately be available throughout the system, e.g., to describe new datasets. All communication is done via SPARQL-queries. This would also allow to swap out the local unit triple store, with a publicly available SPARQL-endpoint possibly maintained by the data providers themselves.

Before describing the functionalities provided by the Unit Store, the concept of a *conversion graph* shall be discussed. Part of OM is the modeling of conversions of units. These conversions form a directed graph, where each unit is represented by a node and two nodes are connected if and only if a conversion consisting of a factor and possibly an offset between both units is given. The direction of the edge is determined by the direction of the conversion, i.e. an edge from unit A to unit B contains the factor and offset to convert a measurement using A to one using B.

A conversion graph constructed in the aforementioned way will contain multiple clusters consisting of nodes having no connection to nodes in other clusters. Each of these clusters represents a set of compatible units. The topology of individual clusters can take different shapes as exemplified in Figure 13.3[8]. For the complete and star-shaped topologies, a conversion can be determined by a single query. In the complete topology (cf. Figure 13.3(a)), each conversion is explicitly represented within the ontology and, hence, can be queried directly. In a star-shaped topology (cf. Figure 13.3(b)), the path between any two units is at most of length two. Both steps can also be retrieved in a single query. However, within snowflake topology (cf. Figure 13.3(c)) the conversion path can be of arbitrary length, making aggregation functions necessary. As of version 1.1, SPARQL does not include an aggregation function to multiply multiple values [web88, §18.5][9]. So with this topology, the conversion between two arbitrary units can not be determined using a single query. OM uses a snowflake topology. The implementation to deal with the resulting challenges will be described later on.



(a) Complete. (b) Star-shaped. (c) Snowflake-shaped.

Figure 13.3: Reasonable conversion graph topologies.

The Unit Store provides endpoints to serve different kinds of queries. Units that are part of the input are always given by their respective URLs. Queries like retrieving units by their label as part of the dataset annotation process are considered out of scope for the main Yavaa application and, hence, are not supported by the Unit Store.

[8]These topologies and their implications have also been discussed in [283].
[9]Supported aggregation are: `Count`, `Sum`, `Avg`, `Min`, `Max`, `GroupConcat`, and `Sample`.

Pulling general information for a unit involves its label, its dimension(-vector), and whether it is a scaled unit like temperatures. The data is used within the user interface and within the determination of the resulting unit of a formula (cf. Chapter 9). It is directly modeled within the OM ontology and this requires no additional efforts.

Alternative units are found by exploring the respective conversion graph. This is a more reliable approach compared to a search on compatible dimension vectors as, e.g., in [168]. As an example, consider pressure and (volumetric) energy density. Both share the same dimension vector according to the SI system of units [160], namely $Mass \times Length^{-1} \times Time^{-2}$. However, as they describe different properties of a phenomenon, their respective units can not be used interchangeably and, hence, can not be converted to one another. A search solely based on the respective dimensions vectors will miss that fact and report wrong results. Another advantage of basing the search for alternatives on the conversion graph is the guarantee of a conversion existing. If the underlying ontology was missing some conversion, this could otherwise lead to problems: A unit presented to users as a possible target of a conversion would result in errors when being actually selected and attempted to be applied.

Another function provided by the Unit Store is the conversion between two given (compatible) units. The result is not the converted measurement value, but a function or the corresponding AST representing the conversion. The AST can be used for possible adaptations like optimizations in the context of a user-defined function's AST. As mentioned before, OM uses a snowflake schema for its conversion graphs, so in general, the paths within a cluster can be of arbitrary length. The final conversion factor accumulating all factors along this path can not be determined by a single SPARQL query alone, as discussed before. At the very least this requires aggregating the conversion factors within the application.

Another challenge is the different kinds of conversions represented in OM. It contains three different kinds of conversions: direct conversion between two units only using a factor like from *foot* to *meter*, conversions using prefixes like *kilometer* to *meter*, and conversions involving a change of scale like from *degree Fahrenheit* to *degree Kelvin*. Figure 13.4 shows an overview of possible cases. If a unit is prefixed, the conversion factor to the respective singular unit is given by that prefix. Afterward, there can be an arbitrary number of conversions defined via `om:Measure`-individuals[10]. If the unit uses scales, there is another step using the associated `om:Measurement_scale`-individual. These scales follow the same paradigm as units with respect to their conversions, but besides a conversion factor also include an offset. In total, this results in three different ways to model conversions within OM which all have to be dealt with within the query.

To cope with both challenges, a three-step approach has been taken to gather all necessary information for a requested conversion. In a first step, the conversion graph related to both source and target unit is examined and for both units, all nodes along the path to the center

[10]To be precise, a single multiple of unit A is measured in terms of unit B, thus defining the conversion.

of the respective cluster are collected. This center is uniquely defined for all clusters with the exception of the one representing mass: In that one graph there is a cycle between `om:gram` and `om:kilogram`. While gram is defined via a direct conversion towards kilogram, kilogram also is annotated as a prefixed version of gram. Adhering to the SI definition [160], Yavaa uses `om:kilogram` as the center of this cluster.

With the exception of this sole case, the direction of the conversions is always well defined. So, in a second query, all conversions for the units along the path are gathered. Scaled units pose another obstacle here: Although they are defined using their respective scales, they also have a property using an `om:Measure`-individual. However, this individual omits the conversion offset and, hence, has to be ignored.

Having obtained all the required information, the last step creates the actual conversion path. As both paths might intersect before the center of the cluster, the application searches for the first common node within both paths towards said center. All subsequently shared nodes will be removed. Afterward, the conversion AST is created by traversing the path from the source unit to a common node and then from common node to target unit. All conversion factors along the way are represented within the conversion AST to prevent inaccuracies while accumulating[11].

Finally, the Unit Store offers some functions to handle compound units. The first method returns the respective compounds given a compound unit. If the unit consists of other compound units, those will be resolved recursively. Similarly, the reverse method is provided: For a given virtual unit object[12] consisting of two sets of units, one for the numerator and one for the denominator, the method tries to provide the URL of the respective equivalent compound unit. This compound unit might not exist in all cases, though, so this method might return a null value.

For virtual units also a simplification method is provided. This removes common entries from numerator and denominator using the approach presented in [168]. First, all compound units within both sets are replaced by their compounds. The replacement of a unit present in the numerator set might also affect the denominator set and vice versa when the compound itself is comprised of numerator and denominator. Furthermore, prefixes are removed and their respective numerical values collected for each set. In the second step, common occurrences in both sets are removed and the accumulated prefix factors are aggregated. The resulting sets are then passed to the aforementioned method to resolve a virtual unit. If a single individual for that virtual unit is found, it is returned. Otherwise, a new simplified virtual unit is passed on.

[11] The different factors will be accumulated when the AST is converted to an actual JavaScript function. In this process, certain optimizations take place which includes the aggregation of chains of constants factors. However, this uses the predefined precision settings, so within those boundaries, no additional inaccuracies are introduced.

[12] Virtual units may appear as the result of certain operations. In essence, they are compound units that have not been matched to a corresponding entry in the unit ontology or where no such entry could be found.

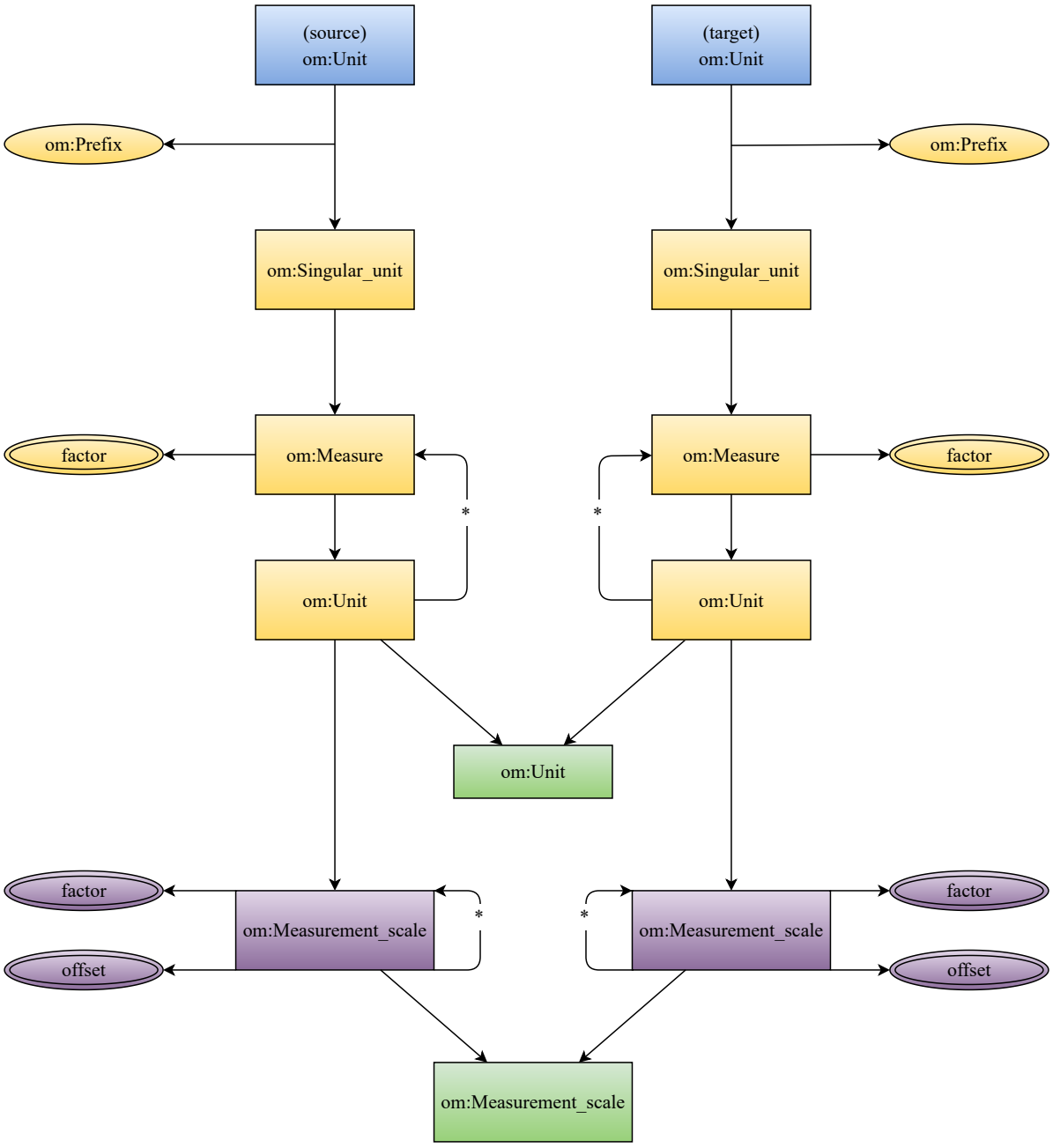


Figure 13.4: OM 1: Generalized conversion path structure.

13.5 Graphical Workflow Layout

The Workflow Visualization converts the workflow maintained internally or in the used PROV dialect (cf. Chapter 12) to a visual representation accessible to users. In the past, various algorithms have been proposed to draw graphs. An extensive survey including a classification of graph layout algorithms can be found in [284]. These algorithms optimize for different criteria like minimizing edge crossings, creating symmetric graphs, preventing node-edge overlaps, separating non-adjacent nodes, or facilitating a uniform node distribution. However, most of them target general graphs or cater to different constraints than the ones at hand.

The workflow graph as created by Yavaa is highly structured (cf. Chapter 12). It consists of a sequence of datasets each consisting of multiple columns[13]. Columns often remain unchanged over multiple or all datasets in a sequence. Another constraint arises from the PROV graph layout conventions [web138]. They suggest providing a chronological ordering either horizontally or vertically with the newest node on the right or bottom respectively. As the datasets in the given workflow depend on one another, this determines their relative positions within the visualization.

The algorithm presented here will adhere to these constraints by implementing the following layout conventions: To reflect the structure of the workflow, each set of columns will receive a dedicated vertical lane within the visualization. This mimics the general layout of a tabular dataset where values of a particular column are also vertically aligned forming a dedicated lane for that column. Activities will span all lanes of columns they modify. Columns that only influence a certain activity without being modified by it will only be referenced by dashed arrows. The chronological order will be represented by the vertical positioning: source datasets at the very top, intermediate activities in the middle section, and the resulting dataset or visualization at the bottom. The vertical positioning of activities will also follow a strictly chronological ordering. If activity a_1 predates activity a_2 its node will be shown more towards the upper end of the visualization than the one representing a_2.

Sample layouts of the workflow graph were given in Figures 12.9 to 12.11. These already adhere to most of the layout conventions listed, but do not represent the final workflow visualization. In the given samples, all entities and relations present in the graph have a visual representation. In particular, this includes entities for a dataset and its respective columns after each operation.

To increase the readability of the generated workflow visualization some elements of the workflow will subsequently be omitted. Firstly, only the first and last column-entity of each lane will be shown. All intermediate column-entities will have no visual representation. As columns are already assigned to their dedicated lane and the starting and final entity for that column is shown, the creation of a new column-entity after each activity is assumed to be implicit and, hence, redundant. The same approach is taken for datasets. Only sources and final results will have a visual representation, while all intermediate datasets are omitted in the visualization.

[13]The term “dataset” within this context also includes the visualization. As discussed before, from an abstract point of view the difference is negligible.

The omission of intermediate column-entities can be justified from the constraint set specified by PROV [web125]. Inference rule five states that, if an activity a_2 `prov:wasInformedBy` a_1, there exists an entity e, such that e `prov:wasGeneratedBy` a_2 and a_1 `prov:used` e. Figure 13.5 shows a visual illustration of this inference rule. So by connecting activities using `prov:wasInformedBy`-edges, the workflow visualization implicitly includes the omitted intermediate column-entities. For the omission of intermediate datasets, there is no equivalent inference rule. So technically, the visualization generated will not represent the exact workflow as serialized by the Export component.

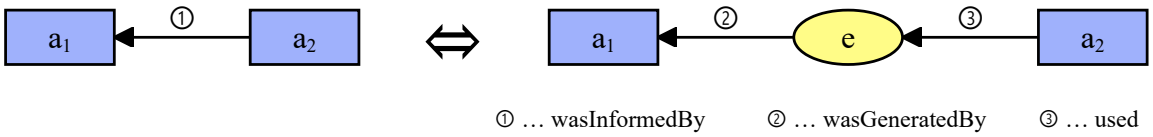


Figure 13.5: PROV-Constraints [web125]: Visual illustration of inference rule 5.

The visualization algorithm itself is separated into two steps. A first layouting step (Listing 13.1) will create visual artifacts to be rendered and determine the horizontal order of column lanes. The second step (Listing 13.2 utilizing Listing 13.3) will then traverse the chronologically ordered list and position all artifacts as well as draw their connections. An example output is shown in Figure 13.6.

The layouting step in Listing 13.1 works as follows. First, artifact-objects are created to hold the data within the visualization (lines 3 to 7). As discussed before, intermediate datasets will not be shown within the visualization and, hence, no artifact is created for them. Similarly, column-entities are discarded as they are directly bound to activities and will be accessed through them. Now, a global column mapping in form of the array `columnMapping` is created by traversing a sorted list of artifacts representing activities (lines 9ff.). The list is sorted in ascending order by the `prov:startTime`-property of the associated activity. The array contains objects representing the aforementioned lanes for each column whose positions within the array also encode their later position in the visualization. If an activity creates new columns, for each of these columns a new lane-object is created (lines 15 to 18). The lane object includes a list of all column-entities it represents in its `wfEntries`-property. The newly created lane-objects will then be inserted into `columnMapping` array (lines 20 to 23). If the activity only creates new columns but uses no existing ones[14], all new lane-objects are appended to the end of the column mapping. However, if the activity works on an intermediate dataset, the lanes will be inserted at the relative positions within `columnMapping`, possibly changing the indices for later entries. The relative position is given by the `yavaa:order` property accessible through the columns of the dataset created by the respective activity and columns already positioned within the column mapping. Finally, all columns involved in the activity are mapped to their respective lane (lines

[14]In the current implementation, this only refers to activities loading external datasets.

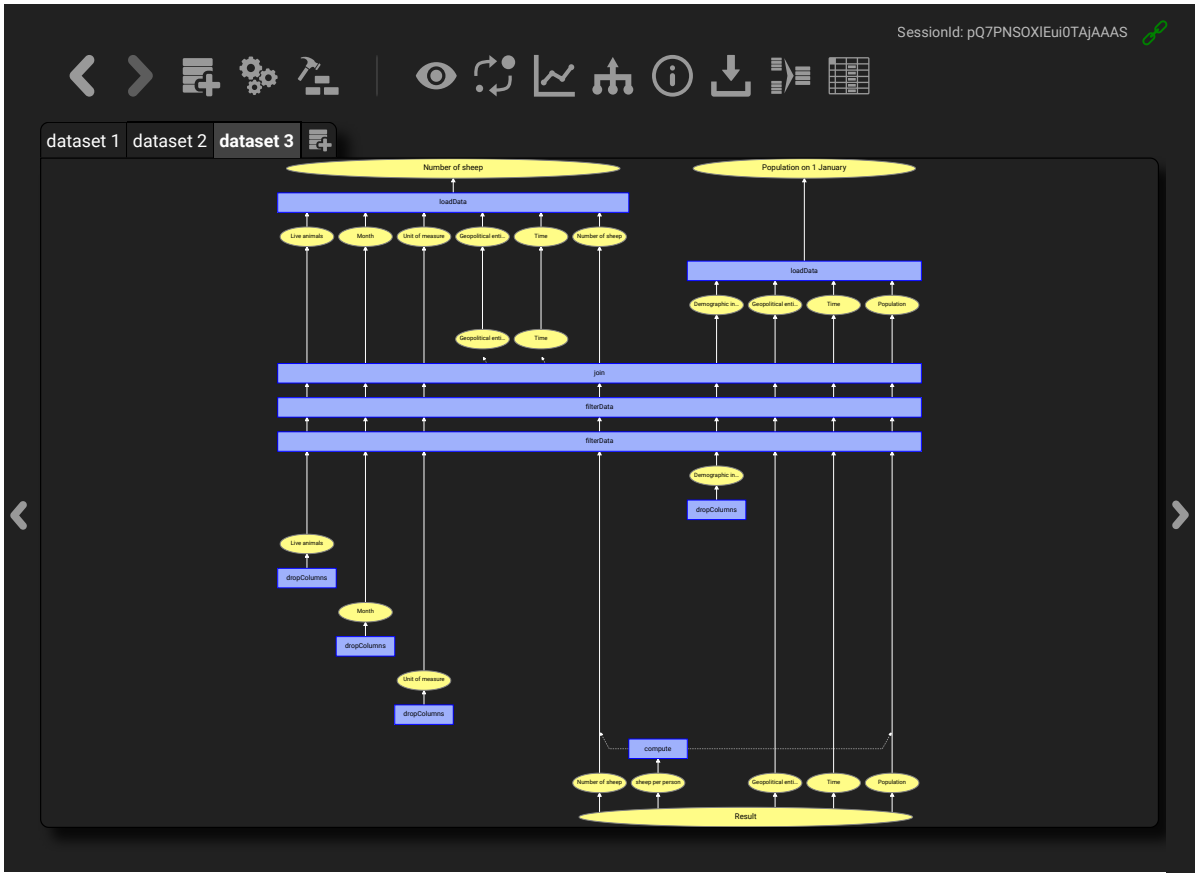


Figure 13.6: Yavaa: Workflow view.
Chronologically ordered from sources (top) to final data products (bottom). Wide ellipsis on the top and bottom indicate data sources and final data products respectively. Columns are arranged into vertical lanes and use smaller ellipses to represent their lifespan within the workflow: The upper one shows their inception, while a lower one symbolizes either end-of-life or their final state. Solid arrows further illustrate the column lanes and connect entities to operations that modified them. Operations are given by rectangles covering all lanes of affected columns. Dashed arrows link operations to used but unchanged columns.

```
FUNCTION layoutGraph( wfGraph )

  artifacts = <empty list>
  FOREACH activity- and entity-item in wfGraph DO
    IF item is not an intermediate dataset or column-entity
      artifact = { wfEntry: entry };
      add artifact to artifacts

  columnMapping = <empty array>
  activities = sort activity-artifacts ascending by the startTime of their respective wfEntry
  FOREACH activity in activities

    IF activity.wfEntry creates new columns

      newColumns = <empty list>
      FOREACH column created by activity.wfEntry DO
        lane = { wfEntries: [ column ] }
        add lane to newColumns

      IF activity.wfEntry loads from an external source
        append newColumns to columnMapping
      ELSE
        insert newColumns at their relative positions to columnMapping

    FOREACH column present in the dataset created by activity.wfEntry DO
      predecessor = column-entity connected via wasDerivedFrom to column
      lane = entry in columnMapping containing predecessor
      add column to lane.wfEntries

  RETURN columnMapping and artifacts

END
```

Listing 13.1: Yavaa: Workflow graph visualization - layouting.

25 to 28). For this, the predecessor of the given column-entity is determined, which is given by a connection via `prov:wasDerivedFrom` in the workflow graph. This predecessor is already listed in the `wfEntries`-property of one entry in the column mapping, which thus defines the lane-object for the current column. The current column is also added to the `wfEntries`-property of the respective lane to allow for further activities' columns to be assigned the same way. The result of the layouting step is a list of artifacts to be rendered and a global ordering of all columns within the workflow in `columnMapping`.

The drawing step of the algorithm, given in Listing 13.2, takes these intermediate results and draws visual artifacts on a canvas[15]. Given inputs are the list of artifacts, the column mapping as created by the previous step, and two constants: Horizontal spacing determines the width of a column-lane. Vertical spacing defines the vertical distance between two rendered artifacts. The algorithm processes all items in chronological order (line 8ff.). Throughout this, the current position along the y-axis is maintained in `curY` (line 3). It is incremented by the vertical spacing constant each time a horizontal series of artifacts has been rendered. In addition to the previous step, the remaining dataset and visualization artifacts are also included in that ordering. As they do not possess a `prov:startTime`-property themselves, the respective property

[15]The implementation relies on SVG [web151] at this point.

```
FUNCTION drawGraph( artifacts, columnMapping, constants )

  curY = constants.verticalSpacing

  FOREACH lane in columnMapping DO
    lane.x = indexOf( lane ) * constants.horizontalSpacing

  sort artifacts chronologically
  FOREACH artifact in artifacts DO

    SWITCH TYPEOF artifact.wfEntry
      CASE activity:

        droppedColumns = lane artifacts for each column dropped by artifact.wfEntry
        IF droppedColumns.length > 0
          FOREACH lane in droppedColumns DO
            render column object using curY and lane.x
            render connector for column object from curY to lane.start
          curY += constants.verticalSpacing

        artifact.y = curY
        render( artifact, columnMapping, constants )
        curY += constants.verticalSpacing
        IF TYPEOF artifact.wfEntry == 'load'
          render connector to respective source entity
        ELSE
          render connector to all droppedColumns

        createdColumns = lane artifacts for each column created by artifact.wfEntry
        IF createdColumns.length > 0
          FOREACH lane in createdColumns DO
            render column object using curY and lane.x
            lane.start = curY
            render connector for column object from curY to artifact.y
          curY += constants.verticalSpacing

      CASE entity:

        IF artifact.wfEntry is source dataset

          artifact.y = 0
          render( artifact, columnMapping, constants )

        ELSE

          lanes = lanes referenced by artifact.wfEntry in columnMapping
          FOREACH lane in lanes
            render column object using curY and lane.x
            render connector for column object from curY to lane.start
          curY += constants.verticalSpacing

          artifact.y = curY
          render( artifact, columnMapping, constants )
          render connectors from curY to previous column objects for each lane
          curY += constants.verticalSpacing

END
```

Listing 13.2: Yavaa: Workflow graph visualization - drawing.

of the activity succeeding them (source datasets and loading activity) or preceding them (result datasets/visualization) are used instead. Depending on the type associated with the artifact, the rendering differs.

If the artifact represents an activity, first all lanes dropped by the activity are drawn (lines 14 to 19). In this case, a final column artifact is drawn and subsequently connected to the respective starting artifact. Now, the artifact for the activity itself is rendered and connected to either the source entity (load activities) or the previously drawn artifacts for dropped columns, if existing (lines 21 to 26). Finally, if the activity created new lanes, those are drawn and connected to the activity artifact (lines 29 to 35). The y-coordinate for these newly created artifacts is maintained to allow for the connection from the final column artifact to be drawn later on (line 33).

Rendering the remaining entity artifacts is simpler. If the artifact represents a data source, it is drawn at the very top of the graph (lines 39 to 42). Otherwise, the entity represents the final dataset or visualization. In this case, for all lanes, a final column artifact is drawn and connected to the starting artifact for that lane (lines 46 to 50). Afterward, the entity is rendered and also connected to all column objects just rendered (lines 52 to 55).

```
FUNCTION render( artifact, columnMapping, constants )

  lanes = lanes referenced by artifact.wfEntry in columnMapping

  artifact.minX = minimum of all col.x for col in lanes
  artifact.maxX = (maximum of all col.x for col in lanes) + constants.horizontalSpacing

  IF artifact is entity
    render entity object using artifact.y and artifact.minX/artifact.maxX
  ELSE
    render activity object using artifact.y and artifact.minX/artifact.maxX

END
```

Listing 13.3: Yavaa: Workflow graph visualization - Render object.

The rendering of all artifacts but the connections need the width of the artifact[16]. For column objects this width is equal to the width of their lanes given by the horizontal spacing minus a certain margin. However, other artifacts need to cover multiple lanes. The respective calculations are given in Listing 13.3. First, all lanes affected by this activity or included in the entity have to be determined (line 3). The width is then given by the minimum and maximum x-coordinates of all these lanes (lines 5 and 6). This allows drawing the appropriate artifact (lines 8 to 10).

13.6 Communication Layer

As mentioned previously, the user interface and worker are loosely coupled and are connected by a separate communication layer. The actual medium of that layer is not fixed, but implementations are provided for communication to a server via WebSockets [web152] or within the browser to a

[16]The height is given by a constant. The implementation uses about half the horizontal spacing.

WebWorker [web143]. The submitted messages are fixed in their structure, though. For a full list of supported commands see Appendix D. At this point only a qualitative description of the messages and their structure shall be given.

The communication channel is considered asynchronous, so the communication layer has to take care of matching messages from the worker to previous requests from the user interface. This is achieved by the use of so-called `job-id`s which will be assigned by the worker to each incoming request. Hence, the exchange of messages is structured as follows: The user interface sends a request to the worker. This is immediately responded to by a message stating the respective `job-id`. The user interface refrains from sending any other request until that first request has received a `job-id`. Previously, other requests may have already be sent without having received the actual result yet. On the worker, site the acknowledgment for a new request in form of the `job-id` is sent as soon as the incoming message has been validated. In particular, this precedes any calculation needed to fulfill said request. After sending the acknowledgment or an error message if the validation failed, the actual computations are run to provide a response. In the meanwhile, other requests will already be accepted. If the result of a request has been prepared, the response is augmented with the respective `job-id` and sent to the user interface. By the use of that id, the user interface will identify the original calling function and relay the results to it.

The decision to assign the `job-id` on the worker's side is almost arbitrary. The scale was tipped by considering necessary validations and potential surface area for attacks. Any client-provided data is a potential security risk [web153] and as such requires validation. Although a particular attack vector might not be known in advance, less surface area for any kind of attack is always preferable. When the `job-id` is already part of the initial request, code for its validation would need to be added. Assigning the `job-id` on the worker's side, this requirement can be omitted to save some pieces of code that might potentially break.

Within the implementation, a `Promise` [web142, §25.4] is created for each request sent by the user interface and returned to the caller at once. When the respective `job-id` is received, it is used to identify the corresponding `Promise`. Upon response, the `Promise` is resolved using the response's payload. This method allows any user interface component to trigger (multiple parallel) asynchronous requests to the worker and thus prevent any blocking behavior.

The structure of messages is defined in an XML [web11] file[17]. The file itself serves two purposes: On the one hand, it documents the external API of the worker and thus allows other vendors to use the worker within their own application or websites. Using a parser[18], also more human accessible documents can be created to serve the same purpose. On the other hand, the file is used to run a basic validation of the messages received and transmitted by the worker.

The message definition is split into three hierarchical levels as shown in Listing 13.4. The first hierarchy splits the list of messages according to topics using `<section>`-elements. On the second level, messages are grouped by the direction they are sent in using `<sectionpart>`-elements

[17] `protocol.xml`
[18] Such a parser is not provided as part of the prototype implementation, though.

```
<root>
  <section title="some title">
    <sectionpart direction="W2UI">
      <command />
    </sectionpart>
    <sectionpart direction="UI2W">
      <command />
    </sectionpart>
  </section>
</root>
```

Listing 13.4: Message definition: General structure.

```
<command name="getData">
    <desc>request a chunk of data</desc>
    <binding module="comm/data" method="getPartialData" />
    <params>
        <param name="data_id" type="Number">ID of the respective dataset</param>
        <param name="start"   type="Number">start index inside the dataset</param>
        <param name="entries" type="Number">amount of entries</param>
    </params>
</command>
```

Listing 13.5: Message definition: Example (Request subset of primary data).

```
{
  "action": "getData",
  "params": {
    "data_id":  1,
    "start":    0,
    "entries":  200
  }
}
```

Listing 13.6: Communication Layer: Example message (Request for a subset of primary data).

and the respective attribute `direction`. The direction might either be from user interface to worker (`direction="UI2W"`) or vice versa (`direction="W2UI"`). Finally, the last hierarchy level represents the actual messages sent that are defined by `<command>`-elements.

An example message definition is given in Listing 13.5. The message is labeled by the `name` attribute of the `<command>`-element. Furthermore, it is comprised of three parts: The `<desc>`-element provides a human-readable description. The part of a worker's code that is to be executed receiving this message is given by the `<binding>`-element. Finally, using `<params>` and its `<param>`-child-nodes all parameters of this message are defined. Each parameter consists of two attributes `name` and `type` as well as a human-readable description in the element's content. The types currently supported by the parser are `Number`, `String`, `Array`, and `Object`. For arrays also the type of its members can be stated. Furthermore, a list of valid values can be given using a JSON-encoded [285, web142, §24.3.2] array holding all allowed values.

A message itself is a JSON encoded instance compliant to this description. Listing 13.6 shows a possible instance for the definition given in Listing 13.5. In this message, a subset of 200 entries from the dataset with id 1 is requested, starting from row zero. On the worker side, this message is validated against the respective definition. The `action`-property defines the respective message definition upon which the presence of all listed properties and their respective types are validated. If any of these checks fails, the message is discarded and an error message returned. Otherwise, the message is accepted and the requested operations are scheduled.

13.7 Visualizations

After users prepared their data and have selected a suitable visualization (cf. Chapter 11), the Visualization Generation will execute the respective code. The result is an SVG representation [web151] of the chosen visualization. SVG is chosen over canvas [web154] at this point for the following reasons: First of all, most statistical visualizations are rather structured in the way they are composed[19]. Data is mapped to visual artifacts that are represented by well-defined geometric shapes. A natural way to represent those shapes within images is vector graphics for which in a Web context the default format is SVG. Using SVG also entails all other benefits of vector over raster graphics like reduced file size and better scaling.

Another reason lies in the environment the visualization is created in. Recall, that the worker may be run using a WebWorker [web143] in the browser or on a Node.js server [web144]. In both environments, there is no direct access to the Document Object Model (DOM) [web155]. A canvas image, however, is created by successively applying commands to a `<canvas>` element to draw individual artifacts. To be transmitted to the browser, the resulting image has to be exported to a raster graphic in any supported format. Contrary, SVG at its core is just an XML document [web11] that can be serialized and transmitted easily. While canvas needs a rather complete DOM implementation, SVG can be created using only a subset of DOM operations[20]. To provide a sufficient DOM environment, Yavaa relies on jsdom [web156] which is a pure JavaScript DOM implementation. However, at the point of writing it does not include canvas support.

The final reason to use SVG over canvas concerns the addition of interactive elements or animations within the visualization. Although not implemented in the prototype, dynamic elements can be added to the visualizations using the given architecture. While for canvas this requires redrawing certain areas of the image over and over again, SVG supports direct manipulation of its elements as well as zooming in and out of the visualization[21]. Annotating certain elements of the SVG graphic then also allows for generic controls, whereas canvas needs customized code for each visualization.

[19] At this point kindly recall the approaches described in Chapter 7.

[20] Actually, no DOM implementation whatsoever would be needed. However, in order to be able to use existing JavaScript charting libraries a subset of commands is necessary to, e.g., create nodes and add attributes.

[21] The redrawing is entirely handled by the browser engine here. From a developer's point of view, only certain attributes have to be changed.

The general workflow of the visualization generation is given in Figure 13.7. After the dataset and respective visualization have been selected, first, the visualization's preprocessor is executed. Here, the visualization implementation has full access to functionality provided by Yavaa. For example, certain aggregations can be calculated if not all values are to be shown directly in the visualization. Afterward, this data is passed on to the actual generation process. This process is executed using the aforementioned DOM environment which enables the use of most JavaScript charting libraries relying on SVG. As of now, all visualizations are realized using Data-Driven Documents (D3.js) [web19]. However, the implementation can easily be adapted to accommodate other libraries as well. After the visualization has been generated within the artificial DOM environment, the respective SVG image is serialized and submitted to the client.

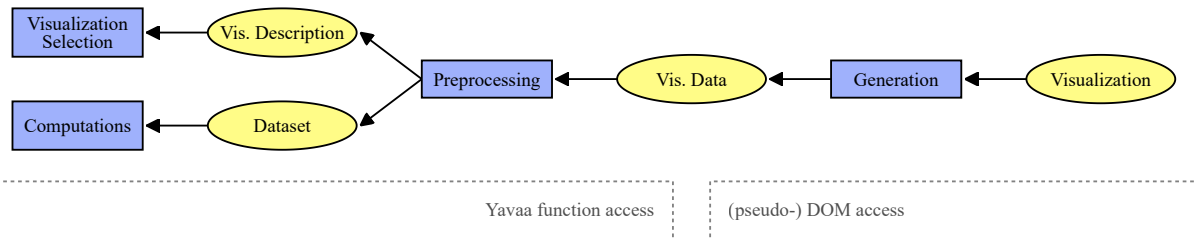


Figure 13.7: Workflow for visualization generation (using PROV-notation).

The implementation of a visualization is split into multiple files according to this workflow. Besides the individual visualizations implementation, Yavaa also includes a general template that provides the general setup for D3.js, so that this is not repeated for each visualization. Overall, each visualization has to provide the following four files. On server start, Yavaa will scan for those and, after basic validation, add them to the internal Visualization Repository. A full list of currently supported visualizations can be found in Appendix F.

Description . a JSON-encoded [39] description as defined in Chapter 7.

Preprocessing . JavaScript code to facilitate necessary data preparations.

Generation . JavaScript code to create the actual visualization.

Preview-Image . a preview image to be shown in the user interface.

13.8 Provenance and Reenactment

As discussed previously in Chapter 12, there are two ways to think of provenance [250, 251, 252]: *Retrospective* provenance documents the origin of a particular entity. In the context of this thesis, this refers to the steps taken to derive a particular dataset or visualization. On the other hand,

prospective provenance is represented by a not yet enacted workflow that describes how to create a particular result. The provenance records exported by Yavaa cater to both views and will be described in the following.

The provided serialization follows the guidelines of PROV-JSON [web131][22]. It serializes a given provenance record into a series of nested objects and as such is, first of all, a form of retrospective provenance. Subsequently, its structure will be illustrated using the example workflow given in Figure 13.8 which presents one way of solving the tasks posed during the user evaluation (cf. Section 14.2).

The first level of nesting is given by the types of components as shown in Listing 13.7. Following the model described in Chapter 12, this results in seven properties here: The nodes of the provenance graph are collected under either `entities` or `activities`. The edges connecting those nodes are separated by their respective type. So following the model presented before, five types of edges and, hence, properties are present in the serialization: `used`, `wasDerivedFrom`, `wasGeneratedBy`, `wasInfluencedBy`, and `hadMember`. Per definition, all unprefixed properties in the serialization are part of the default PROV-namespace.

```
{
  "entity": { },
  "activity": { },
  "used": { },
  "wasDerivedFrom": { },
  "wasGeneratedBy": { },
  "wasInfluencedBy": { },
  "hadMember": { }
}
```

Listing 13.7: Yavaa: Serialized provenance record - top level.

The keys in the nested objects are given by the IDs for each included entry. As most IDs do not carry any meaning outside the particular provenance record, the serialization will choose generic IDs based on the respective type. JSON does not allow for circular references or general referencing which might be needed to properly represent a provenance record. So, the generated IDs are used instead to allow for proper interlinking of components. Listing 13.8 sketches the content of `entities` with three examples. An external source is described in `source6`. It is given by properties used to identify the metadata record (`yavaa:datasetId`) as well as generic information that may be utilized by other tools (remaining properties). An example of an internal dataset is given by `result7` which in this case is an intermediate result. Each provenance record only contains a single non-intermediate result which in turn represents the final outcome of the given workflow. As seen in the example, internal datasets are modeled as instances of `prov:Collection`. They contain multiple columns with one given by `column3` in the example.

[22] At the time of writing this is still a “member submission”, but appears stable enough for production use.

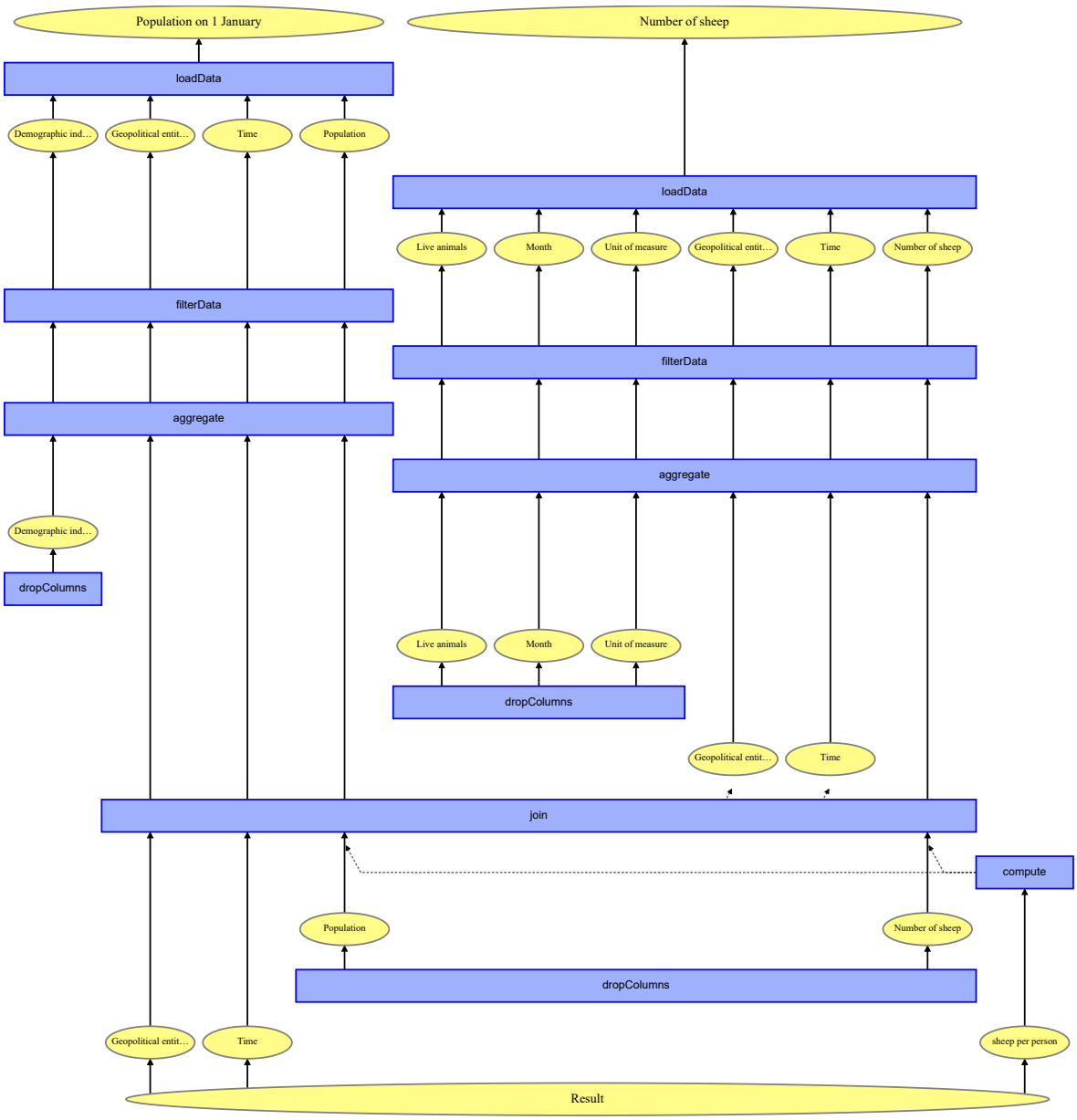


Figure 13.8: Yavaa: Example workflow.

An example for an activity is given in Listing 13.9. Here, the generic provenance information is the start and end time for the execution of this activity. Further, it contains the particular command (`yavaa:action`) and parameters (`yavaa:params`) that triggered this activity. The format of both command and parameters follow the same conventions as the protocol outlined in Section 13.6. The property `yavaa:columns` describes the provenance for the associated columns. In particular, this includes the position of a column within a dataset (`order`) to allow for a

```
{
  "entity": {
    "_:source6": {
      "prov:atLocation": "http://ec.europa.eu/.../BulkDownloadListing?file=data/tps00001.tsv.gz",
      "yavaa:type":      "text/tsv",
      "yavaa:datasetId": "http://yavaa.org/ns/eurostat/dsd#tps00001",
      "dct:publisher":   "http://yavaa.org/ns/Eurostat",
      "dct:title":       "Population␣on␣1␣January"
    },
    "_:result7": {
      "prov:type": { "$": "prov:Collection", "type": "xsd:QName" },
      "yavaa:intermediateResult": true
    },
    "_:column3": {
      "dct:title": "Geopolitical␣entity␣(reporting)"
    }
  }
}
```

Listing 13.8: Yavaa: Serialized provenance record - second level entities.

```
{
  "activity": {
    "_:comp7": {
      "prov:startTime": "2021-04-25T18:50:01.382Z",
      "prov:endTime":   "2021-04-25T18:50:01.414Z",
      "prov:type":      { "$": "yavaa:comp", "type": "xsd:QName" },
      "yavaa:action":   "dropColumns",
      "yavaa:params":   "{\"columns\":[0,1,2]}",
      "yavaa:columns": "[{\"former\":3,\"basedOn\":null,\"order\":0},{\"former\":4,
                        \"basedOn\":null,\"order\":1},{\"former\":5,\"basedOn\":null,\"order\":2}]",
      "yavaa:prevActivity": [ "_:comp8" ]
    }
  }
}
```

Listing 13.9: Yavaa: Serialized provenance record - second-level activities.

consistent interpretation of activities' parameters. Finally, `yavaa:prevActivity` points towards the preceding activity. While this is not strictly necessary and can be derived from the provenance graph itself, it eases the re-enactment of this workflow as discussed later.

Finally, all entities and activities are linked via different relations. Examples are given in Listing 13.10. The particular meaning of each relation was previously described in Chapter 12.

The presented provenance record can also be interpreted prospectively. Users may upload the serialization of a previous workflow and have it executed once again possibly using updated data. Here, the system first parses the uploaded workflow and reconstructs the order of activities or rather operations. Using the `yavaa:prevActivity` of each activity simplifies this process quite substantially as there is no need to derive the order from dependencies among the activities. Once the sequence of operations is determined, the workflow can be executed once again. As both command and parameters already follow the same format as the worker protocol (cf. Section 13.6), no further transformations of provenance records are required.

```
{
  "used": {
    "_:used2": {
      "prov:activity": "_:comp0",
      "prov:entity":   "_:result1"
    }
  },
  "wasDerivedFrom": {
    "_:wasDerivedFrom36": {
      "prov:generatedEntity": "_:column6",
      "prov:usedEntity":      "_:column0"
    }
  },
  "wasInfluencedBy": {
    "_:wasInfluencedBy37": {
      "prov:influencer": "_:column3",
      "prov:influencee": "_:column6"
    }
  },
  "hadMember": {
    "_:hadMember24": {
      "prov:collection": "_:result12",
      "prov:entity":     "_:column0"
    }
  }
}
```

Listing 13.10: Yavaa: Serialized provenance record - second-level relations.

While the current process described before works well enough for the prototype implementation, it has some shortcomings that may impact an immediate use in production. Most of them are a direct result of the workflow itself not being validated. This should not pose a security risk, though, as the execution of operations is isolated within the worker and individual operations are validated before being executed. However, in case datasets' structures have changed since the initial execution of the workflow, the reenactment might lead to undefined behavior. The reenactment is based on the current state of the metadata description as identified by `yavaa:datasetId` and subsequently retrieved from the metadata store. Should the dataset, e.g., not contain the same columns as before, some operations might be applied to the wrong columns and thus yield unexpected errors. While this could be remedied by verifying the column headers upon loading, it was considered out of scope for this prototypical implementation.

13.9 User Interface

Alongside the worker, a corresponding graphical user interface was developed. This interface provides access to most[23] functionalities using a browser via a single-page application. The individual elements were built following the recommendations and examples provided by Google's Material Design [web157]. Figure 13.9 shows the interface with multiple datasets already loaded. Individual functions are controlled via corresponding dialogs which are reachable either via the

[23] Some functions are used primarily during debugging and are not exposed to common users.

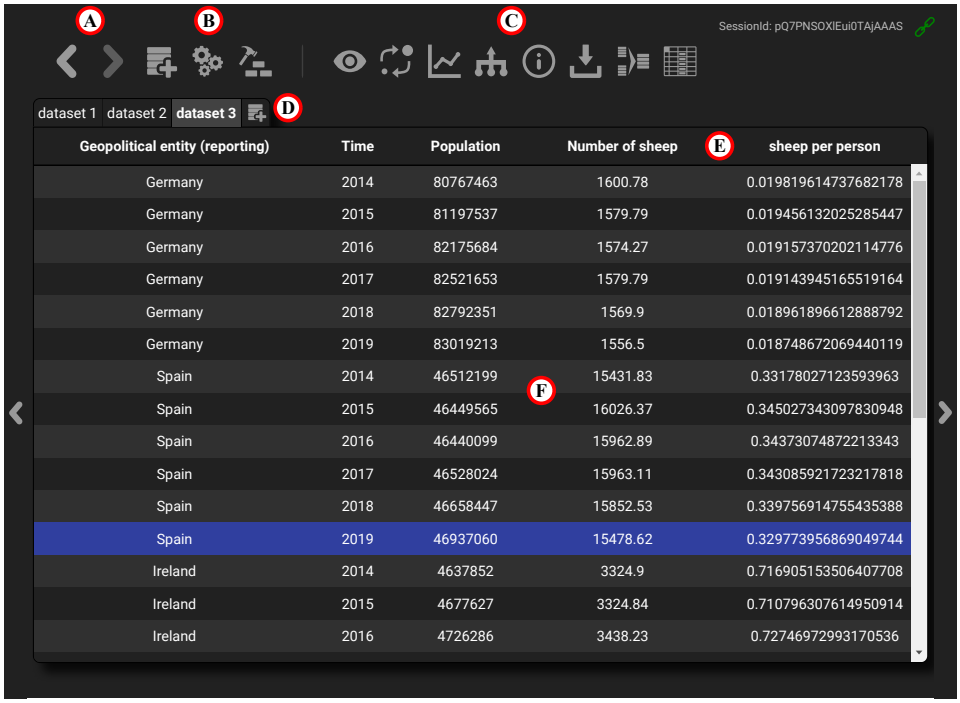


Figure 13.9: Yavaa user interface. (A) undo/redo actions; (B) new datasets; (C) dataset-specific tasks; (D) active datasets; (E) dataset headers; (F) dataset contents

menu on top or context menus available from different components. In the following, the general interface as well as selected dialogs shall be described. For dialogs not discussed here, kindly refer to Appendix H which includes an additional selection of screenshots drawn from the user survey (cf. Section 14.2).

The main interface is separated into three parts (cf. Figure 13.9): The top row, (A) to (C), contains a menu that lets users access individual dialogs. Below is the list of current datasets that were already loaded into the application (D). The remainder of the interface, representing the majority of the available screen estate, is devoted to the actual dataset, (E) and (F). Hereby, the dataset can be represented in different forms. The example of Figure 13.9 features the *Data View* that allows inspecting the current state of the primary data. It follows the default style of tabular display including column headers (E) as well as table content (F). The two other options are a *Workflow View* (cf. Figure H.16) outlining the provenance of the current dataset and the *Visualization View* (cf. Figure H.15) containing the selected visualization.

The top menu itself is again subdivided into three sections. The first section (A) allows traversing the history of the current dataset. In the current implementation, all prior versions of a dataset are still available. The second section (B) contains links to dialogs used to load datasets by different means. From left to right those are as follows:

Load Dataset Datasets can be loaded by using either a keyword-based search or an identifier-based search (cf. Figure H.3). The latter is expected of little use to most users, but allows to skip the actual search in case the required dataset's identifier is already known.

Execute Workflow. The provenance of a particular dataset can be exported and stored inside a file (cf. Section 13.8). This option provides the inverse functionality: It requires a previously created provenance file as input, executes the included workflow based on current data, and adds the resulting dataset to the interface.

Construct Dataset. The required dataset can be posed as a specification of its structure and contents. This implements the approach proposed in Chapter 10. Details of the corresponding interface will be discussed later in this section .

While the second section contains global functions independent of particular datasets and is always available, the third section (C) is dataset-specific and thus only available once at least one dataset has been loaded. All options here apply to the currently active dataset, highlighted in (D) and currently shown in the main content area. Some of the options may also be selected from context menus attached to either the dataset headers (D) or the column headers (E). Dataset specific dialogs include the following dialogs, listed again from left to right:

Change View. As noted before, a dataset can be seen through different views. This dialog as well as the arrows on the outer sides of the main content area allow switching among these views.

Resolve Label. Initially, values for categorical columns are displayed by the label as used within the source dataset. If these are abbreviations, this option allows to expand them into more user-friendly labels (cf. Figure H.6).

Visualize Dataset. With the help of the Visualization Recommender (cf. Chapter 11), this dialog allows users to select and render a visualization matching the current dataset. Details will be discussed later in this section .

Aggregate. The tuples of a dataset can be clustered and merged, in essence representing the `Group By`-statement available in most database systems (cf. Subsection 13.3.2).

Show Metadata. This dialog provides an overview of the current dataset's data. It features each column as well as the currently contained values either as a list (categorical) or a range (time and quantitative).

Export. The current dataset can be exported in different forms (cf. Figure H.17). This currently includes the following: the dataset itself formatted as a TSV-file, its provenance record either as a JSON-based serialization or in a visual display (cf. Section 13.8), and its currently selected visualization, if available, in form of an SVG-file.

Join two Datasets. Oftentimes individual datasets do not hold all required data. This dialog offers the option to join two datasets based on columns shared between both (cf. Subsection 13.3.3 as well as Figure H.4 and H.5).

Apply Function. Based on the current datasets users are able to apply mathematical functions. The result of these functions may either replace one of the current columns or can be added as a new column (cf. Figure H.11).

The context menus associated to individual columns feature some more dialogs. In general, they are available for all columns. The only exception is *Change Unit* which is only available for quantitative columns. The remaining dialogs are as follows:

Drop Column. Columns might become superfluous during a workflow. For the clarity of the interface, this dialog allows dropping individual columns from a dataset (cf. Figure H.9).

Filter Column. To limit the content of a dataset to the actually required data, datasets can be filtered by using the value of individual columns[24]. The dialog's appearance is adapted to the selected column's datatype distinguishing between categorical (cf. Figure H.7) and time/quantitative columns (cf. Figure H.8). Filters can be either excluding or including, i.e. users either specify the values to keep or the values to remove. While this has no impact the result in the current workflow, it might change behavior when re-executing it later on. In the meantime, new tuples might have been added to the dataset, so excluding or including filters determines how to handle those new entries.

Change unit. Values of quantitative columns might be unwieldy in the source dataset or become so during the data modifications. This dialog allows switching the unit of measurement used for the selected column. It will present users with a list of compatible units fetched from the Unit Store and initiates the conversion once a target unit has been selected.

In the remainder, two dialogs shall be presented in more detail. These two represent access to implementations of two major contributions of this thesis: The *Construct Dataset* dialog serves as a frontend to the implementation of concepts previously discussed in Chapter 10. Furthermore, the *Visualize Dataset* dialog supports users in selecting a suitable visualization and corresponding binding by using the recommender described in Chapter 11.

The initial view of the *Construct Dataset* dialog is shown in Figure 13.10. The dataset definition view is vertically split into two parts: On the left side (A), column headers are specified, whereas on the right-hand side included values can be described (B). Consequently, each horizontal row defines one column for the target dataset. In the example shown, four target columns are already defined: one categorical (C), one time (D), and two quantitative columns (E). Columns are

[24] Filter conditions based on multiple columns are supported by the worker, but no corresponding interface has been developed yet.

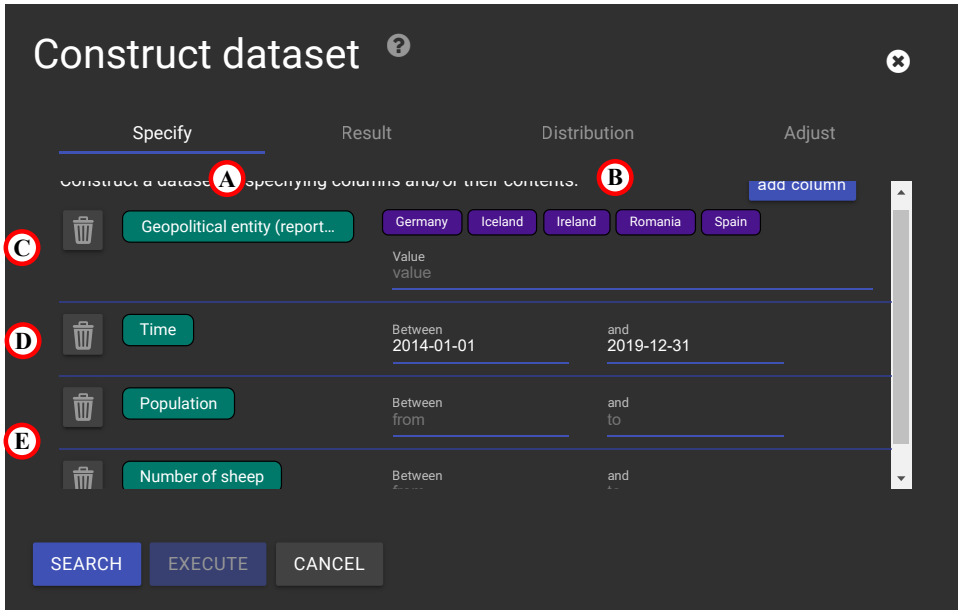


Figure 13.10: Yavaa user interface: Construct Dataset dialog. (A) column headers; (B) column values; (C) categorical column; (D) time column; (E) quantitative columns

selected via an autocomplete field whose options are drawn from the set of all columns available in the dataset repository. So, users start entering keywords and then select from a list containing the most likely known matches.

Once the column is selected, its data type determines the shape of the input for the corresponding value range. For categorical columns (C), this is another autocomplete field that is now bound to possible values from the selected column. As the particular instance of the column, i.e. its incarnation within a particular dataset, is not yet determined, values will be drawn from all possible datasets. On the other hand, time (D) and quantitative columns (E) provide corresponding inputs to define the range by lower and upper bound. It is also possible to specify no further restrictions on a column, which will match any column of that type during the search process.

After the search has been executed, two more views, *Result* and *Distribution*, of the dialog are enabled. The third, *Adjust*, only becomes available if the suggested combination involves an aggregation over several columns and allows users to choose appropriate aggregation functions. The Result view shown in Figure 13.11 summarizes the efforts by the system to provide a complete dataset. As outlined in Chapter 10, this result might be the combination of several datasets. For further insight, the Distribution section shows which datasets by which provider were included (cf. Figure H.20).

The overall layout of the Result view mirrors the one of the specification view: One column per row with column headers on the left side and corresponding value ranges on the right side. However, this view does not offer interaction elements to further refine the result. Instead, it visualizes for each column whether it is included in the result and if so with which values. For

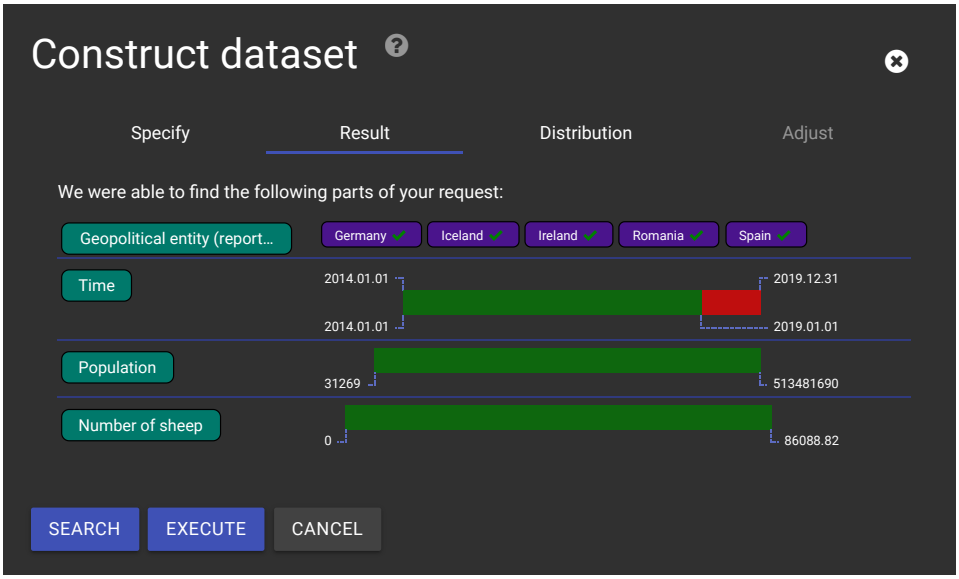


Figure 13.11: Yavaa user interface: Results of Construct Dataset.

categorical columns, the latter is indicated by an icon after the respective value: a checkmark for included values and a cross for omitted ones. Similarly, time and quantitative columns' coverage is visualized in a bar with green sections for the matched range and red sections for the remainder. The borders of both ranges are further given by their respective values next to the bars. If the suggested result does not meet expectations, users can refine their request by returning to the specification view. Once the result is acceptable, a click on *Execute* will trigger the system to run the underlying workflow and present the result (cf. Figure H.10).

With a prepared dataset, users may select a suitable visualization from the *Visualize Dataset* view. Figure 13.12 shows an example of the first step hereby. Here, users may select among visualizations that may be used to represent the current dataset. Following the strategies discussed in Chapter 11, this includes only those visualizations whose requirements can be met. Others are omitted from this view. Figure 13.12 includes examples for possible recommendations made by the system. Visualizations may be recommended without further changes needed (A). Here, all available columns in the dataset could be matched to one of the requirements of the visualization. Visualizations might be suitable but can not cover all available columns (B). They are still suggested, but icon in the lower right hand corner indicate how many columns may need to be omitted. Finally, nested visualizations might be suggested (C) where some of the dimensions may be mapped to rows and/or columns in a matrix representation. Some combinations for nested visualizations might still require columns to be omitted from the result (D).

With one visualization selected, users proceed to the second step illustrated in Figure 13.13 to assign individual columns to the artifacts of the visualization. In the upper section (A), all columns of the current dataset are listed. Individual colors indicate a column's role: dimension or measurement. The lower section includes one row per artifact of the visualization. Using

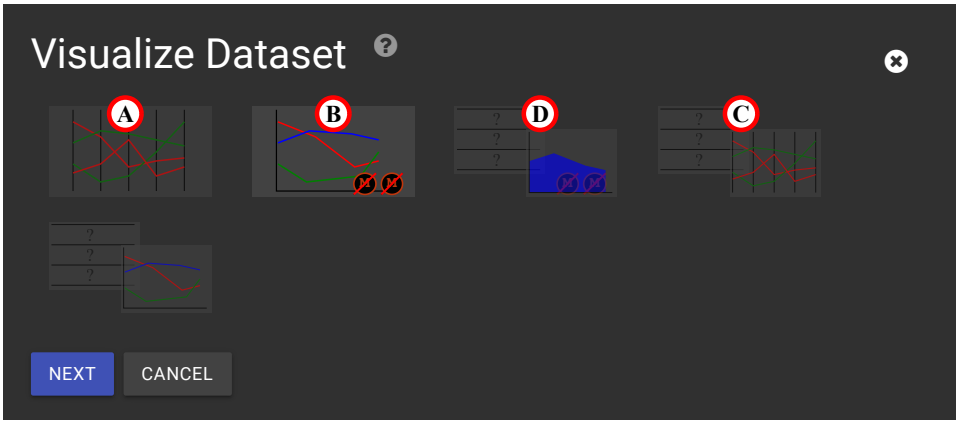


Figure 13.12: Yavaa user interface: Selecting a visualization.
Recommendation types: (A) plain vis.; (B) vis. with changes needed; (C) nested vis.; (D) nested vis. with changes needed

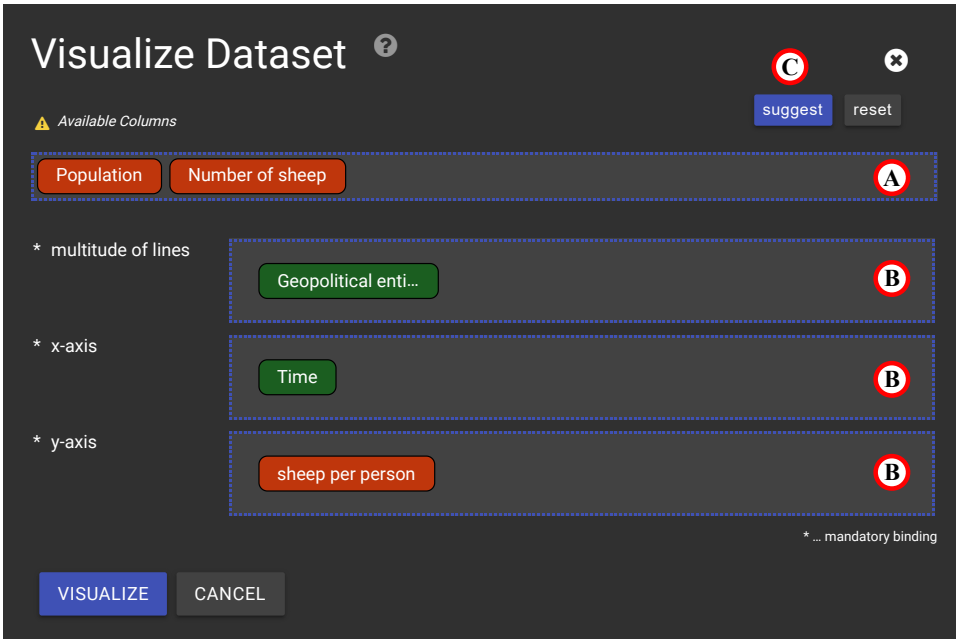


Figure 13.13: Yavaa user interface: Visualization specific bindings.
(A) available columns; (B) visualization component; (C) recommended mapping

drag&drop interactions, users can assign each column to a corresponding artifact of the visualization. During that, borders of hovered drop areas (B) turn green for suitable mappings and red in case column and artifact are incompatible. Instead of providing the mapping manually, users may also ask the system to suggest a suitable mapping (C) (cf. Chapter 11). When all mandatory mappings (marked by a $*$ next to the label) are assigned, the *Visualize* button is enabled and users can trigger the rendering of the selected configuration.

13.10 Summary

This chapter introduced the overall structure of Yavaa. As a prototypical implementation, Yavaa demonstrates the feasibility of the concepts presented in this thesis (cf. Part II). Yavaa's interface allows users to cover the entire visualization workflow[25] from within a single tool. This includes the ability to search across multiple data providers, adapt the data to the particular needs, and finally visualize the result. Over the entire course of the process, all interactions are recorded as part of the provenance record for the final results.

The implementations is fully based on web technologies — HTML [web140], CSS [web141], and JavaScript [web142] — allowing access via broadly available web browsers. Further, the message-based architecture allows to switch between two modes: In a server-based setup, the heavy lifting of operations is done on dedicated machines and browsers act as mere interfaces. In a purely client-based setup, all calculations are performed in the browser and workflows can stay completely private if needed. During its design, special attention was given to the extensibility of the system. Parts like visualization modules or wrappers for accessing data providers can be developed almost independently from the core system and are made available through registration in so-called stores. This allows the system to rather quickly be extended or adapted without the need to access other parts of the code.

Implementing all core concepts of this thesis, Yavaa lays the ground for an evaluation of these concepts. Its preparation, conduct, and results shall be described in the following Chapter 14.

[25] The reference model of such a workflow is depicted in Figure 1.1 and was discussed in Chapter 1.

CHAPTER 14

EVALUATION

Previous chapters outlined core concepts to ease individual tasks during visualization workflows and discussed the validity of the chosen approaches. A developed prototype also showed that those concepts and methods can be transferred into a working implementation. What remains is an evaluation of whether users perceive an actual benefit over existing solutions used to perform comparable assignments. This chapter will provide this last piece of the puzzle.

The primary goal, as stated in Objective 1, is to support users in the process of creating a visualization as a whole. So a natural way of assessing the presented system is to evaluate it in a common workflow and to compare it against other tools that are usually used in similar scenarios. Assuming that the target audience has only rudimentary experience with scripting languages in general and visualization software in particular, the comparison will use standard spreadsheet software like LibreOffice [web17] or Microsoft Excel [web31] as a baseline.

Besides evaluating the user experience of the developed prototype, also its performance with respect to technical aspects shall be analyzed. This includes in particular the time it takes to execute a sample workflow. However, such measures are difficult to gather for monolithic software like the spreadsheet tools of the user evaluation. So for the technical evaluation, different solutions are chosen: As an overall baseline for performance a relational database management system (RDBMS), SQLite [web158], will be used. Similarly, representing scripting languages, the workflow will be executed through a Python/pandas script [web48].

The following sections describe the individual steps necessary: Section 14.1 outlines the efforts to replicate a realistic environment using data provided by Eurostat [web1]. Afterward, Section 14.2 presents a fictitious scenario as well as anticipated strategies to solve it. The results of the subsequent user evaluation are discussed in Section 14.3. Finally, Section 14.4 is dedicated to exploring Yavaa's performance using the same scenario.

14.1 Evaluation Setup

Providing a somewhat realistic environment requires a sufficiently large corpus of datasets annotated according to the specifications of Chapter 8. This corpus has to be publicly available in order to allow all tools within the comparison equal access to the raw data. Eurostat [web1] fulfills both these requirements: A multitude of statistical datasets about the situation in European Union member states and other selected countries is published in regular intervals[1]. Datasets can be accessed via a bulk download facility [web159] that provides TSV [web32] files for each dataset in a Pivot table format (cf. Chapter 6). The bulk download facility also includes metadata descriptions for said datasets: Column headers and categorical values are codified and can be resolved using a corresponding dictionary. Each dataset represents a particular combination of dimensions and a single type of measurement given in the dataset's title. Similar measurements aggregated for different combinations of dimensions are stored in separate datasets. In these cases, the name of the dataset will oftentimes include a list of used dimensions like "*Population on 1 January by age, sex and NUTS 2 region*" [data3]. Unfortunately enough, there does not seem to be a compulsory naming scheme. As a consequence, additional preparations became necessary to harmonize datasets' labels, details of which will be described later.

In the past, projects already attempted to convert Eurostat data to RDF, but they seem to have gone either dormant [web160] or do not provide sufficient details [web161]. These efforts focused on converting the datasets' primary data to RDF. However, for the purpose of this evaluation suitable metadata descriptions were required which, as a result, had to be generated anew. A workflow was set up to automatically create descriptions based on the current set of Eurostat data holdings[2]. Intermediate results are held in an SQLite database [web158], so subsequent runs of the workflow only need to process updated datasets. The workflow was implemented with the help of Maximilian Stiede who improved an earlier implementation during an internship. The implementation is published under a free license [data17, web162] and may serve as a template for the integration of other data portals.

Retrieve a list of changed datasets. Eurostat supplies an XML file[3] that lists all available datasets alongside some basic metadata. In particular, it contains the time of the most recent update for each dataset. This information is extracted and compared to the respective entry stored in the database to gather a list of datasets that changed since the last execution.

[1] At the time of writing, over six thousand datasets are publicly available with updates provided in 12h intervals [web30].

[2] The workflow is not supposed to run continuously but only takes a snapshot of Eurostat's data holdings. An initial snapshot was performed on 2nd February 2020. An update became necessary on 21st May 2020 due to Eurostat adding new entries in the datasets used in this survey as a consequence of Great Britain leaving the European Union [286].

[3] `http://ec.europa.eu/eurostat/estat-navtree-portlet-prod/BulkDownloadListing?file=table_of_contents.xml`

Retrieve and parse datasets. All updated datasets are downloaded and parsed using the respective wrapper that is also part of the Yavaa implementation. The result is a plain table format that allows extracting the range for each column easily. For quantitative columns, the range corresponds to the minimum and maximum value, whereas for time[4] and categorical columns an enumeration of distinct values is extracted.

Harmonize measurement headers. As mentioned before, some datasets' titles include a list of dimensions used within. Those suffixes are removed by a simple heuristic that detects phrasings starting with certain key-phrases like "by" or "between" as well as some special characters like ":" and "-". This harmonized dataset title is then assigned as the label of its respective measurement column.

Annotate with units of measurement. All quantitative columns are annotated with a matching unit of measurement which is derived by different means depending on the dataset: Some datasets contain a specific column named `unit`. If such a column exists, the respective unit is used for the measurement column in that dataset[5]. Furthermore, the aforementioned XML file contains units of measurement for a subset of its entries. If the entry for a dataset has such an annotation, the respective unit is added as well. Finally, some datasets' unit is assigned manually to increase the number of available datasets during the evaluation.

The units determined this way are subsequently mapped to the concepts of OM [164]. However, OM does not include fitting concepts for all units. In particular, units regarding the number of entities or objects are missing. Individuals for those units have been added where needed in accordance with the schema used by OM.

This mapping remains incomplete for different reasons. For one, some extracted units remain too vague to be mapped to a specific concept. For example, the same unit "Annual average" is used for two datasets "*Hours worked per week of part-time employment*" [data18] and "*Unemployment rates of the population aged 25-64 by educational attainment level*" [data19]. However, it is obvious that both actually relate to different unit concepts which prevents a generalized mapping here. Datasets, whose units could not be mapped, were dropped from the evaluation corpus (cf. Table 14.1). In other instances, a pragmatic approach for the mapping is employed: While Eurostat uses different units for counting different things like persons, employees, or farms, those have been assigned to the same unit. It might be worth discussing, whether those are actually the same unit[6], but it seems sufficient for the purpose of this evaluation.

[4]Time columns are recognized by their header "time", which is uniformly used throughout all datasets.

[5]Some datasets contain more than one value in the `unit`-column. As this does not conform to the assumptions made in Chapter 8, no unit is extracted for these datasets.

[6]One advantage would be, that different units prevent the aggregation of unrelated objects like persons and farms. However, this opens the issue, which objects are actually compatible, and entails questions far beyond the scope of this thesis.

Determine date format. Values contained in time-columns come in different formats and granularity. Examples include years like `2015`, months like `2011M04`, or quarters like `2012Q3`. Furthermore, periods of time can be included like `2011-2018`. To determine which date format is used by a particular column, for each encountered format a regular expression was created. Using these expressions all time-columns are annotated with a concept representing the respective format[7]. If multiple different formats are detected within a single time-column, the corresponding dataset is dropped. Similar to quantitative columns, time-columns are annotated with their minimum and maximum value.

Convert column headers and categorical values to concepts. The headers for all columns are converted to RDF concepts [web10] using a camel case conversion of the respective title and a default namespace. Mapping to other knowledge bases like Wikidata [web69] or DBpedia [web163] is considered out of scope for the time being. While this would prove vital to integrate datasets of different providers, it is not strictly necessary to evaluate the implemented concepts. The same procedure is applied to the values of categorical columns. Further, all concepts created are annotated with the labels from the dictionaries of Eurostat's bulk download facility [web30].

Creation of code list. Similar to quantitative and time columns, categorical columns are annotated with a list of included concepts. In a naïve implementation, this would result in numerous identical code lists being stored which results in an unnecessary overhead within the database. To reduce this redundancy, identical code lists are subsumed into a single code list entity with all associated columns referring to the same one.

RDF creation. All entries in the database — i.e. datasets, columns, code lists, and categorical values — are converted to RDF using a set of prepared templates. Each template represents a part of the dataset description proposed in Chapter 8. Finally, all created RDF alongside other predefined RDF datasets like the OM ontology are pushed to a triple store that serves as the data source for Yavaa's Metadata Store.

The described process adds the datasets available in Eurostat's bulk download facility [web159] to Yavaa's dataset repository. As already mentioned on different occasions, datasets had to be excluded from this import for several reasons:

- Datasets containing a measurement provided in multiple units of measurement.
- Datasets containing units of measurement that could not be mapped to OM and were not relevant for the evaluation.
- Datasets containing time-columns using multiple time formats.

[7]The RDF files describing the respective date format were manually prepared. The format is illustrated in Figure 8.8.

- Datasets that could not be downloaded or parsed in the first place.
- Datasets that would require further manual corrections, but are not relevant for the evaluation task.

Table 14.1 lists the number of remaining entities imported into the Metadata Store and thus available throughout the evaluation. This also documents the fact that not even half (~47%) of the datasets from Eurostat were included in the evaluation. However, this does not imply that all other datasets are unusable for Yavaa in general. In fact, many of these datasets could have been included given additional manual effort. For the scope of this evaluation, this was deemed unnecessary, though, and thus omitted.

Number of datasets available from Eurostat	6,283
Imported datasets	2,943
Imported codelists	6,182
Imported distinct dimension concepts	381
Imported distinct measurement concepts	1,369
Imported distinct value concepts	197,301
Imported distinct value concepts (used)	81,107
Imported time formats	4

Table 14.1: Yavaa Metadata Store (Evaluation): Number of entities.

14.2 User Evaluation

The efforts outlined in Section 14.1 resulted in a repository of real-world datasets ready to be used within a user survey. Now, a scenario has to be established that finds the right balance between covering as many aspects of the supported workflow as possible and, at the same time, staying simple and concise enough to be easily understood. The latter requirement serves the following two goals in particular: For one, the audience is envisioned to consist mostly of non-experts, so complex scenarios involving many unfamiliar concepts might distract from the actual task. Furthermore, complex scenarios require more time and effort to complete the survey up to the point when this has a negative impact on participation and completion rates.

The desired scenario's subtasks are given by the generic workflow depicted in Figure 1.1 in general and the contributions of this thesis in particular. As interactive visualizations are out of scope, this results in four subtasks to be performed: First, participants should locate multiple datasets and integrate them into a single one for subsequent processing (cf. Chapter 10). Second, this dataset should be transformed to be ready for visualization (cf. Chapter 9). Third, a suitable visualization should be created (cf. Chapter 11). Finally, the results, both in form of a dataset and a visualization, should be exported (cf. Chapter 12).

> As part of a study concerned with public health, you are tasked to look into the quality of sleep. As the primary indicator the number of available sheep per person is chosen. (Nothing is worse than trying to get to sleep and running out of sheep to count, right?) To eliminate short-term effects, you are interested in the development over the last 5 years. This will also show you, which countries actually make an effort to improve their citizens nightly life!

Figure 14.1: User evaluation: Scenario.

Next, a choice about the baseline has to be made that the developed systems is evaluated against. The almost ubiquitous presence of office suites like Microsoft Office [web164] or LibreOffice [web165] has also established their respective spreadsheet components as common tools to create ad hoc visualizations. The resulting widespread familiarity with these tools triggered the decision to use them as a comparison baseline. However, spreadsheet software, in general, does not feature a built-in search for data repositories. So for the discovery phase of the workflow, participants will have to rely on the standard search capabilities provided by Eurostat [web1]. As a limitation, spreadsheet software usually does not track the provenance of the performed actions. So any comparison of this aspect has been dropped within this evaluation.

These constraints, requirements, and the chosen baseline, lead to the scenario as given in Figure 14.1. In an attempt to reduce sequence effects, a within-subject design with counterbalancing was chosen [287]: Each participant is asked to solve the tasks given in Figure 14.2 twice – once using the spreadsheet software and once using Yavaa (cf. Chapter 13). The order of tools is randomized to account for possible sequence effects like an increased familiarity with the task or fatigue effects over the course of the study. Early tests revealed that some participants were unable to locate the proper datasets. As this would have prevented any further evaluation of the workflow, the accompanying survey offers additional hints in such cases (cf. Figure G.6 and G.12).

After each tool, participants are asked to upload both the final dataset as well as the created visualization. For each individual task, the overall time spend is measured and participants are requested to assess the approximate distribution of time spent for each subtask. This allows for a comparison between the two passes. In Yavaa, all actions are logged alongside the respective timings, but there is no comparable data available from the spreadsheet tools. Subsequently, participants are asked to rate the difficulty of each subtask and the overall usability of the respective approach using a standard System Usability Scale (SUS) questionnaire [288, 289]. The survey concludes by gathering demographic data about the participants as well as their former experience with the topic. A full account of the survey is given in Appendix G.

For the assigned task, two strategies were anticipated as illustrated in Figure 14.3. While the visualization step itself is shared between both strategies, the steps needed for data preparation differ: The conventional approach involves locating the data manually, followed by multiple steps to prepare its final form. In the enhanced approach, the actual data gathering is mostly

Your task is to create a dataset that holds the amount of sheep per inhabitant for the following European countries (the shortlist of vacation destinations of your superior - purely coincidental, of course) and period of time (previous five years):

- Countries: Germany, Iceland, Ireland, Romania, Spain
- Period of time: 2014 - 2019

After the dataset has been assembled, choose an adequate graph to present your results to your fellow colleagues and the general public. The suggested order of steps is as follows. Your personal workflow might deviate, though.

1. Identify suitable datasets.
   While in general Eurostat has all the data you need, it is not provided as a single dataset to start with, so you will need to combine multiple ones.
2. Prepare a single dataset.
   Eurostat's datasets contain more data than needed, so you will have to filter for the requested values. You may also need to join multiple source datasets.
3. Calculate the desired metric.
   The requested metric is not included in Eurostat's raw data, so you will have to calculate it manually.
4. Select a proper visualization.
   Once the dataset contains only the requested values, you can choose a suitable visualization.
5. Export your results.
   Store your results (data and visualization) locally and then upload them on the next page.

Figure 14.2: User evaluation: Task description.

handed off to the system and only the final creation of the derived column (sheep per inhabitant) requires manual effort. The baseline toolchain (Eurostat web-search and Excel or LibreOffice) only supports the conventional approach, whereas Yavaa allows for both routes to be taken. The individual steps necessary for both strategies using Yavaa are documented in Appendix H. Participants were provided with a tutorial that illustrated all required functionalities [data20].

14.3 Assessment

The user evaluation was conducted in an unsupervised, remote fashion. This has two major reasons: First, compared to other evaluations the one conducted takes a rather extensive amount of time. Letting participants choose a time and place convenient to them is supposed to increase

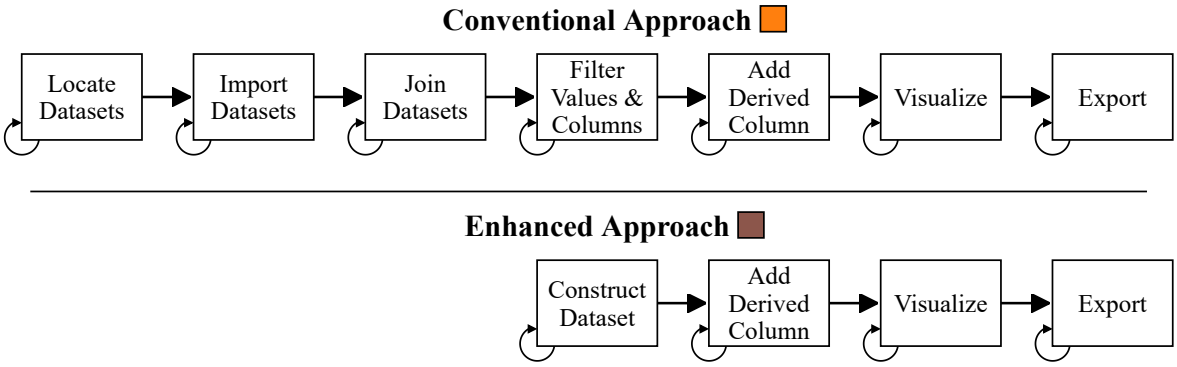


Figure 14.3: User evaluation: Anticipated Strategies.

participation rates. Second, any evaluation with the evaluator present bears the risk of introducing additional bias. This may be due to comments and answers by evaluators themselves or by communication with other participants that might be present. In a remote evaluation, chances are that participants work on their own and thus are influenced less by the decisions and opinions of others.

The survey described before[8] was distributed over the course of several months via different mailing lists, social media channels, as well as personally approaching potential participants. This resulted in well above two thousand individuals being contacted for taking part in the survey. However, over the course of the survey only 16 fully completed responses could be collected from a pool of 92 recorded accesses to the survey. Reasons can only be speculated upon, but discussions with participants after completing the survey point towards the amount of time allotted as the dominating factor. This indicated that either a qualitative, interview-based study or a smaller scenario might have been a more appropriate choice to evaluate the system at hand. While the former has been rejected for the aforementioned reasons, the latter would not have allowed to get an adequate picture of the whole workflow and thus would also compromise the goal of the study.

Participants For the following discussions only complete responses were considered. As stated before, this amounts to 16 entries in total. The raw responses (excluding personal data), submitted files, and derived attributes are publicly available [data21]. The gender ratio was rather balanced with 9 male and 6 female participants (1× no answer). The professional background of participants was mostly in scientific areas: 10 individuals stated computer science, 2 geography, and 3 other or generic areas of science (1× no answer). Most participants hold an academic degree (5× BSc, 4× MSc, 2× Diploma, and 3× PhD) with only 1 individual(s) having not (yet) achieved this (1× no answer). Finally, in terms of age, most participants were in the range of 25 to 34 years (7×), followed by the 35- to 44-year-old (5×), and the ones between 18 and 24 years of age (4×). A visual summary of the participants' demographics is given in Figure 14.4.

[8]For a full account refer to Appendix G.

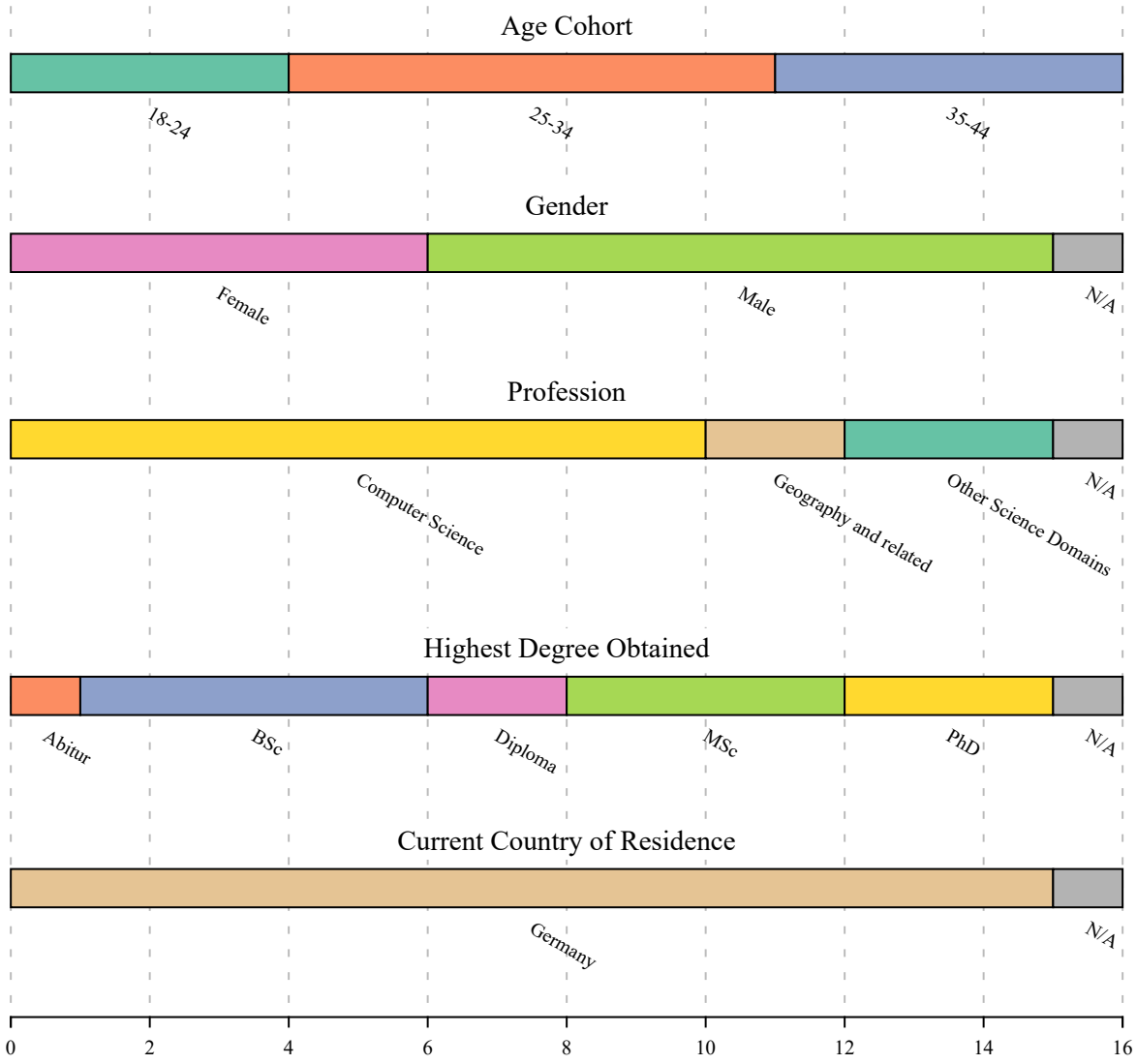


Figure 14.4: Evaluation participants: Demographic composition.

To put the responses into perspective, participants were asked to rate their prior experience in four areas on a Likert-scale [290] ranging from “beginner” over “intermediate” to “expert”. Assessments of language and programming skills were intended to serve as proxies for the general understanding of the task description and overall technical adeptness respectively. Questions about the experience with spreadsheet software and information visualization, on the other hand, are directed at the tasks required throughout the evaluation.

The respective results are illustrated in Figure 14.5. All participants rated their understanding of English as intermediate or better. Similarly, most participants have at least an intermediate understanding of programming. Both outcomes support the assumption that any issues during the evaluation are not caused by participants, but are either rooted in the task description

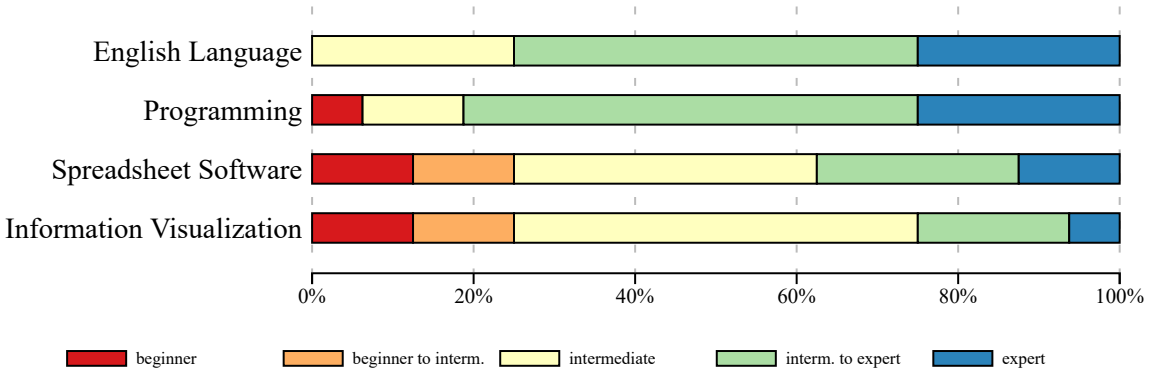


Figure 14.5: Evaluation participants: Prior experience.

or caused by the tools themselves. The self-assessments regarding spreadsheet software and visualization show a rather broad spectrum of prior experience. So despite its small size, the gathered sample seems to represent a rather complete cross-section of potential users.

Dimensions of Evaluation. The actual results of the evaluation will be discussed along the following four dimensions. The first one is a successful task execution: Were participants able to perform the posed task and generate suitable results? After that, the time each of them required will be analyzed. The remaining two dimensions are concerned with the opinions as stated by participants. Users were asked to rate the employed tools with respect to both the perceived difficulty of executing the tasks as well as the general usability of the respective tool. The former attempts to measure the experience needed to successfully command the tools and by extension how suited they are for new users. The latter is concerned with the appropriateness of the provided interface in supporting users in their given tasks.

Successful Task Execution. A prime indicator for the usefulness of a tool is the ability of users to perform given tasks with said tool. However, measuring that success is a rather complicated endeavor. In the following, different criteria are applied to compare the suitability of the evaluated tools. A first measure relies on user interactions during the evaluation and uses two proxies: First, participants could request additional assistance in case they were not able to locate the appropriate datasets (cf. Figure G.5 and Figure G.11). As the spreadsheet tools used in this evaluation do not include search capabilities, this does not compare Yavaa with spreadsheet software but with the default search capabilities of the data provider, namely Eurostat [web1]. A second proxy is given by participants stating that they were unable to complete the task with the given tool (cf. Figure G.6 and Figure G.12) as well as the respective reason. Some also used this opportunity to explain why they were not able to locate the datasets in the first place.

Additional assistance was requested both for Yavaa (1×) as well as Eurostat (2×) with two of these cases including a corresponding explanation. The problem for Yavaa was rooted in technical issues at the point of taking the survey which prevented any successful search due to a crash of

the backend triple store. For Eurostat, one participant stated that searching using the keyword "sheep" only yielded results for 2018 and 2019, but a search for the German "Schafe" would subsequently return the appropriate dataset. Although this behavior could be reproduced neither by using the German keyword nor the English one, this seems to indicate an issue with different language versions of Eurostat's search interface.

Looking at the inability to complete the tasks at all, the picture is more complex. Two participants reported a (partial) failure to finish the task using Yavaa. In one case, the participant was able to assemble the dataset up to the point of creating the derived column. However, the corresponding comment only mentions issues during search as described in the previous paragraph. So any conclusions on possible reasons fall within the realm of speculation. The other participant unable to complete the task, mentions technical problems while filtering the dataset. This issue was caused by a change in Eurostat's schema[9]. This change was not properly reflected in the underlying metadata store which caused problems when trying to remove the values unknown to the system. By updating the metadata store this issue was fixed for subsequent participants. Using spreadsheet software, five users reported problems completing the given task (2× using LibreOffice and 3× using Excel). One participant merely criticized the task definition, but still created results in line with the original intent. Three participants (2× using Libre Office and 1× using Excel) did not find an appropriate visualization but were able to create a suitable dataset. Finally, one participant using Excel reported that they were unable to normalize and subsequently join the data.

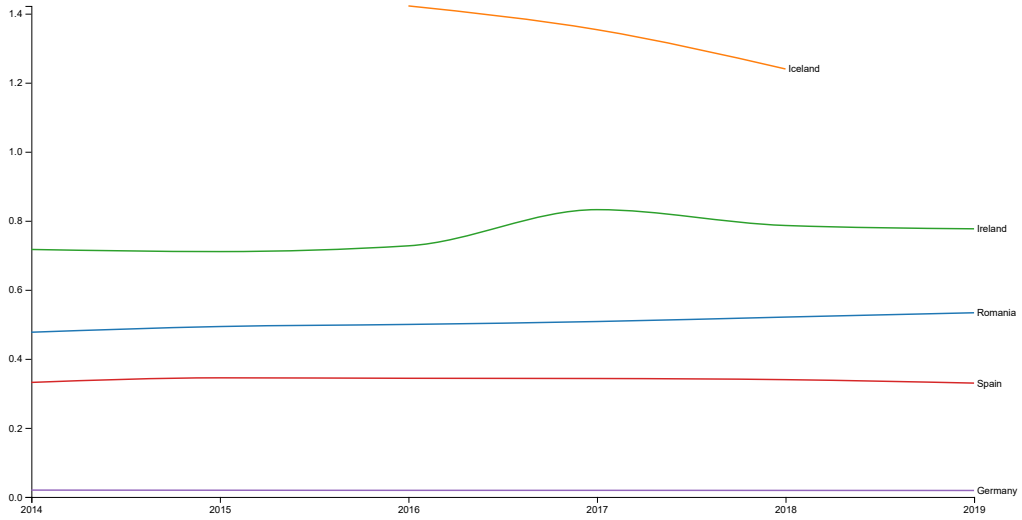
In addition to the previous accounts, one participant did not upload any result for either tool and also did not comment about if or why they were unable to complete the task. However, a comment in the final section of the survey (cf. Figure G.17) suggests they did indeed finish the task. Due to these uncertainties, the corresponding submission is excluded from the discussions here.

Besides the aforementioned proxies, the uploaded results for both visualization and underlying dataset can give insight into the success rate of the task execution. In an attempt to reduce the observer bias, the results were manually evaluated against a list of possible deviations from the sample solution (cf. Figure 14.6). This strategy allows for a more detailed and objective measure, whether the results can be considered successful. The list was extended throughout the evaluation upon encountering new deviation-types. There are fewer deviations evaluated for visualizations compared to the underlying dataset(s) for mainly two reasons: First, many aspects of the visualization depend on the underlying dataset. An assessment of these aspects would only repeat the previous evaluation of the datasets themselves. Second, judging whether a particular visualization itself is suitable for the task can be highly subjective and would require

[9] In Q2 2020 Eurostat added new designations to represent accumulated statistics for the EU as a whole. Those became necessary as Great Britain left the EU earlier that year [286] and subsequently designations for an EU with and without Great Britain became necessary.

Geopolitical entity (reporting)	Time	sheep per person
Germany	2014	0.019819614737682178
Germany	2015	0.019456132025285447
Germany	2016	0.019157370202114776
Germany	2017	0.019143945165519164
Germany	2018	0.018961896612888792
Germany	2019	0.018748672069440119
Spain	2014	0.33178027123593963
Spain	2015	0.345027343097830948
Spain	2016	0.34373074872213343
...	...	...

(a) Dataset.



(b) Visualization.

Figure 14.6: User evaluation: Sample Solution.

an extensive survey of its own which is beyond the scope of this thesis. The prevalences of these deviations are given in Figures 14.7 to 14.9. They are classified into the following three groups based on their impact on the task result.

- *HIGH*-severity deviations render the submitted artifact(s) unsuitable for the given task.
- *MODERATE*-severity deviations violate some of the task constraints, but the artifact(s) can still be considered suitable. In particular, the results include all required facts and no errors. However, there might be additional information added or clarifications needed to interpret the result in the right way.
- *LOW*-severity deviations are mostly cosmetic in nature. The suitability of artifact(s) is not affected.

Hereinafter, the encountered deviations and possible causes shall be discussed based, of course, only on the submitted artifacts. However, some participants did not upload (parts of) their results as summarized in Table 14.2. Only in one case per tool, this coincides with the respective participants stating that they were unable to complete the task. Apart from that, there is no

	No Dataset	No Visualization
Yavaa	2×	3×
Spreadsheet	2×	3×

Table 14.2: User evaluation: Missing artifacts in submissions.

correlation between the reported ability to complete the tasks and the actual submission of the results. Some submissions lack (parts of) the required artifacts without giving a reason, whereas others include all artifacts despite stating the opposite.

The first deviations among those of high severity (cf. Figure 14.7) concern missing data within the dimensions of the assembled dataset. With regard to countries included, one submission substituted a country each in both task executions. As different countries were mixed up, the cause seems to be a lack of focus rather than a systematic error. Another submission included only a single year (2018) instead of the whole range, again for both task executions. Finally, a third submission included only 2017 and 2018 in the spreadsheet-execution. The submitted file seems to be the result of copy-pasting the required values manually, so the cause of incompleteness is likely the repetitiveness of the chosen approach. The next crucial step is joining the required source datasets (population and number of sheep). Here, one submission for the Yavaa used only a single join condition (country) performing a cross join on the other (time). For the spreadsheet-execution, two submissions failed to perform the correct join. The aforementioned submission with swapped countries ignored the country name during the join and seems to have done a purely position-based join. As the switch of countries only appears for one source, this aggregates data for two unrelated countries thus invalidating the respective values. The other submission used a single value per country for the number of sheep therein. This hints towards omitting the year as a proper join condition.

The next subtask involves creating a derived value giving the number of sheep per population. Three submissions for Yavaa did not include any such column with no reason given in the comments, whereas this did not occur throughout the spreadsheet-submissions. But also with the derived column being present, some submissions contained other substantial deviations. Whereas for Yavaa the sole reason was a mix up of operands (2×), the causes using spreadsheet software were more diverse: Swapping operands (1×), using a wrong conversion factor (1×), and an unknown reason[10] (1×) were encountered in the submitted artifacts.

The final step concerns the visualization of the results. Here, major deviations occurred only in the spreadsheet-tasks. One submission neglected the development over time and just visualized the state in one year. Two other submissions represented the data in a way that no longer allows deducing anything meaningful from the visualization. Finally, in three visualizations data points

[10]Here, the submitted values were incorrect, but the process leading up to them remained unclear.

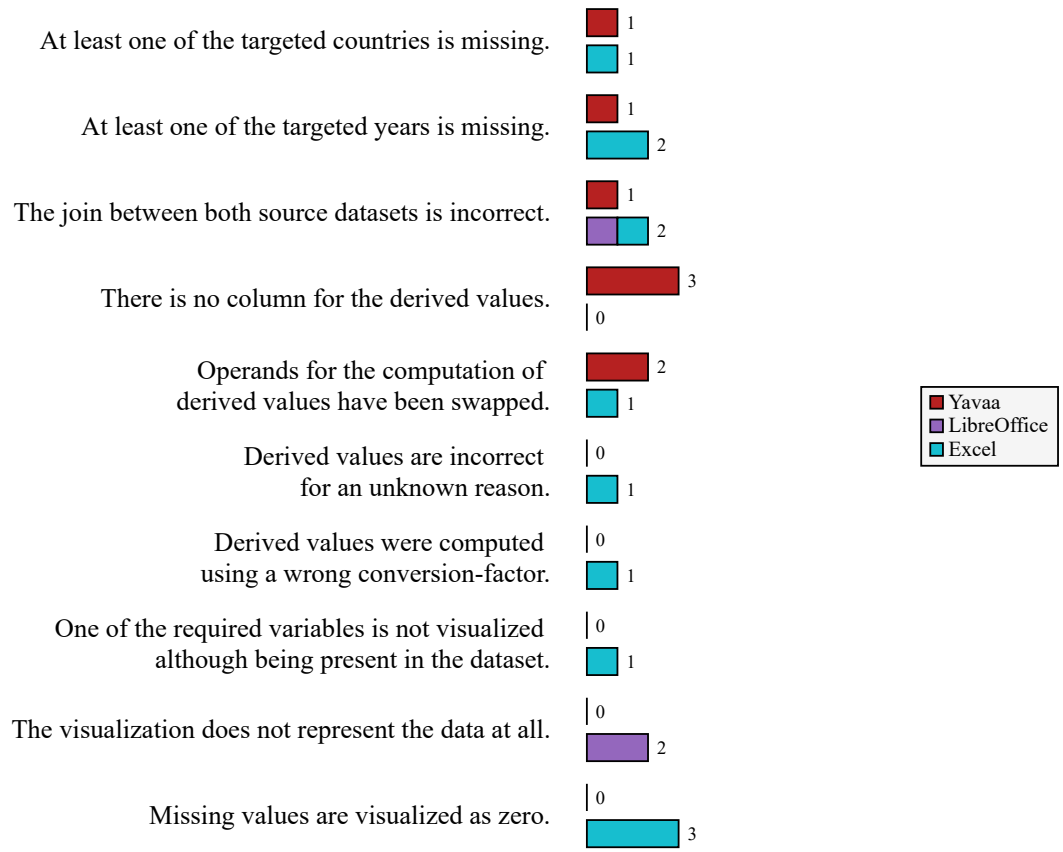


Figure 14.7: User evaluation: *HIGH*-severity deviations from the sample solution.

missing in the dataset[11] were represented as zero in the visualization instead of omitting them in some way. In a standalone visualization without the accompanying dataset, this will easily lead to the wrong conclusion that the values for those year(s) were not missing but were indeed zero.

Deviations of moderate severity (cf. Figure 14.8) can be divided into three groups: additional data, partial completeness, and omitting conversions. Among the occurrences of additional data, the aforementioned submission substituting countries accounted for one case in each Yavaa as well as spreadsheet-submissions. Furthermore, another spreadsheet-solution included a non-required country (Poland). The remaining cases of additional data are all caused by omitting the filter-operations on country and/or time. All issues regarding partial completeness refer to a single spreadsheet-submission: Based on a complete source file for population data, only a subset of entries were augmented with the data requested by the given task. While technically correct, the dataset remains incomplete with the majority of entries lacking most of the data.

The final deviation of moderate-severity is omitting the conversion at all, occurring five times across the spreadsheet-submissions. Contrary to using a wrong conversion factor, this deviation has only been classified as moderate. The task description did not call for a target

[11]The cause is incomplete data in the source files and not the result of a mistake by the respective participants.

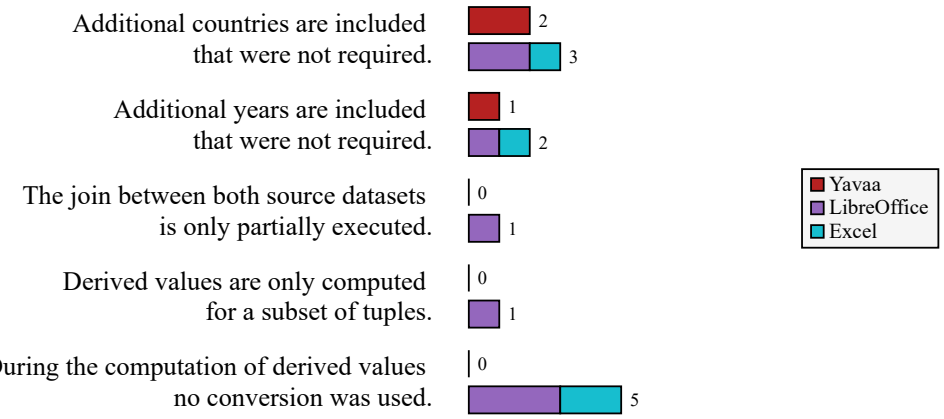


Figure 14.8: User evaluation: *MODERATE*-severity deviations from the sample solution.

unit for the derived column or even mention this issue. Hence, the classification is made under the assumption that the actual unit is mentioned alongside the future uses of the respective visualization, e.g., as part of an image caption. However, the wrong conversion factor mentioned in Figure 14.7 indicates an actual error possibly due to a typo or a flawed interpretation of the source datasets' metadata.

The last class of deviations contains those of low severity (cf. Figure 14.9). In many cases, both the dataset as well as the visualization referred to countries only by their abbreviation (Yavaa 13×, spreadsheet-software 9×). There are two things to note here. First, while Yavaa offers the option to resolve those abbreviations to the full labels, this functionality was neither part of the tutorial nor were participants required to make use of it. Overall this led to only one participant actually applying that feature. Second, the situation seems slightly better for the spreadsheet results at a first glance. However, inspecting the files with resolved country names strongly suggests that the data was manually copy-pasted from the Eurostat website and not via downloading and processing the underlying files. In contrast to the plain files available for download, Eurostat's web-frontend already resolves the respective abbreviations before displaying datasets to its users.

This immediately leads to the next observation. Although spreadsheet-software generally offers ways to automate almost all tasks of this evaluation, quite some participants resorted to manually perform at least some steps. This stretches across most of the data manipulation subtasks: Instead of using the download facilities for the raw datasets, tables were copy-pasted directly from Eurostat's web-frontend (4×). Instead of using the built-in `VLOOKUP`-mechanism, the joins between datasets were performed manually (4×). Instead of applying a formula to compute the number of sheep per population, seemingly the values were computed using a different tool (3×). The numbers presented here only reflect the most obvious cases. If there was any doubt about the process, in an in-dubio-pro-reo fashion the use of all built-in capabilities was assumed. Other submissions might also have neglected these facilities, but the resulting artifact does not

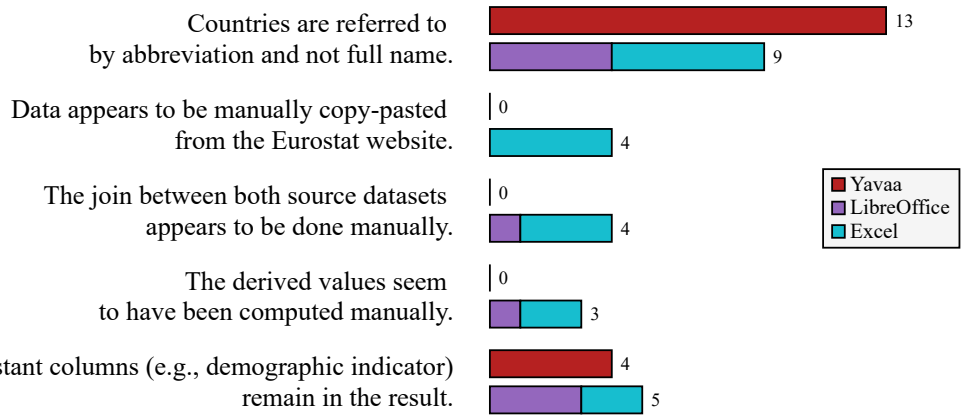


Figure 14.9: User evaluation: *LOW*-severity deviations from the sample solution.

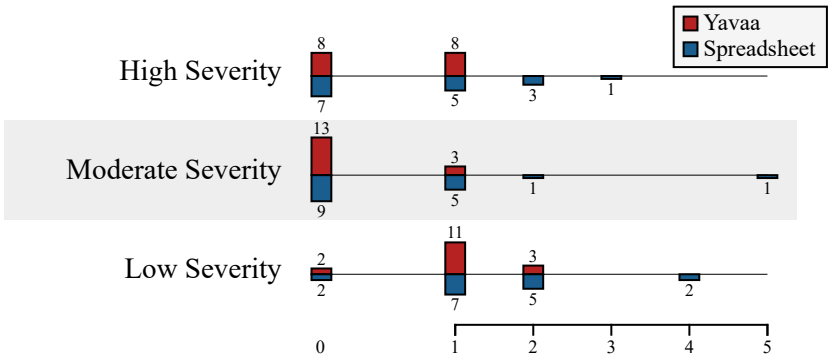


Figure 14.10: User evaluation: Issues per submission by severity.

provide sufficient evidence anymore[12]. Another important aspect is that the individual manual execution of steps does not necessarily entail the same for other steps. Overall, seven submissions indicate the manual execution of at least one step with basically all combinations of the above mentioned being present at least once.

Finally and of least importance, are remnants of the source datasets that are not needed for the visualization. The sources include constant columns, e.g., to state the type or unit of the measurement. In some submissions, these columns were still included (Yavaa 4×, spreadsheet-tools 5×). While they do not have a negative impact on either the process or the result, they constitute unnecessary noise in the artifact and as such should be removed.

Figure 14.10 summarizes the spread of deviations among the submitted artifacts subdivided by their severity. Across all categories, a reduction in deviations per submission can be observed when choosing Yavaa over standard spreadsheet software. This becomes especially noteworthy due to the fact that most participants reported at least an intermediate knowledge of spreadsheet

[12]When a CSV file was submitted applied functions are replaced by the resulting values. As a result, only in rare cases do some indicators of them remain, like a value of `#VALUE!` representing a computation gone wrong.

software while this evaluation was their first encounter with Yavaa. Keeping in mind that the training materials provided for each tool were equivalent, which should favor spreadsheet software considering the prior experience.

Reviewing the deviations and participants' comments, one can speculate about possible reasons and relate them to the tools used. The following discussion will focus mostly on deviations of high and moderate severity, as those impact the results, whereas deviations of low severity merely affect the presentation or process. A first noteworthy set of deviations is concerned with differences between covered dimensions of the results compared to the actual task description. While most participants were able to gather the information in principle, one can observe a number of lacking or added values in some artifacts. This is likely not an issue of the employed tools themselves but can be attributed to negligence on behalf of the participants. Looking at the frequency, using spreadsheet software seemingly increase the chance that values are added or missed out on.

The analysis of submitted artifacts showed that a substantial number of participants disregarded the built-in capabilities during data preparation when using spreadsheet software. Instead, individual values oftentimes seem to be the result of manually copy-pasting them to their intended positions. While not necessarily affecting the results by itself, such an approach may lead to certain mistakes that go unnoticed due to the extent of repetitive tasks. The repetitiveness and the resulting additional effort can result in a kind of fatigue which in turn reduces participants' diligence and causes the observed deviations. As Yavaa by design only allows to edit columns as a whole and thus prevents such manual editing of individual values, this particular reason does not apply here and thus its consequences are less likely to appear.

The second group of deviations is likely caused by shortcomings of the respective user-interfaces. It stretches across two main areas: performing the join of the two data sources and calculating the derived value. In addition to the observed manual copy-pasting of values, in particular, the inability to directly use a multi-column join condition appears to be an obstacle. Yavaa does not suffer from this limitation, but despite a semi-automatic suggestion of a proper join condition, at least one participant also used a single-column-join. This suggests that an additional warning in such cases might be in order.

Using Yavaa, three subjects failed to compute the derived values necessary. As no explicit reasons were provided by those participants, one can only speculate. A possible reason might be issues with the interface to create the derived values (cf. Figure H.11). The list of shortcuts to be used in the function editor is initially collapsed, so users might have not been able to refer to the correct column designations here. Similar problems might also arise in spreadsheet tools, where columns are generally also referred to by rather abstract designations[13]. However, in spreadsheet software they are visible at almost any time, so users are more used to them. On the other hand, Yavaa attempts to use the actual labels of columns wherever possible and only resorts to the

[13]There are means to define alternative designations for columns, but for an ad-hoc task like in this survey, their use seems disproportionate.

abbreviations where necessary. At least in this particular case, this seems to have backfired. Possible changes addressing this could include expanding the list of shortcuts when needed or adapting the function editor to use the same column names as the remainder of the interface.

This might also address the other deviation that can likely be traced to a lack of support in the user interface: the swapping of columns in creating the derived value. So instead of calculating the number of sheep per inhabitant, the affected datasets contain the number of inhabitants per sheep. In general, both values might be valid and a system may not detect which one is more correct. Again, by using only abstract designations for the columns involved, users can easily miss such issues and proceed without further checks. This affects both spreadsheet software and Yavaa equally, although in the survey Yavaa-artifacts suffered from this slightly more often.

A final set of deviations could have been prevented assuming proper checks, support, or automation by the tool used. This includes issues concerning unit-conversions and the observed deviations during visualization. Spreadsheet software offers no support in both areas. It is solely the responsibility of users to make sure their data is consistent. Yavaa takes care of this behind the scenes and thus prevents many such errors. Regarding the visualization, spreadsheet software sometimes falls victim to its own capabilities. In general, there are few constraints about how the data is structured within a worksheet and which parts thereof are used for the visualization. However, this approach also opens up a wide range of possible mistakes. In the survey, this has been witnessed by artifacts failing to represent the data in any meaningful way or missing the distinction between missing values and zero in the visualization. The more restrictive approach used by Yavaa managed to prevent or reduce those issues at least for the given set of tasks and participants.

In summary, the greater flexibility of spreadsheet software sometimes seems to be more of a curse than a blessing in this particular task. It allows users to bypass the intended workflows or capabilities in favor of simpler but also more error-prone approaches. On the other hand, Yavaa does not allow for such improvised solutions and forces its users to adhere to prescribed ways. While this removes an “easy way out” for inexperienced users, it also reduces the number of crucial errors. Accounting for the aforementioned difference in prior experience with both tools, the results obtained by this user survey support this restrictive approach’s appropriateness at least for the evaluated workflow.

Time Taken. Another aspect of the usefulness of a tool is the time required for a certain task. Generally, the better-suited tool will allow users to perform their work faster compared to its competitors. In the following, two aspects shall be examined in comparing Yavaa and standard spreadsheet software: First, the overall time needed to solve the given task from start to finish. Second, how much time is spent on the individual steps like locating appropriate data, joining datasets, etc. For this comparison, all submissions were used that resulted in an actual file being submitted. In particular, this includes submissions by participants stating to be unable to fulfill the task albeit producing a proper result.

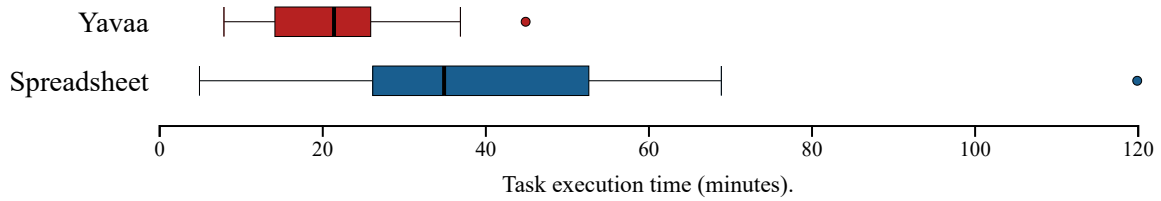


Figure 14.11: Evaluation results: Time spent overall. Where relevant, results for Yavaa a given as total (thick bar) an split by strategy (thin bars).

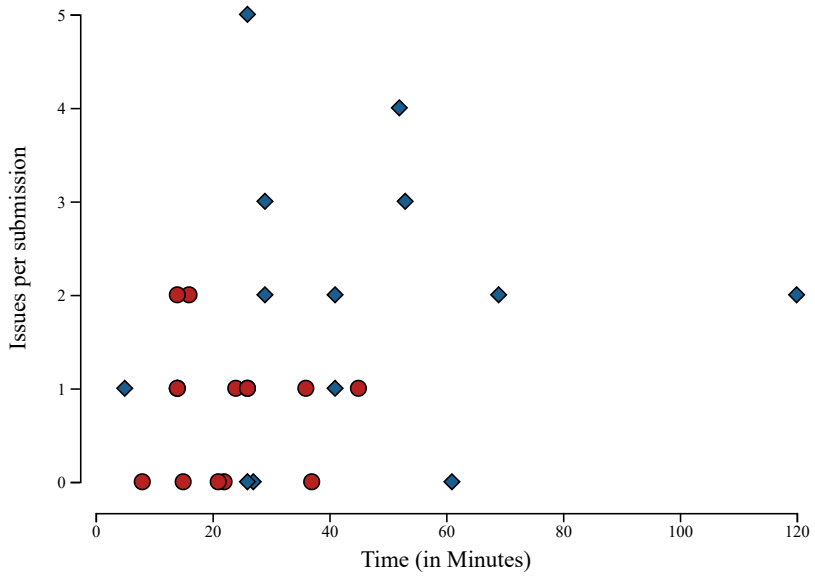


Figure 14.12: Evaluation results: Time spent vs. issues in submissions.

Figure 14.11 shows how much time participants spent on the posed task using each tool. It documents a rather substantial advantage towards Yavaa. Participants on average spent only about 22.7min (median: 21.5min) to finish the task using Yavaa, which is about half of the 42.4min (median: 35.0min) it took them using common spreadsheet software. As noted before, the nature of collected submissions did not allow to investigate the influence of tool order. Similarly, the low number of overall submissions did not allow to establish any correlations between the time spent on the task and the quality of results. Both, statistical tests (e.g., via t-tests) and visual inspection (cf. Figure 14.12), did not reveal any significant relationship.

The time spent on individual steps is rather hard to measure precisely as tasks may overlap and are rarely performed in linear order. As an approximation, participants were asked about the fraction of time that was spent on each step relative to the overall time spent (cf. Figure G.7 / G.13). In conjunction with the timestamp of starting the task (cf. Figure G.4 / G.10) and the time of finishing the task (cf. Figure G.5 / G.11), these fractions were used to calculate the time spent for each individual step.

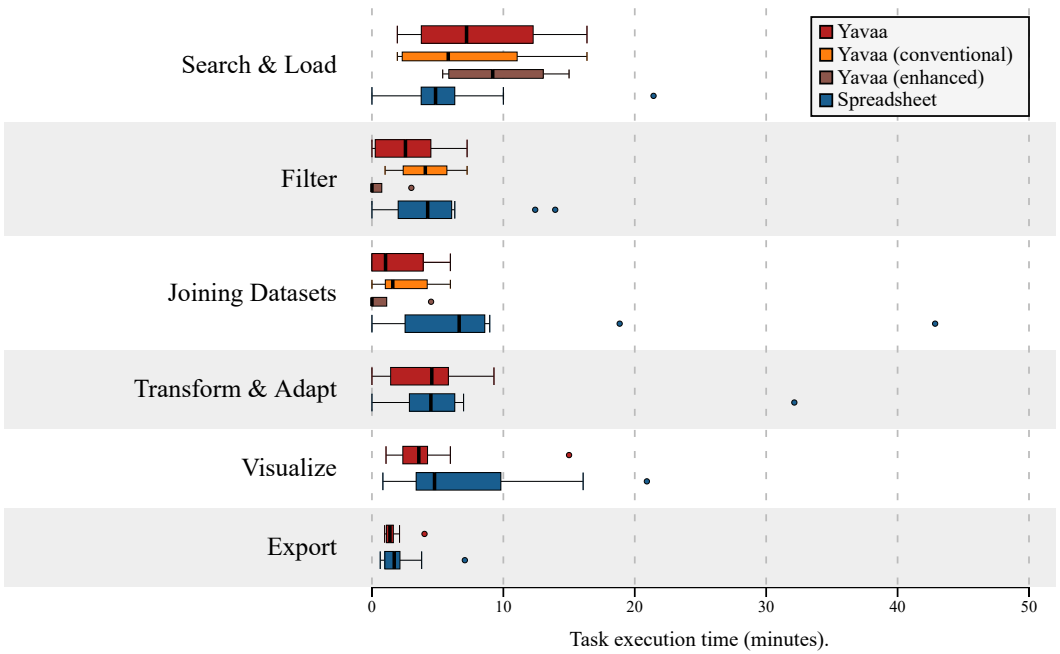


Figure 14.13: Evaluation results: Time taken per tool and step. Results for Yavaa a given as total (thick bar) and split by strategy (thin bars).

The distribution of time over tasks is shown in Figure 14.13. One caveat of note is due to participants stating a relative share of zero for a particular step. In general, each step is required for successfully finishing the task. So assigning a zero-share – basically skipping a step – seems like a mistake while filling the respective form entry. The only exception to this rule might be the *Filter* and *Joining Datasets* steps in Yavaa's enhanced strategy, as here those tasks are performed implicitly by the system, so users spend no explicit time here. Entries assigning 100% of the time spent to a single task were excluded. Here, participants either misunderstood the respective question or willfully entered incorrect data. This affects five responses for Yavaa as well as five for spreadsheet software.

In the following, Yavaa's different strategies will only be distinguished during the first three steps – (*Search & Load*, *Filter*, *Joining Datasets*). For all other steps, the workflow does not differ between the two strategies, so any differences can not be attributed to the chosen strategy (cf. Figure 14.3). Furthermore, the stated times represent the respective median across all valid responses in an effort to reduce the influence of zero-entries as well as other outliers occurring.

Traditional spreadsheet software does not provide a built-in search facility for data repositories, so the actual search was delegated to the Eurostat web portal [web1]. Having identified a dataset, the respective data had to be loaded into a separate tool, the actual spreadsheet software, to continue. Despite this necessity to involve at least two different tools, the first step,

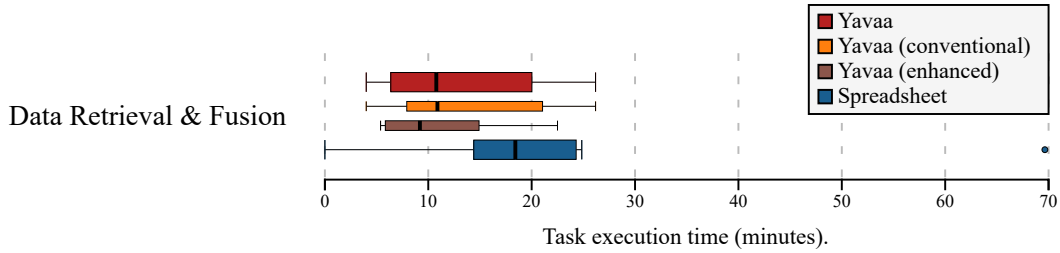


Figure 14.14: Evaluation results: Aggregated time for *Search & Load*, *Filter*, and *Joining datasets*.

Search & Load, Yavaa's conventional strategy (5.8min) took participants slightly longer compared to Eurostat/spreadsheet software (4.8min). However, Yavaa's enhanced strategy demanded considerably more time (9.2min).

In the subsequent step, *Filter*, Yavaa's conventional strategy and spreadsheet software are again quite close (4.1min vs. 4.2min). In the enhanced strategy such an additional step is not necessary in general. The corresponding median time of 0min for this step supports this statement. For the *Joining Datasets* step, the same argument applies to Yavaa's enhanced strategy and again a median of 0min can be observed. Further, also the conventional strategy (1.6min) outperforms spreadsheet software (6.6min) quite substantially here. Noteworthy are in particular the apparent outliers, which coincide with participants presumably attempting to perform a manual join of the two involved datasets.

In the *Transform & Adapt* step no meaningful differences can be observed. Yavaa (4.6min) and spreadsheet software (4.5min) require almost the same time. The *Visualization* step shows a small advantage towards Yavaa: Here, participants took only 3.6min compared to the 4.8min while using spreadsheet software. The final *Export* step shows again no noteworthy difference between Yavaa (1.4min) and spreadsheet software (1.7min).

As noted before, Yavaa's enhanced strategy combines several steps into a single interface. This makes it even harder to distinguish the respective time demands of each individual step. For a more realistic comparison, the respective times are combined in Figure 14.14. In this aggregated view, Yavaa's enhanced strategy requires only about half the time (9.2min) compared to spreadsheet software (18.4min). Due to the substantial advantage when joining datasets, also the conventional strategy of Yavaa outperforms common spreadsheet software (10.9min).

By its design, the survey assigned a random order of tools to each participant individually. However, due to the rather low number of complete survey responses, all successful responses placed the Yavaa task before the spreadsheet software one. As both tasks are identical with the exception of the tool to be used, this places spreadsheet software at an advantage: Once participants have proceeded to this phase of the survey, they already solved the same task using Yavaa and likely increased their overall understanding.

Nevertheless, the presented time measurements show that Yavaa for the given task can compete with common spreadsheet software across the board. In the two areas explicitly addressed, Yavaa excels in particular. The time necessary to assemble a dataset by combining multiple sources is reduced to almost half of what is needed using spreadsheet software. Similarly, the time required to pick a proper visualization for this dataset has been reduced by about 25%. The savings in these two areas are the main reason that allowed participants to perform the given task almost twice as fast using Yavaa than with common spreadsheet software.

Difficulty. Next, participants were asked for a (subjective) assessment of the difficulty of each step on a Likert-scale ranging from "very difficult" (−2) over "neutral" (0) to "very easy" (+2). During the subsequent discussions, the individual ratings are considered equidistant to their respective neighbors. While this might be an oversimplification, it allows for an easier comparison of the involved tools and making an admittedly sketchy estimation of any potential improvement or lack thereof by Yavaa.

The questions were posed immediately after each task. So again due to the one-sided distribution of task-sequence, Yavaa's assessment might be more neutral, while the one for spreadsheet software is influenced by the perception of the former. The distribution of responses for both spreadsheet software as well as Yavaa is shown in Figure 14.15. Furthermore, Figure 14.16 presents the relative difference between the assessments for Yavaa and spreadsheet software. Like before, the difficulty assessments for *Search & Load*, *Filter*, and *Joining datasets* are split between Yavaa's two strategies.

Eurostat's search is deemed rather difficult by most: Only three participants labeled it "very easy", while the remainder chose "neutral" or worse. On the other hand, Yavaa's results are distributed quite evenly across the range from "difficult" to "very easy". Of note is the polarized assessment for the enhanced strategy. Out of five participants employing that strategy, two rated it "very easy" and two others as "difficult" with a final "neutral" vote being cast. Considering the overall average of ratings, Yavaa leads spreadsheet software by about 0.5. Here, the difference between Yavaa's two strategies seems negligible.

Filtering using spreadsheet software is deemed "easy" or "very easy" by about two-thirds of participants. Yavaa improves this result even further with only one "neutral" vote. Two out of five participants that adhered to the enhanced strategy rated the filtering process, although it does not constitute a distinct step here. The result is the aforementioned "neutral" as well as another "very easy" vote. This improvement is subsequently reflected in the average ratings as well where Yavaa scores 1.0 (conventional strategy) and 0.5 (enhanced strategy) ahead respectively.

Joining datasets with spreadsheet software was rated "difficult" or "very difficult" by nine participants. Another two voting with "neutral" leaving four "easy" to "very easy" votes. Yavaa's assessments are more favorable with only one vote for "neutral" and the remaining twelve votes split evenly between "easy" to "very easy". The same two participants employing the enhanced

strategy that assessed filtering did so here as well. Also, the respective votes are the same with one being “neutral” and the other “very easy”. In the comparisons of averages, Yavaa can pull ahead by more than 2.0 for the conventional strategy (1.5 for the enhanced strategy).

The next step, Transform & Adapt, is again mostly rated as “neutral” to “very difficult” for spreadsheet software. Only a single vote each has been cast for “easy” and “very easy”. The situation is reversed for Yavaa. A single vote each for “very difficult” and “difficult” with the rest given to “neutral” or higher. The resulting averages put Yavaa ahead by over 1.0.

The ratings for Visualization using spreadsheet software are split almost evenly between all options with the exception of “very easy” receiving just a single vote. In contrast, Yavaa provides again a rather polarized result. Three participants voted for “difficult”, whereas all others chose either “easy” or “very easy”. On average, Yavaa outperforms spreadsheet software by about 1.3.

Exporting the results from spreadsheet software seems complicated for some. Three participants rated this step with “very difficult” or “difficult”, while four more voted for “neutral”. However, the majority of eight participants chose “very easy”. The opinion about Yavaa is almost unanimous. With a single vote cast for “difficult” and another to “easy”, all other participants voted for “very easy”. This results in an advantage for Yavaa of almost 1.0 on average.

Overall, Yavaa is perceived as easier to use across all examined steps. Common spreadsheet software especially falls shot when it comes to joining different datasets. As this functionality is not directly supported but has to be emulated by `VLOOKUP` statements, this is hardly surprising. While most steps’ results are in line with expectations, two results are quite curious and shall be discussed in more detail.

Search & Load shows the smallest difference. For the conventional strategy, the interfaces of Yavaa and Eurostat offer a similar interface with both providing keyword-based search. The differences in the assessment are thus likely rooted in the integration in the overall workflow referenced by the “Load” component of this step. The average difficulty for the enhanced strategy shows only a minor improvement. The option of building a dataset bottom-up by defining the intended contents and letting the system handling the rest does not seem to resonate with all participants. This is also supported by only a third actually choosing this strategy. Possible reasons could lie with the lack of familiarity with such approaches. Popular search systems nowadays most often follow the same workflow[14]: Users enter a list of keywords and can then pick from a returned list of possible matches. This represents more of a top-down approach in contrast to the bottom-up of the enhanced strategy.

Yavaa’s results for the Visualize step are quite polarized by comparison. This seems to indicate that the interface was accepted rather well by some participants, but failed considerably for others. There is no correlation between these two groups and the respective responses in other parts of the survey like prior experience, time taken, or issues in the submitted artifacts. Also, the provided comments provide no hint for this separation. A potential reason could be issues

[14]Previously available catalog-based systems seem to have lost their importance over time.

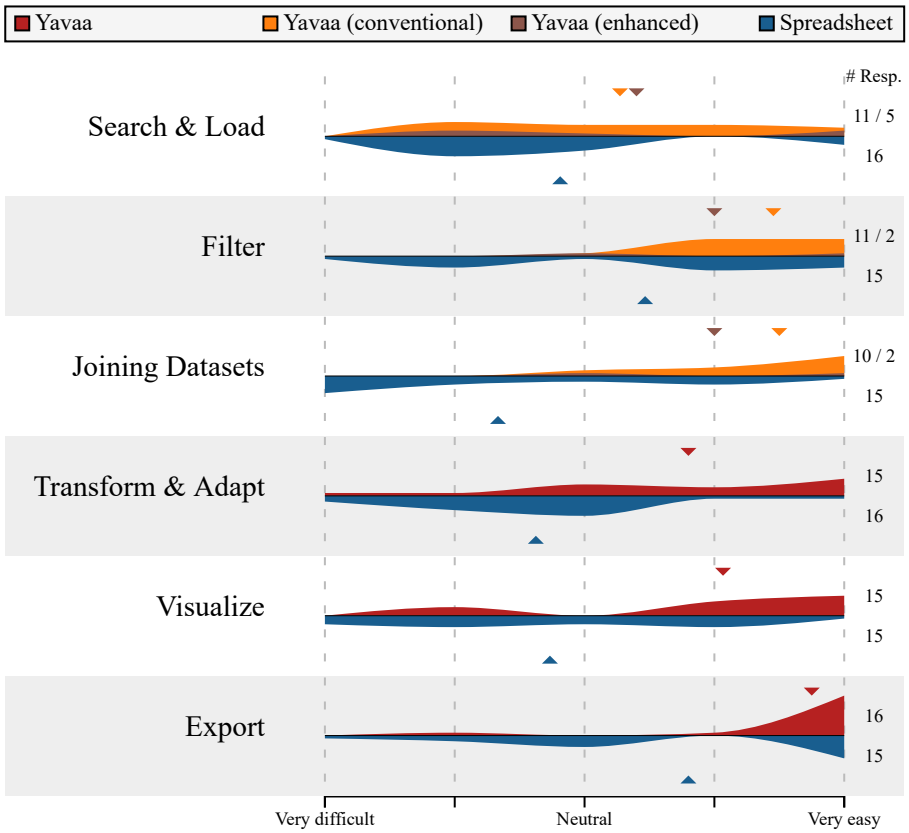


Figure 14.15: Evaluation results: Difficulty assessments. Triangles indicating average value.

in the accessibility of the interface. Once the main idea and intuition behind the workflow is grasped, it is rather easy to use. But failing to do so may result in perceiving the interface as less intuitive, thus resulting in the “difficult” rating.

Usability. A final aspect of the survey was concerned with the overall usability of the systems. Here, the evaluation was conducted via a standard SUS questionnaire [288, 289]. The detailed results for each item herein are given in Figure 14.17. Brooke, the author of SUS, notes that “scores for individual items are not meaningful on their own” [288] – a statement also reinforced by meta-analyses over a wide range of SUS-surveys [291]. Instead, he advises an aggregated form, the SUS-score, that combines the individual ratings into a single score ranging from zero (worst) to one hundred (best). In addition to this numeric value, several ordinal scales have emerged among practitioners in order to better communicate the respective results. Later analyses showed that these scales correlate quite well with the numeric SUS-scores [291, 292]. Figure 14.18 visualizes the results of this user evaluation and includes some alternative, ordinal

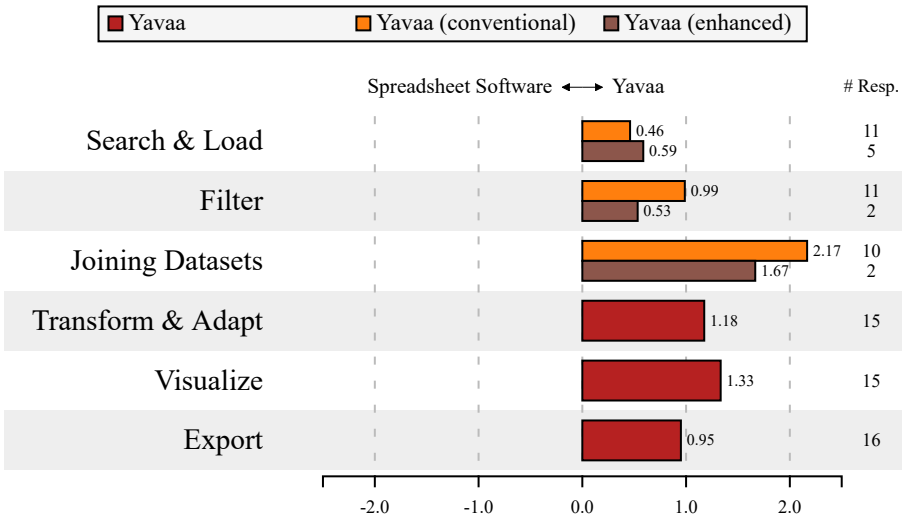


Figure 14.16: Evaluation results: Average difference of difficulty assessments.

scales for orientation. Following the previous admonition, the subsequent discussion will focus on SUS-scores, while the individual distributions for each question are only given for completeness' sake.

The majority of participants rated common spreadsheet software with an SUS-score below 60. Ten ratings are even below 50, labeling the tool not acceptable. Only two ratings are above 60 which puts them in the (marginally) acceptable range. The results for Yavaa are almost swapped. Only two ratings fall in the not acceptable range, whereas ten are well in the acceptable one. This contrast is also represented in the average SUS-score, which puts spreadsheet software at 44.46 while Yavaa resides at 73.57. Spreadsheet software's assessment thus corresponds (on average) to a poor-to-ok rating, whereas Yavaa's is associated with a good rating.

The acquired SUS-scores put Yavaa in a substantial lead over common spreadsheet software. At least for the posed task, the latter seems to be the worse choice despite being a rather common one for comparable tasks. Comparing the results on a participant-level (cf. Figure 14.19) reveals that the two outliers in each SUS-score distribution actually belong to the same two participants (represented by squares in the figure). Within the comments, one of those participants reports the aforementioned technical issues. The other one mentions actual deficiencies in the user interface that require further attention. Both participants stated an intermediate to expert prior knowledge of spreadsheets, which might have further biased the decision. However, from the gathered data this correlation can not be generalized as other participants with similar conditions did not come to a similar conclusion.

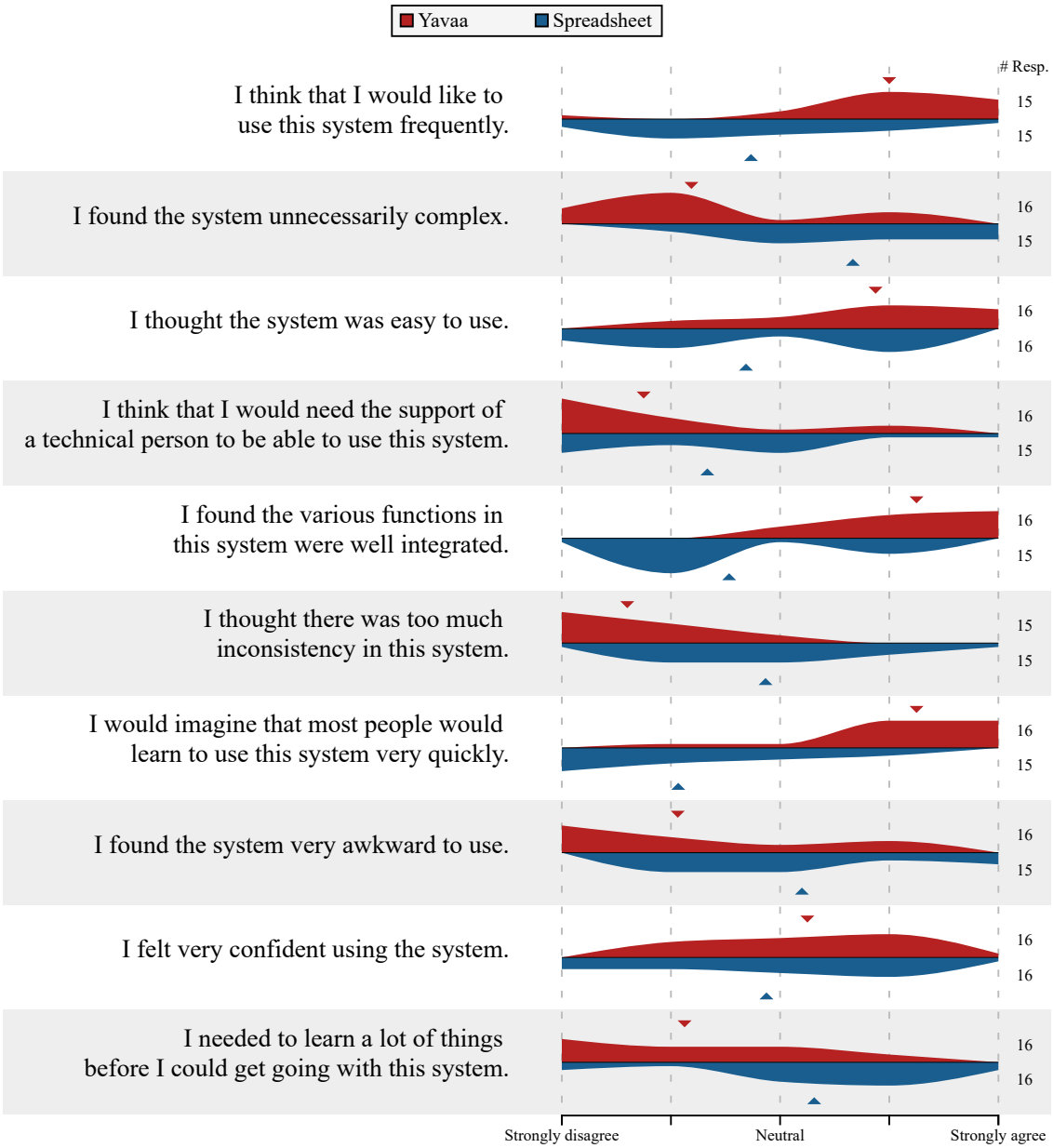


Figure 14.17: Evaluation results: Usability assessments. Triangles indicating average value.

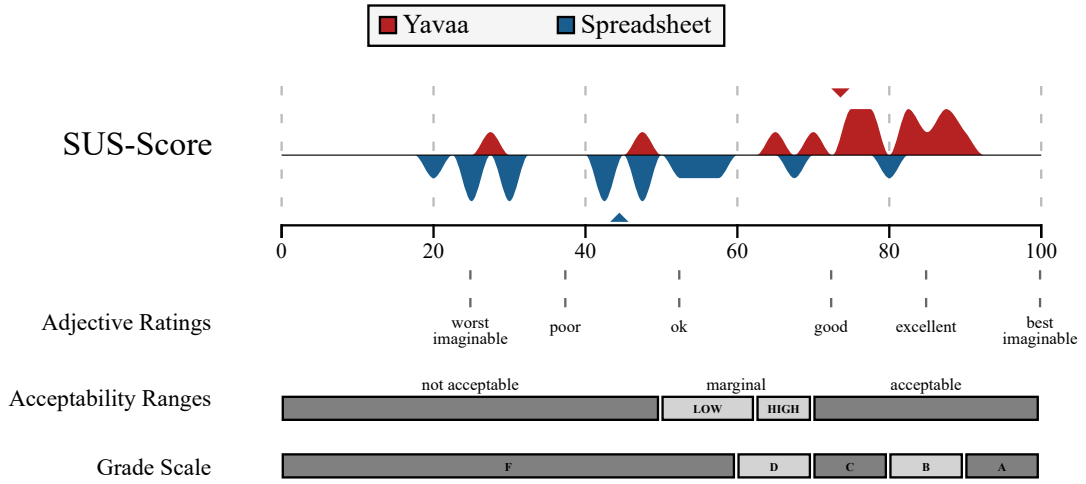


Figure 14.18: Evaluation results: SUS-scores. Triangles indicating average value. Responses missing ratings had to be removed. This leaves 14 scores remaining for spreadsheet software as well as 14 for Yavaa.

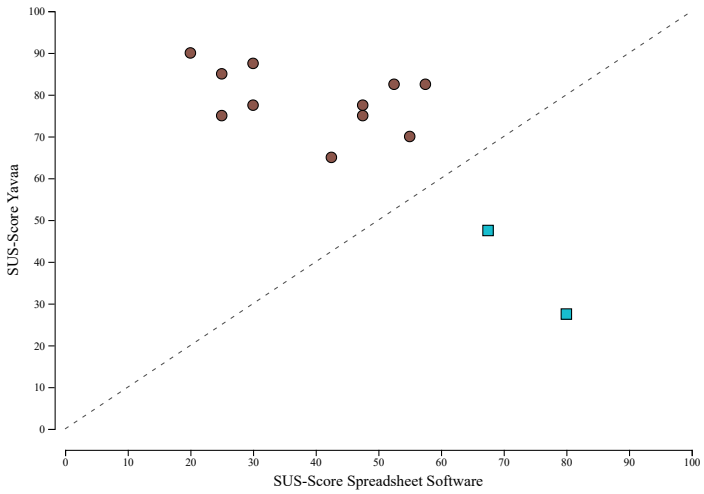


Figure 14.19: Relation of SUS-scores for responses that allowed the calculation of both (13×).

14.4 Technical Evaluation

The performance of the provided implementation is not the prime focus of this thesis. Nevertheless, this section shall provide a first impression of its runtime characteristics compared to its established counterparts. In particular, two aspects will be examined: First, how does the enhanced search strategy proposed in Chapter 10 compare to conventional keyword-based searches? Second, how does the prototypical implementation (cf. Chapter 13) fare against established tools like relational databases or a Python-based implementation?

All measurements presented in the following used the same infrastructure that was prepared for the user evaluation. In particular, it is hosted on a single virtual machine provided by Amazon Web Services [web166], i.e. a "t2.medium" EC2 instance with 2 vCPUs and $4GB$ of memory running Amazon Linux [web167]. All measurements were executed from the same machine. While this reduces the impact of network issues, requests will still have to undergo the cycle of serialization and deserialization common in such scenarios.

14.4.1 Search Strategies Performance

Yavaa supports two strategies to identify dataset(s) needed for the current task (cf. Figure 14.3 in Section 14.2). The conventional strategy implements the common approach to return datasets whose titles include the given keywords. The enhanced strategy allows users to describe the desired dataset. The system will try to fulfill this request by possibly combining multiple datasets.

An intuitive measure for the performance of both strategies is the number of requests per minute the system is able to serve. The scenario for such requests is taken from the user evaluation (cf. Section 14.2): Two datasets are required, one containing the human population for European countries and another one their respective sheep populations. Following the conventional strategy one may thus issue two queries: One using the keyword "population" and another one using "sheep". On the other hand, the dataset definition for the enhanced strategy will contain four[15] columns ("country", "time", "population", "sheep").

For each strategy, the corresponding requests have been issued in sequential order over the course of one minute. So once a request was successfully processed, the next one was issued. Figure 14.20 visualizes the respective number of iterations during that time. In addition to the two aforementioned strategies, it also includes the number of iterations for each keyword-based query in isolation.

A first, interesting observation is the rather large difference between the individual queries. In the given time frame the system could respond to far more requests for "sheep" compared to "population". The underlying reason is likely rooted in the nature of the dataset repository. In the repository used throughout the evaluation, only two datasets mention the term "sheep", whereas

[15] Kindly be reminded at this point that the derived column "sheep per capita" has to be computed in a separate step, as it is not available from the sources directly.

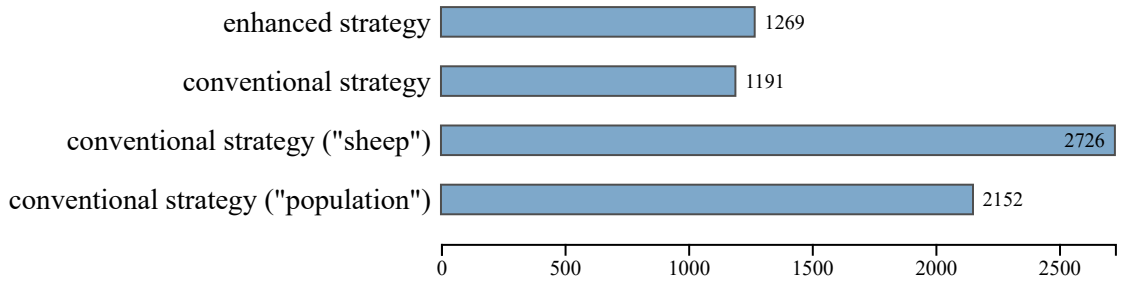


Figure 14.20: Benchmark search strategies: Executions per minute.

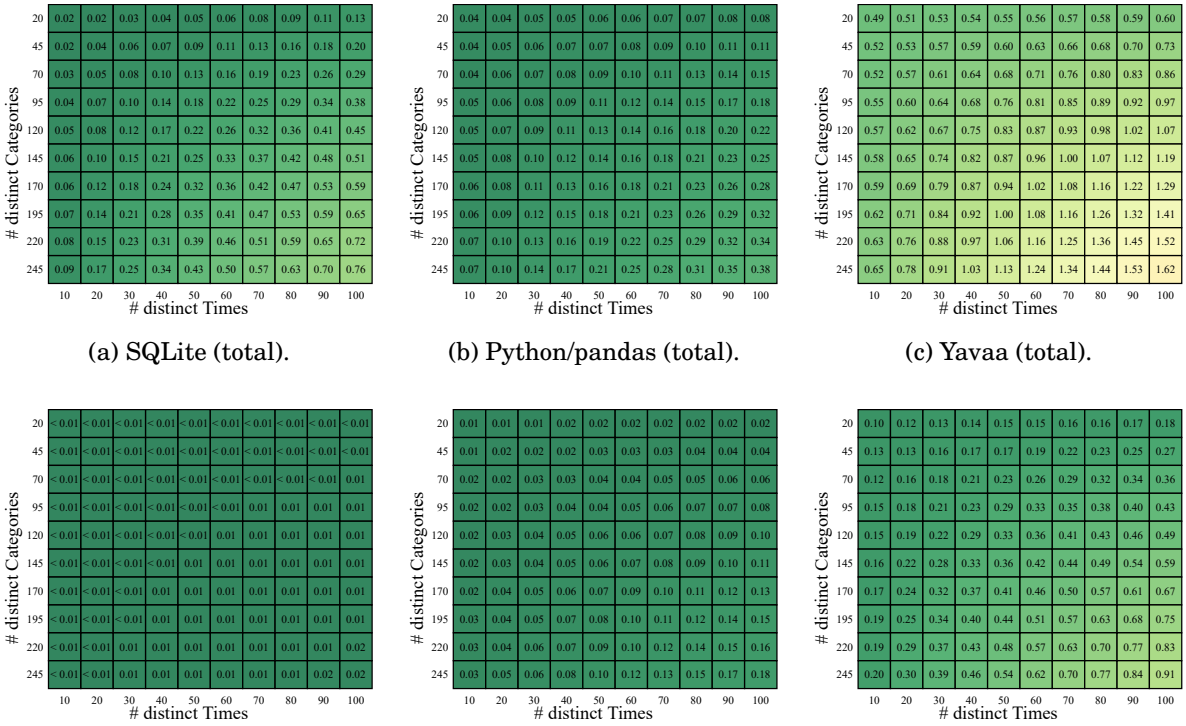
171 mention the term “population”. This large difference in the number of results is amplified by the need to serialize the respective results in order to transfer them. These two aspects are the probable cause of the difference witnessed between the two queries.

Another observation concerns the actual strategies. Here, both strategies are almost on the same level with a small advantage towards the enhanced strategy. On a first glimpse, this is rather surprising, as the enhanced strategy issues more queries to the triple store and further has to combine the intermediate results subsequently. A naïve view would thus expect fewer requests to be served. The core difference here is how the respective queries are framed. The conventional query used keywords, i.e. strings which have to be matched against other strings, the dataset titles. However, in the enhanced approach, those strings have been mapped to concepts of the dataset description model beforehand[16]. So instead of string comparisons, the queries to the triple store now rely on graph traversal which is faster in this case. Summarizing, even the overhead of more queries, applying constraints, and combining the results does not put the enhanced strategy at a disadvantage. In terms of pure execution performance, it seems to be comparable to a conventional keyword-based search.

14.4.2 Compute Engine Performance

A second experiment was conducted to compare the execution time using Yavaa’s prototypical implementation against other well-established tools. Two such tools were chosen: An implementation using Python’s pandas library (version: 1.0.4) [web48] serves as an example of a scripting language often used by Data Science professionals. Further, the same workflow was implemented using SQLite [web158] (version: 3.31.1 via `better-sqlite3` [web168]). As a relational database, this represents years of experience in managing tabular data. The underlying workflow is again modeled after the one used in the user study of Section 14.2. It omits the preceding search step,

[16] During creating a query for a combined dataset, users are presented with the labels of concepts for columns and values. Internally the system relies on their respective IRIs, though.



(a) SQLite (total). (b) Python/pandas (total). (c) Yavaa (total).

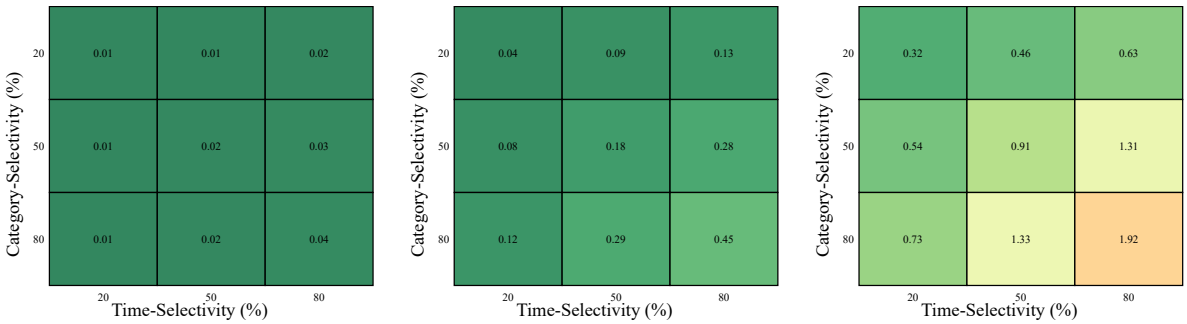
(d) SQLite (non-I/O). (e) Python/pandas (non-I/O). (f) Yavaa (non-I/O).

Figure 14.21: Benchmark results: Workflow execution time in seconds (cat. sel. 50%; time sel. 50%).

but follows the general approach of loading two datasets, filtering entries, joining them, and deriving a new column. Finally, the results are exported into a CSV file. The source code to conduct the measurements as well as the respective data generator is publicly available [data22].

For the evaluation, the size and characteristics of datasets have been varied along the following dimensions[17]. The number of distinct countries started at 20 and was increased in steps of 25 up to a maximum of 245 (user study: 55 and 39). Distinct time values ranged from 10 to 100 in steps of 10 (user study: 12 for both datasets). An additional modifier for data items[18] is added with a probability of 10% (user study: 5% and 10%). Missing values occurred with a probability of 15% (user study: 32% and 4%). During the workflow, the selectivity of filters was applied in three variations of 20%, 50%, and 80%. Overall this leads to $10 \times 10 \times 3 \times 3 = 900$ different configurations (number of categories × number of time periods × category selectivity × time selectivity). The datasets required by each configuration were generated beforehand and reused for all tools. Each measurement was repeated 42 times out of which the best run will be reported in the following.

[17] The corresponding values for the two datasets from the user study are provided in parentheses.
[18] Eurostat uses those to add additional information about the value presented, e.g., to indicate provisional values.



(a) SQLite. (b) Python/pandas. (c) Yavaa.

Figure 14.22: Benchmark results: Workflow execution time in seconds for different selectivities excluding I/O-operations (distinct categories: 245; distinct times: 100).

The results for one slice of the evaluation are given in Figure 14.21(a) to 14.21(c). For the smallest configuration SQLite outperforms the other tools with 0.02*s* against 0.04*s* (Python/pandas) and 0.49*s* (Yavaa). Looking at only the largest configuration, Python/pandas takes the lead with 0.38*s* versus 0.76*s* (SQLite) and 1.62*s* (Yavaa).

The overall time spent on the workflow is predominantly consumed by I/O-operations. Removing the initial reading of data[19] and the export of the final results yields much lower executions times, as shown in Figure 14.21(d) to 14.21(f). In particular, SQLite spent most of its time with these preparatory tasks with only 2% being used in the actual calculation. Both other tools, Python/pandas and Yavaa, split the execution time roughly equal between I/O and non-I/O tasks with 48% (Python/pandas) and 56% (Yavaa) used for calculations in the same configuration. The scaling between the smallest and the largest configuration differs quite substantially among the three tools. The execution times increase 25× (SQLite), 16× (Python/pandas), and 9× (Yavaa).

A final aspect is the influence of the selectivity for filters used, i.e. the fraction of values that are kept after applying said filters. Figure 14.22 summarizes the corresponding results for the largest configuration of 245 distinct categorical values as well as 100 distinct time entries. Here, the overhead of I/O-operations has been omitted. Comparing the smallest configuration (both selectivities at 20%) with the larges one (both selectivities at 80%), puts SQLite and Yavaa ahead with a 6× increase, whereas Python/pandas' execution time grew 10×.

In summary, the following observation can be made. First, Yavaa's current implementation in terms of absolute performance can not compete with state-of-the-art tools dedicated to comparable tasks. The years of experience put into such tools puts them substantially ahead. Second, considering the relative scaling with respect to increasing dataset sizes and selectivity of filters the current implementation performs similar or even better compared to other tools. This suggests that while individual operations might have an inferior implementation, the overall architecture is quite apt. Finally, considered in isolation the Yavaa-prototype tested here is a viable solution to

[19]This includes the conversion of the dataset from a pivot table to a normalized structure, as Yavaa performs that in a single step.

perform the given workflow. Even for larger datasets tested, the absolute runtime is still within acceptable ranges for conversational interactions, commonly given by $2s - 5s$ in standard system response time models [293]. Even more, since workflows are expected to be rarely executed as a whole, but are successively build-up by various user interactions. This breaks it down into smaller steps, each with lower execution time.

Part IV

Epilogue

CHAPTER 15

RETROSPECTIVE

The goals of this thesis were introduced in Section 1.1 and translated into specific requirements in Chapter 2. The previous chapters described possible approaches to solve issues emerging from these requirements (cf. Part II), outlined their implementation (cf. Chapter 13), and, finally, evaluated the resulting prototype (cf. Chapter 14). At this point, the requirements posed in the beginning shall be revisited. The goal is to examine how each of them was addressed and to what extent it was fulfilled. As fulfillment of the requirements entails fulfillment of the corresponding objectives, the latter will not be discussed separately here. In the following, requirements are grouped in accordance to the modules addressing them in Yavaa. At times, this may deviate from the order initially given in Chapter 2.

Requirement 1. Search across Providers
The system has to provide a unified search interface across different data providers. This also includes a translation of provided inputs into the idioms used by the respective providers.

Requirement 5. Mediate Abbreviations
Heterogeneous abbreviation schemes possibly in use by the involved data providers have to be harmonized transparently for users.

Requirement 6. Support Heterogeneous File Formats
When loading primary data, the system has to support the consumption of multiple, heterogeneous file formats transparently to users.

Requirement 7. Support Heterogeneous Dataset Structures
When loading primary data, the system has to be able to automatically harmonize different data structures for further processing.

Requirement 8. Translate Abbreviations
Encoding schemata in use by the different providers have to be consistently translated into human-readable labels.

Yavaa's search relies on the metadata model proposed in Chapter 8. Here, no prior assumptions are made on where the dataset currently resides (Requirement 1). It even allows for multiple locations from which a dataset can be retrieved. This includes the possibility to specify different formats or representations that are available (Requirement 6). Furthermore, the respective idiosyncrasies of each provider are mitigated by linking datasets to code lists used and mapping existing values to semantic concepts therein. This allows to translate across different representations of values without further intervention by users of the system. From a user perspective, all this is hidden. During loading, a file Yavaa automatically normalizes the dataset's structure (Requirement 7) and translates occurring values into their associated concepts. For further operations, it only relies on those concepts and internally omits the display label (Requirement 5). Users, on the other side, will not encounter those concepts, but interact with only the display label. Depending on the current state of the interactions this is either the label used in the original data source or a canonical label attached to the corresponding concept (Requirement 8).

Requirement 2. Search in Primary Data
The system has to support the definition of constraints on the primary data as part of search requests.

Requirement 3. Search by Combination
If a given search request can only be fulfilled by using multiple datasets, this combination has to be suggested as a possible result.

Requirement 4. Materialize Combination-Results
If a search result was given as the combination of multiple datasets, means have to be provided to materialize this combination.

The former two requirements are addressed through the search-by-combination proposed in Chapter 10 and the respective implementation outlined in Section 13.9. Users are allowed to describe their information need in form of a table definition that includes both requested columns as well as constraints posed on their value ranges (Requirement 2). Yavaa will attempt to fulfill this query by possibly combining multiple datasets (Requirement 3). A viable result is given by a workflow that once executed would produce the requested table. Users can choose to execute said workflow to materialize the corresponding table (Requirement 4).

Requirement 9. Allow for Modification of Data
The system has to allow users to adapt datasets to their current needs by, e.g, filtering, aggregating, or adding new columns derived from given formulae.

Requirement 11. Ensure Validity of Operations
During both system-initiated as well as user-driven modifications of the datasets, the system has to ensure the validity of operations. Possible errors have to be transparently corrected. Only if this can not be achieved shall users have to intervene.

The capabilities of Yavaa to modify datasets have been described in Section 13.3. In particular, this includes the three operations requested in Requirement 9. During these operations, the corresponding unit of measure is automatically determined whenever possible. While this still leaves invalid operations in the realm of possibilities and thus only partly satisfies Requirement 11, it removes a prominent source of errors in many workflows. The prevalence of unit errors has been documented by the user study conducted as part of the evaluation (cf. Section 14.2), where related issues appeared in over 40% of submissions using spreadsheet software (6 out of 14; cf. Section 14.3).

Requirement 10. Provide Immediate Feedback on Operations
The results of applied operations have to be apparent to users without unnecessary delays.

Immediate feedback on triggered operations can be interpreted in at least two ways. First, it requires a proper interface to actually represent the feedback. Yavaa has a graphical user interface (cf. Appendix G) that represents datasets from three perspectives: a complete one proving access to the primary data, a summarizing one visualizing the dataset, and a formal one outlining the workflow that leads to the current state (cf. Section 13.9). The second way is given by the time users have to wait before operations take effect. This aspect was examined in Section 14.4. While lagging behind tools commonly used in similar tasks like SQL-databases or Python-scripts, Yavaa can still execute common workflows well within reasonable times allowing for an uninterrupted user experience.

Requirement 12. Recommend Visualizations
The system has to provide recommendations for meaningful visualizations given a certain dataset.

Requirement 13. Recommend Variable to Artifact Mappings
The system has to provide recommendations for a meaningful translation from the current dataset and into the selected, applicable visualization.

Requirement 14. Materialize Visualizations
The system has to be able to create visualizations based only on declarative definitions provided by users.

Chapter 7 contains Yavaa's internal model to describe visualizations. This includes the definition of interfaces that allow mapping columns of existing datasets to corresponding visual artifacts. Based on these descriptions as well as the currently active dataset's corresponding

description (cf. Chapter 8), Yavaa recommends possible visualizations (Requirement 12). As both descriptions primarily operate on the level of columns and visual artifacts, Yavaa is able not only to suggest suitable visualizations but can also provide specific bindings between those components (Requirement 13). The details of deriving these recommendations are described in Chapter 11. Users may adjust the proposed mappings and then can immediately trigger the rendering of a chosen visualization as outlined in Section 13.7. This is done in a purely descriptive manner using a drag&drop-interface (Requirement 14).

Requirement 15. Track Provenance
Each operation that alters a dataset has to induce an entry into the provenance record of this dataset. The corresponding entry has to include all settings that affected the respective operation, including at least inputs, outputs, and any parameters necessary.

Requirement 16. Visualize Provenance Records
The accumulated provenance records for the current dataset have to be presented visually to users. This representation has to include at least the applied operations, their logical order, and their respective inputs and outputs.

Actions triggered by users can be distinguished into two groups: those that modify the underlying dataset and those that do not. While the latter are often of informative nature like retrieving the dataset summary, the former have a lasting impact on the final data product. Consequently, each version of a dataset (cf. Section 13.2) in Yavaa is associated with a description of how it came about (cf. Chapter 12). As this includes a link to its predecessor(s), Yavaa is able to establish a chain of provenance records that includes each of these prior versions (Requirement 15). This rather abstract notion of a workflow is subsequently transformed into a flow chart (cf. Section 13.5) that is more accessible to human users (Requirement 16).

Requirement 17. Share Provenance Records
The system has to provide the means to export the complete provenance record for a given workflow in a standardized format consumable by other applications.

Requirement 18. Allow for Reenactment of Workflows
Based on a previously exported provenance record and under the assumption that the data sources did not change in the meanwhile, the system has to be able to execute a previous workflow with the same results.

Besides the visual inspection of the workflow, Yavaa also allows exporting a JSON-based, PROV-compatible serialization (Requirement 17). These exported workflows can freely be shared and interpreted by other provenance tools. Furthermore, users are able to load and execute them via Yavaa once again using possibly updated data (Requirement 18).

Requirement 19. Usability
The system has to be accessible for novice to intermediate users.

The usability of Yavaa especially in comparison to commonly used spreadsheet tools was a major object of investigation during the user study (cf. Section 14.2). The corresponding results substantiated a considerable improvement over these tools (cf. Section 14.3). As participants also covered a broad range of experience, Yavaa appears as an adequate alternative no matter a user's prior skill level.

Requirement 20. Extensibility
The system has to be structured in a way that allows to easily add new components for critical areas. In particular, this includes adding data providers, data wrappers, and visualizations.

Yavaa's architecture, as outlined in Section 13.1, features several so-called stores to hold both declarative descriptions of crucial components as well as the corresponding implementations. Only a few components are hard-wired, e.g., to facilitate the communication to the interface and orchestrate the access to and execution of modules from those stores. Yavaa's other capabilities in terms of supported data providers, data wrappers, visualizations, and other aspects can be extended by adding to these stores without having to adjust the remaining codebase. This allows to rather easily extend the provided functionalities, thus satisfying the set requirement.

This concludes the review of this thesis with regard to the requirements posed in Chapter 2. It confirms that each of those requirements has been successfully addressed in either concept, implementation, or both. This conclusion can also be extended to the thesis objectives stated in Section 1.1, as the discussed requirements were derived from said objectives and as such represent the same goals. In summary, the preceding discussion illustrates that the proposed concepts and the corresponding implementation, Yavaa, are indeed a valuable contribution to enable easier access to data visualization for a broader range of users.

Limitations. As many research projects, also this thesis is based on a set of assumptions to limit the scope to a feasible subset and promote the bigger picture over handling all corner cases. However, in practice, those assumptions may restrain the applicability of the developed solution in real-world scenarios. In the following, some of the assumptions made in this thesis shall be revisited with respect to the restrictions they entail. Similarly, some general limitations of Yavaa will be discussed to gauge their impact for the intended audience and workflows. Some of the current limitations are not fundamental to the approach itself and, hence, can be a subject of future work. Those will be expanded upon in more detail in Chapter 17.

A key assumption in the dataset descriptions (cf. Chapter 8) and subsequently during dataset combination (cf. Chapter 10) is the homogeneity of datasets: Values within a column have to share the same type (given by the column concept), the same format (in particular for time columns),

and the same unit (applicable only in quantitative columns). As witnessed by the evaluation preparation (cf. Section 14.1), especially the latter two excluded quite a few datasets. However, this does not seem to be an intrinsic limitation. Most issues with the format in time columns are due to datasets providing data for different levels of aggregation like mixing monthly and yearly data in a single dataset. Arguably, those datasets should be split anyways according to the granularity of the data they provide. Similarly, datasets containing measurements represented in different units could be harmonized to use a single unit and thus again conform to the given requirement.

Similar to the homogeneity, Yavaa also assumes all datasets to be complete, i.e. every combination of included dimensional values exists in the datasets and is associated to corresponding measurements. This simplifies the situation of real-world datasets where often enough at least some values are missing. As discussed before in the context of MDLH [193] (cf. Section 10.1), describing these gaps can already get arbitrarily complex. So accounting for them in the metadata descriptions may significantly increase the size of the same up to negating their summarizing nature altogether. Further, evaluating those descriptions in the context of a particular query will be slowed down considerably, as the number of required query splits (cf. Subsection 10.3.3) and thus iterations of the algorithm will grow substantially. For a large number of candidate datasets or a complicated structure of gaps in the corresponding datasets, this might make the approach unfeasible in a live-interaction system.

A different approach could be to fill in the gaps once the result is materialized and they can easily be identified. Other datasets covering these regions could then be fetched to provide the necessary measurements. However, as metadata descriptions do not contain information about possible gaps, this might require fetching all such datasets only to finally recognize that some values are just not available in general. One of the key advantages of the approach presented in this thesis is the comparatively small amount of data needed to respond to a given query. So it can be applied even to rather large collections of datasets. This advantage would be entirely nullified if results would need to be materialized in order to evaluate their suitability. Here, the decision has been made to favor possibly less complete results but against fetching large amounts of data for each query.

On a related account, the presented approach committed to the assumption that results are more consistent if large parts originate from the same source. The idea of fetching all possible datasets that cover (parts of) a query might have been chosen as the alternative approach[1]. This would shift the focus from trying to find a (minimal) combination of datasets to fulfill a query towards having to resolve possible contradictions among different sources. Besides requiring substantially more data to be fetched, this also calls for conflict resolution strategies [205]. From a user perspective, this entails two options: Either conflicts are resolved transparently by the system or user interactions are used at least in controversial cases. For the

[1]This is virtually equivalent to first executing the workflow generated by the approach presented in Chapter 10 and subsequently filling in the blanks in the resulting dataset.

intended audience of novice users, both options are likely undesirable. Transparently handling all conflicts in a background process can hide deficiencies in both data sources as well as the conflict resolution itself. Even when the system is not confident that a certain value is reliable, users would have no feedback and might place exaggerated trust in low-confidence data. On the other hand, the system could provide confidence estimates to users or even include them in the conflict resolution process. This will require substantial additions to the user interface making it more complex and thus likely less accessible to inexperienced users. For these reasons, such an approach has been forgone in this thesis and the aforementioned conflict avoiding strategy has been adopted instead.

Another simplification has been adopted with regard to the concepts describing the contents of datasets' columns. Here, it is assumed that all datasets describe their data using a similar level of granularity and normalization. However, in practice, dataset structures might differ at least between providers. Assume a statistical dataset about the number of unemployed by sex. One way of representing this data is to use, among others, two columns with concepts like "male unemployed" and "female unemployed". A different provider might choose a different structure using a column "sex" and another column "unemployed". While both datasets might contain the same data, Yavaa is currently not able to make the connection between those two. In order to recognize that both descriptions are semantically equivalent, the respective concepts would need to be decomposed and the underlying data transformed accordingly. This has been deemed out of scope for this thesis and is discussed as part of future work (cf. Chapter 17).

The motivation for the visualization recommender of Chapter 11 is the lack of experience by users in choosing suitable visualizations. The matter of what constitutes "suitable" in this context has to be relayed to the authors of the corresponding visualization descriptions. While most of the criteria there are of a technical nature[2], others depend on the opinion of the respective author. This constitutes a rather fundamental restriction to basically all such systems. Certain best practices have evolved that might be applicable to the vast majority of situations [213, 214]. Still, they are driven by opinions and those opinions may differ, e.g., by cultural background and even change over time due to, e.g., broader familiarity with certain representations [294]. Similarly, there is a wide range of potential cognitive biases that may be considered for specific tasks or audiences [295]. Accounting for all potentially relevant aspects in a visualization recommendation is next to impossible or will make interacting with the respective interface excruciating cumbersome. So like all other solutions that venture beyond the mere technical dependencies of visualizations, also the recommender implemented in Yavaa will inevitably involve a certain level of bias. All that remains is to recognize biases — not just in this context, but in basically every situation — and reduce their impact as far as possible.

[2]For example, there is no meaningful way to create a particular visualization if the data types of involved columns do not match the ones expected by the visualization.

CHAPTER 16

CONCLUSION

With the thesis coming to a close, it is time to recapitulate. This chapter will revisit the underlying motivation and summarize the main contributions made. A further look at future directions is deferred to subsequent Chapter 17.

Ever more publicly available data raises questions about how this treasure of information can be exploited not just by a select few but by anyone interested. Possible answers have to cover a rather wide range of relevant aspects: Required data is often scattered over different datasets owned and published by a multitude of providers. Proper means are needed to identify all relevant datasets and to combine them into an exhaustive basis for further analysis. As data is rarely in the form required, several operations will be applied to bring it to its final form. Besides the omnipresent chance for human error, heterogeneity in representations and the resulting need for alignments is another source of errors. Especially novice users may miss some pitfalls and need support to create high-quality results. Raw data is barely understood directly. To reach a wide audience, effective visualizations are essential to communicate one's findings and ideas. However, selecting visualizations matching the current data, let alone effective ones, is a daunting task for many users. Here, recommenders accounting for the current context can guide users towards meaningful results. Finally, when all obstacles are overcome, information about the decisions made is oftentimes lost. This provenance is needed to justify or reproduce the results, though. It has to be tracked along the entire workflow to document the steps leading to a final data product.

Despite past efforts to solve the individual aspects of such visualization workflows, there is no widespread solution addressing all of them holistically. This gap was the starting point for the thesis presented. Its main goal is to develop a platform that is able to support users in their visualization tasks from start to finish. Over the course of the thesis, this workflow was separated into individual steps, each of which was analyzed to see how users can be supported to complete

their tasks. This analysis was guided by two principles: easing challenges posed to users and eliminating sources of errors. In conjunction with an extensive literature review, the results of this analysis subsequently motivated the following contributions of this thesis.

A **semantic metadata model** was developed to describe tabular data which accounts for a large share of generated data. It extends current standards by including information about codification schemes as well as summarizing a dataset's contents. This model paved the way to a **search engine** that allows to **seamlessly combine multiple datasets into a single result** satisfying users' information needs. Instead of manually retrieving and joining the results of a keyword-based search, users can immediately receive a dataset containing all requested data.

The same model allows **translating codified values into semantic entities** to mitigate between various encoding schemes used by different data publishers. In subsequent operations, **consistent handling of units** is ensured. This includes all necessary conversions of values between different units, should those become necessary. For complex formulae, an algorithm was devised to **minimize the number of conversions** improving the numeric stability of results.

A developed **visualization recommender** supports users in selecting suitable visualizations. Besides suggesting visualizations themselves, it also maps individual columns to visual artifacts thereof. The presented algorithm accounts for column roles, datatypes, and value ranges in the matching process. Furthermore, dataset structures and visualizations do not need to match exactly. The recommender also includes suggestions that cover only parts of the dataset but ranks them lower compared to complete mappings.

Over the course of the entire workflow, all actions are tracked and stored in a **provenance model**. This model provides not only the full history of any created data product but also allows to re-execute the workflow on updated data. Provenance is tracked on a column-level, thus allowing not only to follow the applied operations but also to inspect the interdependencies among the individual columns of a dataset. As a means for human users to inspect a given workflow, also a **layouting algorithm for this provenance graph** was devised.

All proposed concepts were **implemented into a prototype, Yavaa**, whose code is publicly available under an open license [data23, web169]. Based on Eurostat's data holdings, it features the **entire visualization workflow in a single application**. To verify the claimed improvements over commonly used tools for similar tasks, Yavaa was subjected to an extensive **user study**. This evaluation confirmed improved usability as well as a reduction of errors in the resulting data products.

The concepts and implementation provided in this thesis can form the basis for a more evenly spread exploitation of publicly available data. Unearthing the secrets hidden in this wealth of information does not have to require expert knowledge. Instead, proper tool support at crucial steps in the workflow can democratize data access and its dissemination. In this spirit, the presented thesis might be a small puzzle piece on the way to more transparency and accountability in all aspects of public life.

CHAPTER 17

FUTURE WORK

As common to research, the concepts presented in this thesis answer some questions, but also bring up new ones. In the following, some of the future directions arising from the contributions of this thesis shall be presented. Some of them will include first ideas on how to approach the respective issues leading the way to continue the work done here. Furthermore, this listing represents an attempt to classify the elements of future work into those that lean towards the software engineering side and those that require additional conceptual work. The boundary between both areas is oftentimes blurry at best, especially in Computer Science. So further investigations into individual topics might change the current assignments in both directions.

17.1 Conceptual Foundations

Decentralize metadata repositories. The architecture proposed in Chapter 13 relies on a centralized metadata store for all available data products. This design is based on the assumption that data providers are either not involved in such a project or are non-cooperative. In both cases, no changes to the existing systems could be made. If this assumption is dropped and data providers actively participate, the metadata store can be decentralized, such that each provider is responsible for maintaining its own store.

The requirements for such a store a rather low, as they only have to provide a public SPARQL endpoint and the metadata within that has to comply with the structure proposed in Chapter 8. While the former is already part of many Linked Data setups, the latter also relies to a large extent on existing and widespread standards. This keeps structural changes to a minimum and allows for low-investment participation. A more challenging task is posed by the harmonization of the vocabularies used to describe the particular datasets' contents.

Extend dataset descriptions for better ranking during dataset combinations. As noted in the later parts of Chapter 15, Yavaa currently assumes datasets to complete, i.e. they do not contain larger gaps in their value coverage. As one way to account for such gaps, the dataset description could be extended with information about this coverage. This information could then be used during the ranking phase of dataset combinations (cf. Subsection 10.3.2) to promote more complete datasets over those with inferior coverage. Similarly, other value-level characteristics like confidence-intervals could be used as additional ranking factors in the dataset combination[1]. However, meaningful aggregations have to be found that allow keeping the metadata descriptions overall within a reasonable size.

Decompose column header concepts. Different data providers will inevitably have diverging opinions on how data should be modeled. One consequence is a certain variety in how datasets are structured. Large parts of this thesis are based on the assumption that datasets are fully normalized and thus column headers are already atomic. In practice, however, this might prove to be too restrictive. As an example, consider data about life expectancy by sex. This can be represented either by two normalized columns, *sex* and *life expectancy*, or by using composed concepts for two columns like *female life expectancy* and *male life expectancy*. In the future, systems should be able to recognize both variations as idempotent and be able to match one to the other. Furthermore, this decomposition might be context-sensitive: What is considered atomic in one context, may need to be decomposed in another. Nonetheless, the information about a possible decomposition of the concepts used to describe a column's content needs to be maintained. For observable properties, this likely requires implementing the recommendations made by the *InteroperAble Descriptions of Observable Property Terminology WG* [157, 158, web100]. Here, the author of this thesis is already involved in efforts to systematically decompose existing vocabularies with the goal of making them interoperable. While this effort is only geared towards observable properties, i.e. measurable characteristics of physical objects, similar techniques can likely be applied to other areas as well.

Model the mathematical relationships among concepts. The example used in the user evaluation (cf. Section 14.2) showed that users still need to define the input operands before being able to calculate a derived value. At least some of these relations could be expressed in the underlying semantic model. One example is the concept of "unemployment rate" which by itself is derived from two other concepts – "people unemployed" and "labor force". Sometimes data providers already offer the derived quantity kind in a dataset of its own, but oftentimes only the individual operands are available. As touched upon in Chapter 9, a similar situation exists for units of measurement: Derived units can be expressed via the relations among their components and some unit ontologies encode this information [161]. This general approach can be extended

[1]This is assuming the corresponding information is offered by data providers.

and applied to generic concepts outside of the unit domain. If such concepts and relations would be available, systems like Yavaa can go beyond union and join operations for dataset combination (cf. Chapter 10) and automatically compute derived values.

Generalizing the approach of derived units is not without challenges, though. Some relationships are more complex than just applying some arithmetic operators, but pose additional restrictions on the operands. Consider "annual increase in population" as an example. The mathematical connection is a mere difference between two operands. However, the remaining context has to be equal (e.g., the region or the age cohort observed) and both operands have to originate from consecutive years. A model would not only have to include the arithmetic operation (subtraction) but also the additional restriction (data from consecutive years). Another challenge is given by the sheer endless possibilities to combine existing values. Any static model will miss out on some links and still be subject to more than exponential growth. This suggests that a more dynamic approach is advised. Instead of modeling certain connections repetitively for all concepts, a combination of Natural Language Processing (NLP) and semantic concepts could separate structure from content. While one part of the model describes the basic concepts (essentially the current status), another part focuses on just the connections without particular instantiations. These connections could, e.g., be just "annual increase" without the connection to population. User input would need to be parsed in the component describing the content ("population") and the one for structure ("annual increase"). By combining the two on-the-fly a system can then trigger the respective computations without the need to materialize all possible combinations beforehand.

Support for user uploaded datasets. Yavaa currently includes only rudimentary support for datasets uploaded by users. In particular, this was disabled throughout the evaluation and used only during early testing. However, for a productive system users need to be able to augment datasets with their own local data. Most modules in Yavaa need some information about the data they are working with in order to function properly. This metadata is initially lacking for local datasets provided by users and needs to be generated on the fly. Determining datatypes, roles, and concepts of and in columns corresponds to tasks within Semantic Table Annotation (STA) [296, 297]. However, the scope is somewhat extended: Where STA assumes only a single subject per row, statistical datasets oftentimes have multiple dimensional columns of equal rank (e.g., two countries and their trade balance). Further, current research into STA assumes that most or even all of the information in the table is already included in the knowledge graph and the task boils down to matching table and knowledge graph information. When considering arbitrary datasets, only the entities by themselves will be included in the knowledge graph, though. Most of their relations and especially the measurements themselves will be missing, making the matching tasks much more challenging.

Explore and evaluate other strategies to cope with an overlap in dataset combination. Chapter 10 describes an algorithm to generate a workflow for a user-requested dataset by combining existing datasets on-the-fly. During such a combination individual parts of the resulting dataset may be provided by different sources that are possibly contradicting one another. The proposed algorithm adopted a strategy of maximum coherence, i.e. by favoring datasets that can cover larger parts of the final result over smaller ones. This follows the assumption that the source datasets in themselves are coherent and any conflicts will arise from combining datasets from different sources. However, other strategies might be possible here. Different approaches might consider the values of all datasets in question. First, they gather all datasets in question and combine all of them resulting in cells with a set of values. In a second conflict resolving step, these sets can be reduced into a single value using different techniques. Naïve techniques might resort to a simple majority voting or are simply compute the average or median of all available values. Others might consider additional aspects like some kind of trustworthiness of the respective sources, outlier detection within a cell before averaging or maximizing consistency with cell in the immediate surrounding[2].

Multiple factors might influence the choice of an approach. The primary goal is likely to provide "correct data", whatever this means in a particular context. Apart from that, other factors can be the availability of information and the performance of computations. Having information about the trustworthiness of sources can also be a deciding factor. Similarly, when a particular approach substantially deteriorates the performance of a system and users have to wait a long time even for rather simple queries, such an approach will not gain any traction. Another aspect to consider is the tracking of the provenance as outlined in Chapter 12. More complex approaches will increase the amount of provenance data to be gathered. This may go up to a point where it is no longer maintainable in the current form and new ways of storing and handling provenance data have to be found.

Track column semantics across computations. Similar to the aforementioned modeling of derived quantity kinds in order to serve more complex user queries directly, the same techniques can be used to track the meaning of individual columns across computations. Right now, many operations lose the semantic meaning of a particular column – Yavaa has no way of knowing what the resulting meaning of a column is that has been derived by combining multiple existing ones. In contrast to dataset combination, here input and output are switched. Instead of determining the necessary computations given a certain concept, the computations are given and the concept is asked for. In general, many techniques will be able to handle both directions. However, formulae

[2]Here, the surrounding is defined according to data dimensions. However, the challenge remains to define a proper measure for closeness for each dimension and determine their importance in relation to one another. In a dataset about population counts, a dimension "country" requires a binary distance measure (basically only assigning equal values as close). On the other hand, a dimension "year" can use a Euclidean distance measure. With regard to the importance, the "country" is essential, whereas the "year" is less so.

used for the computation might vary in their form and thus make it harder to determine the corresponding concept. So before matching against a repository of known concepts and their relations can be performed, formulae have to be normalized into a canonical form.

Explore semantics-driven operations. In this thesis and most other systems, operations that can be applied to a dataset's columns are either generic or at most driven by the corresponding data type. However, when the values of a column are embedded into a semantic network, the intrinsic information could be exploited to offer different ways of interacting with the data. For example, consider a column containing geographic entities like cities. When those values are part of a larger knowledge graph, the interface can offer semantic-driven aggregation functions like "aggregate by country"[3]. Similarly, custom filter operations are possible once the context of values is known to a system.

Extend the usage of data characteristics in visualization recommendation. Chapter 7 and 11 outlined a way to capture the technical requirements a visualization poses on the underlying dataset. The aspects used there are by no means exhaustive. Other data characteristics may in a similar way contribute to the recommendations made by Yavaa. For example, the distinction between continuous and discrete values in a measurement column can drive the distinction between a line and a bar chart. Having a large number of duplicates in a dataset will yield a different visualization than having only unique entries.

Another characteristic to be exploited is the initial thrust of a dataset. Even though their superficial structure might seem closely related, it is all tables, after all, there are different kinds of datasets. In time series data, numeric values will likely be close to one another. On the other hand, graphs will have at least have two categorical columns that draw values from a shared pool, labels of the respective nodes. Further analysis of the correlations among nodes might yield the fact that this is actually a hierarchy instead of a generic graph structure. Determining the category the current dataset belongs to, further enables reducing the number of suggested candidates. For example, a dataset consisting of two categorical dimensions and a numeric measurement can be displayed using a tabular heatmap. However, if the category suggests this is a generic graph, a force feedback layout mapping the numeric values to distances between nodes or edge weights might be more appropriate.

Consider further aspects in visualization recommendation. The approach presented in Chapter 11 is based almost exclusively on technical data characteristics. However, other aspects can contribute as well [298]. As outlined in Chapter 1, visualizations serve a specific purpose or task. So when a system has an understanding of what tasks underlie the current workflow, it can

[3]This particular behavior could be accomplished the Mannheim Search Join Engine [61] (cf. Section 10.1). However, with that system users would first need to extend their current dataset with the respective information, before being able to aggregate it in a second step.

also suggest more appropriate visualizations. Further, users and their communities might have specific preferences for one visualization over another. While the respective information can be queried explicitly from users, the true goal would be to determine them automatically. Having a user model and by extension a community model in addition to the characteristics of the current dataset, a system might be able to make educated guesses[4] on tasks and general preferences. Including these and other inputs more in the visualization recommendation will improve not only its prediction accuracy but also reduce the number of "useless" visualizations.

Explore recommendation of scales in visualizations. Scales are an important part of any visualizations and control how values are mapped to (spatial) artifacts. While Wilkinson stressed this fact in *The Grammar of Graphics* [50], most visualization recommender systems omit this aspect (cf. Chapter 11). The default criterion to choose a scale would be to make efficient use of available space. For example, a scatter plot is rather useless when all values are concentrated in a small section of the canvas. Similarly, a bar chart for vastly different values may not be useful as some bars might not be recognizable anymore. In both cases, a more adequate scale might be of help. Changing the range of the scale might yield better results in the first case, while the second one could benefit from switching to a logarithmic scale. As with visualizations, determining a proper scale is influenced by many factors: Most important is the distribution of values along the scale. But also the conventions for particular visualizations have to be considered. For example, line charts are generally assumed to show the x-axis at a y-value of zero. While technically correct and feasible, deviating from that convention increases the mental capacity needed to understand the visualization and might lead to easier misinterpretation.

17.2 Software Engineering

Increase system performance. While Yavaa's performance was demonstrated to be well suited for the posed tasks, lowering technical requirements and increasing overall performance are natural goals for any system in production. For Yavaa, multiple different venues are possible. On an implementation level, the last years saw most browser vendors implementing support for WebAssembly [web45]. Its code is not parsed and compiled at runtime like JavaScript, but is processed beforehand. This allows to apply a wider range of static analysis and thus optimizations that would be too costly for just-in-time compiler. On a more conceptual level, the workflow itself could be optimized. The database community has a long-standing experience in query planning and execution. The operations performed in both environments, Yavaa and databases, are closely related. So transferring proven techniques from databases to Yavaa may induce better ways to do

[4] A guessing system translates into a probability distribution over available possibilities in its implementation.

certain computations. In particular, these techniques can be applied for the execution of already existing workflows. Here, optimization can not only happen on a local, step-by-step basis but consider the entire workflow at once.

Another indirect way of increasing performance is given by offloading some computations to the data providers themselves. While CSV/TSV-based formats currently do not allow for complex queries[5], other methods of data access offer computational capabilities. For example, publishing datasets using the RDF Data Cube Vocabulary [web9] via a SPARQL endpoint would provide systems with the opportunity to offload some computations to data providers. Initial filter operations triggered by the data combination dialog could be encoded in the corresponding SPARQL queries. Besides reducing the computations necessary on the client-side, this would also reduce the required amount of data transfers between a data provider and client. As with all public offerings of computational capacity, fair-use policies need to be respected and fail-safes have to be put in place in case the respective service is unavailable.

Extend supported data providers. Yavaa's implementation is prototypical and focused on only one data provider: Eurostat. For a production system to demonstrate its full potential, multiple such providers as diverse as possible need to be supported. For the most part, this will boil down to implementation tasks, as Yavaa's architecture already accounts for multiple providers to be used. In particular, two modules would need to be extended or refactored: the data loaders mentioned in Section 13.1 and the crawler described in Section 14.1. Adaptations in these two components could allow to include basically any data provider offering tabular data.

Harmonize vocabulary and root it in common knowledge bases. The prototype used in the evaluation simplified the vocabulary to describe dataset and column contents by minting new IRIs for each header after normalization. However, this removes any semantic connection across different headers almost completely. A more sophisticated approach should reuse the concepts of common knowledge bases like Wikidata [web69] or DBpedia [web163] wherever possible. Such a foundation will further foster data exchange across organizations and ease data integration not only with Yavaa. When metadata stores are distributed across several providers, a common vocabulary can substantially reduce the efforts needed to integrate the data. In particular, it removes the necessity to traverse multiple links to determine whether two IRIs actually refer to the same or similar concepts.

Mine usage data for common operations and visualizations. Once a system like Yavaa is used in production, a wealth of information in form of executed workflows will become available. Such a resource can yield more insight into what data is commonly used, how it is combined and transformed, or how visualizations are chosen based on data semantics. Yavaa and similar

[5] An RFC for "URI Fragment Identifiers for the text/csv Media Type" [299] might have provided a way, but seems to never have gained enough traction.

systems can benefit from this usage data in order to improve and extend the existing recommendation facilities. Of particular interest are relationships between visualizations and data semantics. Users well versed in a particular domain can easily decide on a common data display for their data and domain. However, such knowledge is rather hard to obtain for users new to the field and generic systems. Having a large stockpile of formal descriptions for how specific datasets are translated into a visual representation, might bridge this gap. A system can formalize potential connections and subsequently include them in the recommendations made to users. The basis for such analysis is laid in the detailed tracking of provenance as described in Chapter 12. It includes not only the data semantics in form of concepts for each column but also the detailed mapping of which columns is translated to which artifact.

Extend the visualization store in quantity and quality. Yavaa's current implementation features only a small number of visualizations. While implementing a given visualization is rather easy, determining its requirements is not. This challenge only grows when more visualizations and their descriptions are to be added or new types of requirements are introduced. As outlined in Chapter 7, most of the thresholds used to describe a visualization are fuzzy and subject to personal preferences. The challenge is to determine proper thresholds for a large number of visualizations that still somewhat represent a consensus across all stakeholders. Approaching this task objectively and systematically could be realized by analyzing large quantities of visualizations including their source data. The increasing drive in scholarly publications towards publishing source data along with derived texts and visualizations might provide the raw data at least for the scientific domain. The analysis would then need to involve several steps: Encountered visualizations have to be classified independent of possible visual variations. Further, data and other characteristics have to be deduced from both the source data and the context of the visualization, most often given by a caption and/or surrounding text. Finally, all that information has to be cleaned and analyzed to determine the range for each requirement and visualization that should be included in the system. Such an approach would likely not only expand the range of visualizations covered, but also base the chosen thresholds in their requirements on actual use instead of a few opinions.

List of Tables

List of Figures

List of Code-listings

References

[1] I. Turgenev. *Fathers and Sons*. Trans. Russian by R. Hare.
The University of Adelaide Library, Dec. 17, 2014.

[2] J. Snow. *On the mode of communication of cholera*. London, UK: John Churchill, 1855.

[3] C.-J. Minard. *Des tableaux graphiques et des cartes figuratives*. impr. de Thunot (Paris), 1862.
ark: ark:/12148/bpt6k56900532.

[4] J. Zhang, K. A. Johnson, J. T. Malin, and J. W. Smith.
"Human-centered information visualization".
In: *International workshop on dynamic visualizations and learning, Tubingen, Germany*. 2002.

[5] T.-M. Rhyne, M. Tory, T. Munzner, M. Ward, C. R. Johnson, and D. H. Laidlaw.
"Information and scientific visualization: separate but equal or happy together at last".
In: *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*. IEEE, Oct. 2003.
DOI: 10.1109/visual.2003.1250428.

[6] D. Weiskopf, K.-L. Ma, J. J. van Wijk, R. Kosara, and H. Hauser.
"Scivis, infovis-bridging the community divide".
In: *Proceedings of the IEEE Visualization Conference. IEEE*. 2006.

[7] D. Reinsel, J. Gantz, and J. Rydning.
"Data age 2025: the digitization of the world from edge to core". In: *Seagate* (2018).

[8] M. Hilbert and P. Lopez.
"The World's Technological Capacity to Store, Communicate, and Compute Information".
In: *Science* 332.6025 (Feb. 2011), pp. 60–65. DOI: 10.1126/science.1200970.

[9] A. McAfee and E. Brynjolfsson. "Big Data: The Management Revolution".
In: *Harvard business review* 90 (Oct. 2012), pp. 60–6, 68, 128.

[10] J. Attard, F. Orlandi, S. Scerri, and S. Auer.
"A systematic review of open government data initiatives".
In: *Government Information Quarterly* 32.4 (Oct. 2015), pp. 399–418.
DOI: 10.1016/j.giq.2015.07.006.

[11] OECD. *Open Government Data Report*. OECD, Sept. 2018, p. 264.
DOI: 10.1787/9789264305847-en.

[12] World Wide Web Foundation. *Open Data Barometer: Leaders Edition*. Tech. rep.
Washington DC: World Wide Web Foundation, 2018.

[13] V. Gewin. “Data sharing: An open mind on open data”.
In: *Nature* 529.7584 (Jan. 2016), pp. 117–119. DOI: 10.1038/nj7584-117a.

[14] M. Kindling, H. Pampel, S. van de Sandt, J. Rücknagel, P. Vierkant, G. Kloska, M. Witt, P. Schirmbacher, R. Bertelmann, and F. Scholze.
“The Landscape of Research Data Repositories in 2015: A re3data Analysis”.
In: *D-Lib Magazine* 23.3/4 (Mar. 2017). DOI: 10.1045/march2017-kindling.

[15] “Council of Europe Convention on Access to Official Documents”.
In: *Council of Europe Treaty Series* 205 (June 18, 2006).

[16] H.-J. Schulz, T. Nocke, M. Heitzler, and H. Schumann.
“A Design Space of Visualization Tasks”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (Dec. 2013), pp. 2366–2375. DOI: 10.1109/tvcg.2013.120.

[17] B. Shneiderman.
“The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations”.
In: *Proceedings of the 1996 IEEE Symposium on Visual Languages*. VL ’96.
Washington, DC, USA: IEEE Computer Society, 1996, pp. 336–.

[18] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann.
“Interactive Lenses for Visualization: An Extended Survey”. In: *Computer Graphics Forum*.
Vol. 36. 6. Wiley Online Library. Wiley, May 2016, pp. 173–200. DOI: 10.1111/cgf.12871.

[19] S. K. Card, J. D. Mackinlay, and B. Shneiderman.
Readings in information visualization: using vision to think. Morgan Kaufmann, 1999.

[20] A. Zuiderwijk, M. Janssen, S. Choenni, R. Meijer, and R. Sheikh Alibaks.
“Socio-technical Impediments of Open Data”.
In: *Electronic Journal of e-Government* 10.2 (Dec. 1, 2012).

[21] M. Beno, K. Figl, J. Umbrich, and A. Polleres.
“Perception of Key Barriers in Using and Publishing Open Data”.
In: *JeDEM - eJournal of eDemocracy and Open Government* 9.2 (Dec. 2017), pp. 134–165.
DOI: 10.29379/jedem.v9i2.465.

[22] K. Gregory, P. Groth, A. Scharnhorst, and S. Wyatt.
“Lost or Found? Discovering Data Needed for Research”.
In: *Harvard Data Science Review* (Apr. 2020). DOI: 10.1162/99608f92.e38165eb.

[23] L. M. Koesten, E. Kacprzak, J. F. Tennison, and E. Simperl.
“The Trials and Tribulations of Working with Structured Data”.
In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems - CHI ’17*.
ACM. ACM Press, 2017, pp. 1277–1289. DOI: 10.1145/3025453.3025838.

[24] E. Kacprzak, L. M. Koesten, L.-D. Ibáñez, E. Simperl, and J. Tennison.
“A Query Log Analysis of Dataset Search”. In: *Lecture Notes in Computer Science*.
Ed. by J. Cabot, R. De Virgilio, and R. Torlone. Cham: Springer International Publishing, 2017,
pp. 429–436. DOI: 10.1007/978-3-319-60131-1_29.

[25] N. Noy, M. Burgess, and D. Brickley.
"Google Dataset Search: Building a search engine for datasets in an open Web ecosystem".
In: *The World Wide Web Conference on - WWW '19*. ACM Press, 2019.
DOI: 10.1145/3308558.3313685.

[26] S. Zhang and K. Balog. "Ad Hoc Table Retrieval using Semantic Similarity".
In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*.
ACM Press, 2018. DOI: 10.1145/3178876.3186067.

[27] S. Zhang and K. Balog. "Semantic Table Retrieval Using Keyword and Table Queries".
In: *ACM Transactions on the Web* 15.3 (May 2021), pp. 1–33. DOI: 10.1145/3441690.

[28] S. A. W.M., M. Worring, S. Santini, A. Gupta, and R. Jain.
"Content-Based Image Retrieval at the End of the Early Years". In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 22.12 (Dec. 2000), pp. 1349–1380.
DOI: 10.1109/34.895972.

[29] T. Berners-Lee, J. Hendler, and O. Lassila. "The Semantic Web".
In: *Scientific American* 284.5 (May 2001), pp. 34–43.
DOI: 10.1038/scientificamerican0501-34.

[30] A. Hogan, E. Blomqvist, M. Cochez, C. D'amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann. "Knowledge Graphs".
In: *ACM Computing Surveys* 54.4 (July 2021), pp. 1–37. DOI: 10.1145/3447772.

[31] H. Bast, B. Buchhold, E. Haussmann, et al. "Semantic Search on Text and Knowledge Bases".
In: *Foundations and Trends® in Information Retrieval* 10.1 (2016), pp. 119–271.
DOI: 10.1561/1500000032.

[32] W. Hu, H. Qiu, J. Huang, and M. Dumontier.
"BioSearch: a semantic search engine for Bio2RDF". In: *Database* 2017 (2017).
DOI: 10.1093/database/bax059.

[33] S. Shekarpour, S. Auer, A.-C. N. Ngomo, D. Gerber, S. Hellmann, and C. Stadler.
"Keyword-Driven SPARQL Query Generation Leveraging Background Knowledge".
In: *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. IEEE Computer Society. IEEE, Aug. 2011, pp. 203–210.
DOI: 10.1109/wi-iat.2011.70.

[34] E. M. Voorhees. *Query Expansion using Lexical-Semantic Relations*. 1994.
DOI: 10.1007/978-1-4471-2099-5_7.

[35] F. Löffler and F. Klan. "Does Term Expansion Matter for the Retrieval of Biodiversity Data?"
In: *Joint Proceedings of the Posters and Demos Track of the 12th International Conference on Semantic Systems - SEMANTiCS2016 and the 1st International Workshop on Semantic Change & Evolving Semantics (SuCCESS'16), co-located with the 12th International Conference on Semantic Systems (SEMANTiCS 2016)*.
Ed. by M. Martin, M. Cuquet, and E. Folmer. CEUR Workshop Proceedings, 2016.

[36] T. Tran, H. Wang, and P. Haase.
"Hermes: Data Web search on a pay-as-you-go integration infrastructure". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 189–203.
DOI: 10.1016/j.websem.2009.07.001.

[37] Y. Wu, S. Yang, M. Srivatsa, A. Iyengar, and X. Yan.
"Summarizing answer graphs induced by keyword queries".
In: *Proceedings of the VLDB Endowment* 6.14 (Sept. 2013), pp. 1774–1785.
DOI: 10.14778/2556549.2556561.

[38] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati.
"Linking Data to Ontologies". In: *Journal on Data Semantics X*.
Springer Berlin Heidelberg, 2008, pp. 133–173. DOI: 10.1007/978-3-540-77688-8_5.

[39] *The JSON Data Interchange Format*. Tech. rep. ECMA, Oct. 2013.

[40] ISO. *ISO 17369:2013 - Statistical Data and Metadata eXchange (SDMX)*. ISO Standard. ISO, Jan. 2013.

[41] Eurostat (European Commission), ed. *Regions in the European Union. Nomenclature of territorial units for statistics, NUTS 2016/EU-28 : edition 2018*. Dec. 6, 2018.
DOI: 10.2785/475524.

[42] ISO. *ISO 3166-1:2013 - Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*. ISO Standard. ISO, Nov. 2013.

[43] ISO. *ISO 3166-2:2013 - Codes for the representation of names of countries and their subdivisions – Part 2: Country subdivision code*. ISO Standard. ISO, Nov. 2013.

[44] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer.
"Wrangler: Interactive visual specification of data transformation scripts".
In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*.
ACM. 2011, pp. 3363–3372. DOI: 10.1145/1978942.1979444.

[45] W. H. Inmon. *Building the data warehouse*. John Wiley & Sons, 2005.

[46] G. H. Lockwood. *Final report of the Board of Inquiry investigating the circumstances of an accident involving the Air Canada Boeing 767 aircraft C-GAUN that effected an emergency landing at Gimli, Manitoba on the 23rd day of July, 1983*. English.
Government of Canada [Ottawa], 1985, vi, 199 p.

[47] P. G. Neumann. "Letter from the editor". In: *SIGSOFT Softw. Eng. Notes* 10.3 (1985).

[48] *Mars Climate Orbiter Mishap Investigation Board Phase I Report*. Tech. rep. NASA, Nov. 1999.

[49] J. Bertin. *Semiology of Graphics - Diagrams, Networks, Maps*. ESRI, 2010.

[50] L. Wilkinson. *The Grammar of Graphics*. Springer Science & Business Media, 2006.

[51] S. F. Roth and J. Mattis. "Data characterization for intelligent graphics presentation".
In: *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*. ACM Press, 1990. DOI: 10.1145/97243.97273.

[52] M. Brehmer and T. Munzner. "A Multi-Level Typology of Abstract Visualization Tasks". In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (Dec. 2013), pp. 2376–2385.
DOI: 10.1109/tvcg.2013.124.

[53] D. A. Scheufele and N. M. Krause. “Science audiences, misinformation, and fake news”. In: *Proceedings of the National Academy of Sciences* 116.16 (Jan. 2019), pp. 7662–7669. DOI: 10.1073/pnas.1805871115.

[54] S. Vosoughi, D. Roy, and S. Aral. “The spread of true and false news online”. In: *Science* 359.6380 (Mar. 2018), pp. 1146–1151. DOI: 10.1126/science.aap9559.

[55] R. Peng. “The reproducibility crisis in science: A statistical counterattack”. In: *Significance* 12.3 (June 2015), pp. 30–32. DOI: 10.1111/j.1740-9713.2015.00827.x.

[56] M. Hutson. “Artificial intelligence faces reproducibility crisis”. In: *Science* 359.6377 (Feb. 2018), pp. 725–726. DOI: 10.1126/science.359.6377.725.

[57] L. Hatton and M. van Genuchten. “Computational Reproducibility: The Elephant in the Room”. In: *IEEE Software* 36.2 (Mar. 2019), pp. 137–144. DOI: 10.1109/ms.2018.2883805.

[58] S. Khalsa, P. Cotroneo, and M. Wu. *A survey of current practice of data search services*. May 2018. DOI: 10.17632/7j43z6n22z.1.

[59] F. Löffler, V. Wesp, B. König-Ries, and F. Klan. “Dataset Search In Biodiversity Research: Do Metadata In Data Repositories Reflect Scholarly Information Needs?” In: *PLOS ONE* 16.3 (Mar. 2021). Ed. by H. Suleman, e0246099. DOI: 10.1371/journal.pone.0246099.

[60] S. Schindler, M. Paradies, and A. Twele. “Here is my query, where are my results? A search log analysis of the EOWEB® Geoportal”. In: *Conference on Big Data from Space: Turning Data into Insights (BiDS’19)*. 2019, pp. 1–4.

[61] O. Lehmberg, D. Ritze, P. Ristoski, R. Meusel, H. Paulheim, and C. Bizer. “The Mannheim Search Join Engine”. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 35 (Dec. 2015), pp. 159–166. DOI: 10.1016/j.websem.2015.05.001.

[62] R. Tuchinda, P. Szekely, and C. A. Knoblock. “Building data integration queries by demonstration”. In: *Proceedings of the 12th international conference on Intelligent user interfaces - IUI ’07*. ACM Press, 2007. DOI: 10.1145/1216295.1216328.

[63] M. J. Cafarella, A. Halevy, and N. Khoussainova. “Data integration for the relational web”. In: *Proceedings of the VLDB Endowment* 2.1 (Aug. 2009), pp. 1090–1101. DOI: 10.14778/1687627.1687750.

[64] S. Kasica, C. Berret, and T. Munzner. “Table Scraps: An Actionable Framework for Multi-Table Data Wrangling From An Artifact Study of Computational Journalism”. In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (Feb. 2021), pp. 957–966. DOI: 10.1109/tvcg.2020.3030462.

[65] C. Scaffidi, B. Myers, and M. Shaw. “Intelligently creating and recommending reusable reformatting rules”. In: *Proceedingsc of the 13th international conference on Intelligent user interfaces - IUI ’09*. ACM Press, 2008. DOI: 10.1145/1502650.1502692.

[66] E. Rahm and H. H. Do. “Data cleaning: Problems and current approaches”.
In: *IEEE Data Eng. Bull.* 23.4 (2000), pp. 3–13.

[67] R. A. Tariq and S. Sharma. “Inappropriate Medical Abbreviations”. In: *StatPearls [Internet]*. StatPearls Publishing, 2019.

[68] K. A. Baggerly and K. R. Coombes. “Deriving chemosensitivity from cell lines: Forensic bioinformatics and reproducible research in high-throughput biology”.
In: *The Annals of Applied Statistics* 3.4 (Dec. 2009), pp. 1309–1334.
DOI: 10.1214/09-aoas291.

[69] T. Herndon, M. Ash, and R. Pollin.
“Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff”.
In: *Cambridge Journal of Economics* 38.2 (Dec. 2013), pp. 257–279.
DOI: 10.1093/cje/bet075.

[70] M. Ziemann, Y. Eren, and A. El-Osta.
“Gene name errors are widespread in the scientific literature”.
In: *Genome Biology* 17.1 (Aug. 2016). DOI: 10.1186/s13059-016-1044-7.

[71] E. Santos, L. Lins, J. Ahrens, J. Freire, and C. Silva.
“VisMashup: Streamlining the Creation of Custom Visualization Applications”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009), pp. 1539–1546.
DOI: 10.1109/tvcg.2009.195.

[72] L. Grammel, M. Tory, and M.-A. Storey.
“How Information Visualization Novices Construct Visualizations”.
In: *IEEE Transactions on Visualization and Computer Graphics* 16.6 (Nov. 2010), pp. 943–952.
DOI: 10.1109/tvcg.2010.164.

[73] B. E. Rogowitz, L. A. Treinish, and S. Bryson. “How Not to Lie with Visualization”.
In: *Computers in Physics* 10.3 (1996), p. 268. DOI: 10.1063/1.4822401.

[74] P. Fox and J. Hendler. “Changing the Equation on Scientific Data Visualization”.
In: *Science* 331.6018 (Feb. 2011), pp. 705–708. DOI: 10.1126/science.1197654.

[75] H. E. Plesser. “Reproducibility vs. Replicability: A Brief History of a Confused Terminology”.
In: *Frontiers in Neuroinformatics* 11 (Jan. 2018). DOI: 10.3389/fninf.2017.00076.

[76] R. Abraham and M. Erwig.
“Header and unit inference for spreadsheets through spatial analyses”.
In: *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*. IEEE. 2004, pp. 165–172.

[77] P. W. Koch, B. Hofer, and F. Wotawa. “Static Spreadsheet Analysis”. In: *Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on*. IEEE. 2016, pp. 167–174. DOI: 10.1109/issrew.2016.8.

[78] H. Gonzalez, A. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, and W. Shen.
“Google fusion tables: data management, integration and collaboration in the cloud”.
In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010, pp. 175–180.
DOI: 10.1145/1807128.1807158.

[79] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. "Bigtable: A Distributed Storage System for Structured Data." In: *Inc. In proceeding of the OSDI* 6 (2006). DOI: 10.1145/1365815.1365816.

[80] C. R. Stolte, D. Tang, and P. Hanrahan. "Polaris: A System for Query, Analysis, and Visualization of Multidimensional Relational Databases". In: *IEEE Trans. Vis. Comput. Graph.* 8.1 (2002), pp. 52–65. DOI: 10.1109/2945.981851.

[81] ISO. *ISO/IEC 9075-1:2016 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*. ISO Standard. ISO, Dec. 2016.

[82] J. Gray, A. Bosworth, A. Lyaman, and H. Pirahesh. "Data cube: a relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS". In: *Proceedings of the Twelfth International Conference on Data Engineering*. Feb. 1996, pp. 152–159. DOI: 10.1109/ICDE.1996.492099.

[83] K. Thomas, R.-K. Benjamin, P. Fernando, G. Brian, B. Matthias, F. Jonathan, K. Kyle, H. Jessica, G. Jason, C. Sylvain, I. Paul, A. Damián, A. Safia, W. Carol, and J. D. Team. "Jupyter Notebooks - a publishing format for reproducible computational workflows". In: 0 (2016), pp. 87–90. DOI: 10.3233/978-1-61499-649-1-87.

[84] D. E. Knuth. "Literate Programming". In: *The Computer Journal* 27.2 (Feb. 1984), pp. 97–111. DOI: 10.1093/comjnl/27.2.97.

[85] F. Perez and B. E. Granger. "IPython: A System for Interactive Scientific Computing". In: *Computing in Science & Engineering* 9.3 (2007), pp. 21–29. DOI: 10.1109/mcse.2007.53.

[86] B. Abbott, R. Abbott, T. Abbott, et al. "Observation of Gravitational Waves from a Binary Black Hole Merger". In: *Physical Review Letters* 116.6 (Feb. 2016), p. 061102. DOI: 10.1103/physrevlett.116.061102.

[87] E. Brunk, K. W. George, J. Alonso-Gutierrez, M. Thompson, E. Baidoo, G. Wang, C. J. Petzold, D. McCloskey, J. Monk, L. Yang, E. J. O'Brien, T. S. Batth, H. G. Martin, A. Feist, P. D. Adams, J. D. Keasling, B. O. Palsson, and T. S. Lee. "Characterizing Strain Variation in Engineered E. coli Using a Multi-Omics-Based Workflow". In: *Cell Systems* 2.5 (May 2016), pp. 335–346. DOI: 10.1016/j.cels.2016.04.004.

[88] R. Connelly and V. Gayle. "An investigation of social class inequalities in general cognitive ability in two British birth cohorts". In: *The British Journal of Sociology* 70.1 (Dec. 2017), pp. 90–108. DOI: 10.1111/1468-4446.12343.

[89] S. B. Rosenthal, J. Len, M. Webster, A. Gary, A. Birmingham, and K. M. Fisch. "Interactive network visualization in Jupyter notebooks: visJS2jupyter". In: *Bioinformatics* 34.1 (Sept. 2017). Ed. by O. Stegle, pp. 126–128. DOI: 10.1093/bioinformatics/btx581.

[90] M. Reich, T. Tabor, T. Liefeld, H. Thorvaldsdóttir, B. Hill, P. Tamayo, and J. P. Mesirov. "The GenePattern Notebook Environment". In: *Cell Systems* 5.2 (Aug. 2017), 149–151.e1. DOI: 10.1016/j.cels.2017.07.003.

[91] J. F. N. Pimentel, V. Braganholo, L. Murta, and J. Freire. “Collecting and Analyzing Provenance on Interactive Notebooks: When IPython Meets noWorkflow”. In: *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*. Edinburgh, Scotland: USENIX Association, July 2015.

[92] S. Samuel and B. König-Ries. “ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility”. In: *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, USA, October 8th - to - 12th, 2018*. Ed. by M. van Erp, M. Atre, V. López, K. Srinivas, and C. Fortuna. Vol. 2180. CEUR Workshop Proceedings. CEUR-WS.org, 2018.

[93] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire. “A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks”. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, May 2019. DOI: 10.1109/msr.2019.00077.

[94] A. Rule, A. Tabard, and J. D. Hollan. “Exploration and Explanation in Computational Notebooks”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2018. DOI: 10.1145/3173574.3173606.

[95] C. Goble, C. Wroe, and R. Stevens. *The myGrid Project: Services, Architecture and Demonstrator*. 2003.

[96] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, et al. “The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud”. In: *Nucleic acids research* 41.W1 (2013), W557–W561. DOI: 10.1093/nar/gkt328.

[97] T. Oinn, M. J. Addis, J. Ferris, D. J. Marvin, M. Greenwood, C. Goble, A. Wipat, P. Li, and T. Carver. *Delivering Web Service Coordination Capability to Users*. Conference or Workshop Item; PeerReviewed. 2004.

[98] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. “Taverna, reloaded”. In: *International conference on scientific and statistical database management*. Springer. 2010, pp. 471–481. DOI: 10.1007/978-3-642-13818-8_33.

[99] J. Bhagat, F. Tanoh, E. Nzuobontane, T. Laurent, J. Orlowski, M. Roos, K. Wolstencroft, S. Aleksejevs, R. Stevens, S. Pettifer, R. Lopez, and C. A. Goble. “BioCatalogue: a universal catalogue of web services for the life sciences”. In: *Nucleic Acids Research* 38 (2010). DOI: 10.1093/nar/gkq394.

[100] M. Kanehisa. “KEGG: Kyoto Encyclopedia of Genes and Genomes”. In: *Nucleic Acids Research* 28.1 (Jan. 2000), pp. 27–30. DOI: 10.1093/nar/28.1.27.

[101] *Tracking Workflow Execution With Tavernaprov* (PROV: Three Years Later, Provenance Week 2016, June 6, 2016). Zenodo, 2016. DOI: 10.5281/zenodo.51314.

[102] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. "VisTrails: visualization meets data management". In: *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*. SIGMOD '06. Chicago, IL, USA: ACM Press, 2006, pp. 745–747. DOI: 10.1145/1142473.1142574.

[103] J. Freire, D. Koop, F. S. Chirigati, and C. T. Silva. "Reproducibility using vistrails". In: *Implementing Reproducible Research* 33 (2014).

[104] R. Fielding and J. Reschke. "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing". In: (June 2014). DOI: 10.17487/RFC7230.

[105] W. Schroeder, K. Martin, and W. Lorensen. "The design and implementation of an object-oriented toolkit for 3D graphics and visualization". In: *Proceedings of Seventh Annual IEEE Visualization '96*. IEEE, 1996. DOI: 10.1109/visual.1996.567752.

[106] L. Moreau, J. Freire, J. Futrelle, R. McGrath, J. Myers, and P. Paulson. *The Open Provenance Model*. Dec. 18, 2007.

[107] L. Bavoil, S. P. Callahan, C. E. Scheidegger, H. T. Vo, P. J. Crossno, C. T. Silva, and J. Freire. "VisTrails: Enabling Interactive Multiple-View Visualizations". In: *Visualization Conference, IEEE* 0 (2005), p. 18. DOI: http://doi.ieeecomputersociety.org/10.1109/VIS.2005.113.

[108] S. Callahan, J. Freire, E. Santos, C. Scheidegger, C. Silva, and H. Vo. *Using Provenance to Streamline Data Exploration through Visualization*. SCI Institute Technical Report UUSCI-2006-016. University of Utah, 2006.

[109] C. Scheidegger, H. Vo, D. Koop, J. Freire, and C. Silva. "Querying and Creating Visualizations by Analogy". In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1560–1567. DOI: 10.1109/tvcg.2007.70584.

[110] D. Koop, C. Scheidegger, S. Callahan, J. Freire, and C. Silva. "VisComplete: Automating Suggestions for Visualization Pipelines". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (Nov. 2008), pp. 1691–1698. DOI: 10.1109/tvcg.2008.174.

[111] D. L. Parnas. "Software aging". In: *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press. 1994, pp. 279–287.

[112] D. Koop, C. E. Scheidegger, J. Freire, and C. T. Silva. "The Provenance of Workflow Upgrades." In: *IPAW*. Springer. 2010, pp. 2–16. DOI: 10.1007/978-3-642-17819-1_2.

[113] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva. "Bridging Workflow and Data Provenance Using Strong Links." In: *SSDBM*. Vol. 10. Springer. 2010, pp. 397–415. DOI: 10.1007/978-3-642-13818-8_28.

[114] D. Koop, E. Santos, P. Mates, H. T. Vo, P. Bonnet, B. Bauer, B. Surer, M. Troyer, D. N. Williams, J. E. Tohline, et al. "A provenance-based infrastructure to support the life cycle of executable papers". In: *Procedia Computer Science* 4 (2011), pp. 648–657. DOI: 10.1016/j.procs.2011.04.068.

[115] M. D. McIlroy, E. N. Pinson, and B. A. Tague. “UNIX Time-Sharing System: Foreword”. In: *Bell System Technical Journal* 57.6 (July 1978), pp. 1899–1904. DOI: 10.1002/j.1538-7305.1978.tb02135.x.

[116] A. Doan. *Principles of data integration*. Waltham, MA: Morgan Kaufmann, 2012.

[117] W. McKnight. “Data Virtualization”. In: *Information Management*. Elsevier, 2014, pp. 86–96. DOI: 10.1016/b978-0-12-408056-0.00009-6.

[118] M. Shokouhi and L. Si. “Federated Search”. In: *Foundations and Trends® in Information Retrieval* 5.1 (2011), pp. 1–102. DOI: 10.1561/1500000010.

[119] A. Chapman, E. Simperl, L. Koesten, G. Konstantinidis, L.-D. Ibáñez, E. Kacprzak, and P. Groth. “Dataset search: a survey”. In: *The VLDB Journal* 29.1 (Aug. 2019), pp. 251–272. DOI: 10.1007/s00778-019-00564-x.

[120] C. Bizer, T. Heath, and T. Berners-Lee. “Linked Data - The Story So Far”. In: *Int. J. Semantic Web Inf. Syst.* 5.3 (2009), pp. 1–22. DOI: 10.4018/jswis.2009081901.

[121] D. Vrandečić and M. Krötzsch. “Wikidata: A Free Collaborative Knowledgebase”. In: *Communications of the ACM* 57.10 (Sept. 2014), pp. 78–85. DOI: 10.1145/2629489.

[122] M. M. Zloof. “Query by example”. In: *Proceedings of the June 7-10, 1976, national computer conference and exposition on - AFIPS ’76*. ACM. ACM Press, 1976, pp. 431–438. DOI: 10.1145/1499799.1499914.

[123] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. *Design Patterns*. Prentice Hall, Dec. 1, 1995.

[124] S. S. Stevens. “On the Theory of Scales of Measurement”. In: *Science* 103.2684 (June 1946), pp. 677–680. DOI: 10.1126/science.103.2684.677.

[125] S. Dowdy, S. Wearden, and D. Chilko. *Statistics for research*. Vol. 512. John Wiley & Sons, 2011.

[126] J. Mackinlay, P. Hanrahan, and C. Stolte. “Show Me: Automatic Presentation for Visual Analysis”. In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1137–1144. DOI: 10.1109/tvcg.2007.70594.

[127] F. Bouali, A. Guettala, and G. Venturini. “VizAssist: an interactive user assistant for visual data mining”. In: *The Visual Computer* 32.11 (May 2016), pp. 1447–1463. DOI: 10.1007/s00371-015-1132-9.

[128] “IEEE Standard for Floating-Point Arithmetic”. In: *IEEE Std 754-2008* (Aug. 2008), pp. 1–70. DOI: 10.1109/IEEESTD.2008.4610935.

[129] ISO. *ISO 8601:2004 - Data elements and interchange formats – Information interchange – Representation of dates and times*. ISO Standard. ISO, Dec. 2004.

[130] E. F. Codd. “A relational model of data for large shared data banks”. In: *Communications of the ACM* 13.6 (6 June 1970), pp. 377–387. DOI: 10.1145/362384.362685.

[131] Y. Shafranovich. “Common Format and MIME Type for Comma-Separated Values (CSV) Files”. In: (Oct. 2005). DOI: 10.17487/rfc4180.

[132] B. Jelen and M. Alexander. *Pivot Table Data Crunching: Microsoft Excel 2010*. Pearson Education, 2010.

[133] E. F. Codd, S. B. Codd, and C. T. Salley. “Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate”. In: *Codd and Date* 32 (1993).

[134] *SDMX 2.1 User Guide*. Sept. 19, 2012.

[135] C. H. Coombs. “A theory of data.” In: (1964).

[136] A. Satyanarayan, K. Wongsuphasawat, and J. Heer. “Declarative Interaction Design for Data Visualization”. In: *ACM User Interface Software & Technology (UIST)*. ACM Press, 2014. DOI: 10.1145/2642918.2647360.

[137] B. Mutlu, E. Veas, and C. Trattner. “Vizrec: Recommending personalized visualizations”. In: *ACM Transactions on Interactive Intelligent Systems (TiiS)* 6.4 (2016), p. 31. DOI: 10.1145/2983923.

[138] C. R. Stolte. “Query, Analysis, and Visualization of Multidimensional Databases”. AAI3090686. PhD thesis. Stanford, CA, USA, 2003.

[139] P. Hanrahan. “VizQL: a language for query, analysis and visualization”. In: *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD ’06*. SIGMOD ’06. Chicago, IL, USA: ACM Press, 2006, pp. 721–721. DOI: 10.1145/1142473.1142560.

[140] M. Voigt, M. Franke, and K. Meissner. “Using expert and empirical knowledge for context-aware recommendation of visualization components”. In: *Int. J. Adv. Life Sci* 5 (2013), pp. 27–41.

[141] B. Mutlu, E. Veas, C. Trattner, and V. Sabol. “Vizrec: a two-stage recommender system for personalized visualizations”. In: *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion*. ACM. 2015, pp. 49–52. DOI: 10.1145/2732158.2732190.

[142] W. K. Michener, J. W. Brunt, J. J. Helly, T. B. Kirchner, and S. G. Stafford. “Nongeospatial Metadata for the Ecological Sciences”. In: *Ecological Applications* 7.1 (Feb. 1997), pp. 330–342. DOI: 10.1890/1051-0761(1997)007[0330:nmftes]2.0.co;2.

[143] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. ’. Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop,

A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, and B. Mons.
"The FAIR Guiding Principles for scientific data management and stewardship".
In: *Scientific Data* 3.1 (Mar. 2016). DOI: 10.1038/sdata.2016.18.

[144] S. Neumaier, J. Umbrich, and A. Polleres.
"Automated Quality Assessment of Metadata across Open Data Portals".
In: *Journal of Data and Information Quality* 8.1 (Nov. 2016), pp. 1–29.
DOI: 10.1145/2964909.

[145] A. Mauranen.
"Signaling and preventing misunderstanding in English as lingua franca communication".
In: *International Journal of the Sociology of Language* 2006.177 (Jan. 2006).
DOI: 10.1515/ijsl.2006.008.

[146] *ANSI/NISO Z39.85-2012 The Dublin Core Metadata Element Set*. NISO Standard (ANSI).
NISO, Feb. 25, 2013.

[147] J. A. Kunze and T. Baker. "The Dublin Core Metadata Element Set". In: (Aug. 2007).
DOI: 10.17487/rfc5013.

[148] ISO.
ISO 15836:2009 - Information and documentation – The Dublin Core metadata element set.
ISO Standard. ISO, Feb. 2009.

[149] R. Cyganiak, F. Maali, and V. Peristeras.
"Self-service linked government data with dcat and gridworks".
In: *Proceedings of the 6th International Conference on Semantic Systems - I-SEMANTICS '10*.
I-SEMANTICS '10. Graz, Austria: ACM Press, 2010, 37:1–37:3.
DOI: 10.1145/1839707.1839754.

[150] M. Nilsson, A. Powell, P. Johnston, and A. Naeve.
"Expressing Dublin Core metadata using the Resource Description Framework (RDF)".
In: *DCMI Recommendation* (2008).

[151] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao. "Describing Linked Datasets On the Design and Usage of voiD , the Vocabulary Of Interlinked Datasets".
In: *Design* 19 (2009). Ed. by C. Bizer, T. Heath, T. J. Berners-Lee, and K. Idehen.

[152] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer.
"DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia".
In: *Semantic Web* 6 (2015), pp. 167–195. DOI: 10.3233/SW-140134.

[153] R. Cyganiak, S. Field, A. Gregory, W. Halb, and W. Halb.
"Semantic Statistics : Bringing Together SDMX and SCOVO".
In: *Proceedings of the WWW2010 Workshop on Linked Data on the Web* (2010), pp. 2–6.

[154] E. H. Fegraus, S. Andelman, M. B. Jones, and M. Schildhauer.
"Maximizing the Value of Ecological Data with Structured Metadata: An Introduction to Ecological Metadata Language (EML) and Principles for Metadata Creation".
In: *Bulletin of the Ecological Society of America* 86.3 (July 2005), pp. 158–168.
DOI: 10.1890/0012-9623(2005)86[158:mtvoed]2.0.co;2.

[155] R. L. Rivest. “The MD5 Message-Digest Algorithm”. In: (Apr. 1992). DOI: 10.17487/rfc1321.

[156] P. Murray-Rust and H. S. Rzepa.
“STMML. A markup language for scientific, technical and medical publishing”.
In: *Data Science Journal* 1 (2002), pp. 128–192. DOI: 10.2481/dsj.1.128.

[157] B. Magagna, G. Moncoiffe, A. Devaraju, P. L. Buttigieg, M. Stoica, and S. Schindler.
“Towards an interoperability framework for observable property terminologies”.
In: (Mar. 2020). DOI: 10.5194/egusphere-egu2020-19895.

[158] B. Magagna, G. Moncoiffe, M. Stoica, A. Devaraju, A. Pamment, S. Schindler, and R. Huber.
“The I-ADOPT Interoperability Framework: a proposal for FAIRer observable property descriptions”. In: (Mar. 2021). DOI: 10.5194/egusphere-egu21-13155.

[159] P. W. Bridgman. *Dimensional analysis*. New Haven: Yale University Press, 1922.

[160] *The International System of Units (SI). Basic and general concepts and associated terms*.
8th ed. International Bureau of Weights and Measures (BIPM). 2014.

[161] J. M. Keil and S. Schindler.
“Comparison and evaluation of ontologies for units of measurement”.
In: *Semantic Web Journal* 10 (2018), pp. 33–51. DOI: 10.3233/SW-180310.

[162] M. D. Steinberg, S. Schindler, and J. M. Keil.
“Use Cases and Suitability Metrics for Unit Ontologies”.
In: *OWL: Experiences and Directions - Reasoner Evaluation - (OWLED 2016), (ORE 2016)*.
2016, pp. 40–54. DOI: 10.1007/978-3-319-54627-8_4.

[163] S. Schindler and J. M. Keil. “Building Ontologies for Reuse”.
In: *2nd International Workshop on Bad Or Good Ontology (BOG 2019)*. 2019.

[164] H. Rijgersberg, M. van Assem, and J. Top.
“Ontology of Units of Measure and Related Concepts”. In: *Semantic Web* 4.1 (2013), pp. 3–13.
DOI: 10.3233/SW-2012-0069.

[165] S. Krings and M. Leuschel. “Inferring physical units in formal models”.
In: *Software and System Modeling* 16 (2015), pp. 25–47. DOI: 10.1007/s10270-015-0458-0.

[166] M. Hills, F. Chen, and G. Rosu.
“A Rewriting Logic Approach to Static Checking of Units of Measurement in C”.
In: *Electr. Notes Theor. Comput. Sci.* 290 (2012), pp. 51–67.
DOI: 10.1016/j.entcs.2012.11.011.

[167] L. Jiang and Z. Su.
“Osprey: a practical type system for validating dimensional unit correctness of C programs”.
In: *Proceeding of the 28th international conference on Software engineering - ICSE ’06*.
ACM. ACM Press, 2006, pp. 262–271. DOI: 10.1145/1134285.1134323.

[168] G. S. Novak. “Conversion of Units of Measurement”.
In: *IEEE Trans. Software Eng.* 21.8 (1995), pp. 651–661. DOI: 10.1109/32.403789.

[169] G. S. Novak. “GLISP: A Lisp-Based Programming System with Data Abstraction”.
In: *AI Magazine* 4.3 (1983), pp. 37–47.

[170] O. Lehmberg, D. Ritze, P. Ristoski, K. Eckert, H. Paulheim, and C. Bizer.
"Extending tables with data from over a million websites".
In: *Semantic Web Challenge* 2014 (2014).

[171] C. Bizer. "Search Joins with the Web".
In: *In Proc. of the 17th Int. Conf. on Database Theory (ICDT)*. 2014.

[172] C. S. Bhagavatula, T. Noraset, and D. Downey.
"Methods for exploring and mining tables on wikipedia".
In: *Proceedings of the ACM SIGKDD Workshop on Interactive Data Exploration and Analytics*.
ACM. 2013, pp. 18–26. DOI: 10.1145/2501511.2501516.

[173] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer.
"A large public corpus of web tables containing time and context metadata".
In: *Proceedings of the 25th International Conference Companion on World Wide Web*.
International World Wide Web Conferences Steering Committee. 2016, pp. 75–76.
DOI: 10.1145/2872518.2889386.

[174] R. Meusel, P. Petrovski, and C. Bizer.
"The webdatacommons microdata, rdfa and microformat dataset series".
In: *International Semantic Web Conference*. Springer. 2014, pp. 277–292.
DOI: 10.1007/978-3-319-11964-9_18.

[175] J. Wang, G. Li, and J. Fe.
"Fast-join: An efficient method for fuzzy token matching based string similarity join".
In: *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE. 2011,
pp. 458–469. DOI: 10.1109/icde.2011.5767865.

[176] D. Rinser, D. Lange, and F. Naumann.
"Cross-lingual entity matching and infobox alignment in Wikipedia".
In: *Information Systems* 38.6 (2013), pp. 887–907. DOI: 10.1016/j.is.2012.10.003.

[177] M. Fabian, K. Gjergji, W. Gerhard, et al.
"Yago: A core of semantic knowledge unifying wordnet and wikipedia".
In: *16th International World Wide Web Conference, WWW*. 2007, pp. 697–706.

[178] C. Felbaum. *Wordnet, an Electronic Lexical Database for English*. 1998.

[179] P. Resnik. "Using information content to evaluate semantic similarity in a taxonomy".
In: *arXiv preprint cmp-lg/9511007* (1995).

[180] V. I. Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals".
In: *Doklady. Akademii Nauk SSSR* 163.4 (1965), pp. 845–848.

[181] J. Bleiholder and F. Naumann. "Data fusion".
In: *ACM Computing Surveys (CSUR)* 41.1 (2009), p. 1. DOI: 10.14778/1687553.1687620.

[182] S. Zhang and K. Balog. "On-the-fly Table Generation". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM, June 2018.
DOI: 10.1145/3209978.3209988.

[183] J. M. Ponte and W. B. Croft. “A language modeling approach to information retrieval”. In: *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’98*. ACM Press, 1998. DOI: 10.1145/290941.291008.

[184] J. Guo, Y. Fan, Q. Ai, and W. B. Croft. “A Deep Relevance Matching Model for Ad-hoc Retrieval”. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM, Oct. 2016. DOI: 10.1145/2983323.2983769.

[185] K. Sparck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Journal of Documentation* 28.1 (Jan. 1972), pp. 11–21. DOI: 10.1108/eb026526.

[186] S. Zhang and K. Balog. “EntiTables”. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, Aug. 2017. DOI: 10.1145/3077136.3080796.

[187] A. Kopliku, M. Boughanem, and K. Pinel-Sauvagnat. “Towards a framework for attribute retrieval”. In: *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM ’11*. ACM Press, 2011. DOI: 10.1145/2063576.2063654.

[188] S. Zhang and K. Balog. “Auto-completion for Data Cells in Relational Tables”. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. ACM, Nov. 2019. DOI: 10.1145/3357384.3357932.

[189] G. Sathe and S. Sarawagi. “Intelligent rollups in multidimensional OLAP data”. In: *VLDB*. Vol. 1. 2001, pp. 531–540.

[190] L. V. Lakshmanan, R. T. Ng, C. X. Wang, X. Zhou, and T. J. Johnson. “The generalized MDL approach for summarization”. In: *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment. 2002, pp. 766–777. DOI: 10.1016/b978-155860869-6/50073-1.

[191] A. O. Mendelzon and K. Q. Pu. “Concise descriptions of subsets of structured sets”. In: *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS ’03*. ACM. ACM Press, 2003, pp. 123–133. DOI: 10.1145/773153.773166.

[192] A. Guttman. *R-trees: A dynamic index structure for spatial searching*. Vol. 14. 2. ACM, 1984.

[193] S. Bu, L. V. S. Lakshmanan, and R. T. Ng. “MDL summarization with holes”. In: *Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment. 2005, pp. 433–444.

[194] A. Y. Halevy. “Answering queries using views: A survey”. In: *The VLDB Journal* 10.4 (Dec. 2001), pp. 270–294. DOI: 10.1007/s007780100054.

[195] A. K. Chandra and P. M. Merlin. “Optimal implementation of conjunctive queries in relational data bases”. In: *Proceedings of the ninth annual ACM symposium on Theory of computing*. ACM. ACM Press, 1977, pp. 77–90. DOI: 10.1145/800105.803397.

[196] A. Y. Levy, A. Rajaraman, and J. O. Joann.
"Querying heterogeneous information sources using source descriptions". In: *Proceedings of the 22nd International Conference on Very Large Databases, VLDB-96, Bombay, India*. 1996.

[197] A. Y. Levy and Y. Sagiv. "Queries independent of updates". In: *VLDB*. Vol. 93. 1993, pp. 171–181.

[198] K. A. Morris. "An algorithm for ordering subgoals in NAIL?" In: *Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '88*. ACM. ACM Press, 1988, pp. 82–88. DOI: 10.1145/308386.308419.

[199] R. Pottinger and A. Y. Levy. "A scalable algorithm for answering queries using views". In: *VLDB*. 2000, pp. 484–495.

[200] R. Pottinger and A. Halevy.
"MiniCon: A scalable algorithm for answering queries using views".
In: *The VLDB Journal* 10.2 (Sept. 2001), pp. 182–198. DOI: 10.1007/s007780100048.

[201] S. Abiteboul and O. M. Duschka. "Complexity of answering queries using materialized views". In: *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems - PODS '98*. ACM. ACM Press, 1998, pp. 254–263.
DOI: 10.1145/275487.275516.

[202] R. van der Meyden. "The complexity of querying indefinite information: defined relations, recursion and linear order". PhD thesis. Rutgers University, Department of Computer Science, Laboratory for Computer Science Research, 1992.

[203] U. Dayal, N. Goodman, and R. H. Katz.
"An extended relational algebra with control over duplicate elimination". In: *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*.
ACM. ACM Press, 1982, pp. 117–123. DOI: 10.1145/588111.588132.

[204] A. Klausner and N. Goodman. "Multirelations-Semantice and Languages." In: *VLDB*. Vol. 85. 1985, pp. 251–258.

[205] X. L. Dong and F. Naumann. "Data fusion: resolving data conflicts for integration".
In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1654–1655.
DOI: 10.14778/1687553.1687620.

[206] M. Jarke and J. Koch. "Query Optimization in Database Systems".
In: *ACM Computing Surveys* 16.2 (June 1984), pp. 111–152. DOI: 10.1145/356924.356928.

[207] M. Friendly. "A Brief History of Data Visualization".
In: *Handbook of data visualization* (2008), pp. 15–56. DOI: 10.1007/978-3-540-33037-0_2.

[208] M. Harrower and C. A. Brewer.
"ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps".
In: *The Cartographic Journal* 40.1 (June 2003), pp. 27–37.
DOI: 10.1179/000870403235002042.

[209] A. Dasgupta, J. Poco, B. Rogowitz, K. Han, E. Bertini, and C. T. Silva.
"The Effect of Color Scales on Climate Scientists' Objective and Subjective Performance in Spatial Data Analysis Tasks". In: *IEEE Transactions on Visualization and Computer Graphics* 26.3 (Mar. 2020), pp. 1577–1591. DOI: 10.1109/tvcg.2018.2876539.

[210] K. R. Gegenfurtner and L. T. Sharpe. *Color vision: From genes to perception*.
Cambridge University Press, 2001.

[211] I. Abramov, J. Gordon, O. Feldman, and A. Chavarga.
"Sex & vision I: Spatio-temporal resolution".
In: *Biology of sex differences* 3.1 (Sept. 2012), p. 20. DOI: 10.1186/2042-6410-3-20.

[212] I. Abramov, J. Gordon, O. Feldman, and A. Chavarga.
"Sex & vision II: Color appearance of monochromatic lights".
In: *Biology of Sex Differences* 3.1 (2012), p. 21. DOI: 10.1186/2042-6410-3-21.

[213] A. McNutt and G. Kindlmann.
"Linting for Visualization: Towards a Practical Automated Visualization Guidance System". In: Oct. 2018.

[214] A. Diehl, A. Abdul-Rahman, M. El-Assady, B. Bach, D. Keim, and M. Chen.
"VisGuides: A Forum for Discussing Visualization Guidelines". In:
The Eurographics Association, 2018. DOI: 10.2312/EUROVISSHORT.20181079.

[215] J. Mackinlay. "Automating the design of graphical presentations of relational information".
In: *ACM Transactions on Graphics* 5.2 (Apr. 1986), pp. 110–141. DOI: 10.1145/22949.22950.

[216] W. S. Cleveland and R. McGill. "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods".
In: *Journal of the American Statistical Association* 79.387 (Sept. 1984), pp. 531–554.
DOI: 10.1080/01621459.1984.10478080.

[217] Y. Sun, J. Leigh, A. E. Johnson, and S. Lee. "Articulate: A Semi-automated Model for Translating Natural Language Queries into Meaningful Visualizations." In: *Smart Graphics*.
Vol. 6133. Springer. Springer Berlin Heidelberg, 2010, pp. 184–195.
DOI: 10.1007/978-3-642-13544-6_18.

[218] Y. Sun. "Articulate: Creating Meaningful Visualizations from Natural Language".
PhD thesis. University of Illinois at Chicago, 2012.
DOI: 10.4018/978-1-4666-4309-3.ch011.

[219] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger. "The Penn Treebank: annotating predicate argument structure".
In: *Proceedings of the workshop on Human Language Technology*.
Association for Computational Linguistics. 1994, pp. 114–119.
DOI: 10.3115/1075812.1075835.

[220] M. Voigt, S. Pietschmann, and K. Meißner. "A semantics-based, end-user-centered information visualization process for semantic web data".
In: *Semantic models for adaptive interactive systems*. Springer, 2013, pp. 83–107.
DOI: 10.1007/978-1-4471-5301-6_5.

[221] J. Polowinski and M. Voigt. “VISO: A Shared, Formal Knowledge Base As a Foundation for Semi-automatic Infovis Systems”.
In: *CHI ’13 Extended Abstracts on Human Factors in Computing Systems*. CHI EA ’13. Paris, France: ACM, 2013, pp. 1791–1796. DOI: 10.1145/2468356.2468677.

[222] D. Lin. “An information-theoretic definition of similarity.” In: *Icml*. Vol. 98. 1998. 1998, pp. 296–304.

[223] O. Gilson, N. Silva, P. W. Grant, and M. Chen.
“From Web Data to Visualization via Ontology Mapping”. In: *Computer Graphics Forum*. Vol. 27. 3. Wiley Online Library. Wiley, May 2008, pp. 959–966.
DOI: 10.1111/j.1467-8659.2008.01230.x.

[224] B. Mutlu, P. Hoefler, G. Tschinkel, E. Veas, V. Sabol, F. Stegmaier, and M. Granitzer.
“Suggesting visualisations for published data”. In: *Information Visualization Theory and Applications (IVAPP), 2014 International Conference on*. IEEE. 2014, pp. 267–275.

[225] K. Pearson. “Note on regression and inheritance in the case of two parents”.
In: *Proceedings of the Royal Society of London* 58 (1895), pp. 240–242.

[226] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock.
“Methods and metrics for cold-start recommendations”.
In: *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR ’02*. ACM. ACM Press, 2002, pp. 253–260.
DOI: 10.1145/564376.564421.

[227] M. Srinivas and L. M. Patnaik. “Genetic algorithms: A survey”.
In: *Computer* 27.6 (June 1994), pp. 17–26. DOI: 10.1109/2.294849.

[228] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. “Formalizing Visualization Design Knowledge as Constraints: Actionable and Extensible Models in Draco”.
In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), pp. 438–448.
DOI: 10.1109/tvcg.2018.2865240.

[229] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer.
“Vega-Lite: A Grammar of Interactive Graphics”.
In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017), pp. 341–350.
DOI: 10.1109/tvcg.2016.2599030.

[230] T.-Y. Liu. “Learning to Rank for Information Retrieval”.
In: *Foundations and Trends® in Information Retrieval* 3.3 (2007), pp. 225–331.
DOI: 10.1561/1500000016.

[231] R. Herbrich. “Support vector learning for ordinal regression”.
In: *9th International Conference on Artificial Neural Networks: ICANN ’99*. IEE, 1999.
DOI: 10.1049/cp:19991091.

[232] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, and M. Schneider.
“Potassco: The Potsdam Answer Set Solving Collection”.
In: *AI Communications* 24 (2011), pp. 107–124. DOI: 10.3233/AIC-2011-0491.

[233] J. Nielsen and H. Loranger. *Prioritizing web usability*. Pearson Education, 2006.

[234] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[235] L. Ramshaw and R. E. Tarjan.
"On minimum-cost assignments in unbalanced bipartite graphs".
In: *HP Labs, Palo Alto, CA, USA, Tech. Rep. HPL-2012-40R1* (2012).

[236] H. W. Kuhn. "The Hungarian method for the assignment problem".
In: *Naval Research Logistics (NRL)* 2.1-2 (1955), pp. 83–97.

[237] J. Edmonds and R. M. Karp.
"Theoretical improvements in algorithmic efficiency for network flow problems".
In: *Journal of the ACM (JACM)* 19.2 (1972), pp. 248–264.

[238] R. Agrawal, T. Imieliński, and A. Swami.
"Mining association rules between sets of items in large databases". In: *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*. Vol. 22. 2. ACM. ACM Press, 1993, pp. 207–216. DOI: 10.1145/170035.170072.

[239] T. Hastie, R. Tibshirani, and J. Friedman.
The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2002.

[240] F. Thabtah. "A review of associative classification mining".
In: *The Knowledge Engineering Review* 22.1 (Mar. 2007), pp. 37–65.
DOI: 10.1017/s0269888907001026.

[241] B. Liu, Y. Ma, C. K. Wong, and S. Y. Philip. "Scoring the data using association rules".
In: *Applied intelligence* 18.2 (2003), pp. 119–135.

[242] R. Agrawal, R. Srikant, et al. "Fast algorithms for mining association rules".
In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994, pp. 487–499.

[243] V. Stodden, D. H. Bailey, J. Borwein, R. J. LeVeque, W. Rider, and W. Stein. "Setting the default to reproducible: Reproducibility in computational and experimental mathematics".
In: *Institute for Computational and Experimental Research in Mathematics* (2013).

[244] Y. L. Simmhan, B. Plale, and D. Gannon. "A survey of data provenance in e-science".
In: *ACM SIGMOD Record* 34.3 (Sept. 2005), pp. 31–36. DOI: 10.1145/1084805.1084812.

[245] N. Prat and S. Madnick. "Measuring Data Believability: A Provenance Approach".
In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. IEEE, Jan. 2008. DOI: 10.1109/hicss.2008.243.

[246] M. A. C. Johnson, M. Paradies, M. Dembska, K. Lackeos, H.-R. Klöckner, D. J. Champion, and S. Schindler. "Astronomical Pipeline Provenance: A Use Case Evaluation".
In: *13th International Workshop on Theory and Practice of Provenance (TaPP 2021)*.
USENIX Association, July 21, 2021.

[247] W. Han and H.-J. Schulz. "Beyond Trust Building — Calibrating Trust in Visual Analytics".
In: *2020 IEEE Workshop on TRust and EXpertise in Visual Analytics (TREX)*. IEEE, Oct. 2020.
DOI: 10.1109/trex51495.2020.00006.

[248] A. Gupta. "Data Provenance". In: *Encyclopedia of Database Systems*. Springer US, 2009, pp. 608–608. DOI: 10.1007/978-0-387-39940-9_1305.

[249] W.-C. Tan. “Provenance”. In: *Encyclopedia of Database Systems*. Springer US, 2009, pp. 2202–2202. DOI: 10.1007/978-0-387-39940-9_283.

[250] Y. Zhao, M. Wilde, and I. Foster. “Applying the Virtual Data Provenance Model”. In: *International Provenance and Annotation Workshop*. Springer. Springer Berlin Heidelberg, 2006, pp. 148–161. DOI: 10.1007/11890850_16.

[251] J. Freire, D. Koop, E. Santos, and C. T. Silva. “Provenance for Computational Tasks: A Survey”. In: *Computing in Science & Engineering* 10.3 (May 2008), pp. 11–21. DOI: 10.1109/mcse.2008.79.

[252] C. Lim, S. Lu, A. Chebotko, and F. Fotouhi. “Prospective and retrospective provenance collection in scientific workflow environments”. In: *Services Computing (SCC), 2010 IEEE International Conference on*. IEEE. 2010, pp. 449–456. DOI: 10.1109/scc.2010.18.

[253] D. D. Chamberlin and R. F. Boyce. “SEQUEL: A structured English query language”. In: *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*. SIGFIDET ’74. Ann Arbor, Michigan: ACM, 1974, pp. 249–264. DOI: http://doi.acm.org/10.1145/800296.811515.

[254] P. Buneman, S. Khanna, and W. C. Tan. “Why and Where: A Characterization of Data Provenance”. In: *ICDT*. Ed. by J. V. den Bussche and V. Vianu. Vol. 1973. Lecture Notes in Computer Science. Springer, 2001, pp. 316–330.

[255] T. J. Green, G. Karvounarakis, and V. Tannen. “Provenance semirings”. In: *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems - PODS ’07*. PODS ’07. Beijing, China: ACM Press, 2007, pp. 31–40. DOI: 10.1145/1265530.1265535.

[256] J. Cheney, L. Chiticariu, and W.-C. Tan. “Provenance in Databases: Why, How, and Where”. In: *Found. Trends databases* 1 (4 Apr. 2009), pp. 379–474. DOI: 10.1561/1900000006.

[257] F. Bancilhon and N. Spyratos. “Update semantics of relational views”. In: *ACM Transactions on Database Systems* 6.4 (Dec. 1981), pp. 557–575. DOI: 10.1145/319628.319634.

[258] L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. “DBNotes: a post-it system for relational databases based on provenance”. In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. SIGMOD ’05. Baltimore, Maryland: ACM, 2005, pp. 942–944. DOI: http://doi.acm.org/10.1145/1066157.1066296.

[259] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. “An annotation management system for relational databases”. In: *The VLDB Journal* 14 (4 2005). 10.1007/s00778-005-0156-6, pp. 373–396.

[260] A. Woodruff and M. Stonebraker. “Supporting fine-grained data lineage in a database visualization environment”. In: *Data Engineering, 1997. Proceedings. 13th International Conference on*. Apr. 1997, pp. 91–102. DOI: 10.1109/ICDE.1997.581742.

[261] L. Murta, V. Braganholo, F. Chirigati, D. Koop, and J. Freire.
"noWorkflow: capturing and analyzing provenance of scripts".
In: *International Provenance and Annotation Workshop*. Springer. 2014, pp. 71–83.
DOI: 10.1007/978-3-319-16462-5_6.

[262] J. Van Zundert. "If you build it, will we come? Large scale digital infrastructures as a dead end for digital humanities".
In: *Historical Social Research/Historische Sozialforschung* (2012), pp. 165–186.

[263] C. Bochner, R. Gude, and A. Schreiber.
A Python Library for Provenance Recording and Querying, Provenance and Annotation of Data and Processes: Second International Provenance and Annotation Workshop, IPAW 2008, Salt Lake City, UT, USA, June 17-18, 2008. Revised Selected Papers. 2008.

[264] A. Davison. "Automated capture of experiment context for easier reproducibility in computational research". In: *Computing in Science & Engineering* 14.4 (2012), pp. 48–56.
DOI: 10.1109/mcse.2012.41.

[265] E. Angelino, D. Yamins, and M. Seltzer.
"StarFlow: A script-centric data analysis environment".
In: *Provenance and Annotation of Data and Processes* (2010), pp. 236–250.
DOI: 10.1007/978-3-642-17819-1_27.

[266] D. Tariq, M. Ali, and A. Gehani.
"Towards Automated Collection of Application-Level Data Provenance." In: *TaPP*. 2012.

[267] B. S. Lerner and E. R. Boose. "Collecting provenance in an interactive scripting environment".
In: *Workshop on the Theory and Practice of Provenance (TaPP), Cologne, Germany*. 2014.

[268] J. F. Pimentel, J. Freire, V. Braganholo, and L. Murta.
"Tracking and analyzing the evolution of provenance from scripts".
In: *International Provenance and Annotation Workshop*. Springer. 2016, pp. 16–28.
DOI: 10.1007/978-3-319-40593-3_2.

[269] T. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, K. Bocinsky, Y. Cao, F. Chirigati, S. Dey, J. Freire, D. Huntzinger, C. Jones, D. Koop, P. Missier, M. Schildhauer, C. Schwalm, Y. Wei, J. Cheney, M. Bieda, and B. Ludäscher. "YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts".
In: *International Journal of Digital Curation* 10.1 (May 2015), pp. 298–313.
DOI: 10.2218/ijdc.v10i1.370.

[270] J. F. Pimentel, S. Dey, T. McPhillips, K. Belhajjame, D. Koop, L. Murta, V. Braganholo, and B. Ludäscher.
"Yin & Yang: demonstrating complementary provenance from noWorkflow & YesWorkflow".
In: *International Provenance and Annotation Workshop*. Springer. 2016, pp. 161–165.
DOI: 10.1007/978-3-319-40593-3_13.

[271] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire.
"noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts".
In: *Proceedings of the VLDB Endowment* 10.12 (2017), pp. 1841–1844.
DOI: 10.14778/3137765.3137789.

[272] "FIPS 180-1: Secure Hash Standard".
In: *NIST, US Dept. of Commerce, Washington DC April* 9.21 (1995), p. 32.

[273] P. Missier, K. Belhajjame, and J. Cheney.
"The W3C PROV family of specifications for modelling provenance metadata".
In: *Proceedings of the 16th International Conference on Extending Database Technology*.
ACM. 2013, pp. 773–776. DOI: 10.1145/2452376.2452478.

[274] L. Moreau, P. Groth, J. Cheney, T. Lebo, and S. Miles. "The rationale of PROV".
In: *Journal of Web Semantics* 35 (Dec. 2015), pp. 235–257.
DOI: 10.1016/j.websem.2015.04.001.

[275] L. Moreau, B. Ludäscher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. Chin, B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble, J. Golbeck, P. Groth, D. A. Holland, S. Jiang, J. Kim, D. Koop, A. Krenek, T. McPhillips, G. Mehta, S. Miles, D. Metzger, S. Munroe, J. Myers, B. Plale, N. Podhorszki, V. Ratnakar, E. Santos, C. Scheidegger, K. Schuchardt, M. Seltzer, Y. L. Simmhan, C. Silva, P. Slaughter, E. Stephan, R. Stevens, D. Turi, H. Vo, M. Wilde, J. Zhao, and Y. Zhao. "Special Issue: The First Provenance Challenge".
In: *Concurrency and Computation: Practice and Experience* 20.5 (2008), pp. 409–418.
DOI: 10.1002/cpe.1233.

[276] Y. Simmhan, P. Groth, and L. Moreau. "Special section: The third provenance challenge on using the open provenance model for interoperability".
In: *Future Generation Computer Systems* 27.6 (2011), pp. 737–742.
DOI: 10.1016/j.future.2010.11.020.

[277] Y. Cao, C. Jones, V. Cuevas-Vicenttin, M. B. Jones, B. Ludäscher, T. McPhillips, P. Missier, C. Schwalm, P. Slaughter, D. Vieglais, L. Walker, and Y. Wei.
"ProvONE: extending PROV to support the DataONE scientific community".
In: *PROV: Three Years Later*. June 6, 2016.

[278] D. J. Abadi, S. R. Madden, and N. Hachem.
"Column-stores vs. row-stores: How different are they really?" In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*.
ACM. ACM Press, 2008, pp. 967–980. DOI: 10.1145/1376616.1376712.

[279] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters".
In: *OSDI* (2004), p. 13.

[280] B. Ford. "Parsing expression grammars: a recognition-based syntactic foundation".
In: *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '04*. Vol. 39. 1. ACM. ACM Press, 2004, pp. 111–122.
DOI: 10.1145/964001.964011.

[281] B. Ford. "Packrat parsing: simple, powerful, lazy, linear time, functional pearl".
In: *ACM SIGPLAN Notices*. Vol. 37. 9. ACM. ACM Press, 2002, pp. 36–47.
DOI: 10.1145/581478.581483.

[282] P. Mishra and M. H. Eich. "Join processing in relational databases".
In: *ACM Computing Surveys* 24.1 (Mar. 1992), pp. 63–113. DOI: 10.1145/128762.128764.

[283] M. D. Steinberg. “Bewertung von Ontologien zum Thema Maßeinheiten”.
Bachelor’s thesis. Friedrich Schiller University Jena, Aug. 19, 2016.

[284] H. Gibson, J. Faith, and P. Vickers.
“A survey of two-dimensional graph layout techniques for information visualisation”.
In: *Information visualization* 12.3-4 (2013), pp. 324–357.

[285] “The JavaScript Object Notation (JSON) Data Interchange Format”.
In: (Dec. 2017). Ed. by T. Brady. DOI: 10.17487/RFC8259.

[286] “Agreement on the withdrawal of the United Kingdom of Great Britain and Northern Ireland from the European Union and the European Atomic Energy Community”.
In: *Office Journal of the European Union* 63 (Jan. 31, 2020).

[287] M. Allen, ed. *The SAGE Encyclopedia of Communication Research Methods*. Mar. 7, 2017.
DOI: 10.4135/9781483381411.n103.

[288] J. Brooke. “SUS: a "quick and dirty" usability scale”. In: *Usability Evaluation in Industry*.
Ed. by P. W. Jordan, B. Thomas, B. Weerdmeester, and I. L. McClelland.
London: Taylor and Francis, 1996.

[289] J. Brooke. “SUS: A Retrospective”. In: *J. Usability Studies* 8.2 (Feb. 2013), pp. 29–40.

[290] R. Likert. “A technique for the measurement of attitudes.” In: *Archives of psychology* (1932).

[291] A. Bangor, P. T. Kortum, and J. T. Miller.
“An Empirical Evaluation of the System Usability Scale”.
In: *International Journal of Human-Computer Interaction* 24.6 (July 2008), pp. 574–594.
DOI: 10.1080/10447310802205776.

[292] A. Bangor, P. Kortum, and J. Miller.
“Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale”.
In: *J. Usability Studies* 4.3 (May 2009), pp. 114–123.

[293] J. Dabrowski and E. V. Munson.
“40years of searching for the best computer system response time”.
In: *Interacting with Computers* 23.5 (Sept. 2011), pp. 555–564.
DOI: 10.1016/j.intcom.2011.05.008.

[294] Z. Stachoň, Č. Šašinka, J. Čeněk, Z. Štěrba, S. Angsuesser, S. I. Fabrikant, R. Štampach, and K. Morong. “Cross-cultural differences in figure–ground perception of cartographic stimuli”.
In: *Cartography and Geographic Information Science* 46.1 (May 2018), pp. 82–94.
DOI: 10.1080/15230406.2018.1470575.

[295] E. Dimara, S. Franconeri, C. Plaisant, A. Bezerianos, and P. Dragicevic.
“A Task-Based Taxonomy of Cognitive Biases for Information Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.2 (Feb. 2020), pp. 1413–1432.
DOI: 10.1109/tvcg.2018.2872577.

[296] N. Abdelmageed and S. Schindler. “JenTab: Matching Tabular Data to Knowledge Graphs”. In: *The Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab 2020) co-located with the 19th International Semantic Web Conference (ISWC 2020)*. Ed. by E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, and V. Cutrona. Vol. 2775. CEUR Workshop Proceedings. CEUR-WS.org, 2020, pp. 40–49.

[297] N. Abdelmageed and S. Schindler. “JenTab: A Toolkit for Semantic Table Annotations”. In: *Second International Workshop on Knowledge Graph Construction (KGCW 2021) co-located with the Extended Semantic Web Conference (ESWC 2021)*. June 6, 2021.

[298] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. Parameswaran. “Towards Visualization Recommendation Systems”. In: *ACM SIGMOD Record* 45.4 (May 2017), pp. 34–39. DOI: 10.1145/3092931.3092937.

[299] M. Hausenblas, E. Wilde, and J. Tennison. “URI Fragment Identifiers for the text/csv Media Type”. In: (Jan. 2014). DOI: 10.17487/RFC7111.

[300] B. N. Taylor and A. Thompson. *The International System of Units (SI). National Institute of Standards and Technology Special Publication 330 (2008 Edition)*. National Institute of Standards and Technology / U.S. Department of Commerce, 2008.

Web Resources

Snapshots of all web resources have been deposited with the Internet Archive's Wayback Machine (`https://web.archive.org/`) at the time of last access.

[web1] *Eurostat*. European Comission.
URL: `https://ec.europa.eu/eurostat` Last Access: Aug. 7, 2021.

[web2] *Uber Movement: Let's find smarter ways forward, together.* Uber Technologies, Inc.
URL: `https://movement.uber.com` Last Access: Aug. 7, 2021.

[web3] *Willkommen - Open-Data-Portal – Deutsche Bahn Datenportal*. Deutsche Bahn AG.
URL: `https://data.deutschebahn.com/` Last Access: Aug. 7, 2021.

[web4] *GFBio website - Welcome*. GFBio e.V.
URL: `https://www.gfbio.org/` Last Access: Aug. 7, 2021.

[web5] *Find open data - data.gov.uk*. URL: `https://data.gov.uk/` Last Access: Aug. 7, 2021.

[web6] *Bing*. URL: `https://www.bing.com/` Last Access: Aug. 7, 2021.

[web7] 百度一下，你就知道. URL: `https://www.baidu.com/` Last Access: Aug. 7, 2021.

[web8] *Google*. URL: `https://www.google.com/` Last Access: Aug. 7, 2021.

[web9] R. Cyganiak and D. Reynoolds, eds. *The RDF Data Cube Vocabulary*. W3C. Jan. 16, 2014.
URL: `https://www.w3.org/TR/vocab-data-cube/` Last Access: Aug. 7, 2021.

[web10] R. Cyganiak, D. Wood, and M. Lanthaler, eds. *RDF 1.1 Concepts and Abstract Syntax*. W3C. Feb. 25, 2014. URL: `https://www.w3.org/TR/rdf11-concepts/` Last Access: Aug. 7, 2021.

[web11] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau.
Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C. Nov. 26, 2008.
URL: `https://www.w3.org/TR/xml/` Last Access: Aug. 7, 2021.

[web12] *SDMX - Statistical Data and Metadata eXchange*.
URL: `https://sdmx.org/` Last Access: Aug. 7, 2021.

[web13] *GBIF REST API*. Global Biodiversity Information Facility.
URL: `https://www.gbif.org/developer/summary` Last Access: Aug. 7, 2021.

[web14] *Wikidata Query Service*. Wikimedia Foundation.
URL: `https://query.wikidata.org/` Last Access: Aug. 7, 2021.

[web15] *FIPS Codes Replacement Chart 2015*.
URL: `https://www.nist.gov/document/fipscodesreplacementchart2015pdf` Last Access: Aug. 7, 2021.

[web16] S. Ribecca. *The Data Visualisation Catalogue*.
URL: `https://datavizcatalogue.com/` Last Access: Aug. 7, 2021.

[web17] *Calc | LibreOffice - Free Office Suite - Fun Project - Fantastic People*.
The Document Foundation.
URL: `https://www.libreoffice.org/discover/calc/` Last Access: Aug. 7, 2021.

[web18] *Choosing a Chart Type - LibreOffice Help*. The Document Foundation. URL: https://help.libreoffice.org/Chart/Choosing_a_Chart_Type Last Access: Aug. 7, 2021.

[web19] M. Bostock. *D3.js - Data-Driven Documents*. URL: https://d3js.org/ Last Access: Aug. 7, 2021.

[web20] *gnuplot homepage*. June 2021. URL: http://www.gnuplot.info Last Access: July. 21, 2021.

[web21] *WTF Visualizations*. URL: https://viz.wtf/ Last Access: Aug. 7, 2021.

[web22] P. Groth and L. Moreau, eds. *PROV-Overview*. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-overview/ Last Access: Aug. 7, 2021.

[web23] *Artifact Review and Badging Version 1.1*. ACM. Aug. 24, 2020. URL: https://www.acm.org/publications/policies/artifact-review-and-badging-current Last Access: Aug. 7, 2021.

[web24] *Introduction - Google Fusion Tables*. Google. URL: https://sites.google.com/site/fusiontablestalks/home Last Access: Feb. 3, 2019.

[web25] *Business Intelligence and Analytics*. Tableau Software, LLC, a Salesforce Company. URL: https://www.tableau.com Last Access: Aug. 7, 2021.

[web26] *Project Jupyter | Home*. Project Jupyter. URL: https://jupyter.org/ Last Access: Aug. 7, 2021.

[web27] *Taverna - Apache Incubator*. The Apache Software Foundation. URL: https://taverna.incubator.apache.org/ Last Access: Aug. 7, 2021.

[web28] *VisTrails*. Sept. 19, 2019. URL: http://www.vistrails.org Last Access: Aug. 7, 2021.

[web29] *Copyright notice and free re-use of data - Eurostat*. European Comission. URL: https://ec.europa.eu/eurostat/about/policies/copyright Last Access: Aug. 7, 2021.

[web30] *Bulk download - Eurostat*. European Comission. URL: https://ec.europa.eu/eurostat/data/bulkdownload Last Access: Aug. 7, 2021.

[web31] *Microsoft Excel Spreadsheet Software | Microsoft 365*. Microsoft. URL: https://www.microsoft.com/en/microsoft-365/excel Last Access: Aug. 7, 2021.

[web32] *IANA - Media Types - tab-separated-values*. IANA. URL: https://www.iana.org/assignments/media-types/text/tab-separated-values Last Access: Aug. 7, 2021.

[web33] *Google Sheets - create and edit spreadsheets online, for free.* Google. URL: https://www.google.com/sheets/about/ Last Access: Aug. 7, 2021.

[web34] *ECMA-376-1:2016: Office Open XML File Formats*. Ecma. Dec. 2016. URL: https://www.ecma-international.org/publications-and-standards/standards/ecma-376/ Last Access: Aug. 7, 2021.

[web35] P. Durusau and M. Brauer, eds.
Open Document Format for Office Applications (OpenDocument) Version 1.2. OASIS.
Sept. 29, 2011.
URL: http://docs.oasis-open.org/office/v1.2/os/OpenDocument-v1.2-os.html Last Access: Aug. 7, 2021.

[web36] *FAQ: Google Fusion Tables - Fusion Tables Help*. Google.
URL: https://support.google.com/fusiontables/answer/9551050?visit_id=637211893913799211-597987236&rd=1 Last Access: Aug. 7, 2021.

[web37] *G Suite Updates Blog: Google Fusion Tables to be shut down on December 3, 2019*. Google.
URL: https://gsuiteupdates.googleblog.com/2018/12/google-fusion-tables-to-be-shut-down-on.html Last Access: Aug. 7, 2021.

[web38] *Fusion Tables REST API | Google Developers*. Google.
URL: https://developers.google.com/fusiontables/ Last Access: Aug. 10, 2019.

[web39] *Welcome to Python.org*. Python Software Foundation.
URL: https://www.python.org/ Last Access: Aug. 7, 2021.

[web40] *The Jupyter Notebook — Jupyter Notebook 6.4.0 documentation*. Project Jupyter.
URL: https://jupyter-notebook.readthedocs.io/en/stable/ Last Access: Aug. 7, 2021.

[web41] J. Gruber. *Daring Fireball: Markdown*. The Daring Fireball Company LLC. Dec. 7, 2004.
URL: https://daringfireball.net/projects/markdown/ Last Access: Aug. 7, 2021.

[web42] B. Ragan-Kelley, A. J. Winstein, Y. Panda, and J. Kanner. *minrk/ligo-binder: Black holes!*
URL: https://github.com/minrk/ligo-binder Last Access: Aug. 7, 2021.

[web43] *JupyterLite*. URL: https://github.com/jupyterlite/ Last Access: Aug. 7, 2021.

[web44] J. Tuloup. *JupyterLite: Jupyter ♡ WebAssembly ♡ Python*. July 13, 2021.
URL: https://blog.jupyter.org/jupyterlite-jupyter-%5C%EF%5C%B8%5C%8F-webassembly-%5C%EF%5C%B8%5C%8F-python-f6e2e41ab3fa Last Access: Aug. 7, 2021.

[web45] A. Rossberg, ed. *WebAssembly Core Specification*. W3C. Dec. 5, 2019.
URL: https://www.w3.org/TR/wasm-core/ Last Access: Aug. 7, 2021.

[web46] *SciPy.org — SciPy.org*. URL: https://www.scipy.org/ Last Access: Aug. 7, 2021.

[web47] *NumPy*. URL: https://numpy.org/ Last Access: Aug. 7, 2021.

[web48] *pandas - Python Data Analysis Library*.
URL: https://pandas.pydata.org/ Last Access: Aug. 7, 2021.

[web49] J. Hunter, D. Darren, E. Firing, M. Droettboom, and the Matplotlib development team.
Matplotlib: Python plotting — Matplotlib 3.4.2 documentation.
URL: https://matplotlib.org/ Last Access: Aug. 7, 2021.

[web50] Jupyter Development Team. *The Jupyter Notebook Format — nbformat 5.1 documentation*.
Project Jupyter.
URL: https://nbformat.readthedocs.io/en/latest/ Last Access: Aug. 7, 2021.

[web51] *GitHub: Where the world builds software · GitHub*. GitHub, Inc.
URL: https://github.com/ Last Access: Aug. 7, 2021.

[web52] *myGrid*. URL: http://www.mygrid.org.uk/ Last Access: Nov. 8, 2017.

[web53] *The Apache Software Foundation*.
URL: https://www.apache.org/ Last Access: Aug. 7, 2021.

[web54] *Taverna - Apache Incubator*.
URL: http://incubator.apache.org/projects/taverna.html Last Access: Aug. 7, 2021.

[web55] *Release taverna-commandline 3.1.0-incubating · apache/incubator-taverna-commandline*.
June 30, 2016. URL: https://github.com/apache/incubator-taverna-commandline/releases/tag/3.1.0-incubating Last Access: Aug. 7, 2021.

[web56] *Release Apache Taverna Server 3.1.0-incubating · apache/incubator-taverna-server*.
Jan. 18, 2018. URL: https://github.com/apache/incubator-taverna-server/releases/tag/3.1.0-incubating Last Access: Aug. 7, 2021.

[web57] *Release apache-taverna-language-0.15.1-incubating · apache/incubator-taverna-language*.
Mar. 11, 2016. URL: https://github.com/apache/incubator-taverna-language/releases/tag/0.15.1-incubating Last Access: Aug. 7, 2021.

[web58] *BioCatalogue: The Life Science Web Service Registry*.
URL: http://www.biocatalogue.org/ Last Access: Apr. 26, 2021.

[web59] A. Williams, ed. *User Manual - Taverna 2.5 - myGrid developer wiki*. May 7, 2014.
URL: http://dev.mygrid.org.uk/wiki/display/tav250/User+Manual Last Access: Aug. 7, 2021.

[web60] K. Wolstencroft, ed. *myExperiment - Workflows - NCBI Gi to Kegg Pathway Descriptions (Katy Wolstencroft) [Taverna 2 Workflow]*. Jan. 30, 2013.
URL: http://www.myexperiment.org/workflows/2659.html Last Access: Aug. 7, 2021.

[web61] *National Center for Biotechnology Information (NCBI)*. Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information. 1988.
URL: https://www.ncbi.nlm.nih.gov/ Last Access: Aug. 7, 2021.

[web62] *Apache Taverna - SCUFL2 Taverna Language*. Apache Taverna.
URL: https://taverna.incubator.apache.org/documentation/scufl2/ Last Access: May. 15, 2020.

[web63] D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. *RDF 1.1 Turtle*.
Ed. by E. Prud'hommeaux and G. Carothers. W3C. Feb. 25, 2014.
URL: https://www.w3.org/TR/turtle/ Last Access: Aug. 7, 2021.

[web64] F. Gandon and G. Schreiber, eds. *RDF 1.1 XML Syntax*. W3C. Feb. 25, 2014.
URL: https://www.w3.org/TR/rdf-syntax-grammar/ Last Access: Aug. 7, 2021.

[web65] S. Soiland-Reyes, ed. *Taverna, SCUFL2 and wfdesc*. Oct. 21, 2014.
URL: https://groups.google.com/g/common-workflow-language/c/eGrNfpjuq2E/m/mKj2pawNwakJ Last Access: Aug. 7, 2021.

[web66] *VTK - The Visualization Toolkit*. Kitware.
URL: http://www.vtk.org Last Access: Aug. 7, 2021.

[web67] *Facebook*. URL: https://www.facebook.com/ Last Access: Aug. 7, 2021.

[web68] *Google Workspace | Business Apps Collaboration Tools*. Google.
URL: https://workspace.google.com/ Last Access: Aug. 7, 2021.

[web69] *Wikidata*. Wikimedia Foundation.
URL: https://www.wikidata.org Last Access: Aug. 7, 2021.

[web70] *Microsoft Excel Data Types | Microsoft Docs*. Microsoft. Jan. 19, 2017.
URL: https://docs.microsoft.com/en-us/sql/odbc/microsoft/microsoft-excel-data-types Last Access: Aug. 7, 2021.

[web71] *Data Types Used by Excel*. July 1, 2011. URL:
https://msdn.microsoft.com/en-us/library/bb687869.aspx Last Access: Aug. 7, 2021.

[web72] *Information Functions - LibreOffice Help*. The Document Foundation.
URL: https://help.libreoffice.org/Calc/Information_Functions#TYPE Last Access: Aug. 7, 2021.

[web73] *TYPE - Docs editors Help*. Google.
URL: https://support.google.com/docs/answer/3267375 Last Access: Aug. 7, 2021.

[web74] *Data Types in Analysis Services | Microsoft Docs*. Microsoft.
URL: https://docs.microsoft.com/en-us/analysis-services/multidimensional-models/olap-physical/data-types-in-analysis-services Last Access: Aug. 7, 2021.

[web75] *OLAP DML Data Types*. Oracle. URL: https://docs.oracle.com/cd/B28359_01/olap.111/b28126/dml_expression001.htm#OLADM138 Last Access: Aug. 7, 2021.

[web76] *Data types of columns - IBM Documentation*. IBM.
URL: https://www.ibm.com/docs/en/db2-for-zos/12?topic=columns-data-types Last Access: Aug. 7, 2021.

[web77] *Data Types - MariaDB Knowledge Base*. MariaDB Foundation.
URL: https://mariadb.com/kb/en/mariadb/data-types/ Last Access: Aug. 7, 2021.

[web78] J. Tennison. *CSV on the Web: A Primer*. W3C. Feb. 25, 2016.
URL: https://www.w3.org/TR/tabular-data-primer/ Last Access: Aug. 7, 2021.

[web79] P. Salas. *A bit of personal PR (at least I labeled it so you can skip it)*. Aug. 8, 2005.
URL: https://salas.com/2005/08/08/200588a-bit-of-personal-pr-at-least-i-labeled-it-so-you-can-skip-i-html/ Last Access: Aug. 7, 2021.

[web80] *Cubes | Data Brewery*. Data Brewery.
URL: http://cubes.databrewery.org/ Last Access: Aug. 7, 2021.

[web81] *Oracle OLAP*. Oracle.
URL: http://www.oracle.com/technetwork/database/options/olap/index.html Last Access: Aug. 7, 2021.

[web82] *MariaDB.org - Ensuring continuity and open collaboration*. MariaDB Foundation.
URL: https://mariadb.org/ Last Access: Aug. 7, 2021.

[web83] *IBM Db2 – Data Management Software | IBM*. IBM.
URL: https://www.ibm.com/analytics/db2 Last Access: Aug. 7, 2021.

[web84] *VizAssist - Your visualization Web assistant*. Mar. 24, 2016.
URL: http://www.vizassist.fr/ Last Access: Aug. 7, 2021.

[web85] *Kaiser Family Foundation - Health Policy Research, Analysis, Polling, Facts, Data and Journalism*. Kaiser Family Foundation.
URL: http://www.kff.org/ Last Access: Aug. 7, 2021.

[web86] W3C OWL Working Group, ed.
OWL 2 Web Ontology Language Document Overview (Second Edition). W3C. Dec. 11, 2012.
URL: https://www.w3.org/TR/owl2-overview/ Last Access: Aug. 7, 2021.

[web87] A. Greiner, A. Isaac, C. Iglesias, C. Laufer, C. Guéret, D. Lee, E. G. Stephan, E. Kauz, G. A. Atemezing, H. Beeman, I. I. Bittencourt, J. P. Almeida, M. Dekkers, P. Winstanley, P. Archer, R. Albertoni, S. Purohit, and Y. Córdova. *Data on the Web Best Practices*. Ed. by B. F. Lóscio, C. Burle, and N. Calegari. W3C. Dec. 15, 2016.
URL: https://www.w3.org/TR/dwbp/ Last Access: Aug. 7, 2021.

[web88] C. B. Aranda, O. Corby, S. Das, L. Feigenbaum, P. Gearon, B. Glimm, S. Harris, S. Hawke, I. Herman, N. Humfrey, N. Michaelis, C. Ogbuji, M. Perry, A. Passant, A. Polleres, E. Prud'hommeaux, A. Seaborne, and G. T. Williams, eds. *SPARQL 1.1 Overview*. W3C. Mar. 21, 2013.
URL: https://www.w3.org/TR/sparql11-overview/ Last Access: Aug. 7, 2021.

[web89] *DCMI Home: Dublin Core® Metadata Initiative (DCMI)*. The Dublin Core Metadata Initiative.
URL: http://dublincore.org/ Last Access: Aug. 7, 2021.

[web90] *Dublin Core Metadata Element Set, Version 1.1*. The Dublin Core Metadata Initiative. June 14, 2012.
URL: http://www.dublincore.org/documents/dces/ Last Access: Aug. 7, 2021.

[web91] *DCMI Metadata Terms*. June 14, 2012.
URL: http://dublincore.org/documents/dcmi-terms/ Last Access: Aug. 7, 2021.

[web92] F. Maali and J. Erickson, eds. *Data Catalog Vocabulary (DCAT)*. W3C. Jan. 16, 2014.
URL: https://www.w3.org/TR/vocab-dcat/ Last Access: May. 10, 2018.

[web93] R. Albertoni, D. Browning, S. Cox, A. Gonzalez Beltran, A. Perego, and P. Winstanley, eds. *Data Catalog Vocabulary (DCAT) - Version 2*. W3C. Feb. 2, 2020.
URL: https://www.w3.org/TR/vocab-dcat-2/ Last Access: Aug. 7, 2021.

[web94] K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao. *PROV-O: The PROV Ontology*. Ed. by T. Lebo, S. Sahoo, and D. McGuinness. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-o/ Last Access: Aug. 7, 2021.

[web95] K. Alexander, R. Cyganiak, M. Hausenblas, and J. Zhao.
Describing Linked Datasets with the VoID Vocabulary. W3C. Mar. 3, 2011.
URL: https://www.w3.org/TR/void/ Last Access: Aug. 7, 2021.

[web96] P. V. Biron and A. Malhotra, eds. *XML Schema Part 2: Datatypes Second Edition*. W3C. Oct. 28, 2004. URL: https://www.w3.org/TR/xmlschema-2/ Last Access: Aug. 7, 2021.

[web97] M. Jones, M. O'Brien, B. Mecum, C. Boettiger, M. Schildhauer, M. Maier, T. Whiteaker, S. Earl, and S. Chong. *Ecological Metadata Language version 2.2.0*. Feb. 22, 2021. DOI: 10.5063/f11834t2. URL: https://eml.ecoinformatics.org Last Access: Aug. 7, 2021.

[web98] *URL*. WHATWG. Aug. 5, 2021. URL: https://url.spec.whatwg.org/ Last Access: Aug. 7, 2021.

[web99] T. Berners-Lee. *Linked Data - Design Issues*. June 18, 2009. URL: https://www.w3.org/DesignIssues/LinkedData.html Last Access: Aug. 7, 2021.

[web100] B. Magagna, A. Devaraju, G. Moncoiffé, and M. Stoica. *InteroperAble Descriptions of Observable Property Terminology WG (I-ADOPT WG) | RDA*. Research Data Alliance. URL: https://www.rd-alliance.org/groups/interoperable-descriptions-observable-property-terminology-wg-i-adopt-wg Last Access: Aug. 7, 2021.

[web101] *RDA | Research Data Sharing without barriers*. Research Data Alliance. URL: https://www.rd-alliance.org Last Access: Aug. 7, 2021.

[web102] G. Schadow and C. J. McDonald. *The Unified Code for Units of Measure*. Nov. 21, 2017. URL: http://unitsofmeasure.org/ucum.html Last Access: Aug. 7, 2021.

[web103] *Wolfram Mathematica: Modern Technical Computing*. Wolfram. URL: http://www.wolfram.com/mathematica/ Last Access: Aug. 7, 2021.

[web104] *MATLAB - Mathworks*. URL: https://www.mathworks.com/products/matlab.html Last Access: Aug. 7, 2021.

[web105] *R: The R Project for Statistical Computing*. The R Foundation. URL: https://www.r-project.org/ Last Access: Aug. 7, 2021.

[web106] J. W. Eaton. *GNU Octave*. URL: https://www.gnu.org/software/octave/ Last Access: Aug. 7, 2021.

[web107] *IBM SPSS - IBM Analytics*. IBM. URL: https://www.ibm.com/analytics/us/en/technology/spss/ Last Access: Aug. 7, 2021.

[web108] *Data Analysis and Statistical Software | Stata*. StataCorp LLC. URL: http://www.stata.com/ Last Access: Aug. 7, 2021.

[web109] E. Pebesma, T. Mailund, T. Kalinowski, J. Hiebert, and I. Ucar. *CRAN - Package units*. URL: https://cran.r-project.org/package=units Last Access: Aug. 7, 2021.

[web110] M. A. Birk. *CRAN - Package measurements*. URL: https://cran.r-project.org/package=measurements Last Access: Aug. 7, 2021.

[web111] *Mannheim Search Joins Engine*. University of Mannheim. URL: http://searchjoins.webdatacommons.org/ Last Access: Aug. 7, 2021.

[web112] *Apache Lucene*. The Apache Software Foundation. URL: https://lucene.apache.org/ Last Access: Aug. 7, 2021.

[web113] T. Käfer and A. Harth. *Billion Triples Challenge 2014 Dataset*. 2014. URL: http://km.aifb.kit.edu/projects/btc-2014/ Last Access: Aug. 7, 2021.

[web114] B. Caldwell, M. Cooper, L. Guarino Reid, and G. Vanderheiden, eds. *Web Content Accessibility Guidelines (WCAG) 2.0*. W3C. Dec. 11, 2008. URL: http://www.w3.org/TR/WCAG20/ Last Access: Aug. 7, 2021.

[web115] *VIZBOARD - GAINING INSIGHTS FROM SEMANTIC DATA*. TU Dresden. URL: http://www.vizboard.de/ Last Access: Oct. 31, 2019.

[web116] *CODE Linked Data Vis Wizard*. Know-Center Graz. URL: https://code.know-center.tugraz.at/vis Last Access: May. 12, 2015.

[web117] D. Moritz, C. Wang, G. L. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. *Draco*. Interactive Data Lab, University of Washington. URL: https://uwdata.github.io/draco/ Last Access: Aug. 7, 2021.

[web118] *gems-uff/noworkflow: Supporting infrastructure to run scientific experiments without a scientific workflow management system.* URL: https://github.com/gems-uff/noworkflow Last Access: Aug. 7, 2021.

[web119] B. Ludaescher, T. Kolisnik, and T. McPhillips. *YesWorkflow*. URL: https://github.com/yesworkflow-org Last Access: Aug. 7, 2021.

[web120] *Javadoc Tool Home Page*. Oracle. URL: https://www.oracle.com/java/technologies/javase/javadoc.html Last Access: Aug. 7, 2021.

[web121] *Use JSDoc: Index*. URL: http://usejsdoc.org/ Last Access: Aug. 7, 2021.

[web122] *World Wide Web Consortium (W3C)*. W3C. URL: https://www.w3.org Last Access: Aug. 7, 2021.

[web123] K. Belhajjame, H. Deus, D. Garijo, G. Klyne, P. Missier, S. Soiland-Reyes, and S. Zednik. *PROV Model Primer*. Ed. by Y. Gil and S. Miles. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-primer/ Last Access: Aug. 7, 2021.

[web124] J. Cheney, ed. *Semantics of the PROV Data Model*. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-sem/ Last Access: Aug. 7, 2021.

[web125] T. De Nies. *Constraints of the PROV Data Model*. Ed. by J. Cheney, P. Missier, and L. Moreau. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-constraints/ Last Access: Aug. 7, 2021.

[web126] L. Moreau, O. Hartig, Y. Simmhan, J. Myers, T. Lebo, K. Belhajjame, S. Miles, and S. Soiland-Reyes. *PROV-AQ: Provenance Access and Query*. Ed. by G. Klyne and P. Groth. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-aq/ Last Access: Aug. 7, 2021.

[web127] S. Miles, C. M. Trim, and M. Panzer. *Dublin Core to PROV Mapping*. Ed. by D. Garijo and K. Eckert. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-dc/ Last Access: Aug. 7, 2021.

[web128] P. Missier, L. Moreau, J. Cheney, T. Lebo, and S. Soiland-Reyes. *PROV-Dictionary: Modeling Provenance for Dictionary Data Structures*. Ed. by T. De Nies and S. Coppens. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-dictionary/ Last Access: Aug. 7, 2021.

[web129] K. Belhajjame, R. B'Far, J. Cheney, S. a. C. S. Coppens, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes. *PROV-DM: The PROV Data Model*. Ed. by L. Moreau and P. Missier. W3C. Apr. 30, 2013.
URL: https://www.w3.org/TR/prov-dm/ Last Access: Aug. 7, 2021.

[web130] T. D. Huynh, P. Groth, and S. Zednik, eds. *PROV Implementation Report*. W3C. Apr. 30, 2013.
URL: https://www.w3.org/TR/prov-implementations/ Last Access: Aug. 7, 2021.

[web131] T. D. Huynh, M. O. Jewell, A. S. Keshavarz, D. T. Michaelides, H. Yang, and L. Moreau. *The PROV-JSON Serialization. A JSON Representation for the PROV Data Model*. W3C. Apr. 24, 2013.
URL: https://www.w3.org/Submission/prov-json/ Last Access: Aug. 7, 2021.

[web132] L. Moreau and T. Lebo, eds. *Linking Across Provenance Bundles*. W3C. Apr. 30, 2013.
URL: https://www.w3.org/TR/prov-links/ Last Access: Aug. 7, 2021.

[web133] J. Cheney and S. Soiland-Reyes. *PROV-N: The Provenance Notation*.
Ed. by L. Moreau and P. Missier. W3C. Apr. 30, 2013.
URL: http://www.w3.org/TR/prov-n/ Last Access: Aug. 7, 2021.

[web134] C. McCathie Nevila, ed. *World Wide Web Consortium Process Document*. W3C. Mar. 1, 2017.
URL: https://www.w3.org/Consortium/Process/ Last Access: Aug. 7, 2021.

[web135] *Second Provenance Challenge < Challenge < TWiki*.
URL: https://openprovenance.org/provenance-challenge/SecondProvenanceChallenge.html Last Access: Aug. 7, 2021.

[web136] *The Fourth and Last Provenance Challenge*.
URL: https://openprovenance.org/provenance-challenge/FourthProvenanceChallenge.html Last Access: Aug. 7, 2021.

[web137] L. Moreau. *PROV-XML: The PROV XML Schema*. Ed. by H. Hua, C. Tilmes, and S. Zednik. W3C. Apr. 30, 2013. URL: https://www.w3.org/TR/prov-xml/ Last Access: Aug. 7, 2021.

[web138] *PROV Graph Layout Conventions*. W3C. Dec. 13, 2012.
URL: https://www.w3.org/2011/prov/wiki/Diagrams Last Access: Aug. 7, 2021.

[web139] V. Cuevas-Vicenttin, B. Ludäscher, P. Missier, K. Belhajjame, F. Chirigati, Y. Wei, S. Dey, P. Kianmajd, D. Koop, S. Bowers, I. Altintas, C. Jones, M. B. Jones, L. Walker, P. Slaughter, B. Leinfelder, and Y. Cao.
ProvONE: A PROV Extension Data Model for Scientific Workflow Provenance. May 1, 2016.
URL: https://purl.dataone.org/provone-v1-dev Last Access: Aug. 7, 2021.

[web140] *HTML Living Standard*. WHATWG. Aug. 6, 2021.
URL: https://html.spec.whatwg.org/ Last Access: Aug. 7, 2021.

[web141] T. Atkins, E. J. Etemad, and F. Rivoal, eds. *CSS Snapshot 2017*. W3C. Jan. 31, 2017.
URL: https://www.w3.org/TR/CSS/ Last Access: Aug. 7, 2021.

[web142] *ECMAScript® 2021 Language Specification*. Ecma.
URL: https://262.ecma-international.org/ Last Access: Aug. 7, 2021.

[web143] *HTML Living Standard §10: Web workers*. WHATWG. Aug. 6, 2021. URL:
https://html.spec.whatwg.org/multipage/workers.html Last Access: Aug. 7, 2021.

[web144] *Node.js*. Node.js Foundation. URL: https://nodejs.org/en/ Last Access: Aug. 7, 2021.

[web145] *Apache Hadoop*. The Apache Software Foundation.
URL: http://hadoop.apache.org/ Last Access: Aug. 7, 2021.

[web146] *Microsoft/ChakraCore: ChakraCore is the core part of the Chakra Javascript engine that powers Microsoft Edge*.
URL: https://github.com/Microsoft/ChakraCore Last Access: Aug. 7, 2021.

[web147] *SpiderMonkey — Firefox Source Docs documentation*. Mozilla Foundation. URL:
https://firefox-source-docs.mozilla.org/js/index.html Last Access: Aug. 7, 2021.

[web148] *Chrome V8 | Google Developers*. Google. URL: https://v8.dev/ Last Access: Aug. 7, 2021.

[web149] F.-z. Ryuu. *PEG.js – Parser Generator for JavaScript*. URL: https://pegjs.org/.

[web150] *HTML Living Standard §12: The HTML syntax*. WHATWG. Aug. 6, 2021.
URL: https://html.spec.whatwg.org/multipage/syntax.html Last Access: Aug. 7, 2021.

[web151] E. Dahlström, P. Dengler, A. Grasso, C. Lilley, C. McCormack, D. Schepers, J. Watt, J. Ferrarolo, 藤沢淳, and D. Jackson, eds. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C. Aug. 16, 2011. URL: https://www.w3.org/TR/SVG11/ Last Access: Aug. 7, 2021.

[web152] *HTML Living Standard §9.3: Web sockets*. WHATWG. Aug. 6, 2021.
URL: https://html.spec.whatwg.org/multipage/web-sockets.html Last Access: Aug. 7, 2021.

[web153] *CWE - CWE-20: Improper Input Validation (4.3)*. Version 2020-10-10. July 19, 2006.
URL: https://cwe.mitre.org/data/definitions/20.html Last Access: Aug. 7, 2021.

[web154] *HTML Living Standard §4.12.5: The canvas element*. WHATWG. Aug. 6, 2021.
URL: https://html.spec.whatwg.org/multipage/canvas.html Last Access: Aug. 7, 2021.

[web155] *Document Object Model. Living Standard*. WHATWG. Aug. 2, 2021.
URL: https://dom.spec.whatwg.org/ Last Access: Aug. 7, 2021.

[web156] *jsdom*. URL: https://github.com/jsdom/jsdom Last Access: Aug. 7, 2021.

[web157] *Homepage - Material Design*. Google. URL: https://material.io/ Last Access: Aug. 7, 2021.

[web158] *SQLite Home Page*. SQLite Consortium.
URL: https://www.sqlite.org/ Last Access: Aug. 7, 2021.

[web159] *Eurostat: Bulk Download*. European Comission. URL:
http://ec.europa.eu/eurostat/estat-navtree-portlet-prod/BulkDownloadListing Last Access: Aug. 7, 2021.

[web160] *linked-statistics/eurostat: Back-end code and website for eurostat.linked-statistics.org*.
URL: https://github.com/linked-statistics/eurostat Last Access: Aug. 7, 2021.

[web161] *OntologyCentral Eurostat Mirror*.
URL: http://ontologycentral.com/2009/01/eurostat/ Last Access: Aug. 7, 2021.

[web162] S. Schindler, ed. *Yavaa-Vis/Yavaa$_{E}urostat_{C}rawler$: $Eurostatcrawlertogeneratedatadescriptionsfor Yavaa$*. Aug. 15, 2021. URL: https://github.com/Yavaa-Vis/Yavaa_Eurostat_Crawler Last Access: Aug. 15, 2021.

[web163] *DBpedia*. DBpedia Association. URL: https://www.dbpedia.org/ Last Access: Aug. 7, 2021.

[web164] *Microsoft 365 with Office apps | Microsoft 365*. Microsoft.
URL: https://www.microsoft.com/en-us/microsoft-365 Last Access: Aug. 7, 2021.

[web165] *Home | LibreOffice - Free Office Suite - Fun Project - Fantastic People*.
The Document Foundation. URL: https://www.libreoffice.org/ Last Access: Aug. 7, 2021.

[web166] *Amazon Web Services (AWS) - Cloud Computing Services*. Amazon Web Services, Inc.
URL: https://aws.amazon.com/ Last Access: Aug. 7, 2021.

[web167] *Amazon EC2 Instance Types - Amazon Web Services*. Amazon Web Services, Inc.
URL: https://aws.amazon.com/ec2/instance-types/ Last Access: Aug. 7, 2021.

[web168] J. Wise. *JoshuaWise/better-sqlite3: The fastest and simplest library for SQLite3 in Node.js.*
URL: https://github.com/JoshuaWise/better-sqlite3 Last Access: Aug. 7, 2021.

[web169] S. Schindler, ed.
Yavaa-Vis/Yavaa: Supporting Data Workflows from Discovery to Visualization. Aug. 15, 2021.
URL: https://github.com/Yavaa-Vis/Yavaa Last Access: Aug. 15, 2021.

[web170] *Units - GNU Project - Free Software Foundation*. Free Software Foundation, Inc.
URL: https://www.gnu.org/software/units/ Last Access: Aug. 7, 2021.

Dataset Resources

[data1] *Olives by production. tag00122*. Eurostat, Aug. 2, 2017.

[data2] *World Population Prospects - Population Division - United Nations*. United Nations - Department of Econimic and Social Affairs. 2017. URL: https://esa.un.org/unpd/wpp/Download/Standard/Population/.

[data3] *Population on 1 January by age, sex and NUTS 2 region. demo_r_d2jan*. Eurostat, May 30, 2017.

[data4] *Physicians or doctors by NUTS 2 regions. tgs00062*. Eurostat, Aug. 2, 2017.

[data5] G. David, P. Carle, and J.-M. Leroux. *France Geojson*. Apr. 16, 2017. URL: https://github.com/gregoiredavid/france-geojson/.

[data6] Minnesota Population Center. *National Historical Geographic Information System: Version 11.0 [Database].* Minneapolis: University of Minnesota, 2016. DOI: 10.18128/D050.V11.0.

[data7] S. Manson, J. Schroeder, D. V. Riper, and S. Ruggles. *IPUMS National Historical Geographic Information System: Version 12.0 [Database].* Minneapolis: University of Minnesota, 2017. DOI: 10.18128/D050.V12.0.

[data8] *The World Factbook*. Central Intelligence Agency. 2017. URL: https://www.cia.gov/library/publications/the-world-factbook/.

[data9] *Fresh vegetables and strawberries by area. tag00115*. Eurostat, Aug. 2, 2017.

[data10] *Permanent crops for human consumption by area. tag00120*. Eurostat, Aug. 2, 2017.

[data11] *Utilised agricultural area by categories. tag00025*. Eurostat, Aug. 2, 2017.

[data12] *Eurovoc, the EU's multilingual thesaurus*. European Commission. URL: http://eurovoc.europa.eu/.

[data13] *Population Distribution by Race/Ethnicity*. US Census Bureau, 2016.

[data14] E. Celeste. *GeoJSON and KML data for the United States*. July 15, 2017. URL: http://eric.clst.org/Stuff/USGeoJSON.

[data15] *Bundeshaushalt-Info.de*. Bundesministerium der Finanzen, Germany. 2017. URL: https://www.bundeshaushalt-info.de/download.html.

[data16] *Number of sheep. tag00017*. Eurostat, Jan. 15, 2020.

[data17] S. Schindler. *Yavaa-Vis/Yavaa_Eurostat_Crawler: 1.0.0*. Version v1.0.0. Aug. 2021. DOI: 10.5281/zenodo.5204518.

[data18] *Hours worked per week of part-time employment. tps00070*. Eurostat, Aug. 3, 2018.

[data19] *Unemployment rates of the population aged 25-64 by educational attainment level. tps00066*. Eurostat, Aug. 3, 2018.

[data20] S. Schindler. *Yavaa - Evaluation Materials*. Version 1.0.0. Mar. 2021. DOI: 10.5281/zenodo.4589337.

[data21] S. Schindler. *Yavaa - User Survey Results*. Version 1.0.0. Zenodo, Aug. 2021. DOI: 10.5281/zenodo.5171103.

[data22] S. Schindler. *Yavaa - Performance Benchmark*. Version 1.0.0. Feb. 2021. DOI: 10.5281/zenodo.4514808.

[data23] S. Schindler. *Yavaa-Vis / Yavaa: 1.0.0*. Version v1.0.0. Aug. 2021. DOI: 10.5281/zenodo.5204516.

Author's Publications

Publications Pertaining to this Thesis

[1] S. Schindler, M. Hauswirth, and B. König-Ries. "Navigating in a heterogeneous data space". In: *3rd International Conference on Web Science (WebSci'11)*. June 2011.

[2] M. D. Steinberg, S. Schindler, and J. M. Keil. "Use Cases and Suitability Metrics for Unit Ontologies". In: *OWL: Experiences and Directions - Reasoner Evaluation - (OWLED 2016), (ORE 2016)*. 2016, pp. 40–54. DOI: 10.1007/978-3-319-54627-8_4.

[3] J. M. Keil and S. Schindler. "Comparison and evaluation of ontologies for units of measurement". In: *Semantic Web Journal* 10 (2018), pp. 33–51. DOI: 10.3233/SW-180310.

[4] S. Schindler and J. M. Keil. "Building Ontologies for Reuse". In: *2nd International Workshop on Bad Or Good Ontology (BOG 2019)*. 2019.

[5] S. Schindler. *Yavaa - Evaluation Materials*. Version 1.0.0. Mar. 2021. DOI: 10.5281/zenodo.4589337.

[6] S. Schindler. *Yavaa - Performance Benchmark*. Version 1.0.0. Feb. 2021. DOI: 10.5281/zenodo.4514808.

[7] S. Schindler. *Yavaa - User Survey Results*. Version 1.0.0. Zenodo, Aug. 2021. DOI: 10.5281/zenodo.5171103.

[8] S. Schindler. *Yavaa-Vis/Yavaa: 1.0.0*. Version v1.0.0. Aug. 2021. DOI: 10.5281/zenodo.5204516.

[9] S. Schindler. *Yavaa-Vis/Yavaa_Eurostat_Crawler: 1.0.0*. Version v1.0.0. Aug. 2021. DOI: 10.5281/zenodo.5204518.

Other Publications

[1] A. Nauerz, S. Schindler, and F. Bakalov. "Adaptive Treemap Based Navigation Through Web Portals". In: *"Lernen, Wissen & Adaptivität" (LWA 2008)*. Vol. 448. Technical Report. Department of Computer Science, University of Würzburg, Germany, 2008, pp. 51–54.

[2] M. Paradies, S. Schindler, S. Kiemle, and E. Mikusch. "Large-Scale Data Management for Earth Observation Data - Challenges and Opportunities". In: *"Lernen, Wissen, Daten, Analysen" (LWDA 2018)*. Vol. 2191. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 285–288.

[3] L. Feddoul, S. Schindler, and F. Löffler. "Automatic Facet Generation and Selection over Knowledge Graphs". In: *Semantic Systems. The Power of AI and Knowledge Graphs - 15th International Conference (SEMANTiCS 2019)*. Vol. 11702. Lecture Notes in Computer Science. Springer, 2019, pp. 310–325. DOI: 10.1007/978-3-030-33220-4_23.

[4] L. Feddoul, S. Schindler, and F. Löffler.
"Semantic Relatedness as an Inter-facet Metric for Facet Selection over Knowledge Graphs".
In: *The Semantic Web: ESWC 2019 Satellite Events - ESWC 2019 Satellite Events, Portorož, Slovenia, June 2-6, 2019, Revised Selected Papers*. Vol. 11762.
Lecture Notes in Computer Science. Springer, 2019, pp. 47–51.
DOI: 10.1007/978-3-030-32327-1_10.

[5] K. Opasjumruskit, S. Schindler, L. Thiele, and P. M. Schäfer.
"Towards Learning from User Feedback for Ontology-based Information Extraction".
In: *1st International Workshop on Challenges and Experiences from Data Integration to Knowledge Graphs co-located with the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019)*. Vol. 2512. CEUR Workshop Proceedings.
CEUR-WS.org, 2019.

[6] S. Schindler, M. Paradies, and A. Twele.
"Here is my query, where are my results? A search log analysis of the EOWEB® Geoportal".
In: *Conference on Big Data from Space: Turning Data into Insights (BiDS'19)*. 2019, pp. 1–4.

[7] M. D. Steinberg, S. Schindler, and F. Klan.
"Software solutions for form-based, mobile data collection".
In: *Datenbanksysteme für Business, Technologie und Web (BTW 2019)*. Vol. P-290. LNI.
Gesellschaft für Informatik, Bonn, 2019, pp. 135–144. DOI: 10.18420/btw2019-ws-14.

[8] N. Abdelmageed and S. Schindler. "JenTab: Matching Tabular Data to Knowledge Graphs".
In: *The Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab 2020) co-located with the 19th International Semantic Web Conference (ISWC 2020)*.
Ed. by E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, and V. Cutrona.
Vol. 2775. CEUR Workshop Proceedings. CEUR-WS.org, 2020, pp. 40–49.

[9] N. Abdelmageed and S. Schindler. "JenTab: A Toolkit for Semantic Table Annotations".
In: *Second International Workshop on Knowledge Graph Construction (KGCW 2021) co-located with the Extended Semantic Web Conference (ESWC 2021)*. June 6, 2021.

[10] M. A. C. Johnson, M. Paradies, M. Dembska, K. Lackeos, H.-R. Klöckner, D. J. Champion, and S. Schindler. "Astronomical Pipeline Provenance: A Use Case Evaluation".
In: *13th International Workshop on Theory and Practice of Provenance (TaPP 2021)*.
USENIX Association, July 21, 2021.

[11] B. Magagna, I. Rosati, M. Stoica, S. Schindler, G. Moncoiffe, A. Devaraju, J. Peterseil, and R. Huber.
"The I-ADOPT Interoperability Framework for FAIRer data descriptions of biodiversity". In: *Proceedings of the 3rd International Workshop on Semantics for Biodiversity (S4BioDiv2021)*. 2021. arXiv: 2107.06547 [cs.AI].

[12] K. Opasjumruskit, S. Schindler, and D. Peters. "Automatic Data Sheet Information Extraction for Supporting Model-based Systems Engineering". In: *The 18th International Conference on Cooperative Design, Visualization and Engineering (CDVE 2021)*. 2021, pp. 1–4.

Posters / Demos / Extended Abstracts / Preprints

[1] A. Ostrowski, K. Y. Bohn, P. Kaur, E. Petzold, S. Schindler, F. Zander, and B. König-Ries. "Just pain, no gain? Data management systems and biodiversity data". In: *Ecology Across Borders: Joint Annual Meeting, Ghent, Belgium*. 2017. DOI: 10.22032/dbt.38468.

[2] A. Ostrowski, E. Petzold, and S. Schindler. "Showcase: Biodiversity Exploratories Information System Report of our current stage of migration to the new BEXIS2 instance". In: *10th International Conference on Ecological Informatics – Translating Ecological Data into Knowledge and Decisions in a Rapidly Changing World. (ICEI 2018)* (Sept. 2018). DOI: 10.22032/dbt.37933.

[3] L. Feddoul, S. Schindler, and F. Löffler. "Automatic Facet Generation and Selection over Knowledge Graphs". In: *RDA Plenary 14*. Oct. 2019.

[4] K. Opasjumruskit, D. Peters, and S. Schindler. "ConTrOn: Continuously Trained Ontology based on Technical Data Sheets and Wikidata". In: *CoRR* abs/1906.06752 (2019).

[5] S. Schindler and F. Klan. "Automated Data Integration - How to enable a not (yet) Redeemed Promise?" In: *W3C Workshop on Web Standardization for Graph Data*. Mar. 2019.

[6] M. Steinberg, S. Schindler, and F. Klan. "Open Data Kit Goes Semantic - A Contribution to the Interpretability and Interoperability of Citizen Science Data". In: *Forum Citizen Science 2019*. 2019.

[7] M. D. Steinberg, S. Schindler, and F. Klan. "Open Data Kit Goes Semantic - A Contribution to the Interpretability and Interoperability of Field Data". In: *iDiv Annual Conference 2019*. 2019.

[8] M. D. Steinberg, S. Schindler, and F. Klan. "Bringing Semantics to Citizen Data Collection - A Semantic Extension of Open Data Kit 1". In: *EGU General Assembly 2019*. 2019.

[9] B. Magagna, G. Moncoiffe, A. Devaraju, P. L. Buttigieg, M. Stoica, and S. Schindler. "Towards an interoperability framework for observable property terminologies". In: (Mar. 2020). DOI: 10.5194/egusphere-egu2020-19895.

[10] K. Opasjumruskit, D. Peters, and S. Schindler. "DSAT: Ontology-based Information Extraction on Technical Data Sheets". In: *ISWC 2020 Demos and Industry Tracks, co-located with 19th International Semantic Web Conference (ISWC 2020)*. Vol. 2721. CEUR Workshop Proceedings. CEUR-WS.org, Nov. 2020, pp. 251–256.

[11] B. Magagna, G. Moncoiffe, A. Devaraju, M. Stoica, S. Schindler, and A. Pamment. "I-ADOPT Framework 1.0.0". In: *RDA Virtiual Plenary 17*. Apr. 2021.

[12] B. Magagna, G. Moncoiffe, M. Stoica, A. Devaraju, A. Pamment, S. Schindler, and R. Huber. "The I-ADOPT Interoperability Framework: a proposal for FAIRer observable property descriptions". In: (Mar. 2021). DOI: 10.5194/egusphere-egu21-13155.

Invited Talks

[1] M. Paradies, S. Schindler, and R. Axmann.
"Datenmanagement für Intelligente Verkehrssysteme und Erdbeobachtung: Unterschiedliche Anwendungsbereiche mit gleichen Big Data Herausforderungen?" In: *POSNAV 2018*.
June 2018.

[2] S. Schindler. "The DLR Institute of Data Science".
In: *Radio2018 and GLOW Annual Assembly*. Oct. 2018.

[3] M. Stoica, B. Magagna, A. Pamment, and S. Schindler.
"Decomposing Observable Property Descriptions into Machine-Readable Components to Increase Interoperability Across Data Standards". In: *RDA US, Webinar 2021 Series*.
July 28, 2021.

Appendices

APPENDIX A

RDF NAMESPACES

RDF examples throughout this work are given in Turtle syntax [web63]. Table A.1 gives a list of namespaces and the respective prefixes used.

Prefix	Namespace
dc	http://purl.org/dc/elements/1.1/
dcat	http://www.w3.org/ns/dcat#
dcterms	http://purl.org/dc/terms/
dctype	http://purl.org/dc/dcmitype/
foaf	http://xmlns.com/foaf/0.1/
owl	http://www.w3.org/2002/07/owl#
prov	http://www.w3.org/ns/prov#
qb	http://purl.org/linked-data/cube#
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
skos	http://www.w3.org/2004/02/skos/core#
void	http://rdfs.org/ns/void#
wd	https://www.wikidata.org/wiki/
xsd	http://www.w3.org/2001/XMLSchema#
yavaa	http://yavaa.org/ns/
yDim	http://yavaa.org/ns/dimension/
yUnit	http://yavaa.org/ns/units/

Table A.1: RDF namespaces used.

APPENDIX B

TESTING SOFTWARE FOR UNIT-SUPPORT

Besides the approaches in the literature described in Section 9.1, there are also software packages supporting computations involving units. Software packages like SPSS [web107] or Stata [web108] focussing mostly on statistical analysis make no mention of any support for units of measurement in their documentation. Other packages like Octave [web106] offer means to convert from one unit to the other, but no automatic dimensional checks or inference of units[1]. The R project [web105] itself provides no unit support either, but there are different packages adding various levels of support. The package `measurements` [web110] adds functions to allow users to convert between predefined units. Another package, `units` [web109], adds separate objects, which include both magnitude as well as unit of a value. Regarding the conversion, however, it uses a naïve approach as it is stated[2]: "When mixing units in sums, comparisons or concatenation, units are automatically converted to those of the first argument".

More sophisticated systems like Mathematica [web103] or Matlab [web104] also have built-in support for units of measurement. However, in their documentation, they do not describe their approach in detail. Mathematica just states[3], that "many symbolic commands are capable of understanding units". Matlab is similarly vague saying[4] "The simplify function automatically chooses the unit to simplify to".

[1] Octave offers a function to integrate the result of GNU units [web170] using the function `units`: `https://octave.sourceforge.io/miscellaneous/function/units.html`.
[2] `https://cran.r-project.org/web/packages/units/vignettes/measurement_units_in_R.html`
[3] `https://reference.wolfram.com/language/tutorial/SymbolicCalculationsWithUnits.html`
[4] `https://mathworks.com/help/symbolic/units-of-measurement-tutorial.html`

To get an idea nonetheless of how these systems handle units of measurements in computations an empirical evaluation by the means of sample formulae was performed. Below, the test cases will be listed alongside the respective rationale behind them[5]. Oftentimes, there will be two almost identical test cases. This is done to account for the naïve approach to always use the unit of the left-hand side operand.

1. **Preference for SI units:** The review of literature in Section 9.1 suggests that systems may have a preference for SI units [300] as compared to units of other systems. So in this test, a non-SI unit is added to an SI unit to see if the system will always return the same unit.
   Test cases:

   - $[m]+[ft]$
   - $[m]+[mile]$
   - $[ft]+[m]$
   - $[mile]+[m]$

2. **Dominant unit:** If a formula consists of multiple (compatible) units, the resulting unit should be the one that appears most often.
   Test cases:

   - $[m]+[ft]+[m]$
   - $[ft]+[m]+[ft]$

3. **Composition of compound units:** Derived units are a composition of multiple base units. An example is the unit Newton which is defined as Kilogram times Meter per Second squared. This test checks, whether the system is able to recognize such compound units and substitutes them in the result.
   Test cases:

   - $[kg]\times[m]\,/\,[s]^2$
   - $[N]\times[m]\times[s]$

4. **Composition of compound units using different units:** In the previous test case, the components of a compound unit were given according to the definition. Here, however, single components are substituted with other compatible units.
   Test cases:

   - $[kg]\times[ft]\,/\,[s]^2$
   - $[kg]\times[cm]\,/\,[s]^2$

[5]Unit names will be abbreviated as follows: Meter $[m]$, Centimeter $[cm]$, Kilometer $[km]$, Foot $[ft]$, Yard $[yd]$, Mile $[mi]$ Kilogram $[kg]$, Second $[s]$, Newton $[N]$.

5. **Partial composition of compound units:** Analogous to the previous one, this test case checks for the recognition of compound units. However, the result is not just a single compound unit, but also another unit in addition.
   Test cases:

   - $[kg]^2 \times [m] \,/\, [s]^2$

6. **Decomposition of compound units:** This test case is symmetrical to the preceding ones. The system should also be able to decompose compound units, if necessary. It also checks under which circumstances that decomposition occurs.
   Test cases:

   - $[N] \,/\, [m]$
   - $[N] \,/\, \frac{[m]}{[s]}$
   - $[N] \,/\, \frac{[m]}{[s]^2}$

7. **Preference for unprefixed units (prefix < 1):** When unifying a unit with a prefixed version of the same unit, does the system choose the prefixed or the unprefixed unit for the result? In this iteration, the prefix represents a factor smaller than one.
   Test cases:

   - $[m] + [cm]$
   - $[cm] + [m]$

8. **Preference for unprefixed units (prefix > 1):** Using the same rationale as before, here prefixes are checked that represent a factor larger than one.
   Test cases:

   - $[m] + [km]$
   - $[km] + [m]$

9. **Preference for smaller or larger units (no prefix, same system):** Similar to the before test cases, a system may have different units for the same system without one being a prefixed version of the other. This test checks, whether the smaller or larger unit will be chosen.
   Test cases:

   - $[ft] + [yd]$
   - $[yd] + [ft]$

Mathematica [web103] has had native support for units since Version 9, which was released on November 28th, 2012. Matlab [web104] introduced units with Version 9.2 (R2017a) released on March 9th, 2017. For both systems, scripts were created to check the above test cases. The scripts are given in Listing B.1 and Listing B.3, while the respective outputs are shown in Listing B.2 and Listing B.4. For both systems, the current versions at the time of writing were used. For Mathematica this is 12.2.0 and for Matlab 9.10 (R2021a).

Both systems seem to employ a more sophisticated approach than the aforementioned naïve one. The symmetric test cases, that check for these simple rules, all come back with identical results independent of operand order. The naïve approach would have yielded different results here.

The first two test cases also show no preference for SI units in both systems. On the contrary, the non-SI unit `Foot` is used for the result no matter the number of input variables using it (see Test Case 2). For Mathematica judging from Test Cases 1, 2, 7, 8, and 9 the preference is towards the "smaller" unit – that is the one to use fewer decimals to represent the same value. For Matlab this assumption fails, however, as in Test Case 8 Kilometer is chosen over Meter. Here, another criterion is used which is not easily deduced from the examined samples.

Composition of compound units seems only to be present in Mathematica. However, even substitutes using different units were recognized. Matlab, on the other hand, does not replace any compound units in Test Cases 3 to 5.

The decomposition of compound units seems to work the same way for both systems. It seems that a compound unit is decomposed as soon as the result has fewer components than the not decomposed version. The term $[N]$ / $[m]$, for example, consists of two components, whereas the decomposed version $[kg]$ / $[s]^2$ has three counting both occurrences of Second. This changes for $[N]$ / $\frac{[m]}{[s]}$ where the base version consists of three components, while the decomposed one has only two: $[kg]$ / $[s]$.

Overall, the support in Mathematica seems more sophisticated than Matlab. This is probably caused by the more recent addition of unit support in Matlab, while the support in Mathematica already has had the time to evolve over several more years and versions.

```
(* 1. Preference for SI units *)
Print[ "1. Preference for SI units" ]
Print[ "[m] + [ft]: ", UnitSimplify[ Quantity[ 1, "Meter" ]
                      + Quantity[ 1, "Feet" ] ] ]
Print[ "[ft] + [m]: ", UnitSimplify[ Quantity[ 1, "Feet" ]
                      + Quantity[ 1, "Meter" ] ] ]
Print[ "[m] + [mile]: ", UnitSimplify[ Quantity[ 1, "Meter" ]
                      + Quantity[ 1, "Mile" ] ] ]
Print[ "[mile] + [m]: ", UnitSimplify[ Quantity[ 1, "Mile" ]
                      + Quantity[ 1, "Meter" ] ] ]

(* 2. Dominant unit *)
Print[ "2. Dominant unit" ]
Print[ "[m] + [ft] + [m]: ", UnitSimplify[ Quantity[ 1, "Meter" ]
                            + Quantity[ 1, "Feet" ] + Quantity[ 1, "Meter" ] ] ]
Print[ "[ft] + [m] + [ft]: ", UnitSimplify[ Quantity[ 1, "Feet" ]
                             + Quantity[ 1, "Meter" ] + Quantity[ 1, "Feet" ] ] ]

(* 3. Composition of compound units *)
Print[ "3. Composition of compound units" ]
Print[ "[kg] * [m] / [s]^2: ", UnitSimplify[ Quantity[ 1, "Kilogram" ]
                               * Quantity[ 1, "Meter" ] / ( Quantity[ 1, "Second" ]^2 ) ] ]
Print[ "[N] * [m] * [s]: ", UnitSimplify[ Quantity[ 1, "Newton" ]
                           * Quantity[ 1, "Meter" ] * Quantity[ 1, "Second" ] ] ]

(* 4. Composition of compound units using different units *)
Print[ "4. Composition of compound units using different units" ]
Print[ "[kg] * [ft] / [s]^2: ", UnitSimplify[ Quantity[ 1, "Kilogram" ]
                                * Quantity[ 1, "Feet" ] / ( Quantity[ 1, "Second" ]^2 ) ] ]
Print[ "[kg] * [cm] / [s]^2: ", UnitSimplify[ Quantity[ 1, "Kilogram" ]
                                * Quantity[ 1, "Centimeter" ] / ( Quantity[ 1, "Second" ]^2 ) ] ]

(* 5. Partial composition of compound units *)
Print[ "5. Partial composition of compound units" ]
Print[ "[kg]^2 * [m] / [s]^2: ", UnitSimplify[ ( Quantity[ 1, "Kilogram" ]^2)
                                 * Quantity[ 1, "Meter" ] / ( Quantity[ 1, "Second" ]^2 ) ] ]

(* 6. Decomposition of compound units *)
Print[ "6. Decomposition of compound units" ]
Print[ "[N] / [m]: ", UnitSimplify[ Quantity[ 1, "Newton" ] / Quantity[ 1, "Meter" ] ] ]
Print[ "[N] / ( [m] / [s] ): ", UnitSimplify[ Quantity[ 1, "Newton" ]
                                / ( Quantity[ 1, "Meter" ] / Quantity[ 1, "Second" ] ) ] ]
Print[ "[N] / ( [m] / [s]^2 ): ", UnitSimplify[ Quantity[ 1, "Newton" ]
                                  / ( Quantity[ 1, "Meter" ] / ( Quantity[ 1, "Second" ]^2 ) ) ] ]

(* 7. Preference for unprefixed units (prefix < 1) *)
Print[ "7. Preference for unprefixed units (prefix < 1)" ]
Print[ "[m] + [cm]: ", UnitSimplify[ Quantity[ 1, "Meter" ] + Quantity[ 1, "Centimeter" ] ] ]
Print[ "[cm] + [m]: ", UnitSimplify[ Quantity[ 1, "Centimeter" ] + Quantity[ 1, "Meter" ] ] ]

(* 8. Preference for unprefixed units (prefix > 1) *)
Print[ "8. Preference for unprefixed units (prefix > 1)" ]
Print[ "[m] + [km]: ", UnitSimplify[ Quantity[ 1, "Meter" ] + Quantity[ 1, "Kilometer" ] ] ]
Print[ "[km] + [m]: ", UnitSimplify[ Quantity[ 1, "Kilometer" ] + Quantity[ 1, "Meter" ] ] ]

(* 9. Preference for smaller or larger units (no prefix, same system) *)
Print[ "9. Preference for smaller or larger units (no prefix, same system)" ]
Print[ "[ft] + [yd]: ", UnitSimplify[ Quantity[ 1, "Feet" ] + Quantity[ 1, "Yard" ] ] ]
Print[ "[yd] + [ft]: ", UnitSimplify[ Quantity[ 1, "Yard" ] + Quantity[ 1, "Feet" ] ] ]
```

Listing B.1: Unit testing script in Mathematica [web103].

```
1  1. Preference for SI units
   [m] + [ft]: 1631/381 ft
   [ft] + [m]: 1631/381 ft
4  [m] + [mile]: 201293/25 m
   [mile] + [m]: 201293/25 m

7  2. Dominant unit
   [m] + [ft] + [m]: 2881/381 ft
   [ft] + [m] + [ft]: 2012/381 ft
10
   3. Composition of compound units
   [kg] * [m] / [s]^2: 1 N
13 [N] * [m] * [s]: 1 s J

   4. Composition of compound units using different units
16 [kg] * [ft] / [s]^2: 381/1250 N
   [kg] * [cm] / [s]^2: 1/100 N

19 5. Partial composition of compound units
   [kg]^2 * [m] / [s]^2: 1 kg N

22 6. Decomposition of compound units
   [N] / [m]: 1 N / m
   [N] / ( [m] / [s] ): 1 kg / s
25 [N] / ( [m] / [s]^2 ): 1 kg

   7. Preference for unprefixed units (prefix < 1)
28 [m] + [cm]: 101 cm
   [cm] + [m]: 101 cm

31 8. Preference for unprefixed units (prefix > 1)
   [m] + [km]: 1001 m
   [km] + [m]: 1001 m
34
   9. Preference for smaller or larger units (no prefix, same system)
   [ft] + [yd]: 4 ft
37 [yd] + [ft]: 4 ft
```

Listing B.2: Results for unit testing script in Mathematica [web103].

```
% init
clc  % clear output
u = symunit; % load units

% 1. Preference for SI units
disp( "1. Preference for SI units" )
disp( [ '[m] + [ft]: ', char( simplify( 1 * u.m + 1 * u.ft ) ) ] )
disp( [ '[ft] + [m]: ', char( simplify( 1 * u.ft + 1 * u.m ) ) ] )
disp( [ '[m] + [mile]: ', char( simplify( 1 * u.m + 1 * u.mi ) ) ] )
disp( [ '[mile] + [m]: ', char( simplify( 1 * u.mi + 1 * u.m ) ) ] )

% 2. Dominant unit
disp( ' ' )
disp( '2. Dominant unit' )
disp( [ '[m] + [ft] + [m]: ', char( simplify( 1 * u.m + 1 * u.ft + 1 * u.m ) ) ] )
disp( [ '[ft] + [m] + [ft]: ', char( simplify( 1 * u.ft + 1 * u.m + 1 * u.ft ) ) ] )

% 3. Composition of compound units
disp( ' ' )
disp( '3. Composition of compound units' )
disp( [ '[kg] * [m] / [s]^2: ', char( simplify( (1 * u.kg) * (1 * u.m) / ((1 * u.s)^2) ) ) ] )
disp( [ '[N] * [m] * [s]: ', char( simplify( (1 * u.N) * (1 * u.m)  * (1 * u.s) ) ) ] )

% 4. Composition of compound units using different units
disp( ' ' )
disp( '4. Composition of compound units using different units' )
disp( [ '[kg] * [ft] / [s]^2: ', char( simplify( (1 * u.kg) * (1 * u.ft) / ((1 * u.s)^2) ) ) ] )
disp( [ '[kg] * [cm] / [s]^2: ', char( simplify( (1 * u.kg) * (1 * u.cm) / ((1 * u.s)^2) ) ) ] )

% 5. Partial composition of compound units
disp( ' ' )
disp( '5. Partial composition of compound units' )
disp( [ '[kg] * [m]^2 / [s]^2: ', char( simplify( 1 * u.kg * (1 * u.m)^2 / (1 * u.s)^2 ) ) ] )

% 6. Decomposition of compound units
disp( ' ' )
disp( '6. Decomposition of compound units' )
disp( [ '[N] / [m]: ', char( simplify( 1 * u.N / (1 * u.m) ) ) ] )
disp( [ '[N] / ( [m] / [s] ): ', char( simplify( 1 * u.N / ( (1 * u.m) / (1 * u.s) ) ) ) ] )
disp( [ '[N] / ( [m] / [s]^2 ): ', char( simplify( 1 * u.N / ( (1 * u.m) / (1 * u.s)^2 ) ) ) ] )

% 7. Preference for unprefixed (prefix < 1)
disp( ' ' )
disp( "7. Preference for unprefixed units (prefix < 1)" )
disp( [ '[m] + [cm]: ', char( simplify( 1 * u.m + 1 * u.cm ) ) ] )
disp( [ '[cm] + [m]: ', char( simplify( 1 * u.cm + 1 * u.m ) ) ] )

% 8. Preference for unprefixed (prefix > 1)
disp( ' ' )
disp( "8. Preference for unprefixed units (prefix > 1)" )
disp( [ '[m] + [km]: ', char( simplify( 1 * u.m + 1 * u.km ) ) ] )
disp( [ '[km] + [m]: ', char( simplify( 1 * u.km + 1 * u.m ) ) ] )

% 9. Preference for smaller or larger units (no prefix, same system)
disp( ' ' )
disp( "9. Preference for smaller or larger units (no prefix, same system)" )
disp( [ '[ft] + [yd]: ', char( simplify( 1 * u.ft + 1 * u.yd ) ) ] )
disp( [ '[yd] + [ft]: ', char( simplify( 1 * u.yd + 1 * u.ft ) ) ] )
```

Listing B.3: Unit testing script in Matlab [web104].

```
   1. Preference for SI units
2  [m] + [ft]: (1631*symunit('ft'))/381
   [ft] + [m]: (1631*symunit('ft'))/381
   [m] + [mile]: (201293*symunit('m'))/125
5  [mile] + [m]: (201293*symunit('m'))/125

   2. Dominant unit
8  [m] + [ft] + [m]: (2881*symunit('ft'))/381
   [ft] + [m] + [ft]: (2012*symunit('ft'))/381

11 3. Composition of compound units
   [kg] * [m] / [s]^2: (symunit('kg')*symunit('m'))/symunit('s')^2
   [N] * [m] * [s]: symunit('N')*symunit('m')*symunit('s')
14
   4. Composition of compound units using different units
   [kg] * [ft] / [s]^2: (symunit('ft')*symunit('kg'))/symunit('s')^2
17 [kg] * [cm] / [s]^2: (symunit('cm')*symunit('kg'))/symunit('s')^2

   5. Partial composition of compound units
20 [kg] * [m]^2 / [s]^2: (symunit('kg')*symunit('m')^2)/symunit('s')^2

   6. Decomposition of compound units
23 [N] / [m]: symunit('N')/symunit('m')
   [N] / ( [m] / [s] ): symunit('kg')/symunit('s')
   [N] / ( [m] / [s]^2 ): symunit('kg')
26
   7. Preference for unprefixed units (prefix < 1)
   [m] + [cm]: 101*symunit('cm')
29 [cm] + [m]: 101*symunit('cm')

   8. Preference for unprefixed units (prefix > 1)
32 [m] + [km]: (1001*symunit('km'))/1000
   [km] + [m]: (1001*symunit('km'))/1000

35 9. Preference for smaller or larger units (no prefix, same system)
   [ft] + [yd]: 4*symunit('ft')
   [yd] + [ft]: 4*symunit('ft')
```

Listing B.4: Results for unit testing script in Matlab [web104].

APPENDIX C

ADDED UNITS AND DIMENSIONS

For a broader spectrum of datasets to be used in the evaluation (cf. Chapter 14) it became necessary to extend the pool of units and dimensions provided by OM [164] (cf. Section 13.4). The following tables list all the entities that were added alongside the information to connect them to the remainder of the ontology. Table C.1 lists added dimensions alongside their dimensional vector. Tables C.2 and C.3 enumerate added units including their dimension and a conversion, where applicable. Finally, compound units as well as their compounds are given in Table C.4.

IRI	Label	Dimensional Vector
yDim:count	dimensionless count	[0, 0, 0, 0, 0, 1, 0]
yDim:massLength	mass times length	[1, 1, 0, 0, 0, 0, 0]
yDim:massPerTime	mass per time	[1, 1, -1, 0, 0, 0, 0]
yDim:perCount	per count	[0, 0, 0, 0, 0, -1, 0]
yDim:shipTonnage	ship volume	[3, 0, 0, 0, 0, 0, 0]

Table C.1: Dimensions added.

IRI (Label)	Dimension	Conversion
yUnit:gigawattHours (gigawatt hours)	om:energy-dimension	$1e+6\times$ om:kilowatt_hour
yUnit:grossTonnage (gross tonnage)	yDim:shipTonnage	
yUnit:hundred (hundred)	yDim:count	$1e+2\times$ yUnit:one
yUnit:kgOilEquivalent (kilogram of oil equivalent)	om:energy-dimension	$4.1868e+4\times$ om:gigajoule
yUnit:million (million)	yDim:count	$1e+6\times$ yUnit:one
yUnit:millionCubicmeter (million cubic meter)	om:volume-dimension	$1e+6\times$ om:cubic_metre
yUnit:millionEuro (million Euro)	om:Amount_of_money	$1e+6\times$ om:euro
yUnit:millionPps (million purchasing power standard)	om:Amount_of_money	$1e+6\times$ yUnit:pps
yUnit:millionSquaremeter (million square meter)	om:area-dimension	$1e+6\times$ om:square_metre
yUnit:millionTonne (million tonnes)	om:mass-dimension	$1e+6\times$ om:tonne
yUnit:one (one)	yDim:count	
yUnit:perOne (per one)	yDim:perCount	
yUnit:perHundred (per one hundred)	yDim:perCount	$1e-2\times$ yUnit:perOne
yUnit:perHundredThousand (per one hundred Thousand)	yDim:perCount	$1e-5\times$ yUnit:perOne
yUnit:perMillion (per one million)	yDim:perCount	$1e-6\times$ yUnit:perOne
yUnit:perThousand (per one thousand)	yDim:perCount	$1e+0\times$ yUnit:perOne
yUnit:pps (purchasing power standard)	om:Amount_of_money	
yUnit:terajoule (terajoule)	om:energy-dimension	$1e+12\times$ om:joule
yUnit:thousand (thousand)	yDim:count	$1e+3\times$ yUnit:one

Table C.2: Units added I.

IRI (Label)	Dimension	Conversion
yUnit:thousandCubicmeter (thousand cubic meter)	om:volume-dimension	$1e+3\times$ om:cubic_metre
yUnit:thousandEuro (thousand Euro)	om:Amount_of_money	$1e+3\times$ om:euro
yUnit:thousandHectar (thousand hectar)	om:area-dimension	$1e+3\times$ om:hectare
yUnit:thousandMinute (thousand minutes)	om:time-dimension	$1e+3\times$ om:minute-time
yUnit:thousandSquaremeter (thousand square meter)	om:area-dimension	$1e+3\times$ om:square_metre
yUnit:thousandTonnes (thousand tonnes)	om:mass-dimension	$1e+3\times$ om:tonne
yUnit:thousandTonnesOilEquivalent (thousand tonnes of oil equivalent)	om:energy-dimension	$1e+3\times$ yUnit:tonneOilEquivalent
yUnit:thousandTonnesPerDay (thousand tonnes per day)	yDim:massPerTime	$1e+3\times$ yUnit:tonnePerDay
yUnit:tonneOilEquivalent (tonne of oil equivalent)	om:energy-dimension	
yUnit:tonnesPerDay (tonnes per day)	yDim:massPerTime	

Table C.3: Units added II.

IRI (Label)	Compound Elements
yUnit:euroPerTonneOilEquivalent (Euro per tonne of oil equivalent)	om:euro / yUnit:tonneOilEquivalent
yUnit:gigawattHour (gigawatt-hour)	om:gigawatt * om:hour
yUnit:kilogrammOilEquivalentPerThousandEuro (kilogram of oil equivalent per thousand Euro)	yUnit:kgOilEquivalent / yUnit:thousandEuro
yUnit:kilogramPerThousandEuro (kilogram per thousand Euro)	om:kilogram / yUnit:thousandEuro
yUnit:millionPpsPerSquarekilometer (million purchasing power standard per square kilometer)	yUnit:millionPps / om:square_kilometre
yUnit:perMile (per mile)	1 / om:mile-US_survey
yUnit:tonnePerDay (tonne per day)	om:tonne / om:day
yUnit:tonnePerInhabitant (tonnes per capita)	om:tonne / yUnit:inhabitant

Table C.4: Compound units added.

APPENDIX D

YAVAA: LIST OF SUPPORTED MESSAGES

In the following, all messages send between worker and interface of Yavaa are listed (cf. Section 13.6). Messages are grouped by area of use and direction of transmission. Messages are sent either from worker to interface or vice versa. This documentation loosely follows JSDoc conventions [web121] to describe parameters. So, optional parameters are denoted by `[parameter]`.

Aggregation

aggregate (*interface → worker*)

Aggregate inside a dataset by the given columns.

Parameters:

`[agg]`	(Array)	...	List of aggregation functions for columns to be aggregated
`cols`	(Array)	...	List of columns to group by
`data_id`	(Number)	...	ID of the base dataset

unbag (*interface → worker*)

Unbag one column.

Parameters:

`agg`	(String)	...	Function used to unbag
`col`	(Number)	...	ID of column to unbag
`data_id`	(Number)	...	ID of the base dataset

Computation

compute (*interface → worker*)

Apply the given operation to a column.

Parameters:

`col_id`	(Number)	...	ID of the respective column
`data_id`	(Number)	...	ID of the respective dataset
`[label]`	(String)	...	Label of the column, if it is a new one
`new_col`	(Boolean)	...	Result stored in a new column? will replace source column otherwise
`op`	(String)	...	Actual operation
`op_type`	(["UDF"])	...	Type of the operation

setUnit (*interface → worker*)

Change the unit for a given dataset and column.

Parameters:

`col_id`	(Number)	...	ID of the respective column
`data_id`	(Number)	...	ID of the respective dataset
`unit`	(String)	...	URI for the target unit

done (*worker → interface*)

Operation has finished.

Parameters:

`data_id`	(Number)	...	ID of the result dataset

progress (*worker → interface*)

Progress of current column operation.

Parameters:

`progress`	(Number)	...	Current progress as value [0,1]

Data Communication

getColumnValues (*interface → worker*)

Get a list of distinct values for a particular column.

Parameters:

`col_id`	(Number)	...	ID of the column
`data_id`	(Number)	...	ID of the dataset

getData (*interface → worker*)

Request a chunk of data from the given dataset.

Parameters:

`data_id`	(Number)	...	ID of the respective dataset
`entries`	(Number)	...	Amount of entries
`start`	(Number)	...	Start index inside the dataset

getMeta (*interface → worker*)

Request the metadata for a dataset.

Parameters:

`data_id`	(Number)	...	ID of the respective dataset

columnValues (*worker → interface*)

List of distinct values for the given column/dataset.

Parameters:

`col_id`	(Number)	...	ID of the respective column
`data_id`	(Number)	...	ID of the respective dataset
`values`	(Object)	...	Distinct column values; using "min"/"max" for numeric and time, "list" otherwise

data (*worker → interface*)

Subset of primary data.

Parameters:

`data`	(Array[Array])	...	Requested chunk of data
`data_id`	(Number)	...	ID of the respective dataset

meta (*worker → interface*)

Metadata for the given dataset.

Parameters:

`data_id`	(Number)	...	ID of the dataset
`entries`	(Number)	...	Number of rows in the dataset
`meta`	(Object)	...	Requested metadata

Data Filtering

dropColumns (*interface → worker*)

Drop the given columns from the given dataset.

Parameters:

`columns`	(Array[Number])	...	IDs of the columns to drop
`data_id`	(Number)	...	ID of the respective dataset

filterData (*interface → worker*)

Filter the given dataset.

Parameters:

`data_id`	(Number)	...	ID of the respective dataset
`filterDef`	(Object)	...	Filter-options as an AST-like object

Data Retrieval

export (*interface → worker*)

Export a dataset to a file.

Parameters:

`data_id`	(Number)	...	ID of the dataset to export
`mime`	(String)	...	Mime type of the download requested
`part`	(String)	...	Part requested for download (ds, wf, vis)
`[visoptions]`	(Object)	...	If a vis is chosen as a part, this has to hold the respective parameters as defined in getStaticViz

loadData (*interface → worker*)

Request loading of a dataset given by the ID.

Parameters:

`id`	(String)	...	ID of the respective dataset

loadFile (*interface → worker*)

Request parsing of the given string as a new dataset.

Parameters:

`content`	(String)	...	Content of the file to parse
`module`	(String)	...	Name of the parser module to use
`parser`	(Array)	...	List of parsers to use for the columns
`settings`	(Object)	...	Settings for the parser

exported (*worker → interface*)

Result of export function.

Parameters:

`data`	(String)	...	Content of the exported file
`data_id`	(Number)	...	Internal id of the retrieved dataset
`mime`	(String)	...	Mime type for the download
`part`	(String)	...	Part triggered for download

Dataset related information

resolveColValues (*interface → worker*)

Resolve values for a given column (has to be of type semantic), i.e. add labels.

Parameters:

`columns`	(Array)	...	IDs of all columns to be resolved
`data_id`	(Number)	...	Referenced dataset

setColLabel (*interface → worker*)

Set the label for a given column.

Parameters:

`col_id`	(Number)	...	ID of the column
`data_id`	(Number)	...	Referenced dataset
`label`	(String)	...	New label

Debug

getMemory (*interface → worker*)

Retrieve the size of currently used memory.

memory (*worker → interface*)

Description of the currently used memory.

Parameters:

`size`	(Number)	...	Size of currently used memory

General

error (*interface → worker*)

Description of error happening on the client side; to be logged on server side.

Parameters:

`msg`	(String)	...	Error message
`src`	(String)	...	Module where the error occurred
`[stack]`	(String)	...	Stack trace of the error
`ts`	(Number)	...	Timestamp of the error; as Unix timestamp

error (*worker → interface*)

Description of an error happening on the server side.

Parameters:

`msg`	(String)	...	Error message
`src`	(String)	...	Module where the error occurred
`[stack]`	(String)	...	Stack trace of the error
`ts`	(Number)	...	Timestamp of the error; as Unix timestamp

invalidCommand (*worker → interface*)

Last issued command was invalid.

Parameters:

`command`	(Array)	...	Invalid command
`message`	(String)	...	Error message

queued (*worker → interface*)

Operation queued for execution.

Joins

join (*interface → worker*)

Join the two given datasets.

Parameters:

`augm_data_id`	(Number)	...	ID of the augmenting dataset
`data_id`	(Number)	...	ID of the base dataset
`join_cond`	(Array)	...	List of matching columns

suggestJoin (*interface → worker*)

Suggest a possible join condition for the two given datasets.

Parameters:

`data_id1`	(Number)	...	ID of the left hand dataset
`data_id2`	(Number)	...	ID of the right hand dataset

union (*interface → worker*)

Union the two given datasets.

Parameters:

`augm_data_id`	(Number)	...	ID of the augmenting dataset
`base_data_id`	(Number)	...	ID of the base dataset
`union_cond`	(Array)	...	List of matching columns

queued (*worker → interface*)

Operation queued for execution.

suggestedJoin (*worker → interface*)

Proposed join condition.

Parameters:

`data_id1`	(Number)	...	ID of the left hand dataset
`data_id2`	(Number)	...	ID of the right hand dataset
`join_cond`	(Array[Array])	...	Join condition

Knowledge Base

getCompatibleUnits (*interface → worker*)

Request a list of compatible alternatives for a given unit.

Parameters:

`unit`	(Object)	...	Unit for which alternatives are requested

getDsDetails (*interface → worker*)

Request details for a specific dataset.

Parameters:

`id`	(String)	...	URI of the dataset to get data about

resolveCodelists (*interface → worker*)

Find the associated values and labels for the given list of codelists-URIs.

Parameters:

`uris`	(Array[String])	...	List of codelist-URIs

resolveLabels (*interface → worker*)

Resolve labels for a given list of URIs.

Parameters:

`uris`	(Array[String])	...	List of URIs

compatibleUnits (*worker → interface*)

List of compatible units and the respective systems.

Parameters:

`systems`	(Array)	...	System represented by at least one unit
`unit`	(String)	...	Unit requested
`units`	(Array)	...	List of compatible units

dsDetails (*worker → interface*)

Results of getDsDetails command.

Parameters:

`cols`	(Array[Object])	...	Column data for this dataset
`meta`	(Object)	...	Metadata for this dataset

resolvedCodelists (*worker → interface*)

Labels for resolved URIs.

Parameters:

`results`	(Object)	…	Map from codelist-URI to array of contained values

resolvedLabels (*worker → interface*)

Labels for resolved URIs.

Parameters:

`results`	(Object)	…	Map from URI to label

Search

getDsByCombination (*interface → worker*)

Request a combination of datasets to fit the given description.

Parameters:

`constraints`	(Array)	…	Constraints given per column

search (*interface → worker*)

Search datasets by keyword.

Parameters:

`restrictions`	(Object)	…	Search restrictions

typeAhead (*interface → worker*)

Get a list of possible terms for an autocomplete field.

Parameters:

`[codelist]`	(String)	…	For values using codelist, define that codelist
`needle`	(String)	…	Currently given substring
`type`	(String)	…	Type to search for (dimension, measurement, column, value)

getDsByCombination (*worker → interface*)

Possible combination of datasets found.

Parameters:

`components`	(Array)	…	List of datasets used
`pwf`	(Object | Number)	…	Pseudo-workflow to achieve the proposed result
`result`	(Object)	…	Description of the resulting dataset

search (*worker → interface*)

Results of a previous search.

Parameters:

`results`	(Array)	…	results

typeAhead (*worker → interface*)

List of possible terms.

Parameters:

`terms`	(Object)	…	List of possible terms
`type`	(String)	…	Used type for the query

visualization

getStaticSVG (*interface → worker*)

Get a static SVG visualization.

Parameters:

`data_id`	(Number)	...	ID of the result dataset
`options`	(Object)	...	Settings for the visualization
`type`	(String)	...	Type of requested visualization

suggestViz (*interface → worker*)

Get visualization recommendation for the given dataset.

Parameters:

`data_id`	(Number)	...	ID of the result dataset

viz (*worker → interface*)

Visualization in form of the respective (SVG) code.

Parameters:

`code`	(String)	...	Requested code
`data_id`	(Number)	...	ID of the referenced dataset
`type`	(String)	...	Static or dynamic code returned

vizSuggestions (*worker → interface*)

List of recommended visualizations and the respective scoring.

Parameters:

`data_id`	(Number)	...	ID of the referenced dataset
`omitted`	(Array)	...	Columns omitted from the suggestion; single valued dimensions
`sugg`	(Array)	...	List of suggestions

Workflow

execWorkflow (*interface → worker*)

Execute the given workflow.

Parameters:

`wfType`	(String)	...	Type of the transmitted workflow: "workflow" || "pwf"
`workflow`	(Object)	...	Workflow to be executed; JSON-encoded

getWorkflow (*interface → worker*)

Get a serialized representation of the workflow leading to the given dataset.

Parameters:

`data_id`	(Number)	...	ID of the result dataset
`format`	(String)	...	Format of the workflow
`[includeStyles]`	(Boolean)	...	Include styling information for visualizations?

workflow (*worker → interface*)

Document representing the workflow.

Parameters:

`data_id`	(Number)	...	ID of the dataset to this workflow
`format`	(String)	...	Format of the workflow
`workflow`	(String)	...	Requested workflow

APPENDIX

Formula Parsing Grammar

```
/*
 * Grammar to parse arithmetic formulae in Yavaa
 * adapted from https://github.com/pegjs/pegjs/blob/master/examples/arithmetics.pegjs
 *
 */

/* Term: addition, subtraction */
TERM_ADD =
  _ head:TERM_MUL _ tail:(_ operator:ADD_OPERATOR _ right:TERM_MUL _)* _
  {
    return tail.reduce(function(result, match) {
      return { value: match[1], children:[ result, match[3] ] };
    }, head );
  }

/* Term: multiplication, division */
TERM_MUL =
  _ head:FACTOR _ tail:(_ operator:MUL_OPERATOR _ right:FACTOR _)* _
  {
    return tail.reduce(function(result, match) {
      return { value: match[1], children:[ result, match[3] ] };
    }, head );
  }

/* factor in TERM_MUL */
FACTOR
  = _ "(" _ expr:TERM_ADD _ ")" _
  { return expr; }
  / _ operand:OPERAND _
  { return operand; }

/* operands */
OPERAND =
```

```
    _ operand:(NUMBER / VALUE) _
35    { return { value: operand }; }

  /* numbers */
38  NUMBER =
    _ number: $([0-9]+([,.][0-9]+)?) _
    { return number; }
41
  /* values */
  VALUE =
44    _ value: ("value" / 'col0' / $('col'[1-9][0-9]*) ) _
    { return value; }

47  /* operators */
  OPERATOR =
    ADD_OPERATOR / MUL_OPERATOR
50  ADD_OPERATOR =
    _ operator:("+"/'-') _
      { return operator; }
53  MUL_OPERATOR =
    _ operator:("*"/'/') _
      { return operator; }
56
  /* whitespaces */
  _ =
59    ( "␣" / "\\t" / "\\n" / "\\r" )*
      { return; }
```

Listing E.1: PEG grammar used to parse user defined functions.

APPENDIX F

SUPPORTED VISUALIZATIONS

Table F.1 shows an overview for the visualizations implemented in the Yavaa prototype. Similarly, Table F.2 provides a list of implemented layouts. Given are the name, a list of column-binding descriptions, and the pictogram representing the visualization within the user interface. The syntax for the descriptions roughly follows the one usually seen in regular expressions:

- Left of the arrow are bindings for dimensions, whereas right of the arrow are those for measurements.[1]
- The cross operator × connects two separate properties of a visualization.
- The data types as presented in Chapter 5 are represented using single, upper case letters: `C` (categorical), `Q` (quantitative), `T` (time), or `V` (nested visualization).
- Parentheses are used to group subpatterns.
- Alternatives are connected by a vertical line `|`.
- The multiplicity of a pattern can be modified by using `?` (zero or one), `+` (one or more), or `*` (zero or more). The absence of a modifier signals a single occurrence.

As an example, consider the first description of a Line Chart: $C? \times (Q|T) \longrightarrow Q$. Firstly, `C?` describes an optional categorical binding, which here describes the coloring of the lines. The next component, `(Q|T)` allows for either a quantitative or a time column to be bound. Here, this represents the x-axis. While these first two components relate to dimension columns, the final part `Q` describes the measurement to be displayed. In the example, this will map to the y-coordinate of a line segment.

[1] For Parallel Coordinates this separation actually does not carry over to the visualization itself but is needed to conform to the structure for other visualization types.

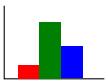
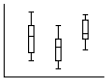
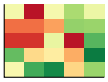
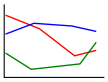
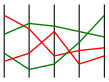
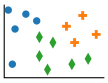
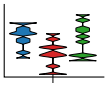
Name	Definitions	Pictogram
Area Chart	$(Q\|T) \longrightarrow Q$	
Bar Chart	$C? \times C \longrightarrow Q$ $C? \times T \longrightarrow Q$	
Box Plot	$C \longrightarrow Q$	
Bubble Chart	$C \longrightarrow Q$	
Heatmap (categorical)	$C \times C \longrightarrow Q$	
Line Chart	$C \times (Q\|T) \longrightarrow Q$ $(Q\|T) \longrightarrow Q+$	
Parallel Coordinates	$C? \times (C\|Q\|T)* \longrightarrow (C\|Q\|T)*$	
Scatter Plot	$Q \times Q \longrightarrow C?$	
Sunburst	$C+ \longrightarrow Q$	
Violin Plot	$C \longrightarrow Q \times Q$	

Table F.1: Yavaa: List of supported visualizations.

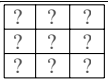
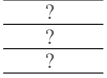
Name	Definitions	Pictogram
Matrix Layout	$C \times C \longrightarrow V$	
Row Layout	$C \longrightarrow V$	

Table F.2: Yavaa: List of supported layouts.

APPENDIX G

USER EVALUATION

The following screenshots document the user evaluation as conducted. For a description of the reasoning kindly refer to Section 14.2. The order given here is not necessarily the order presented in the actual survey as this was randomized between participants. The optional hints in Figure G.6 and G.12 only became visible upon the user stating to be unable to find the required datasets (cf. first question in Figure G.5 and G.11 respectively).

Figure G.1: User evaluation: Welcome page.

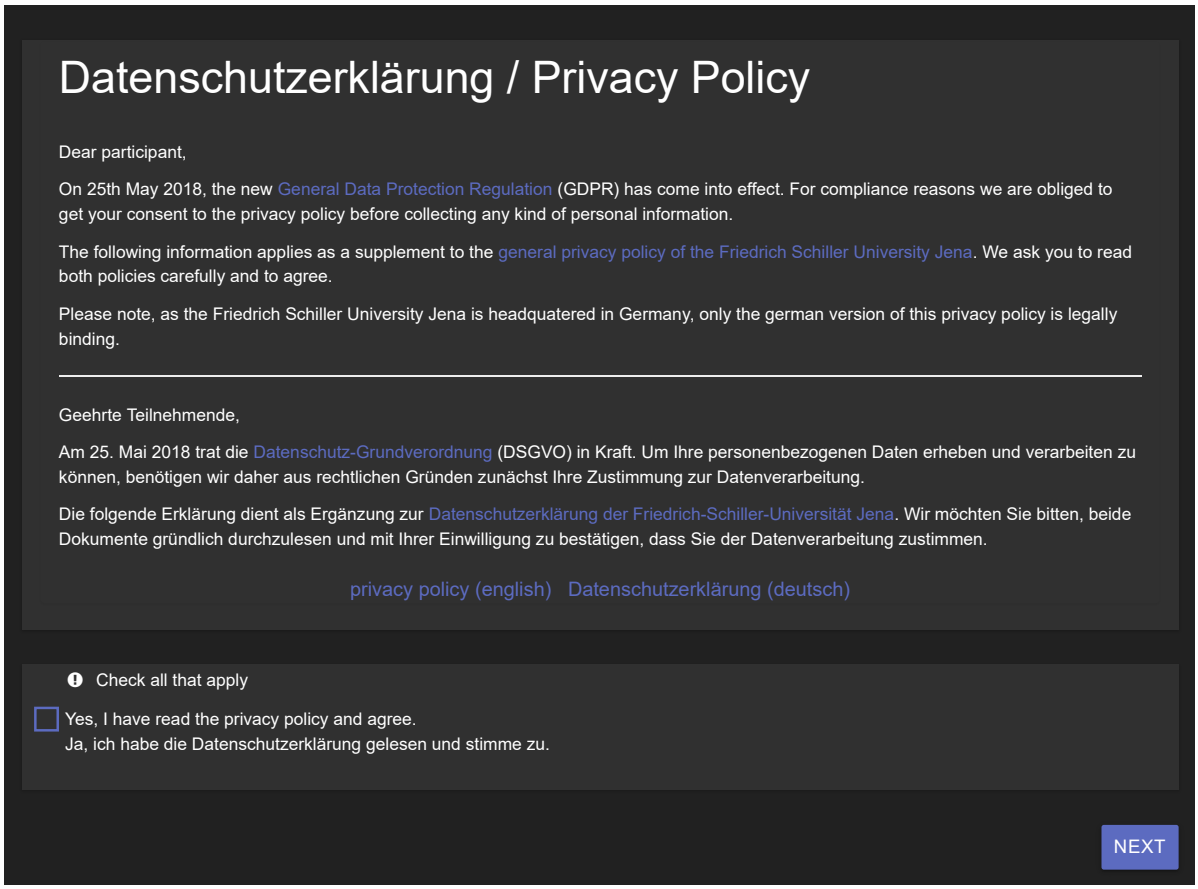


Figure G.2: User evaluation: privacy statement (detailed description omitted).

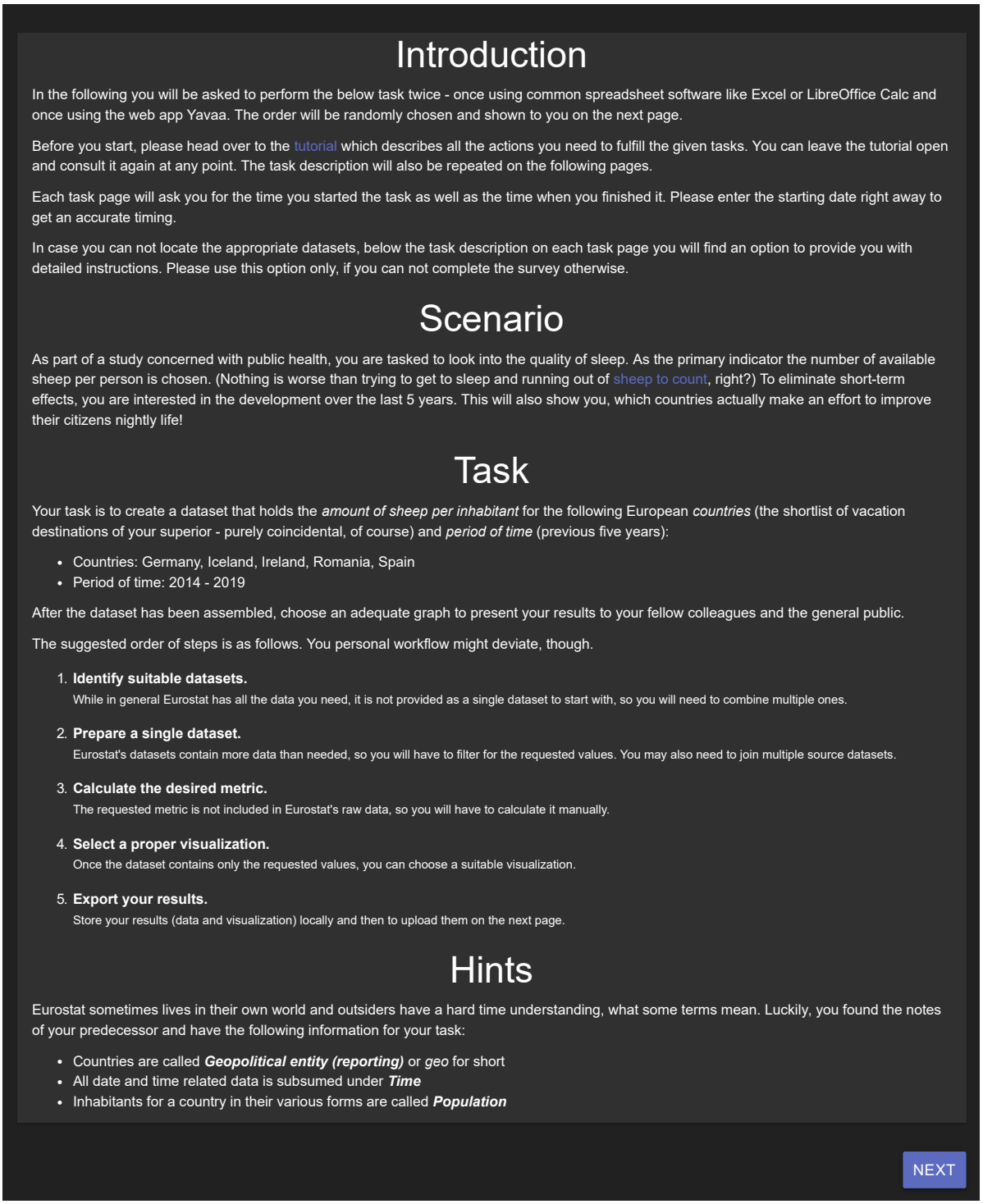


Figure G.3: User evaluation: Scenario and task introduction.

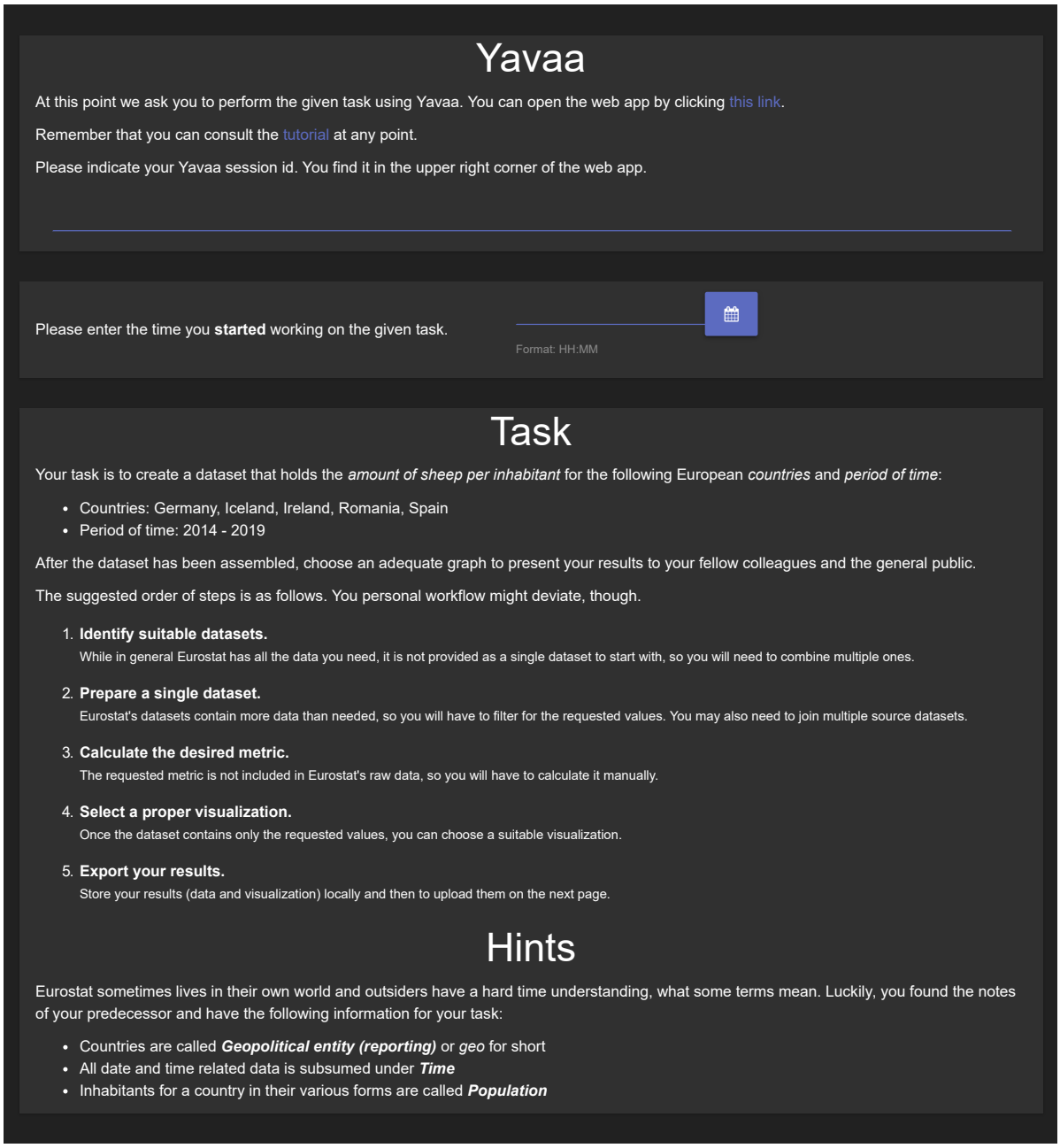


Figure G.4: User evaluation: Task description (Yavaa / part 1).

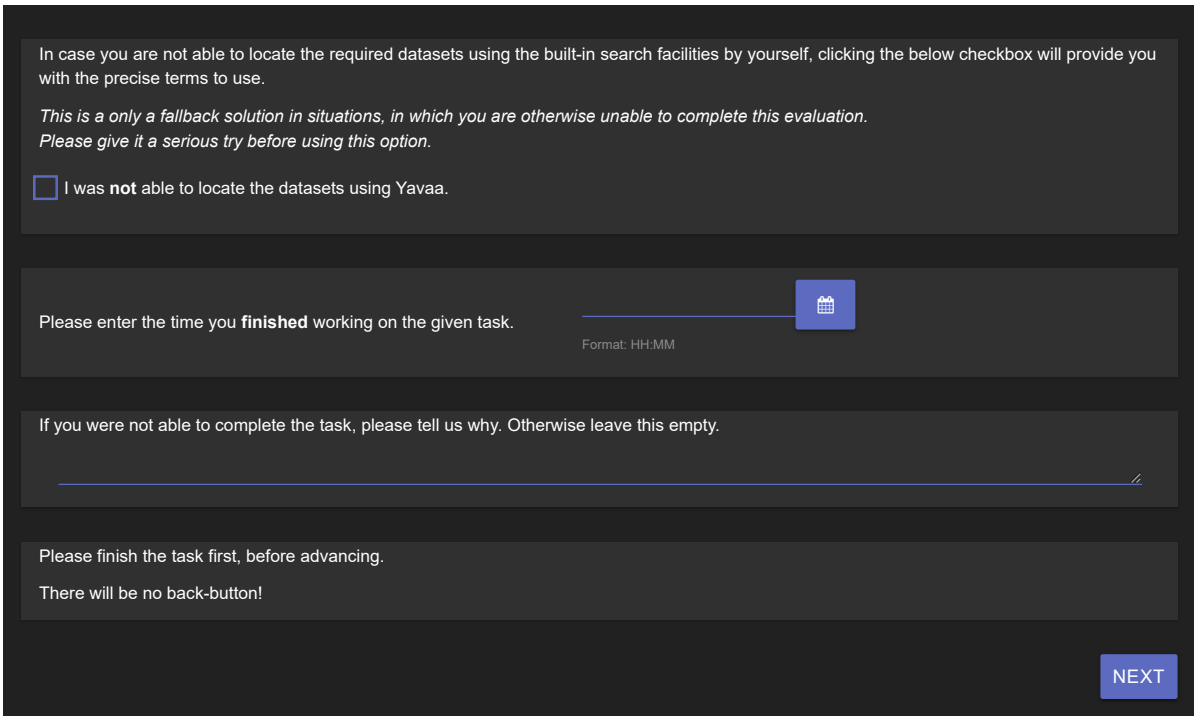


Figure G.5: User evaluation: Task description (Yavaa / part 2).

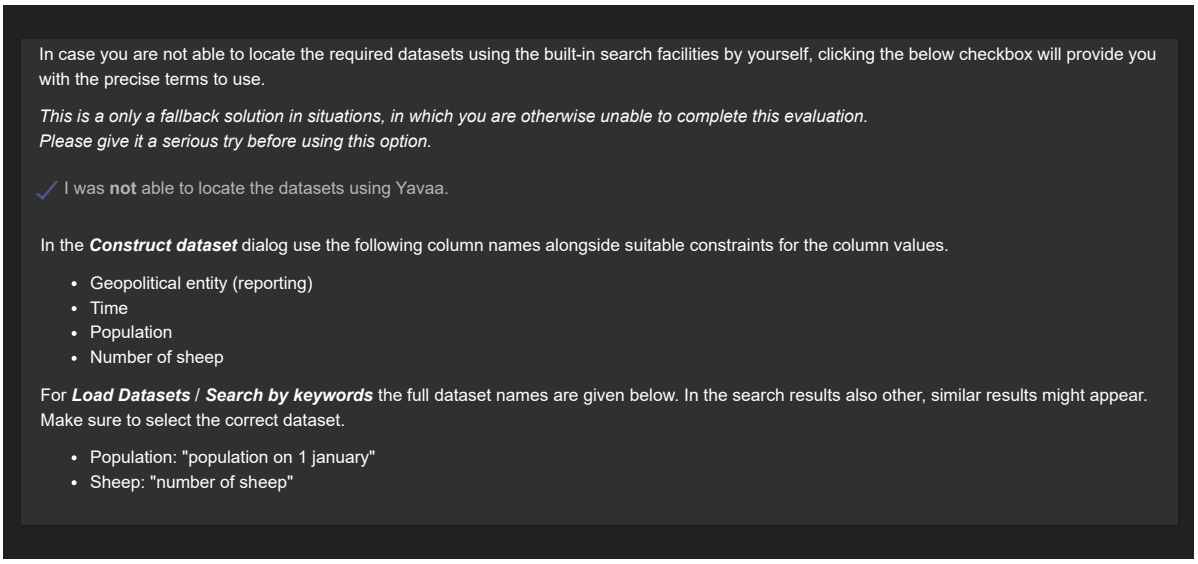


Figure G.6: User evaluation: Task description (Yavaa / optional hint).

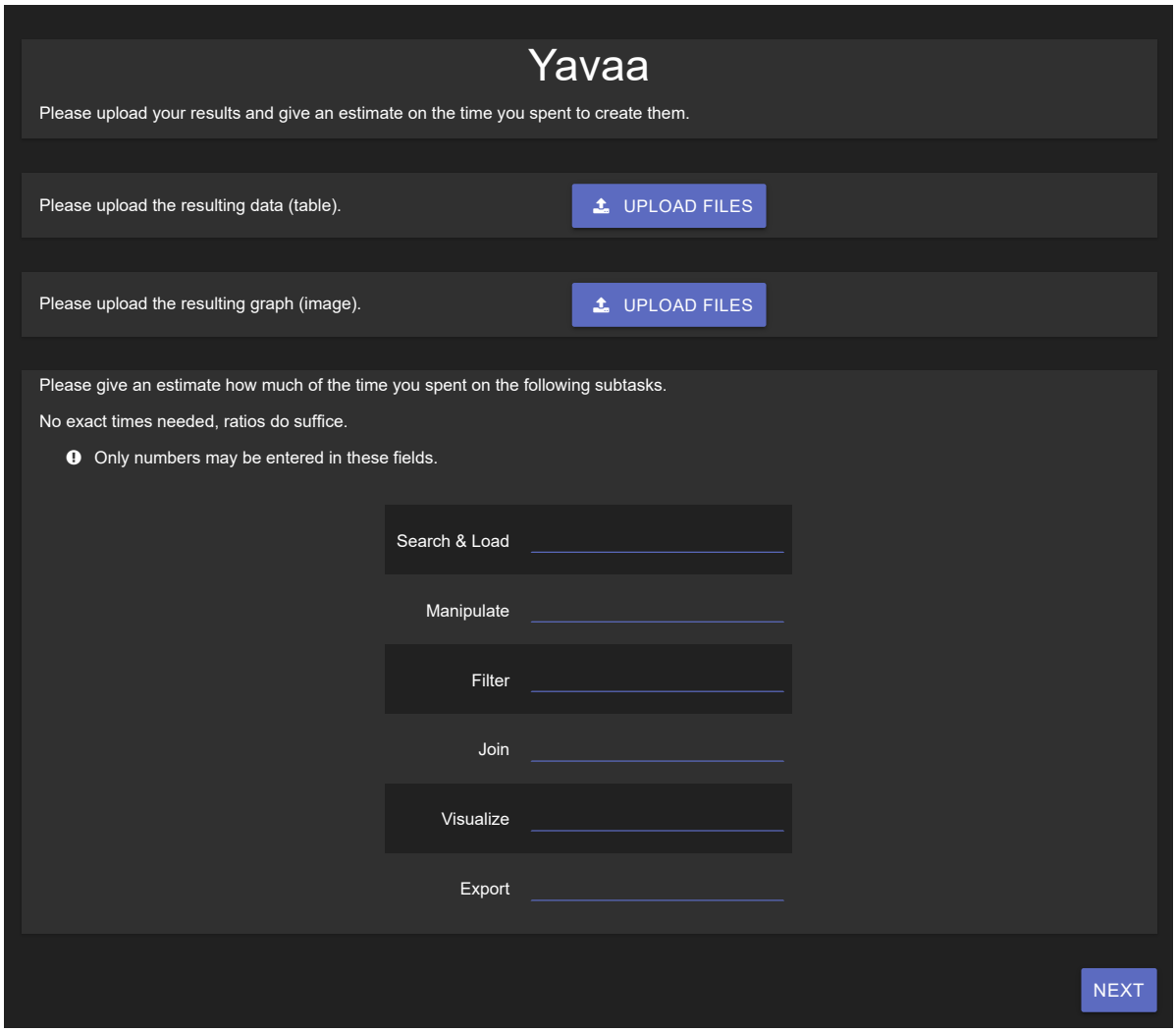


Figure G.7: User evaluation: Artifacts and time distribution (Yavaa).

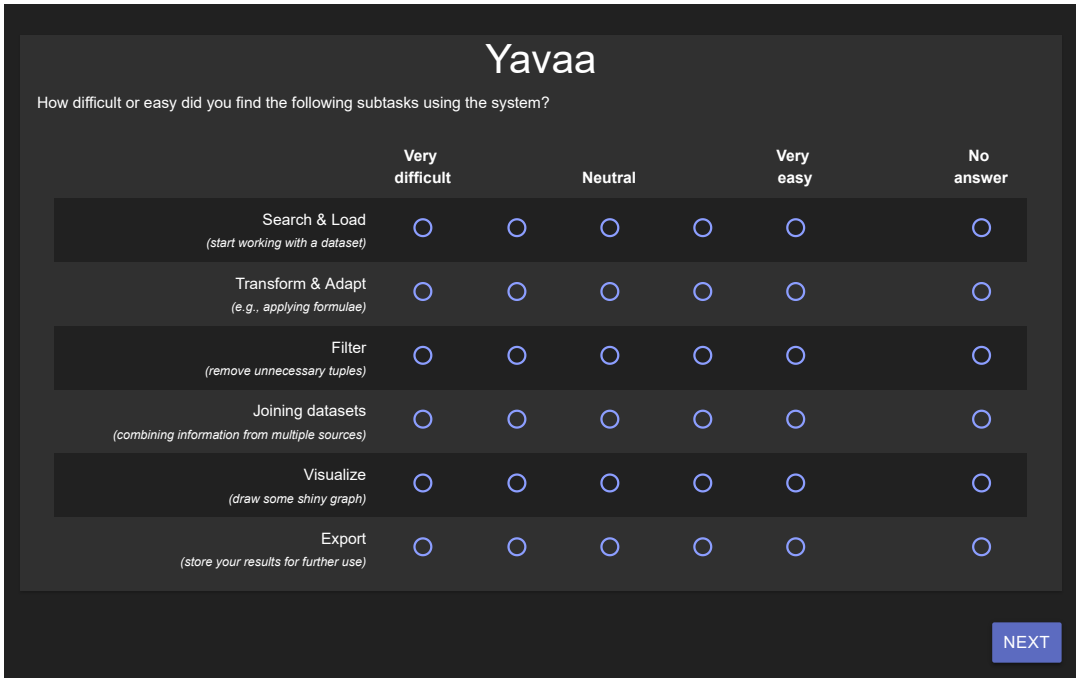


Figure G.8: User evaluation: Difficulty assessment (Yavaa).

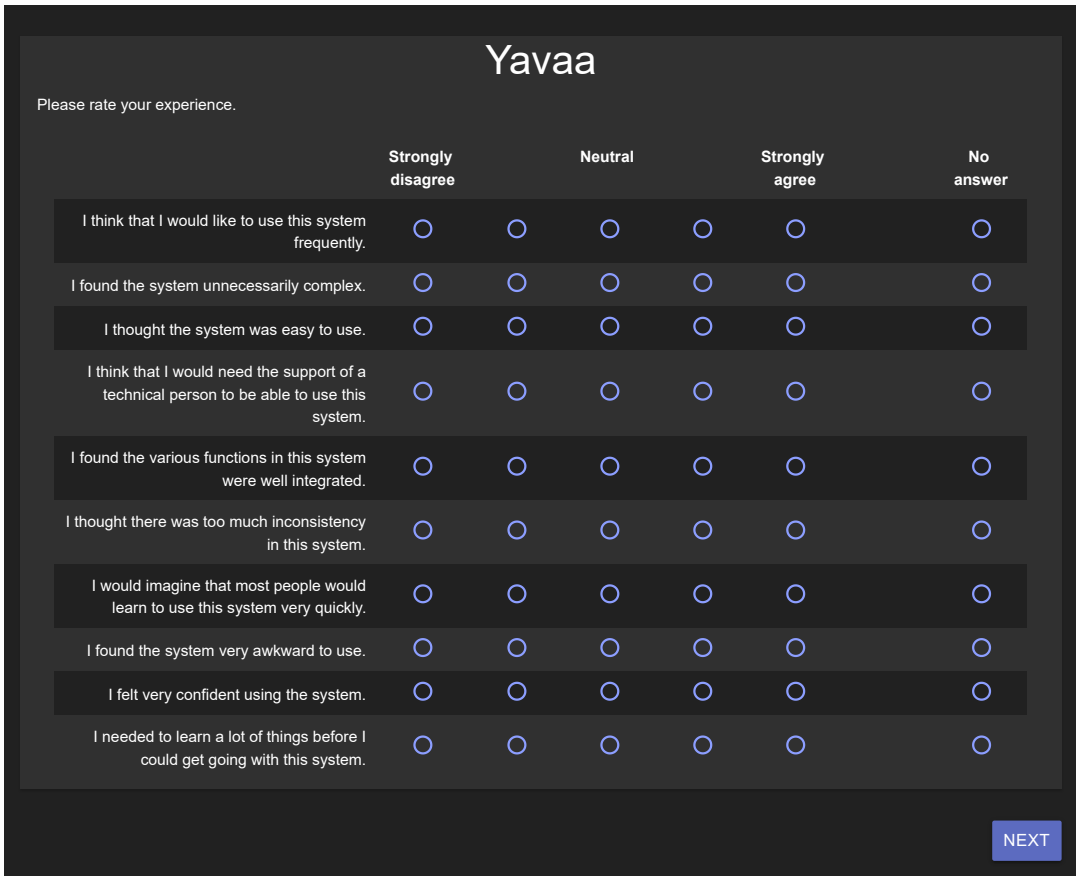


Figure G.9: User evaluation: Usability assessment (Yavaa).

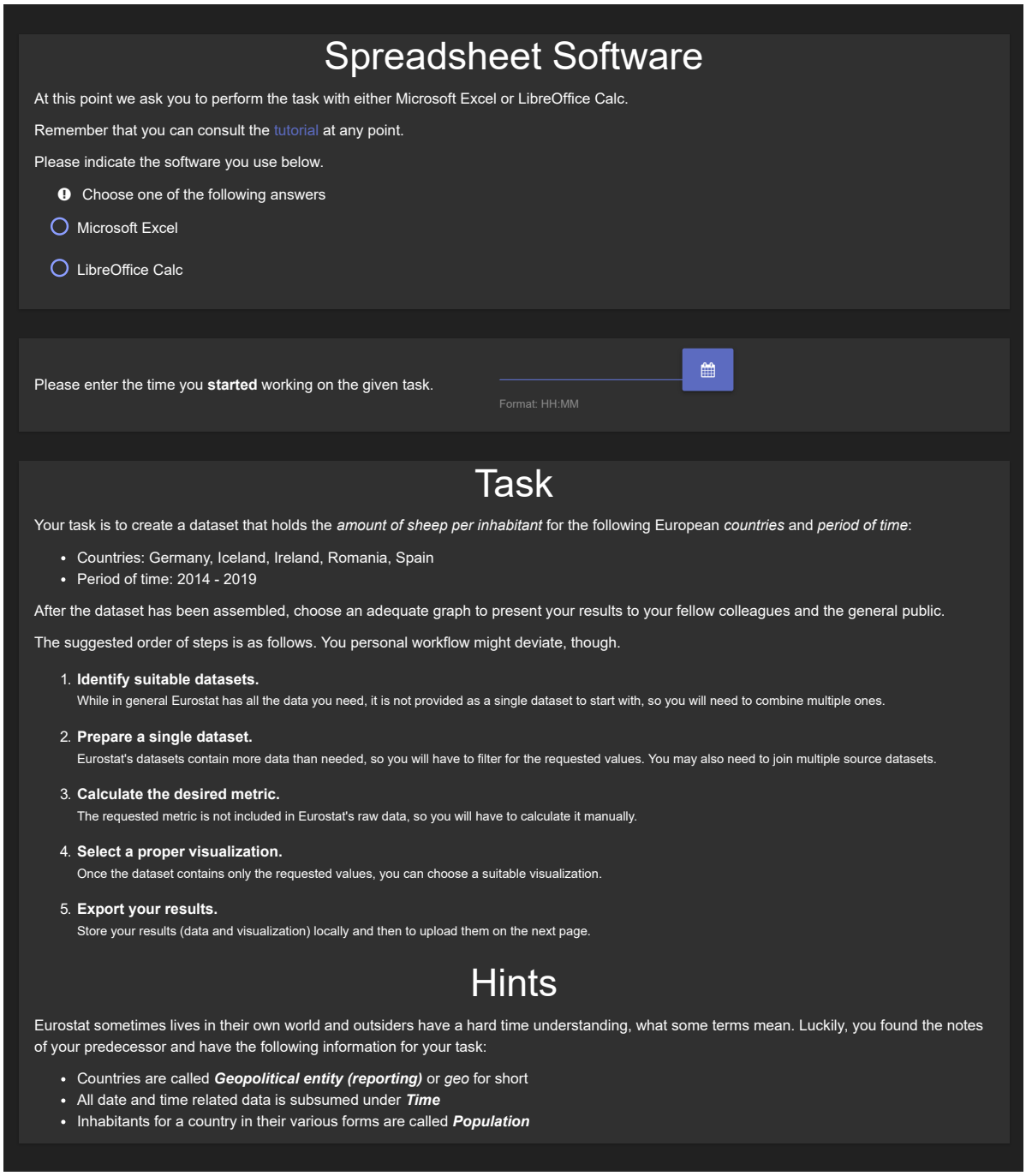


Figure G.10: User evaluation: Task description (Spreadsheet / part 1).

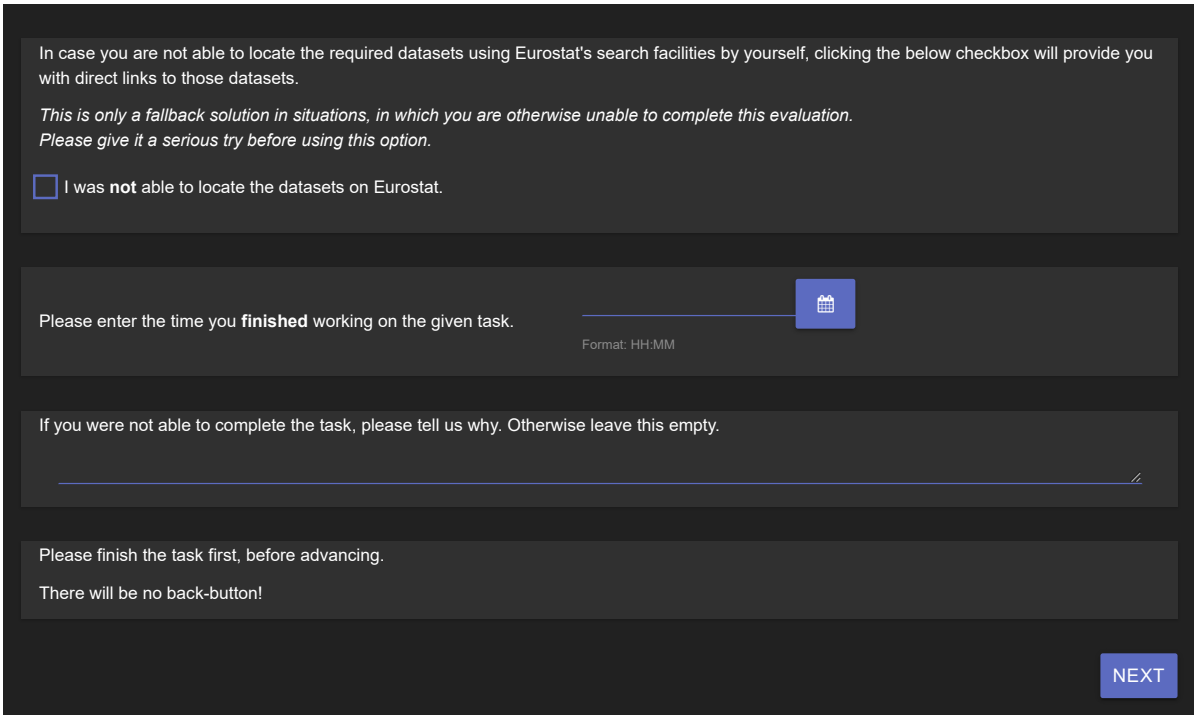


Figure G.11: User evaluation: Task description (Spreadsheet / part 2).

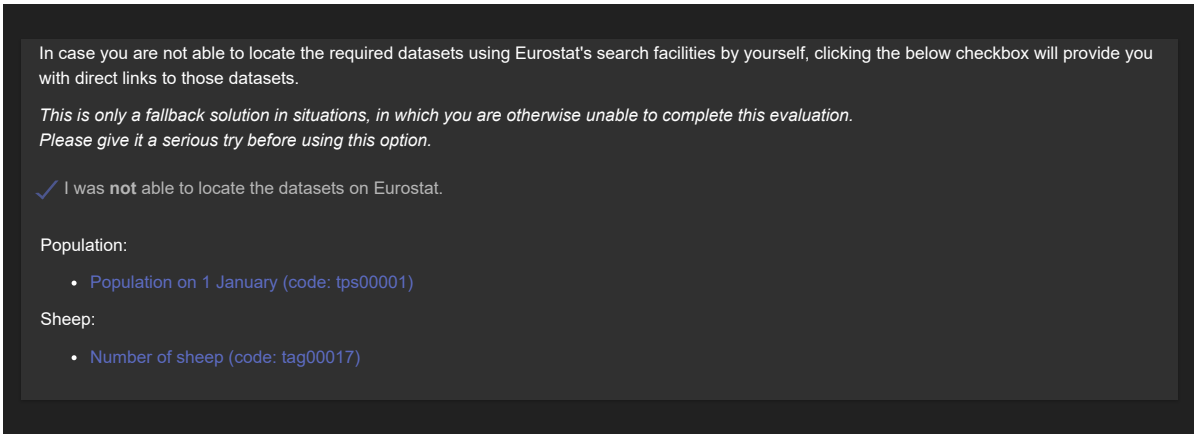


Figure G.12: User evaluation: Task description (Spreadsheet / optional hint).

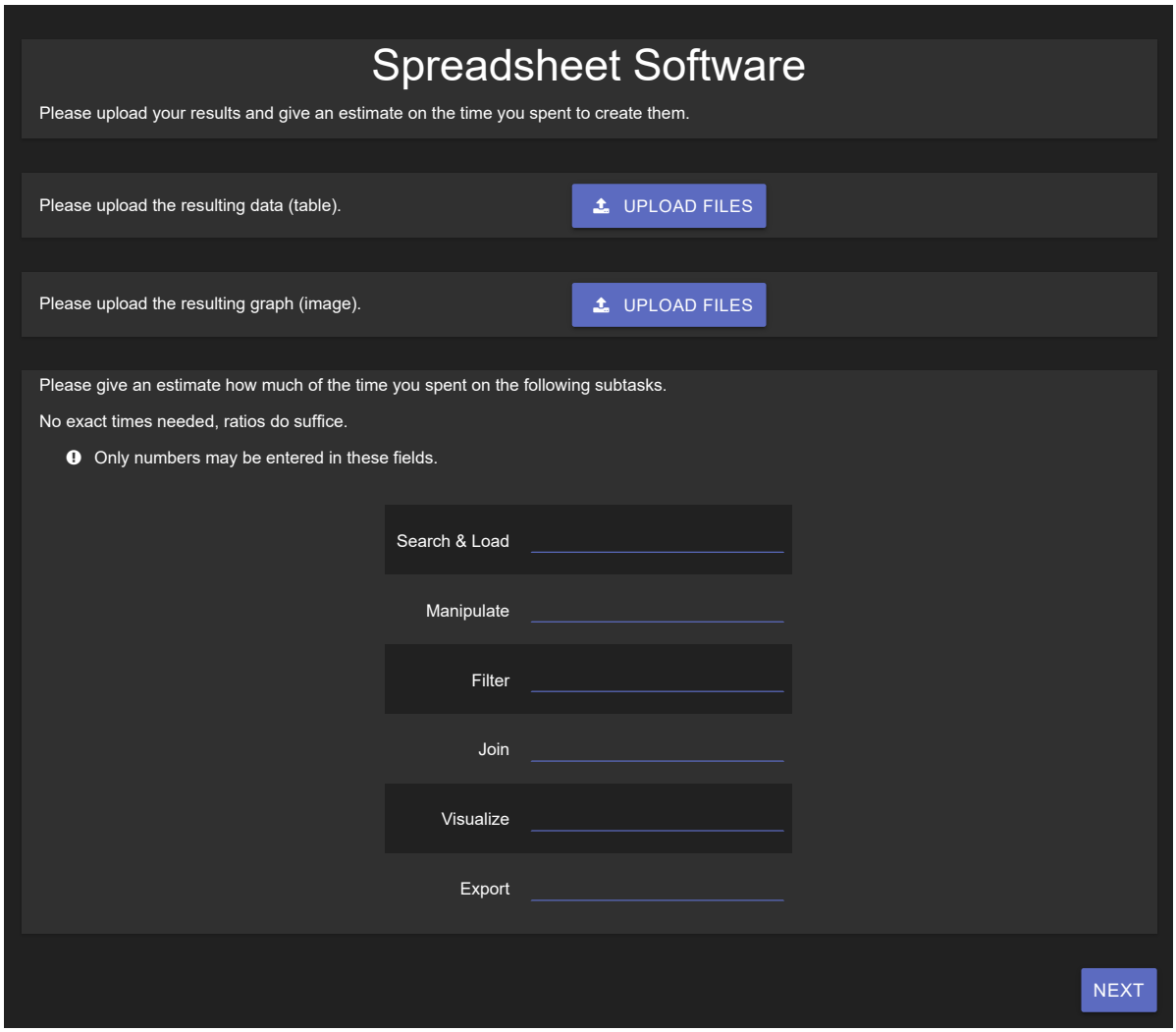


Figure G.13: User evaluation: Artifacts and time distribution (Spreadsheet).

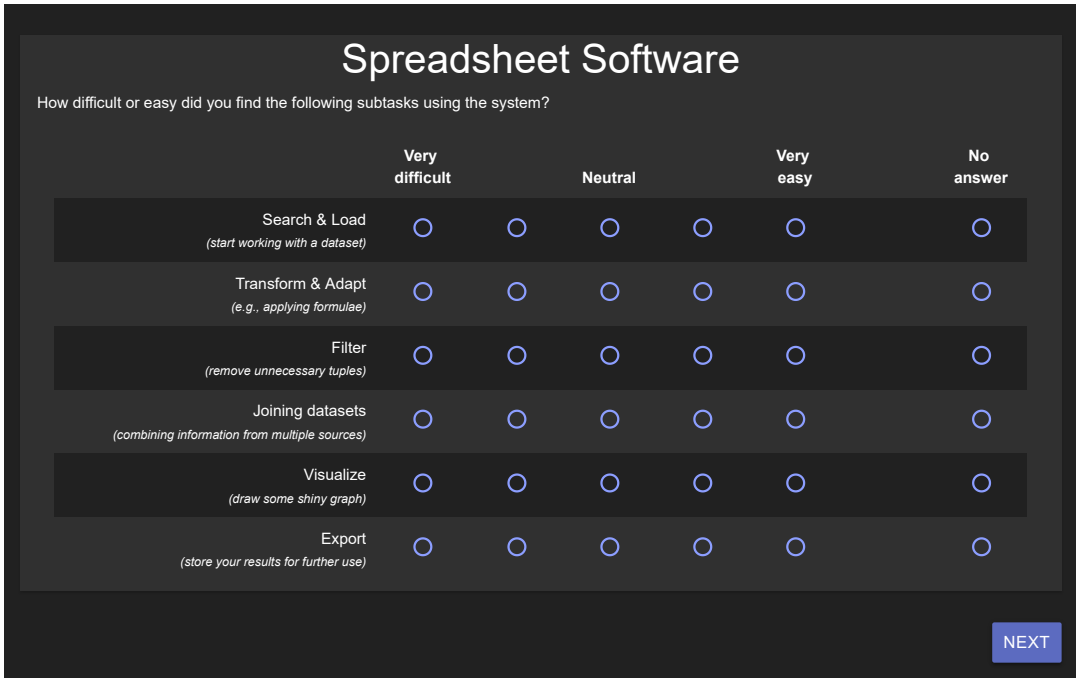


Figure G.14: User evaluation: Difficulty assessment (Spreadsheet).

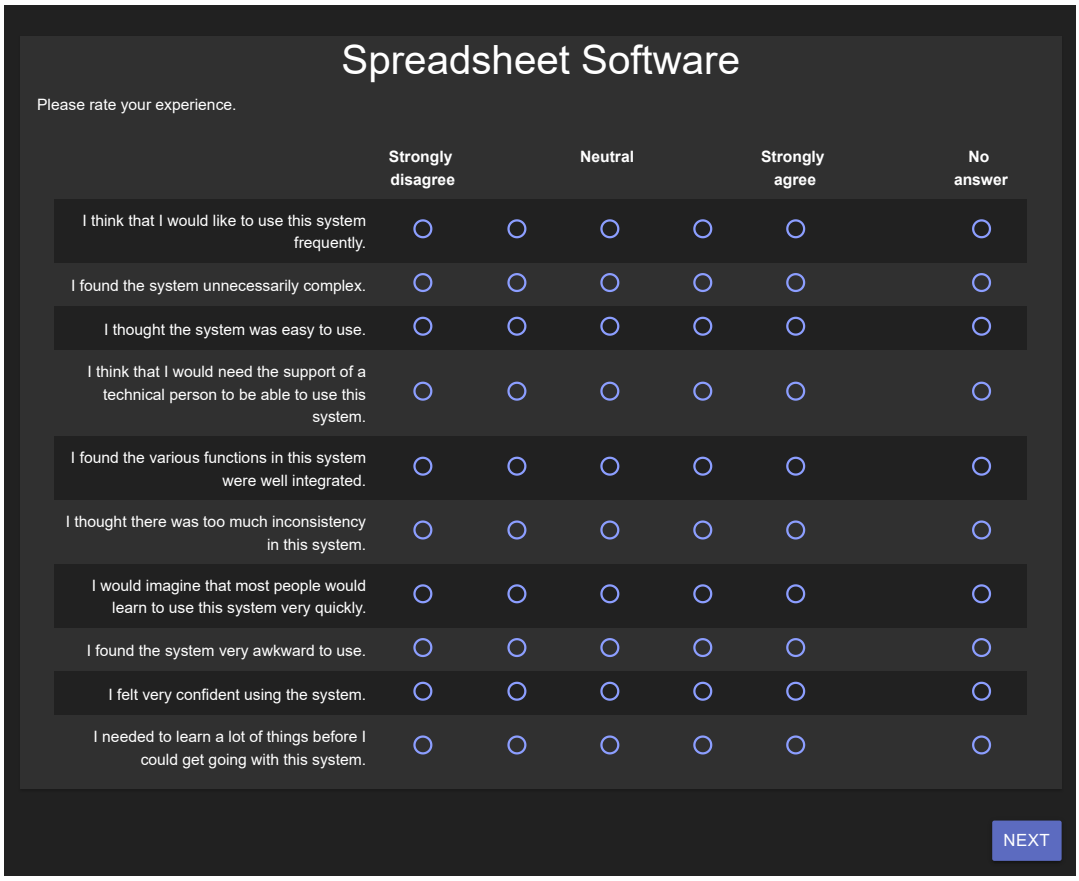


Figure G.15: User evaluation: Usability assessment (Spreadsheet).

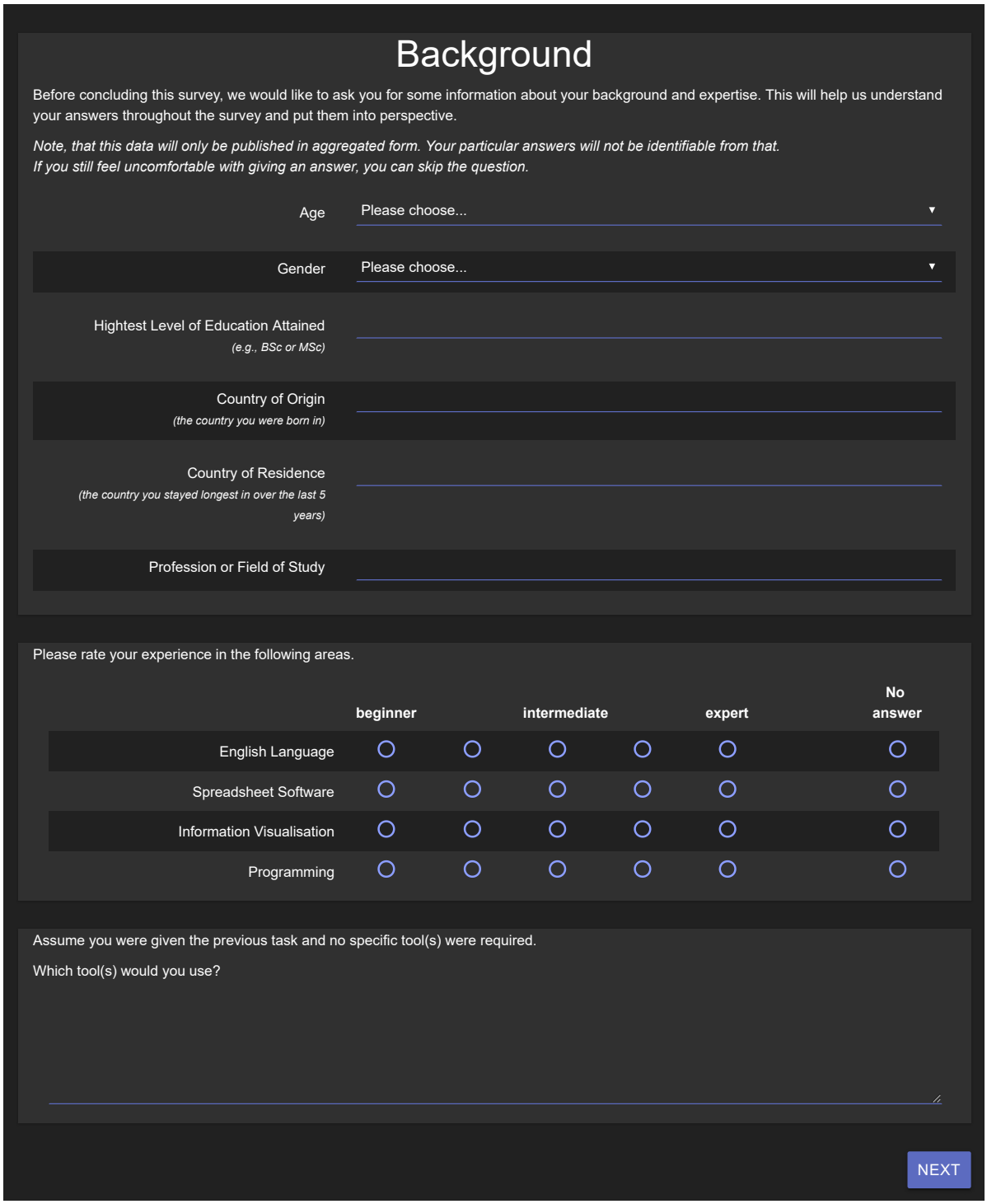


Figure G.16: User evaluation: Background information.

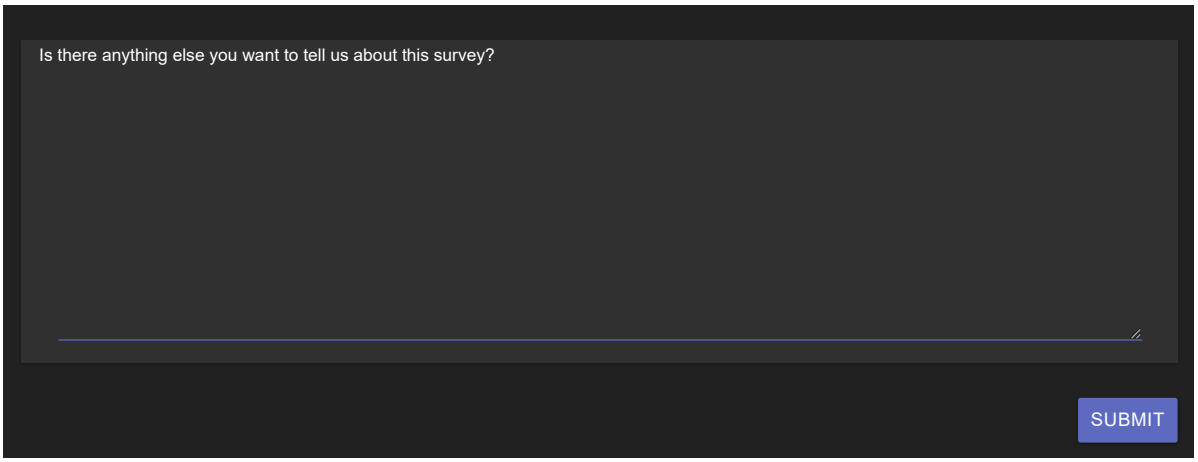


Figure G.17: User evaluation: Final Comments.

APPENDIX H

YAVAA USER INTERFACE

The following screenshots give an impression of the Yavaa user interface. They illustrate the anticipated strategies of the user evaluation outlined in Section 14.2 and depicted in Figure 14.3. Figure H.1 provides an overview of the order in which users of the respective strategy encounter the respective application states. As both strategies only differ in how they retrieve and filter the raw data, the later steps are shared between both. In the enhanced strategy, filtering the datasets is inherent in the query execution and, hence, the filtering steps (illustrated in Figure H.7 through H.9) can be omitted.

There is not necessarily one screenshot per user interaction necessary and vice versa. The following figures are merely intended to illustrate important steps and the respective interface presented to users.

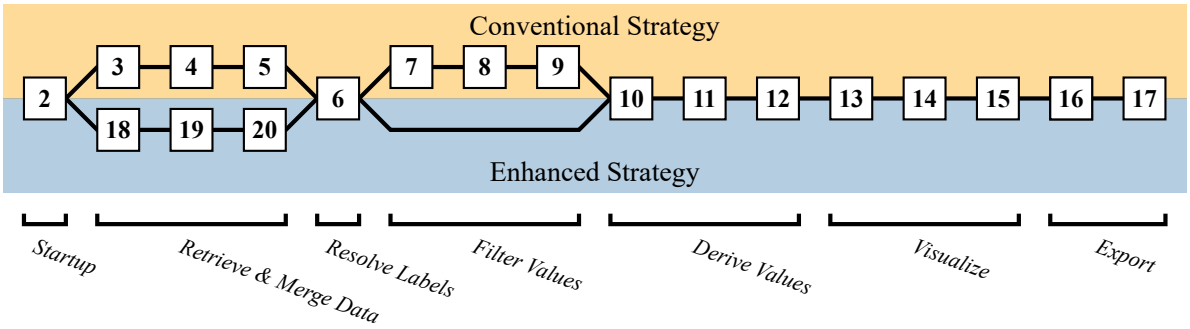


Figure H.1: Yavaa user interface: Workflow per strategy.
Numbers referring to the respective figures of this section.

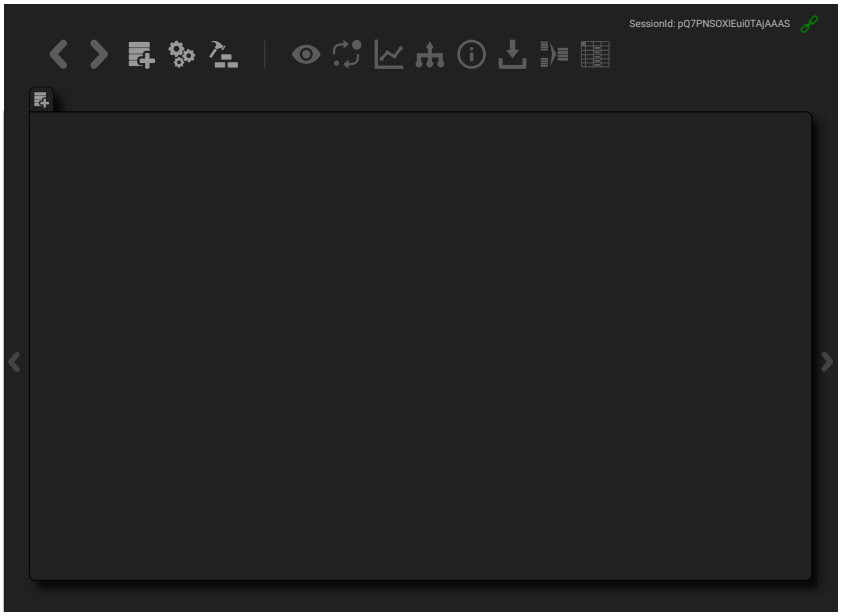


Figure H.2: Yavaa user interface: Landing page.

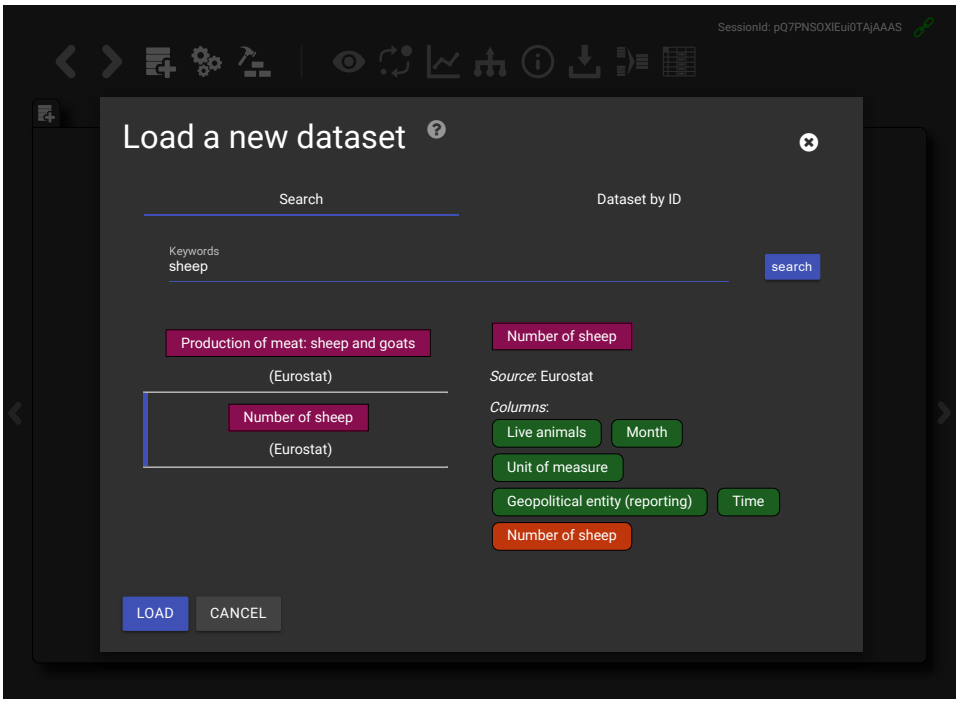


Figure H.3: Yavaa user interface: Keyword based search.

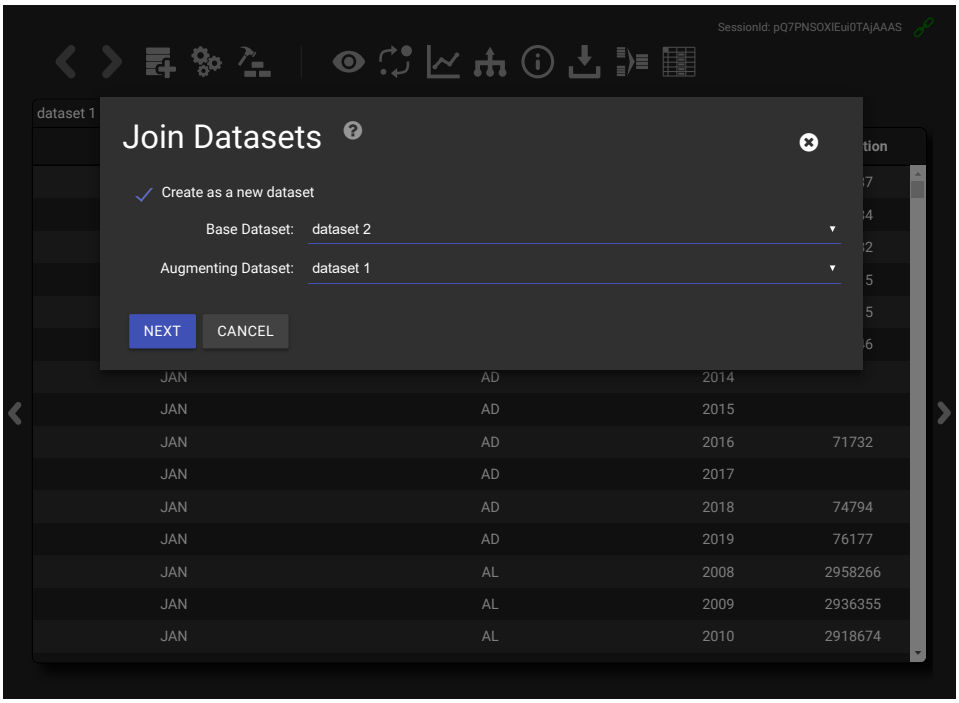


Figure H.4: Yavaa user interface: Select datasets to join.

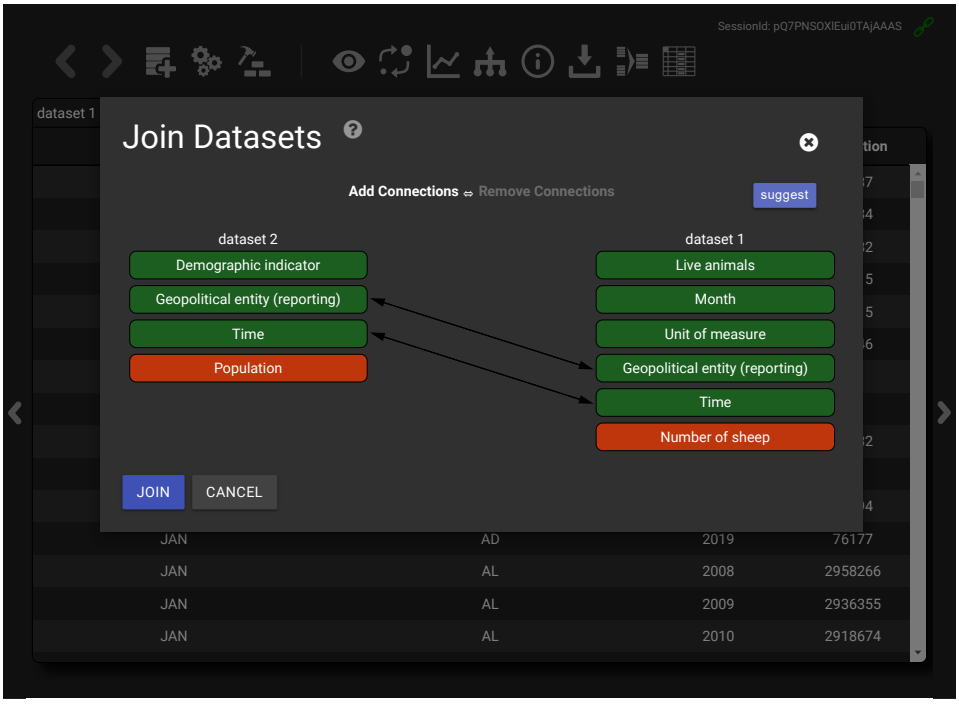


Figure H.5: Yavaa user interface: Join condition.

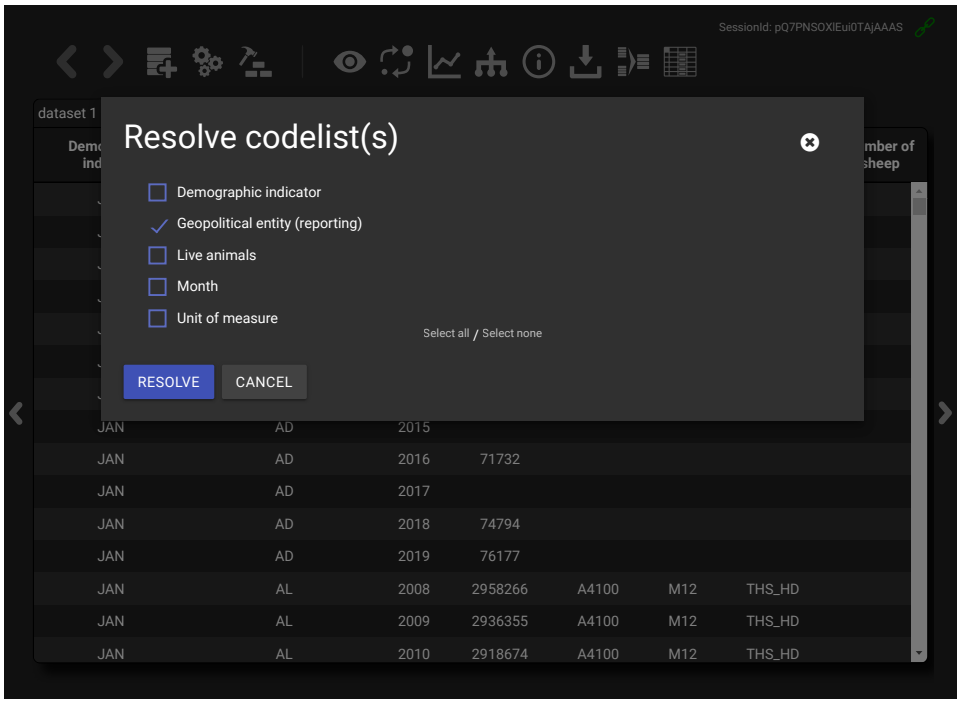


Figure H.6: Yavaa user interface: Resolve value labels.

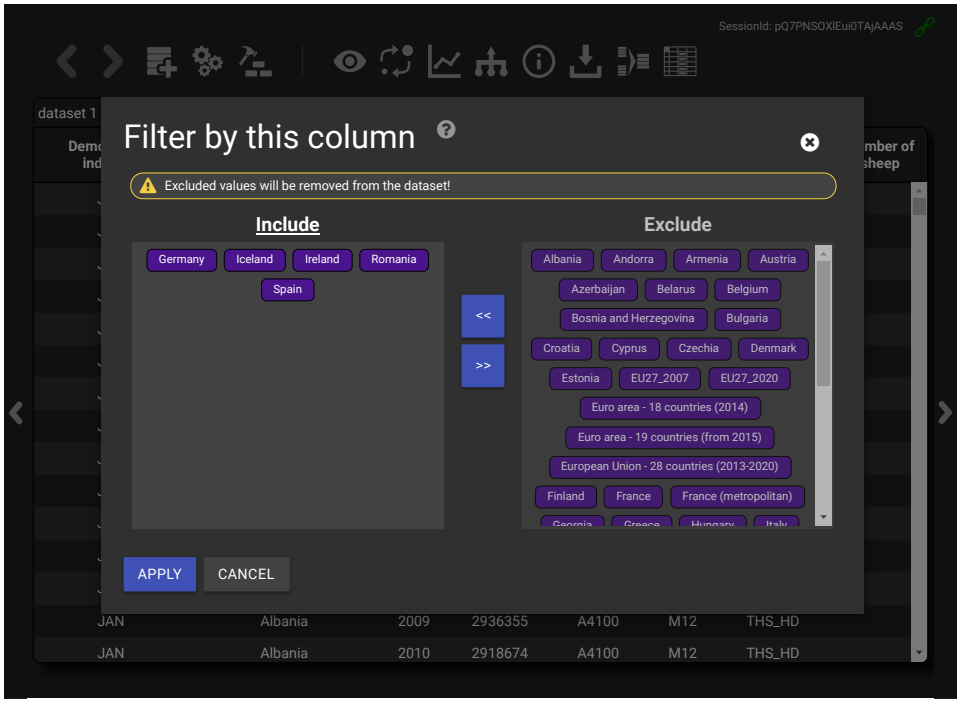


Figure H.7: Yavaa user interface: Filtering a categorical column.

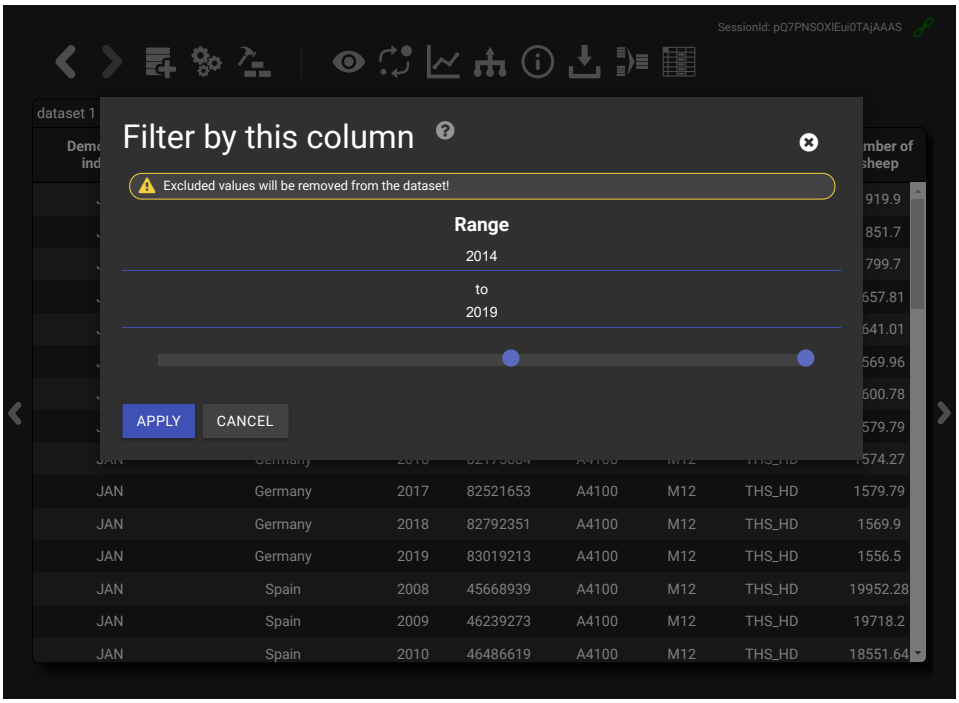


Figure H.8: Yavaa user interface: Filtering a numerical/time column.

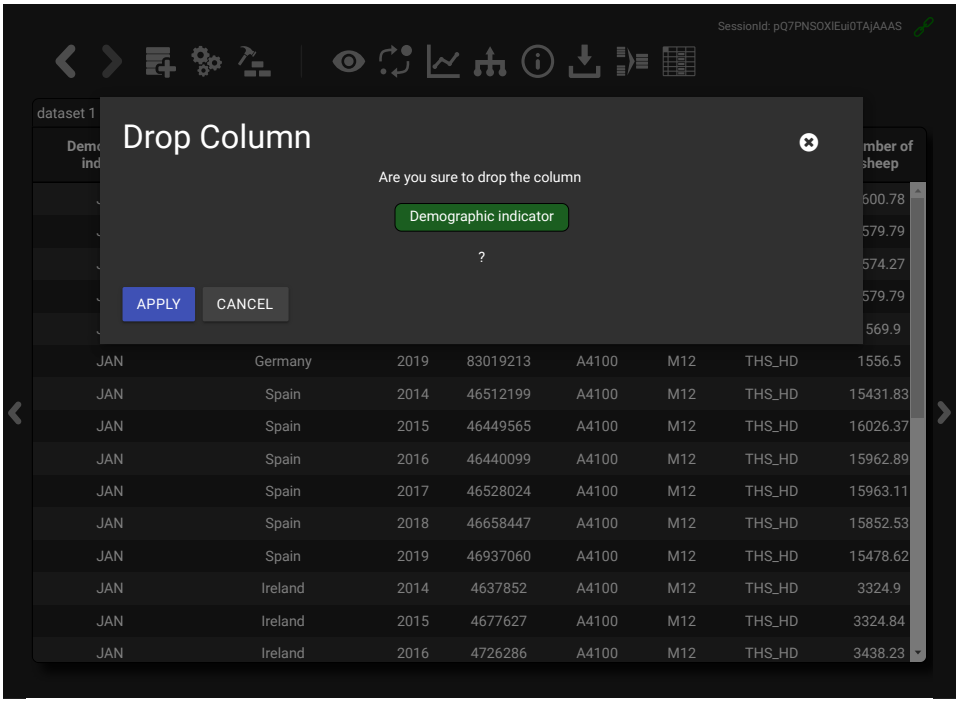


Figure H.9: Yavaa user interface: Dropping columns.

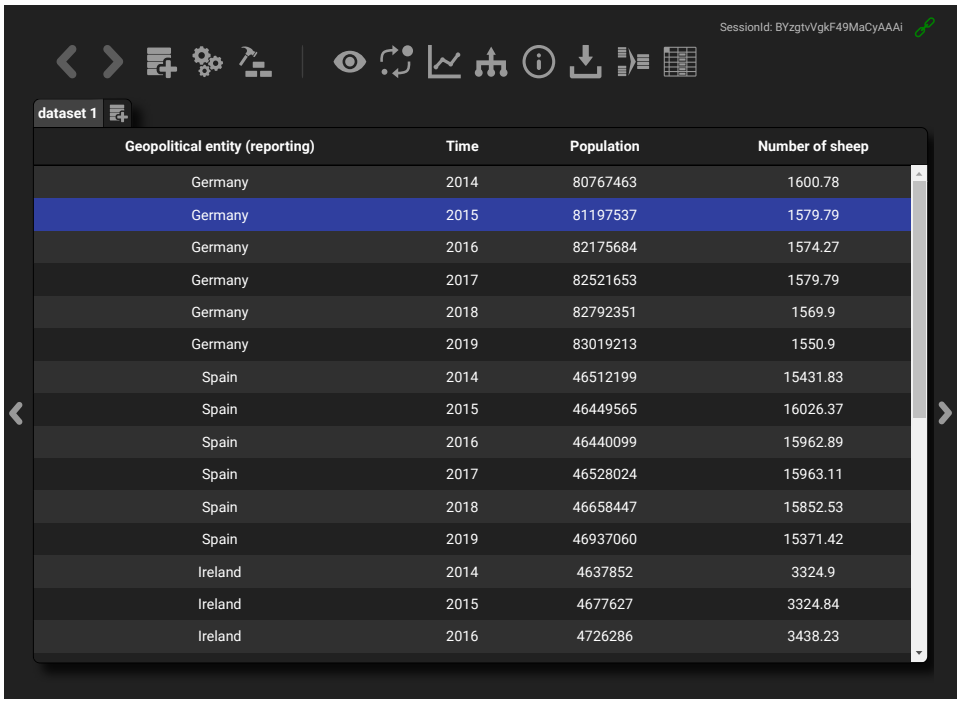


Figure H.10: Yavaa user interface: Merged and filtered dataset.

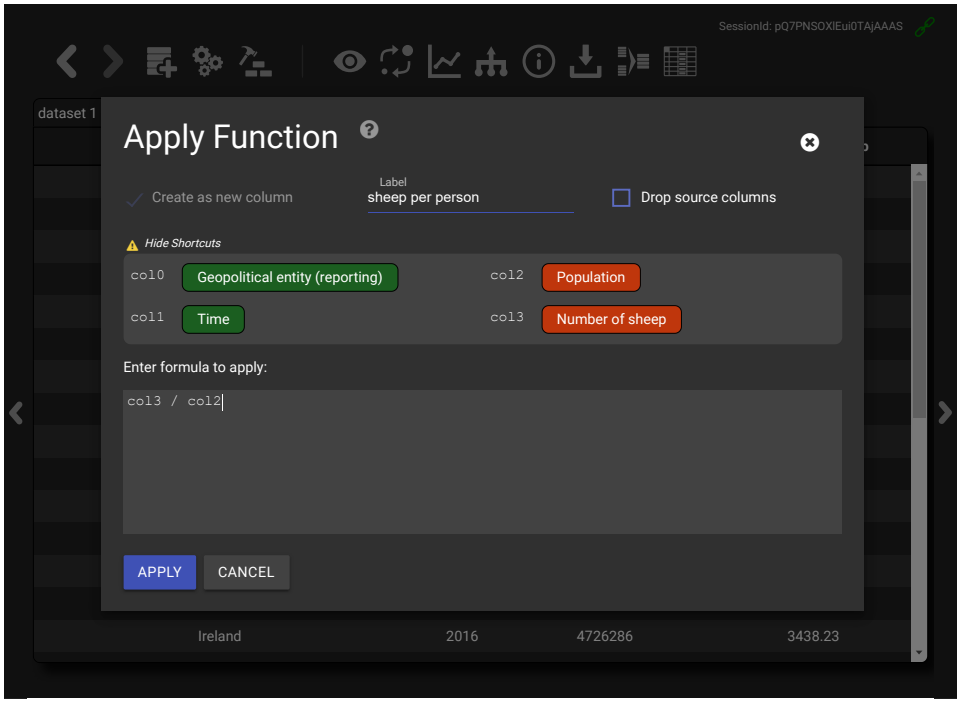


Figure H.11: Yavaa user interface: Adding derived columns.

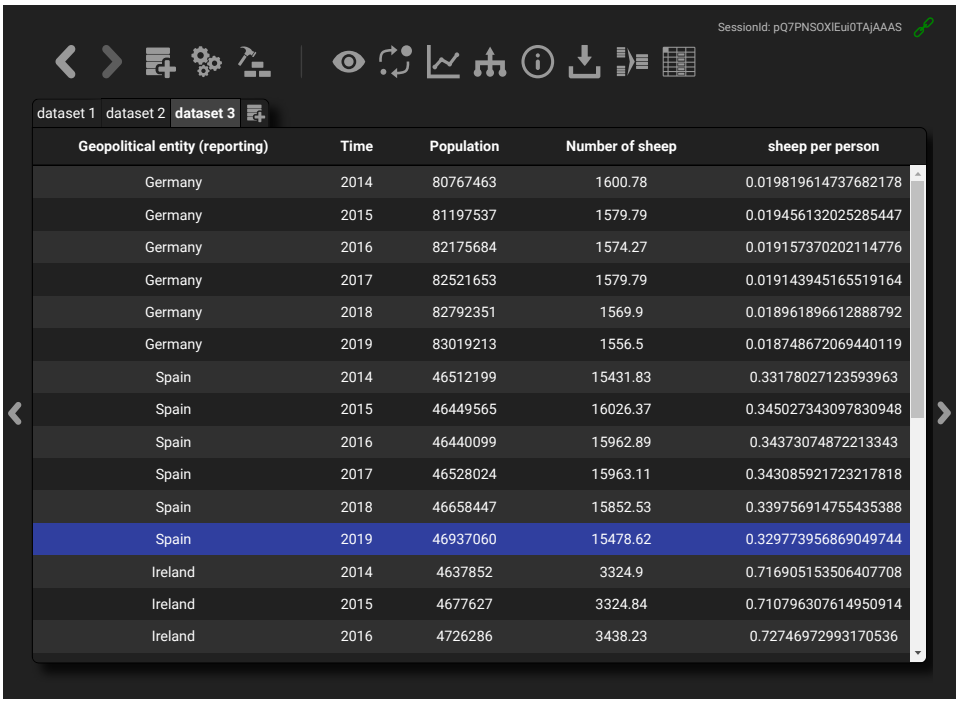


Figure H.12: Yavaa user interface: Dataset ready to be visualized.

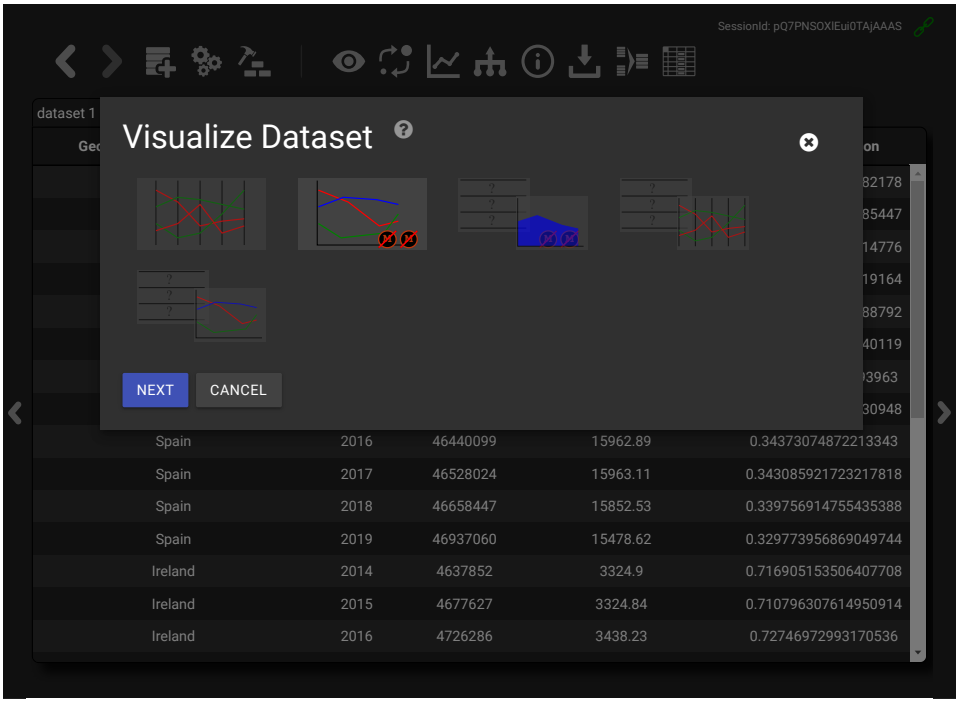


Figure H.13: Yavaa user interface: Selecting a visualization.

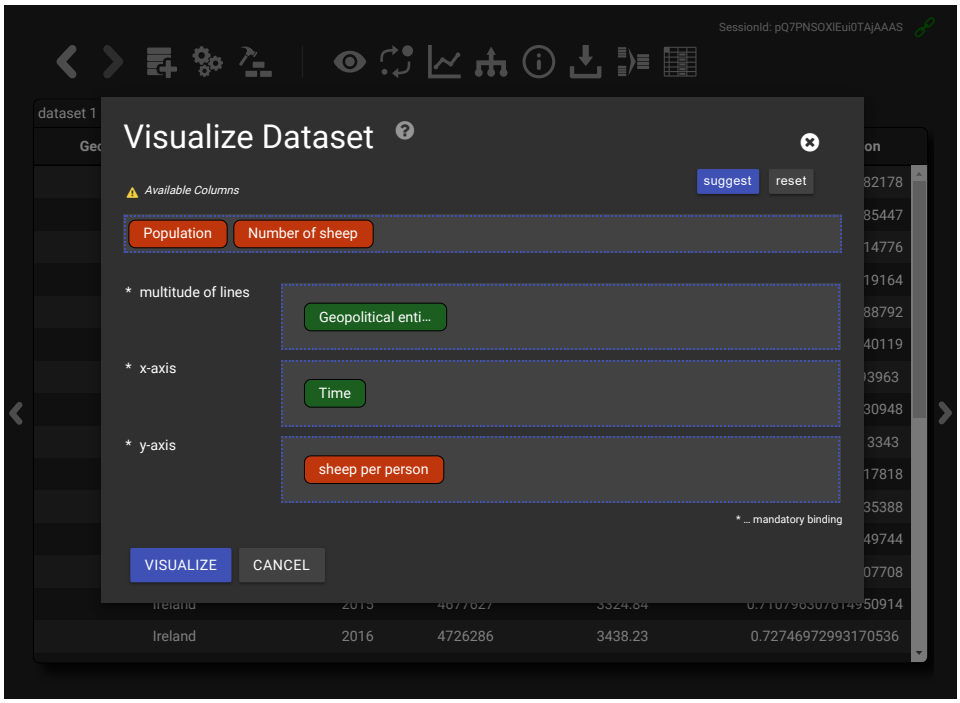


Figure H.14: Yavaa user interface: Binding columns to visual artifacts.

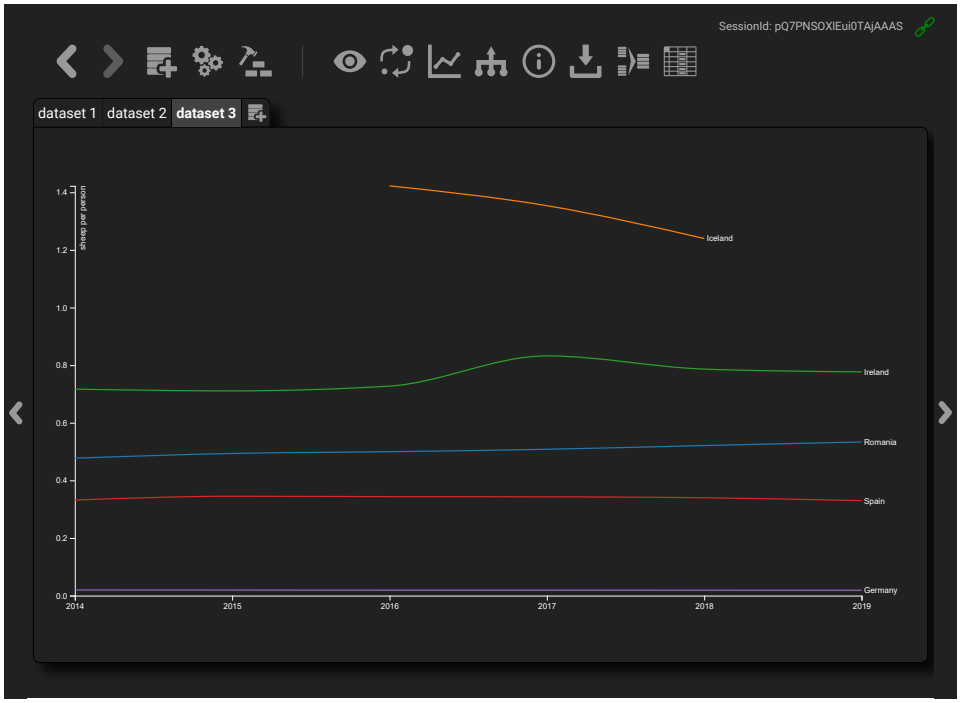


Figure H.15: Yavaa user interface: Visualized dataset.

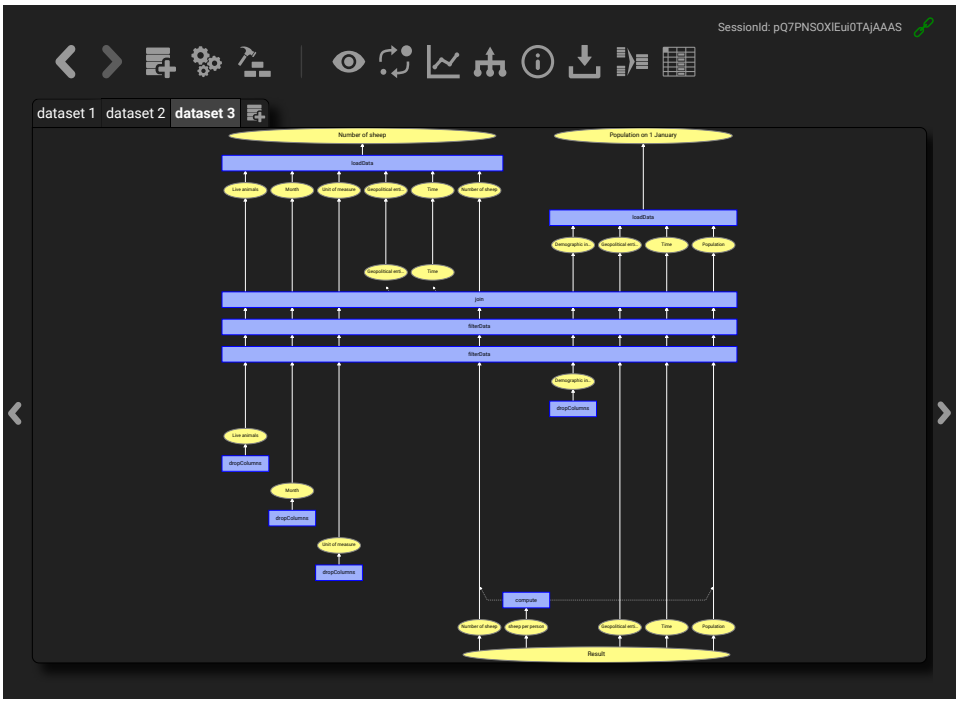


Figure H.16: Yavaa user interface: Workflow view.

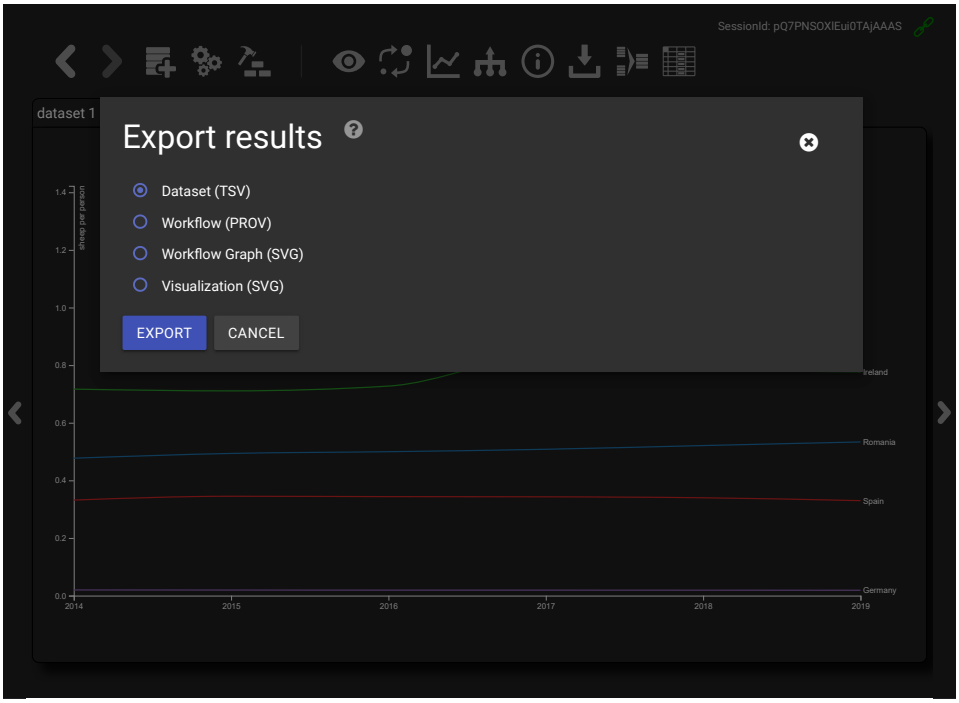


Figure H.17: Yavaa user interface: Export dialog.

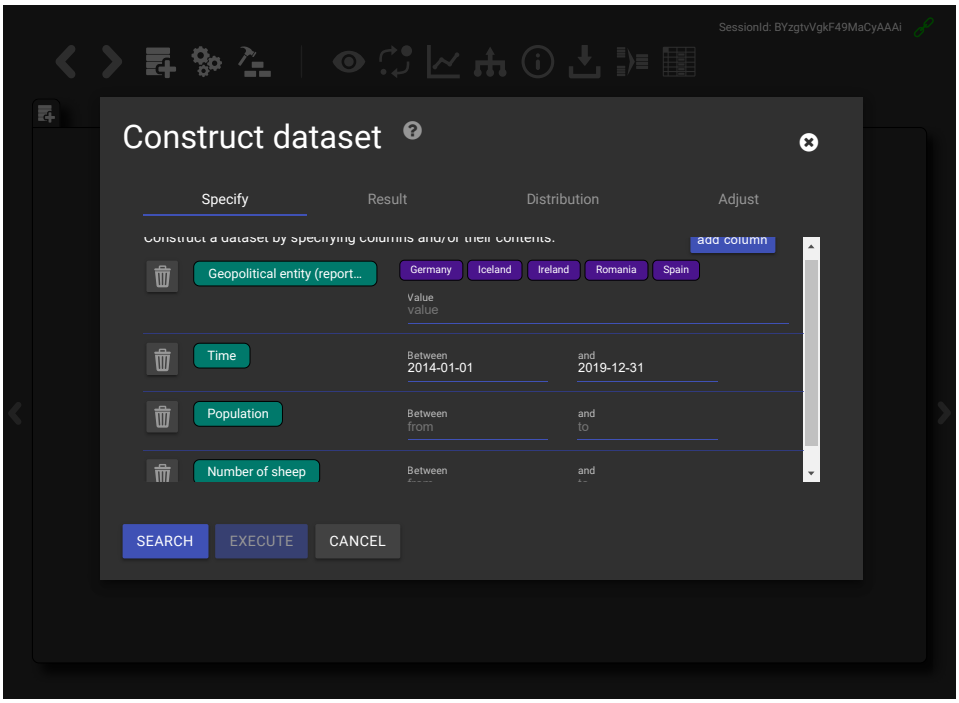


Figure H.18: Yavaa user interface: Constructing a dataset.

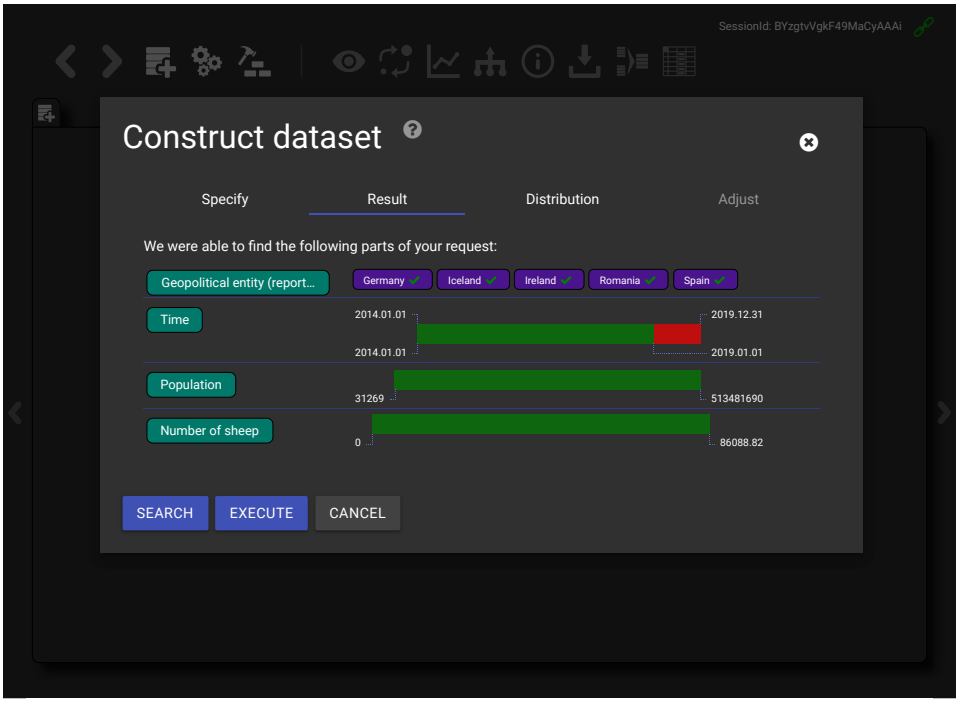


Figure H.19: Yavaa user interface: Search result for a constructed dataset.

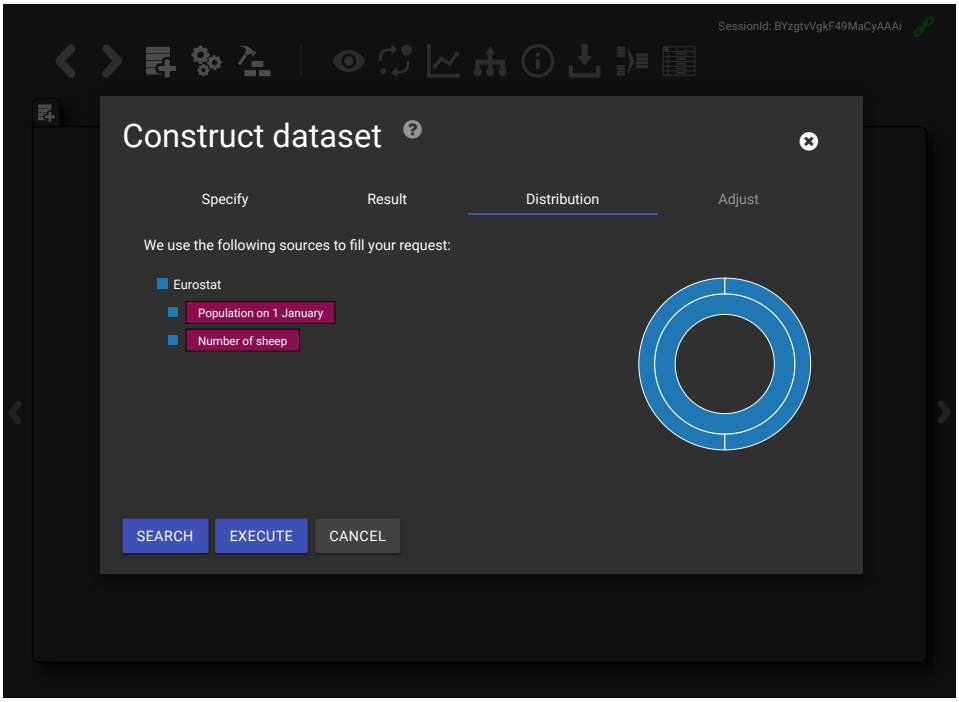


Figure H.20: Yavaa user interface: Source distribution for a constructed dataset.