Cross-Formalism Resource Discovery in Smart Environments

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 M.Eng. Kobkaew Opasjumruskit
geboren am 07.04.1983 in Bangkok, Thailand

Gutachter

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

2. Prof. Dr. Martin Welsch
   IBM Deutschland Research & Development GmbH, D-71032 Böblingen

3. Prof. Dr. Wolf Zimmermann
   Martin-Luther-Universität Halle Wittenberg, D-06120 Halle (Saale)

Tag der öffentlichen Verteidigung: 20 November 2017

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, ...

[Kobkaew Opasjumruskit]

Deutsche Zusammenfassung

Das Internet der Dinge (IoT-Internet of Things) kommt heutzutage durch die Medien immer mehr ins Gespräch. In Anbetracht der Fülle von intelligenten Sensoren, die wir derzeit nutzen, ist kontextbewusstes Computing überall verfügbar und heisst uns also willkommen in der Welt von IoT. Allerdings, wenn es Trillionen von Ressourcen gibt, wie können wir spontan die eine Ressource, die wir brauchen, bestimmen? Daher ist eine der Hauptfragen in der Forschung die Auffindung des Geräts und des Dienstes. Der grundlegende erste Schritt ist zu untersuchen, wie diese Geräte und Dienste beschrieben werden. Viele standardisierte Web-Service-Beschreibungen werden dazu verwendet, nicht nur Web-Services, sondern auch die Geräte physisch zu beschreiben. Diese Geräte sind unter dem Web-Service-Kommunikations-Layer eingekapselt, um sie im Internet verfügbar zu machen. Web-Service-Beschreibungen mit semantischen Annotationen können dazu verwendet werden, die dynamische Auffindung von Quellen zu automatisieren. Diese Technik ermöglicht die automatische Erkennung, Konfiguration und Ausführung von Ressourcen in dynamischen Umgebungen. Wir konzentrieren uns auf die Beschreibungssprache von Ressourcen, die eine semantische Annotation ermöglicht. Dennoch, es gibt keinen eindeutigen standardisierten Formalismus um Ressourcen zu beschreiben. Es ist taktisch von Vorteil mehrere Beschreibungsformalismen gleichzeitig zu behandeln.

Diese Dissertation stellt eine Technik zur Auffindung der formalismus-übergreifenden Ressource vor, die den User Kontext und Ressourcen-Kontext verwendet um die Empfehlung von Ressourcen zu verbessern. Im Gegensatz zu bestehenden Arbeiten ist unsere Technik auf nicht-IT-versierte Anwender ausgerichtet. Der Auffindungsprozess sollte nicht auf einen einzelnen Ressourcenbeschreibungsformalismus beschränkt sein. Darüber hinaus sollte der Matching-Algorithmus benutzersensibel und an die jeweilige Umgebung anpassungsfähig sein, d.h. ausgeführt je nach aktueller Situation des Benutzers, anstatt sich auf die Keyword-basierte Suche zu beschränken. Der größte Nachteil dieses traditionellen Ansatzes ist, dass er von der Kompetenz der Anwender abhängt und meistens mehrere Versuche erfordert. Daher haben wir das Ressourcen-Auffindungsmodul auf der Grundlage existierender Techniken entwickelt, welches entdeckte Ressourcen auflistet, um die Abfragen der Benutzer nach ihren Interessen, Fachwissen und aktuellen Situationen bedienen zu können. Diese Arbeit erläutert die Implementierungsdetails und zeigt die Auswertung jedes implementierten Moduls. Wir wollten beweisen, dass die Qualität der Ergebnisse, im Vergleich zu herkömmlichen Auffindungstechniken, deutlich verbessert werden kann.

Um die Verwendbarkeit der vorgeschlagenen Methode zu demonstrieren, setzen wir sie in MERCURY ein. MERCURY ist eine Plattform, die Zweierlei ermöglicht, den Unternehmen mit ihren Kunden zu interagieren und den Endbenutzern maßgeschneiderte Anwendungen zu erstellen. Es bietet eine webbasierte Schnittstelle, um eine Brücke zwischen den aufblühenden Funktionalitäten, die im IoT verfügbar sind, und den Endbenutzern zu schlagen. Im Rahmen von MERCURY benötigen die Registrierung, die Zusammenstellung und die Ausführung von Ressourcen eine automatisierte Ressourcenfindung. Da die Implementierung dieser Arbeit einen eigenständigen (stand-alone) Service darstellen soll, besteht keine Einschränkung diesen unter der Domäne von MERCURY zu verwen-

den. Somit kann das generische Problem der Ressourcenfindung in einer beliebigen IoT-Anwendung vom Ergebnis dieser Arbeit profitieren.

Acknowledgement

This thesis cannot be completed without the help and support from these people. First, I want to thank Prof. Dr. Birgitta König-Ries for supervising and supporting me not only with the thesis but also the projects I have been working on. She is a great and kind professor, friend, colleague, boss, and mother. I would like to give special thanks to Prof. Dr. Wolf Zimmermann and Prof. Dr. Martin Welsch for reviewing this thesis.

This thesis is initiated by IBM project under the supervision of Prof. Dr. Martin Welsch and Dr. Andreas Nauerz. This work cannot be done without the work of Jesus Expósitos. Even though we did not make it to the end together, but it had been a wonderful time working with great people like you.

Through all these years in Jena, I could not have made it without my colleagues in Fusion group, especially Felicitas Löffler and Dr. Friederike Klan who listened to and helped me through myriad problems. My appreciations also go to all the people whom I met and exchanged our ideas during conferences, meetings or even on my vacations.

I cannot finish this part without saying thank you to dad and mom, who always believe in me and support me for everything I want to do. I have dreamed so many dreams, and one of my dreams is to make you smile. Your smiles have given me all strength to move on every time I fall. Karn, my only brother, and my inspiration. My life has changed since I followed you to a math class one day in 1995. Auntie, I appreciate everything you have done for me. My words are too plain to say thank you to you. Uncle Meechai, even I cannot see you anymore. I know you can see me from somewhere far away. Rest in peace, and you will be in our memories forever.

Looking back to the day I decided to do my Ph.D., many thanks to my colleagues and supervisors at Thomson Reuters Thailand, who understand my dream and supported me to pursue it. Not forgetting my classmates in Electrical Engineering Department, Chulalongkorn University, and Assoc. Prof. Dr. Ekachai Leelarasamee. I am not good at showing my appreciation, but I would not have come this far without you all.

Friends, you know what I have been through. We have shared our joy, sadness, and laughter all these years. Have you warned me about a Ph.D. life? I guess you did. I suffered, I survived, I struggled, I laughed, and I carry on.

Finally, I want to thank myself for being persistent. I always believe that every action has a consequence, and every matter has a reason. I started to write a story of Ph.D. life years ago, and now it is the right time to complete it. After I finish my Ph.D., I will continue pursuing my researching in informatics career and open a new chapter of my book.

Abstract

Nowadays, the Internet of Things (IoT) is becoming progressively colloquial to media. Considering the abundance of smart sensors we are currently using, context-aware computing is available everywhere and thus embrace us into the world of IoT. However, when there are trillions of resources out there, how can we spontaneously specify the resource we need? Therefore, one of the main research questions is the device and service discovery. The fundamental step is to study how these devices and services are described. Many standard web services descriptions are used to describe not only web services but also physical devices. These devices are encapsulated under the web service communication layer to make them available on the Internet. Web service descriptions with semantic annotations can be used to automate dynamic discovery of resources. This technique enables automatic discovery, configuration, and execution of resources in dynamic environments. Thus, we focus on the resource description language that allows semantic annotation. Nevertheless, there is no single standard formalism to describe resources. It is more tactful to handle multiple description formalisms simultaneously.

This thesis presents a cross-formalism resource discovery technique which utilizes the user context and resource's context to improve the recommendation of resources. In contrast to existing work, our technique is geared towards non-IT-savvy users. The discovery process should not be restricted to single resource description formalism. Moreover, the matching algorithm should be user-aware and environmentally adaptive, i.e. depending on the user's current situation, rather than limit to keyword-based search. The major drawback of this traditional approach is that it depends on the users' expertise and mostly requires several tries. Hence, we developed the resource discovery module on top of existing techniques, which will rank discovered resources to serve users' queries according to their interests, expertise, and current situations. This thesis explains the implementation detail and shows the evaluation of each implemented module. We aimed to prove that the quality of the result is improved significantly compared to conventional discovery techniques.

To demonstrate the usability of the proposed method, we deploy it in MERCURY. MERCURY is a platform that allows both businesses to engage with their customers and end users to create custom-made applications. It offers a web-based interface to bridge the flourishing functionalities available in the IoT and the end users. Within the context of MERCURY, registration, assembling, and execution of resources need the automatic resource discovery. Since the implementation of this work is designed to be a standalone service, there is no restriction to use it under the domain of MERCURY. Hence, the generic problem of resource discovery in any IoT application can benefit from the outcome of this thesis.

Contents

List of Tables

List of Figures

Listings

Nomenclature

BPEL	Business Process Execution Language for Web Services
GSN	Global Sensor Network
IaaS	Infrastructure as a Service
IoT	Internet of Things
nDCG	normalized Discounted Cumulative Gain
OWL-S	Web Ontology Language for Web Services
PaaS	Platform as a Service
S3	Semantic Service Selection
SaaS	Software as a Service
SAWSDL	Semantic Annotations for WSDL
SRRs	Search Result Records
TopD	Top Document
W3C	World Wide Web Consortium
WSDL	Web Services Definition Language
WSMO	Web Service Modeling Ontology
XaaS	Everything as a Service

Part I.

Introduction

1

Motivation and Overview

Context-aware pervasive computing has become a prevalent topic over the last decade. With the upcoming of ubiquitous mobile devices[1], which provide heterogeneous sensors and facilitate Internet access to users, at anytime and anywhere, we are close to realizing the idea of the "Internet of Things" (IoT) [MF10]. In the IoT world, physical things are becoming smart web-connected devices. They can create, store, share data, and can be programmed to make decisions based on given information. In addition to these concepts, things have become not only remotely controllable but also able to communicate with each other and able to automatically adapt themselves to the demands of the user. Meanwhile, the enormous growth of RFID embedded devices enables sensors - from personal health monitoring such as blood pressure sensing, to logistic monitoring of cargo shipping - to be accessible as an application of IoT [AIM10]. The usage of IoT is not limited only to proprietary and complex solutions, but almost everyone can also experience IoT via customizable and affordable sensors like Arduino [Ard], Raspberry Pi [Ras] or Ninja Blocks [Nin].

According to the definition of context in informatic computing domain:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. [Dey01]

By measuring the context of users and their preferences, things can be altered or configured to match the requirements of each user. Let us consider the following example:

> *Ann decides to be woken up at 6 A.M. and go jogging if it does not rain in the morning. Otherwise, she prefers to receive an alarm message at 7 A.M. and postpone her jogging schedule to the evening.*

[1] [SGFW10] estimated that there will be 50 billion connected devices by 2020. However, comparing with the recent figures of IoT devices, this popular claim seems to be over-optimistic. From [Sta], there were 15.41 billion IoT devices in 2015. The number is growing approximately 15% per year, giving that there were 17.68 billion IoT devices in 2016. When the trend keeps continuing, we will reach the estimated number by 2023, three years later than the estimation.

To achieve this functionality, a weather report service or a rain sensor needs to be combined with the user's location sensor. This piece of information can be obtained from, for example, the user's calendar, a flight booking service, or any service that provides information on her whereabouts. Meanwhile, the change of her jogging schedule should push a notification to her online calendar so that she will be reminded later via her mobile device.

This simple example already shows the main components of the IoT; sensors (things that sense the environment, e.g. rain gauges), actuators (things that change the environment, e.g. an alarm clock), and services (e.g. weather forecast web services). For the sake of brevity, all three of these will be referred to as resources for the remainder of this thesis.

Major challenges in realizing the IoT arise from its colossal size and ubiquity. One such challenge is the discovery of appropriate resources. It is not realistic to assume that all users will be able or willing to manually find a suitable resource among the thousands that physically surround them or even the billions existing globally. Thus, some support for resource discovery is needed.

One approach to address this problem is to wrap resources (or collections thereof) as (web) services (XaaS - Everything as a Service), as suggested by [ZPG13] and [PZCG13], and to use an approach of automatic resource discovery developed in the semantic web services community [LpNQP11]. However, there is no unified resource description formalism and no unified resource matchmaking approach, and it is unlikely that this will change in the future. This means when a user composes a request using one formalism, an automatic discovery process will find only resources described in this formalism and will miss all others, which is not satisfactory. Moreover, an appropriate use of context, e.g. user's locale, can significantly improve discovery results, e.g. rain sensors nearby. Hitherto, no unified framework that allows "injection" of context information into service offers or requests exists.

The semantically annotated description also enhances the discovery of resources. For instance, when a user is looking for an actuator that receives the date time value to trigger an alarm, the description of this actuator should be matched with an alarm clock. Hence, all the actuators that have been described to have the similar functionality should be picked up and presented to the user.

The usage of resource discovery is not limited only to the recommendation of suitable resources, where a user gets the list of possible resource, then decide as to which resource is suitable for his needs. Let us consider the case of running scenario, where the user combines multiple resources to work as a new application. When one resource immediately fails to function, the dynamic resource discovery should automatically find a temporary replacement. This requires a highly accurate and profound understanding of the behavior and connectivity of the replacing resource. Therefore, the human interaction must be involved at some point in time, and as soon as possible. The resource discovery process should notify the user immediately about the failure and should offer a list of replaceable resources.

Addressing all these challenges is the main goal that this thesis needs to achieve, and we also envision that the outcome of the solution might be tremendously helpful for the realization of IoT.

Thesis Outline

This thesis comprises of four main parts: introduction, solution outline, evaluation, and conclusion.

The introduction provides the motivation and overview of this thesis, which leads us to an example scenario, followed by the system requirement for a solution platform, called MERCURY in chapter 2 "Project Background". Nonetheless, it is important to note that the primary objective of this thesis is to create an advanced method for resource discovery in the context of IoT which is but one part of MERCURY. We discuss the usage of this research via MERCURY throughout this thesis to demonstrate this work in practical application, although the proposed technique is not limited only to one particular platform. Therefore, we state all the challenges and issues which need to be solved in order to realize the resource discovery process in chapter 3 "Thesis requirements". Then we review the existing works that are either similar to this research, or could be used in this thesis in chapter 4 "State of the Art". Respectively, we compare them and indicate the gaps that need to be improved upon, which also reflect the necessity of aforementioned requirements.

In solution part, the overview of the solution is given, where the implementation detail of each part in the resource discovery is described. Chapter 6 "Context Extraction" provides the detail of how to retrieve context information for enhancing the automatic resource discovery. In chapter 7 "Request Analysis", we elaborate on the resource description languages and matching tools used in this thesis as a proof of concept. We explain how we interpret users' query into formatted resource descriptions that are compatible with existing resource matchers in chapter 8 "Request Preparation". All the results returned from resource matchers are merged into one list as explained in chapter 9 "Result Integration". Eventually, the whole solution is applied to MERCURY as elaborated in chapter 10 "Resource Discovery Integration to MERCURY".

The quality measurement and the prerequisites for evaluating the resource discovery are provided in chapter 11 "Evaluation Overview". First, the measurement was done individually as depicted in chapter 12 "Unit Test Results". Then the integrated system was evaluated in chapter 13 "Integrated System Evaluation Results". The application of the resource discovery in MERCURY is also presented here.

We summarize the result of this thesis and compare it with our initial requirements in chapter 14 "Summary". Finally, we provide the foresight of further improvements in chapter 15 "Future Plan".

2

Project Background

This thesis is written in the context of the MERCURY project[1]. Before exploring deeper into detail of the main objective of this thesis, it is helpful to first understand the overview of this project. We aim to develop a platform which offers straightforward, user-centric integration and management of heterogeneous sensors, devices, and services via a Web-based interface. To illustrate how this platform can assist end-users maximizing the benefits of IoT, we analyze example scenarios. From these scenarios, we then formulate the project requirements. Then, we construct an architecture of MERCURY, where we can realize the workflow to achieve the given scenario. Finally, we focus on a specific module which leads to this thesis' requirements in the following chapter.

2.1 Scenarios

Let us consider the previously introduced scenario:

> *Ann decides to be woken up at 6 A.M. and go jogging if it does not rain in the morning. Otherwise, she prefers to receive an alarm notification at 7 A.M. and postpone her jogging schedule to the evening.* (Ex. 1)

To realize the scenario, Ann needs to register a rain sensor and an alarm clock to the platform. Sensors and actuators are connected through gateway components. Thus, they appear to the service discovery and the execution engine as web services. This way, we can overcome the issue of heterogeneity of devices. Gateways support bridging sensor networks to web services, allowing direct access to sensors and actuators. Here, the infrastructures like GSN middleware [PZC+12] can be deployed. These infrastructures will be elaborated on further in Chapter 4.

From MERCURY's perspective, these sensors and devices appear as web services with machine-interpretable descriptions. Therefore, physical devices or web services could be

[1] www.mercury-portals.org

treated similarly, and can be simply referred to as "resources". These resources will be used as fundamental building blocks to create new custom-made applications later.

Based on [BBM], we envision the interactions between end users and IoT platform as follows: after the registration of each resource, the resource's properties should be manageable. A resource management process can be implemented together with the resource registration process. First;

P1. We need a resource registration and management module.

However, the user may experience some difficulties in sensor discovery. During the registration process, if there are sensors with the same name appearing simultaneously, this user may be unable to specify the preferred sensor. The issue can be solved by using technical information to identify the preferred one, like a serial number. However, this approach is apparently inconvenient for non-technical users. Thus;

P2. We need a resource discovery module.

Afterwards, Ann can wire sensors or services to meet the desired functionality. Here, a modeling tool can assist her in combining all sensors and actuators to create a scenario. When the user completes the scenario, there should be a model translator to convert a graphical model into an executable script. Therefore;

P3. We need a scenario modeling interface and model translator.

The executable script will be processed by an execution engine which regularly returns the result from the operation. Besides, the runtime failure should be dealt with, e.g. push a notification to a user or automatically find a replacement of the missing resource. Consequently;

P4. We need an execution environment.

During the day, Ann may need to check if her rain sensor is still responsive and whether there would be a thunderstorm for the whole day. She can monitor the on-going process via a runtime interface. Thereupon;

P5. We need a runtime UI.

Furthermore, each user may have different interests and preferences. We can adapt the behavior of our platform according to each user's context. Consider the following scenario:

> *Ann has connected her online calendar with her email account, which she provided to MERCURY as a user account information. She wants to have a rain sensor internally updated according to the location she put on her calendar. On the other hand, the confirmed jogging scenario should be updated to her calendar as well.* (Ex. 2)

It is possible that the user can register and assign her GPS locator to automatically locate her whereabouts and then look for a nearby rain sensor. However, if she wants MERCURY to rely on her calendar, she can assign her calendar as a sensor. The user can register her calendar without extra effort, since she has already defined it in her user account. Furthermore, if there is no information on her calendar, MERCURY can assume to use her default address from her user profile as her current location. Then;

P6. We need a user management module and need to extract user context from it.

Based on all these project requirements, we propose a system architecture as described in the following section.

2.2 MERCURY Architecture [KRONW12]

From a user's perspective, the essential parts are divided into two categories: the web portal client side, and the server side as shown in Figure 2.1.

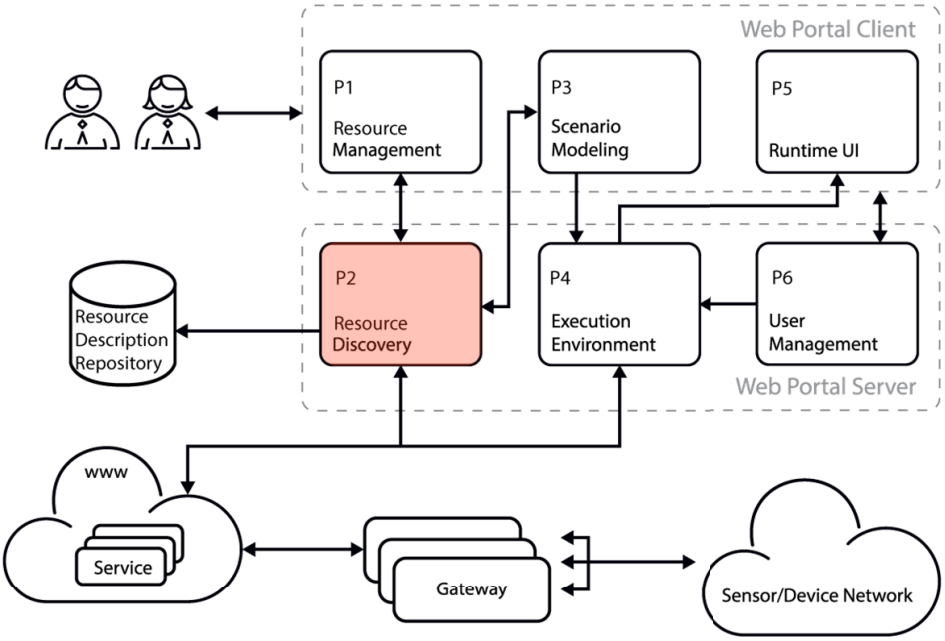


Figure 2.1.: Architecture of MERCURY's framework [KRONW12].

Let us take a closer look at the web portal client side. A resource management module is responsible for registration and management of resources. Corresponding to Requirement

P1, this module allows users to register their devices or services to the system and manage their properties and access rights. Users can choose and combine devices and services via a scenario modeling tool with respect to P3. When a scenario is complete, a model translator will translate it into an executable script, while a runtime UI, as required by P5, is responsible for displaying events and the current status of defined scenarios to the user.

On the server side, it consists of resource discovery[2], execution environment, and user management modules. Following Requirement P2, the resource discovery manages the service's description repository. It is also responsible for retrieving a request from the scenario modeling, the resource management or the execution engine. Then the resource matcher can use the request to match with the description in the repository.

Meanwhile, requirement P4 demands an execution environment to obtain the script-like results of the scenario modeling and then to execute them. A user management maintains user models (such as expertise and preferences), and access rights to devices. Also, it can provide valuable information, such as user context which could fulfill Requirement P6.

MERCURY offers a service implemented for portal technologies such as WebSphere [3] and Liferay [4], and a user-interactive part as a standalone application. This portal allows users to access our framework from any location via any suitable device.

We aim to implement the resource management, the scenario modeling, the runtime UI, the resource discovery, and the execution environment. For the execution environment, we implement a model translator which passes the script to an execution engine such as *Business Process Execution Language for Web Services (BPEL)*, *Business Process Modeling Notation (BPMN)* or *Yet Another Workflow Language (YAWL)*. For the user management, we use a built-in structure from the Portal. The user context retrieval is needed to be implemented additionally. Nevertheless, this thesis focuses on the resource discovery part. The outcome can be utilized not only within MERCURY platform, but also in any solution that deals with the heterogeneity of IoT.

In [OEKR+12], we demonstrated MERCURY. It assists users to accomplish their desired tasks equipped with environment-adaptive functionalities. That is, depending on the user's context, MERCURY can deploy the appropriate resources available at the specific time and place to perform the assignment. The following process flow exemplifies how end users can make use of existing resources to create applications and execute them via MERCURY.

[2]The resource discovery (P2) is the main focus of this thesis. We will discuss more in detail about the motivation, research questions, and requirement of this component in the upcoming chapter. Later in the solution part, we will describe the implementation methods.

[3]http://www-03.ibm.com/software/products/en/websphere-portal-family

[4]http://www.liferay.com

2.3 Process Flow

To depict the usage of MERCURY, let us consider the scenario from Ex. 1. To achieve the scenario, we envision three main steps: Resource Registration and Management, Scenario Modeling, and Scenario Execution.

Obviously, it is unrealistic to register all relevant resources available via the Internet. Therefore, an automated resource discovery approach is combined with the conventional registration process. In the registration process, we can make many subprocesses automatic. First, the nearby resources, like rain sensors, should be discovered (if the current location of resources and the user are provided). Otherwise, the user has to provide at least one keyword that describes the functionality of the resource she is looking for, e.g. rain, weather, meteorology.

During the scenario design, a group of resources can be combined to create a situation building block, such as ”get weather” or ”set calendar” situations. This situation concept offers reusability, modularity, and the component is also counted as one type of resource. An automatic resource discovery is also necessary to help users find suitable building blocks.

When a scenario ”go jogging” is created and being executed, a placeholder for a weather sensor might need to be updated according to the user’s location as required in Ex. 2. If the GPS sensor is not assigned to the scenario, MERCURY should be able to estimate the whereabouts of the user. Such estimation can be made from a user profile’s data or other sources of context, social network status updates, for instance. The resource discovery helps to resolve this issue by deriving user’s context and matching with resource descriptions.

Next, we will elaborate on the definition and detail of each process.

2.3.1. Resource Registration and Management

Web services usually provide human and machine interpretable descriptions. These descriptions can be automatically created in IDE (Interactive Development Environment) so that they are consumable by a machine. A user can register a resource by providing a direct URI of the resource’s description (as provided in ProgrammableWeb [5], for example). However, it is most likely that a keyword(s) will be provided instead of the exact URI. Then, the basic matching would match the keyword with resources’ names via a registration GUI as shown in Figure 2.2. Additionally, this keyword-based search can be done semantically so that the results are not limited to the exact input text. For example, when a user enters "rain sensor" as a keyword, the resource discovery could also return the resource with the name "weather forecast".

[5] www.programmableweb.com

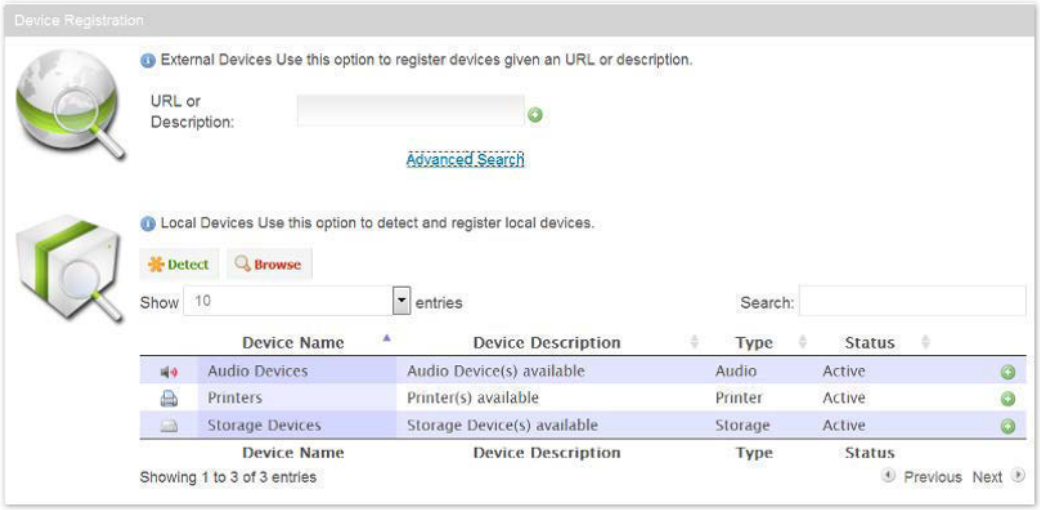


Figure 2.2.: Designed GUI of resource registration in MERCURY.

Furthermore, we can get more accurate results when the user defines keywords for input, output or operation of a resource. With this particular information, the discovery process can look into the description detail and match the given keywords with a more refined algorithm. For instance, from Ex.2, the user might want to use the weather forecast service instead of a local rain sensor. The weather forecast service she is looking for should receive location data as an input and weather condition as an output. However, the input location may have various formats. It could be a latitude-longitude value, postal code, or a city name. Therefore, the user could also specify the data type in a resource discovery GUI.

After the registration, users can configure the resource name, description, attributes and privacy properties via the resource management GUI depicted in Figure 2.3. Here, users can specify which resources they want to share or see resources other users have shared. All resource properties are stored in MERCURY database. Therefore, any changes in properties are applied only to MERCURY context.

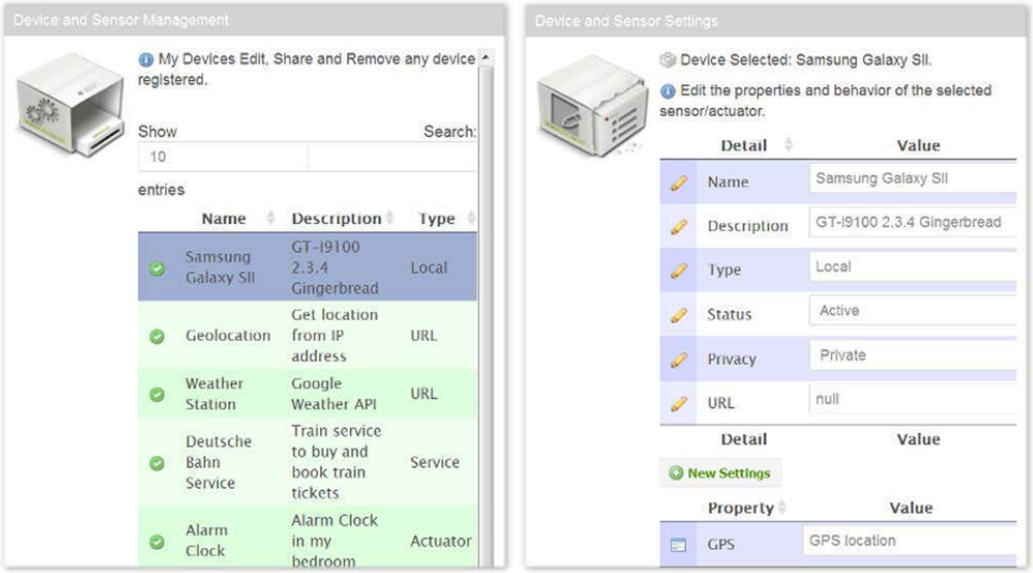


Figure 2.3.: Designed GUI of resource management in MERCURY.

2.3.2. Scenario Modeling

The GUI for modeling scenario is shown in Figure 2.4. On the left side, a toolbox palette displays various possible components: sensors, actuators, web services, operator or predefined groups of components. On the right side, there is a canvas where a user can drag components from the toolbox palette and drop into it. Then, the user can connect these components on this canvas. The canvas provides two sides of the operation, if [a condition on the left canvas] is fulfilled, then activate [an operation on the right canvas]. For example, if a heart rate sensor senses the heartbeat above 190 bpm, then sends a notification email to a medical unit and calls for an ambulance. The complete setting is defined as a scenario. Additionally, parts of a scenario can be saved as a situation for future usage.

2.3.2.1. Situation Design

When a user creates a new situation, the discovery module can assist finding appropriate resources to add to this situation. For instance, while defining a "Get weather condition" situation, a user could be looking for a rain sensor in her backyard. She can start selecting any rain sensors, and the resource discovery will recommend her similar resources which may be more suitable than the first one. Also, the compatible resources which return the weather condition as an output or accept weather condition as an input will be recommended. Therefore, the users can continue their situation design without repeating the manual search.

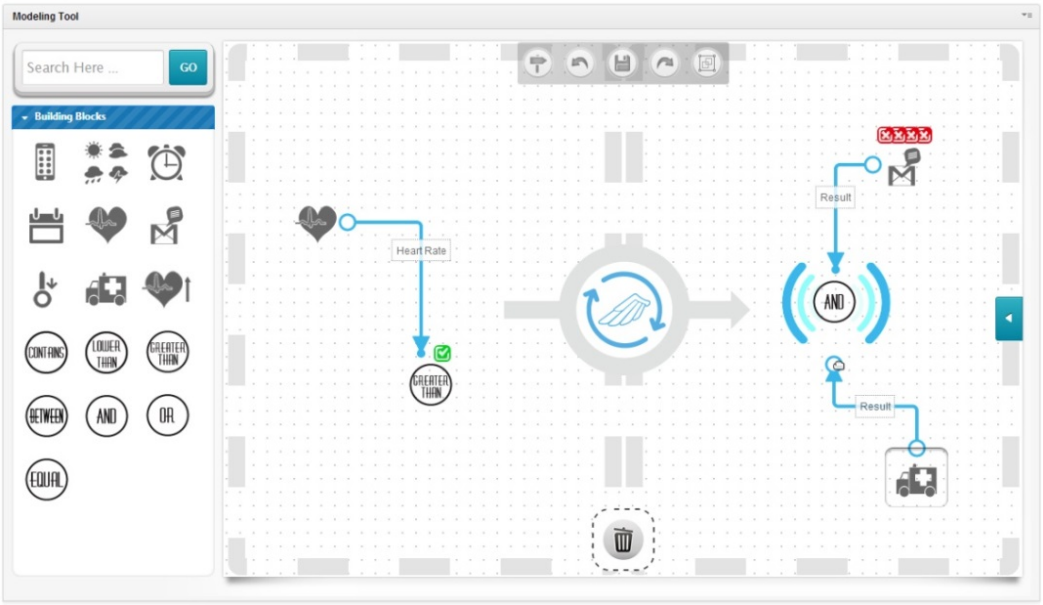


Figure 2.4.: Designed GUI of situation or scenario modeling in MERCURY.

The created situation is also shareable among other users, and this can ease the scenario design process. For example, a user can get a recommendation of resources from other users who have created similar situations.

2.3.2.2. Scenario Design

A new scenario can be created from existing situations or atomic instances. Again, the discovery module can help finding appropriate situations to be added into the scenario. As an example, a user defines: if "Get weather condition." then "Set an alarm clock." Hence, a previously defined situation "Get weather condition." can be used.

Situation templates can be used or modified. For instance, a primary element, "rain sensor", in the situation "Get weather condition" is replaceable with the "German weather service" by changing a resource instance.

2.3.3. Scenario Execution

This step allows a user to monitor resources and scenarios while they are being executed. Even though the user is not observing the process, the execution engine should be able to handle resource connection failures. When the connection failure is detected, the execution

engine should alert the user. In the meantime, the automatic discovery process should start looking for a replacement resource.

Another case for altering the scenario during the runtime is when a placeholder is included in a scenario. When a user needs the service to adjust automatically according to her current context, she can utilize the "on-the-fly" resource discovery. For instance, we have a situation "Detect if there is a weather sensor in my vicinity." The user may select a placeholder to provide value for a parameter "myLocation" in the situation creation. During execution, MERCURY will look for a resource that returns location information.

Note that the processes 2.3.1 and 2.3.2 are entirely user-interactive. MERCURY will propose suitable resources, but it is up to users to decide which one is the most appropriate. In contrast, it is hardly possible for the process 2.3.3 to wait for users' interactions because the decision must be made immediately. In that case, we need to ensure that MERCURY is capable of automatically finding the best resource.

Even if the resource discovery is highly accurate, it is still doubtful whether the replacement should be completely done without a user approval. Although this issue is beyond the scope of this thesis, we would like to present the initial solution. We aim at providing a list of potential resource and push a notification to a user whenever the on-the-fly resource discovery is required.

Moreover, we can utilize a semantic knowledge to elevate the chance of successful resource discovery. This semantic technique can be applied during the preparation of a request message (e.g. appending synonyms) and the matching between the request and resource descriptions.

Although the entire system's architecture is presented here, this thesis is focusing on the resource discovery module and its connection to other parts within the proposed architecture. In the next chapter, we conclude the thesis requirements based on the project requirements described in this chapter.

3

Thesis Requirements

According to the project requirements, one major component is the resource discovery unit. In this chapter, we summarize the problems the resource discovery aims to solve within each process of MERCURY. Consequently, the thesis requirements are defined. Thereupon, we design a resource discovery engine, which will be elaborated on in Part II.

3.1 Problem Statement

> "The acquisition of knowledge is always of use to the intellect, because it may thus drive out useless things and retain the good. For nothing can be loved or hated unless it is first known." — Leonardo da Vinci

To make IoT resources feasible, accessible, and usable, there are IoT search engines like Thingful [Thi] and Shodan [Sho]. Although such services are available, looking up the appropriate resource among numerous resources in IoT is still tedious and even impossible in many scenarios (e.g. dynamic allocation of resources at runtime). Therefore, an automatic resource discovery is needed.

First of all, a registration process must be defined so the system would recognize and understand the functionality of each resource. Let us consider Ex. 1 again. There, Ann wants to create a scenario that sets her alarm at six in the morning on days with nice weather and in the afternoon on the other days. To build this scenario, she needs to register the required sensors and services with MERCURY. The resource discovery module should support her in this task. Ann should be able to specify the resource she is looking for by searching for ”rain sensor” and ”alarm clock” in the registration GUI provided by MERCURY. The system should then match her specification against the descriptions of available sensors/services.

Then we have to answer the following question. What could describe the individual resource in the machine-interpretable fashion so that the resource description can be discovered and used automatically? Thus, we need to study how resources are commonly described. In fact, there are several description formalisms widely in use, and it is less likely that a single resource would be described in various formalisms. If we look for a

particular resource based on one single formalism, we may overlook the relevant resource described in a different formalism. Therefore, the resource discovery should handle more than one description language, in other words, we need a cross-formalism discovery. Moreover, the crucial information of each resource must be stored so that it could be managed and matched with a user's query later.

Next, during the modeling process, MERCURY needs to construct a query to retrieve resources that fit a scenario. This opens up the opportunity to cultivate context information, such as geolocation and user preferences, to improve the resource recommendation, as exemplified in Ex. 2. Considering when the user is looking for a calendar, the highest potential should be her calendar that is linked to her email address. Thus, we need to determine which type of context should be taken into account and when they should be applied to the discovery process. The context information can significantly enhance the relevance of discovery results.

Lastly, during the execution process, the resource's failure to connect or operate should be detected immediately, and the system should offer a user the solution, i.e. replaceable resources. Considering the scenario created in Ex. 2, when a GPS sensor fails to respond, as well as when an empty placeholder for a location sensor is detected in the runtime, the resource discovery should be instantly activated. The resource discovery should suggest possible sensors which can be used by the execution engine in place of the previous one. Here, we need to formulate a query from the qualifications of the missing resource to match with available resources.

Though the outcome of this thesis will be evaluated and deployed on our implemented platform, the problems this work aims to solve are not restricted to this platform. These are generic problems faced by every system that uses automatic discovery. Therefore, the solution from this research can be used for the greater good in the context of IoT.

3.2 Requirements

Based on the problem statement, the requirements need to be fulfilled to construct the proposing system are summarized below.

First, the resource discovery should be responsible for creating a request in pre-defined formats from free-text keywords (R1), so it can be matched with existing descriptions in a resource discovery repository by semantic resource matchers. Also, the solution should be user-friendly so that non-IT users will also be able to formulate resource requests.

Since the description that we are looking for could be described in arbitrary formalisms, the resource discovery should be able to handle different description formalisms (R2). First, we study the commonly used description formalisms. There are many solutions to make a matching between the request and resource descriptions in each formalism. Thus, this thesis does not propose a new resource matcher, but rather use the existing ones. Besides,

using different types of matchers at once does not only yield a better discovery result, it also eliminates the aforementioned problems. The resource discovery should be able to handle multiple description formalisms, and therefore matchers simultaneously (R3).

To support the discovery process, a basic search is needed, for example, it can list the local resources visible on the current device, as well as the conventional text-based search over the description document (R4). However, this alone could be insufficient; the system should offer an advanced search. That is, the system would consider not only the purely semantic meaning of the keyword but also the functionality of the keyword (R5). The example of a semantic search is, when a user is looking for an ambulance service, the ambulance can be associated with terms like *emergency*, *road*, and *vehicle*. Thus, the relevant resources which contain such terms can be included in search results. This semantic search recently becomes more in use [BS16]. The syntactic search is considering the role of a given keyword. For instance, whether the keyword should be found as an *input*, *output* or *operation* description of a resource.

Additionally, the context information can enhance the search results [LdMT+15]. Thus, the resource discovery should use the context information in addition to the semantic search process. We need to specify the source of contexts such as the user profile or sensors' descriptions (R6). When the user context or resource context is available, such as current location, it should be added to the query in order to improve the search result (R7).

The registered resources' descriptions should be stored in the system and the system allows the resource owner or the system administrator to update them later (R8). This functionality can be realized by a resource management module. It will play a major role in the modeling tool. Since a newly created scenario is considered as a new resource, its properties would be used as a resource description so that this scenario can be discovered later on.

In the process of scenario modeling, it is possible that thousands of resources will be available on the platform. Thus, the recommendation and search for the right resource that is adaptive to user's interactions are necessary (R9). When the registered resources and created scenarios become numerous, users might need to use a search function to find resources, and a recommendation could offer potential resources to the users. The recommendation can be based on the users' history of usage ('frequently used' resource, for instance), the compatibility of input/output, and the semantic relevance (e.g. a temperature sensor is relevant to a weather forecast service).

During runtime processing, the resource discovery can be used for replacing a resource in a placeholder or used for handling a failure of a resource (R10). For a placeholder item, it is usable when an actual resource replaces it. The list of suitable resources should be provided to users so that they can easily make a decision. Moreover, as the runtime failure occurs, the users can enable the system to automatically repair the failure, but it is recommended that the users should acknowledge the solution first before it is applied.

Since the resource discovery is involved in all these processes, it is practical to provide the resource discovery as a standalone service (R11).

Despite the fact that the resource discovery is designed to operate as a standalone service, the outcome of the discovery should be presented differently depending on the context of usage (R12). Thus, the recommendation of resources needs to be customized according to the purpose of discovery. For example, in the registration process, the detail from resource descriptions (including service's invoking protocol) can be reviewed before being added to the system. While in the modeling process, such technical detail like the invocation protocol is unimportant. Instead, the operation description, user ratings, and privacy settings are more informative for users. Lastly, during the execution process, the resource discovery should focus on the input/output compatibility of resources. It should allow a user to replace the resource with a single action.

In the next chapter, relevant studies and projects are reviewed to compare and emphasize the importance of these requirements. Other components of the project that can be fulfilled by existing works are discussed as well. The detailed solution for the requirements presented in this chapter is elaborated on in Section II. Finally, all requirements are summarized again in Table 3.1.

	Requirement
R1	The resource discovery unit should be able to construct a free-text query message from end users into pre-defined formats.
R2	The resource discovery should be able to interpret different description formalisms.
R3	The resource discovery should handle multiple description formalisms and matchers simultaneously.
R4	The discovery process should offer a basic search for resource descriptions.
R5	The discovery process should offer an advanced search considering semantic annotations and syntax of keywords.
R6	The source of user context should be defined.
R7	When user context or resource context is available, they should be applied to the search query.
R8	The resources' descriptions should be derived, stored, and made editable by authorized users.
R9	The modeling tool should recommend the potential resources to users by considering users' interactions.
R10	During runtime, a placeholder item or a fail-to-respond resource should be supported by the service discovery.
R11	The resource discovery should operate as a standalone module.
R12	The discovery result should be presented in the registration, scenario modeling, and execution processes in a way that users can apply the result instantly.

Table 3.1.: Summary of requirements in this thesis.

4

State of the Art

In this chapter, the existing works which offer a solution like MERCURY are reviewed and compared in terms of usability. It may turn out, however, that none of them focuses on the resource discovery process and sufficiently takes user and environmental context into account.

Middleware is used as a connection between an application layer and a hardware layer to hide complications in the device integration. Even so, the heterogeneity issue has never been adequately addressed. Nevertheless, the prominent middlewares are reviewed so that we can decide which one can be applied to MERCURY.

Afterwards, the resource descriptions are reviewed. Upon that, representative descriptions are chosen for implementing a proof-of-concept prototype. Along with commonly used formalisms, many description matchers have been implemented, and they perform well in their domain. Therefore, it is more plausible to study and deploy them in this work rather than create a new resource matcher from scratch.

Finally, we examine solutions which apply user and environmental context into their search algorithms. Thus, we can summarize the gap of improvement that has not been answered by any work.

4.1 Similar Existing Solutions

The concept of the Internet of things (IoT) has been around since the 1990s. Since then, many pieces of research have been conducted to actualize the idea. Nowadays, users can control smart devices via web-based platforms. Initially, most of the solutions are aimed at the implementation of platforms for scientists or IT-savvy users. Many attempts have been made to attract non-programming users with more intuitive GUIs. Furthermore, the created workflow can be more versatile and enhanced by being shareable and customizable within the users' community. The big potential revenue for commercial players lies in mash-up tools for social networking applications.

Here, we classify existing solutions into four groups: tools for experts, tools for end users, tools for teams (multiple users), and commercial tools. The comparison between all these solutions is summarized in Table 4.1.

Project name	Sensors Registeration UI	Modeling UI	Runtime Execution UI	Supports non-expert users	Remarks
SenseWeb [KNLZ07]	P	✗	✓	✗	
Web Mashups [Gui09]	P	✓	✓	✗	
Actinium [KLD12]	P	✗	P	✗	
Linked Open Sensor [LpH09]	P	✓	?	✓	
OPEN [GZI11]	P	✓/P	?	✓	context-aware
COBASEN [LdMT+15]	✓	?	?	✓	context-aware, semantic-based
OpenIoT [SKH+]	●	✓	✓	●	semantic-based
CASCOM [PV16]	●	●	✗	✓	context-aware
[WC16]	●	✗	●	●	context-aware
SHIWA [SHI]	SCI-BUS	●	?	?	scientific sensors
myExperiment [DRGS09]	?	●	✓	?	scientific sensors
IFTTT [IFT]	✓/P	✓	✓	●	
Evrythng [Evr]	P	P	✓	●	
Xively [Xiv]	P	P	✓	●	
LSM [LpNQP11]	●	✓	✓	✗	semantic-based

(✓=available, ✗=not available, ●=partially available, P=programming required, ? =not mentioned in the work)

Table 4.1.: Comparison of relevant works regarding functionality and usability.

4.1.1. IoT tools for expert users

SenseWeb [KNLZ07], an infrastructure for data sharing, provides a map-based front-end, called SensorMap [NLZ07], as a visual interface for geocentric datasets. Figure 4.1 depicts how the integrated system works for SenseWeb. It needs a middleware to cope with the integration of the sensor network, and no GUI is available. On the other hand, a presentation layer, like SensorMap, can be explicitly implemented and used to explore the data provided by SenseWeb. Although SenseWeb can handle arbitrary types of sensors, it requires programming skills for registering and managing sensors in the system. Moreover, the project focuses mainly on scientific projects, especially geological studies.

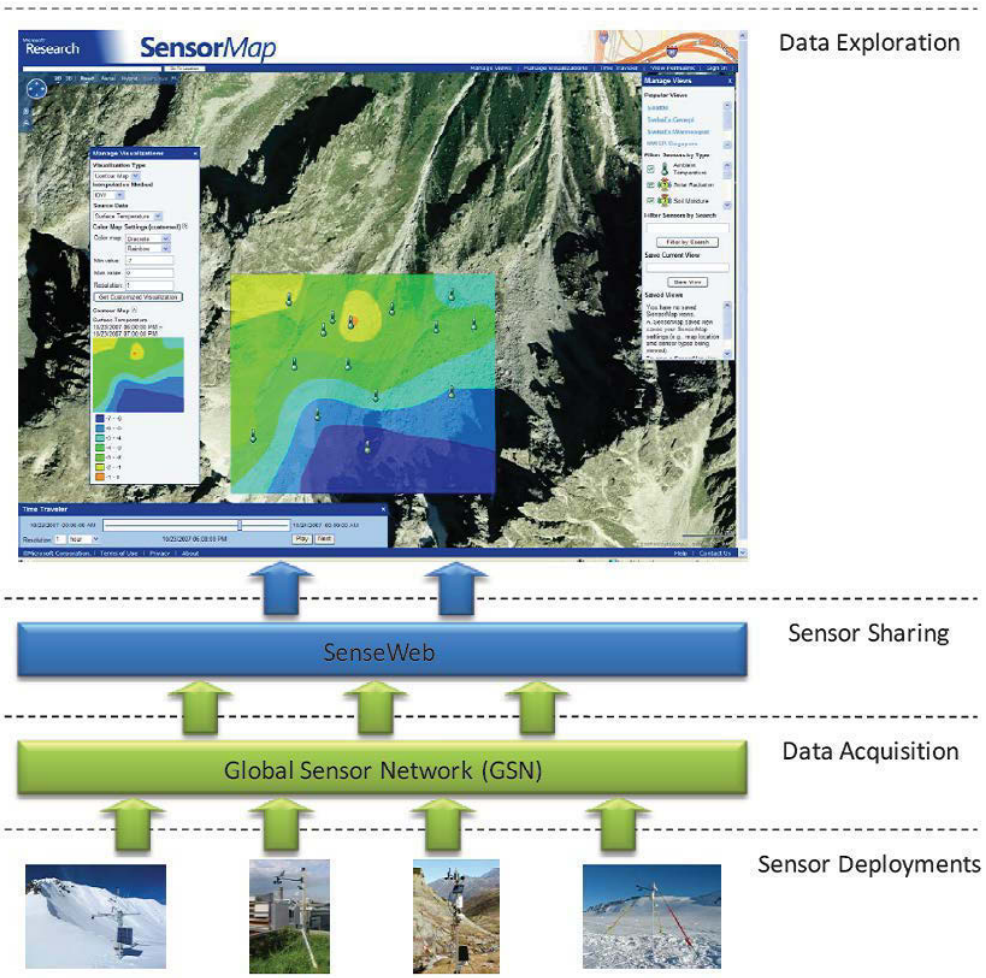


Figure 4.1.: Overview of GUI, SenseWeb [KNLZ07], sensor network, and actual sensors.

Web Mashups [Gui09] and Actinium [KLD12] offer a broad range of sensors by assuming that all sensors are wrapped and presented via RESTful API. Web Mashups provides 'Clickscript', a GUI for connecting sensors and monitoring their values (see Figure 4.2). However, to use the mash-up functionality, all sensors need to be implemented into specific interface instances called Sun SPOT. Although a registration of sensors via Sun SPOT is too complicated for end-users, the mashing up interface is a good example for our scenario modeling.

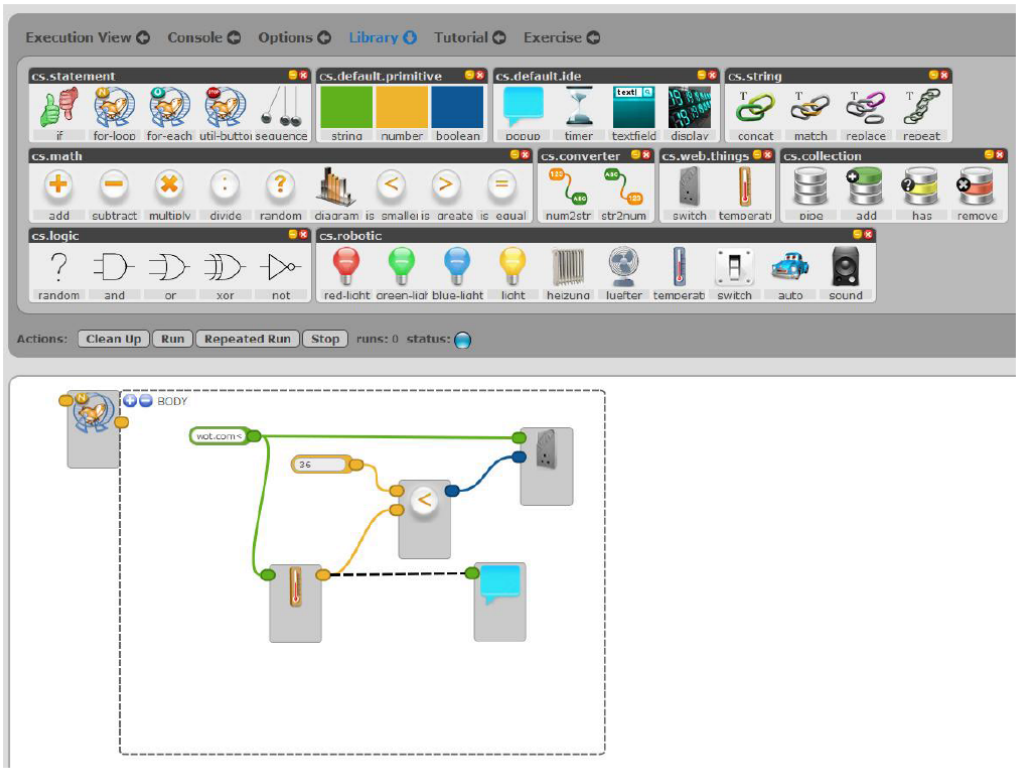


Figure 4.2.: Snapshot of Clickscript, an editor tool for Web Mashups [GTW10].

Meanwhile, Actinium requires Javascript knowledge to create an execution script. Despite the lack of a user-friendly GUI, Actinium provides concepts of "Runtime Container" and "Scripting Sandbox" (see Figure 4.3). These concepts are similar to the runtime UI and the execution environment modules of MERCURY.

We can take a lesson from these works concerning using a sensor gateway to reduce the complexity of device integration.

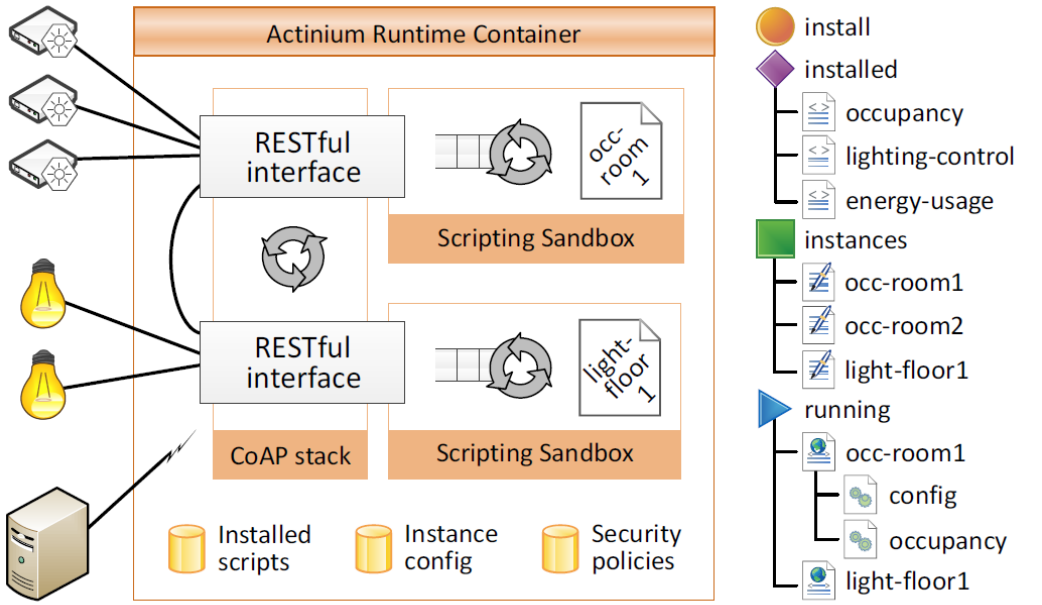
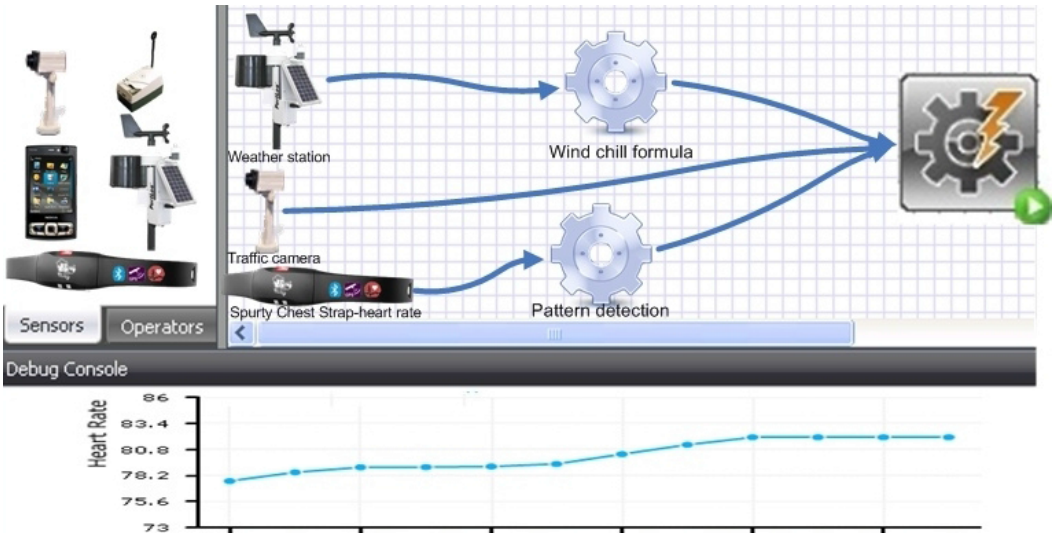


Figure 4.3.: Actinium's system architecture [KLD12].

4.1.2. IoT tools supporting non-programming users

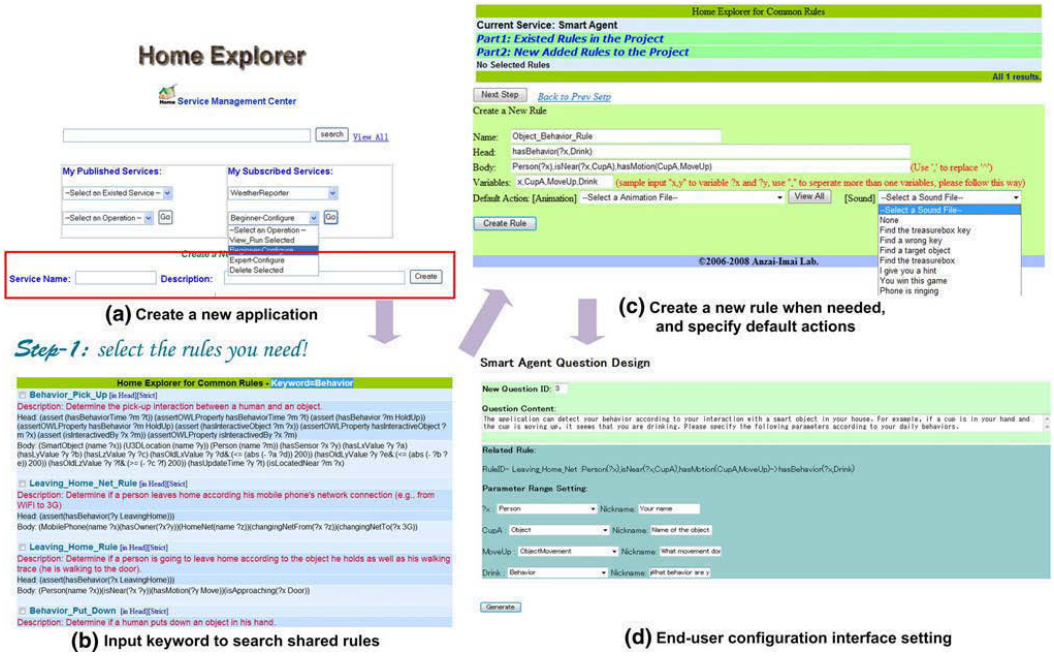
Not only that the following works aim towards non-programming users, but they also support user-defined context sensing applications. Linked Open Sensor [LpH09] attempted to publish sensor data as linked data to enable dynamic discovery, integration, and querying of heterogeneous sensors. It provides an intuitive GUI for mashing up a new application, as shown in Figure 4.4, and claims to already deal with 200,000 sensors.

OPEN [GZI11] supports both professional developers and non-expert users. It provides expert users programming API (see Figure 4.5), while simple applications can be achieved via graphical and tangible interfaces for non-programmer users. Additionally, this work also advocates the use of context-aware applications in collaborative programming environments. However, the ability to achieve sophisticated solutions should not be limited up-front by classification of users, but it should rather be pushed as far as possible by adequate context adaptive system and UI support.



License: ODC-By v1.0.

Figure 4.4.: Snapshot from Sensor Masher, a graphical mashup tool for Linked Open Sensor [Phu09].



Copyright © 2010, Springer-Verlag London Limited, RightsLink license.

Figure 4.5.: Screenshots of OPEN's main-interface (resource-search) (a) pages for incremental/composition programming mode (b-d) [GZI11].

COBASEN [LdMT+15] is a software framework includes a context-based search engine focusing on industrial IoT resources. It also tightly couples with COMPaaS [ATdMH15], an IoT middleware supporting communication between software and physical devices. The overview architecture of COBASEN in cooperation with COMPaaS is shown in Figure 4.6. COBASEN focuses on the resource discovery technique using context information derived from resources. It does not cope with users' preferences directly. Moreover, there is no device mash-up interface or runtime monitoring UI presented in this work yet.

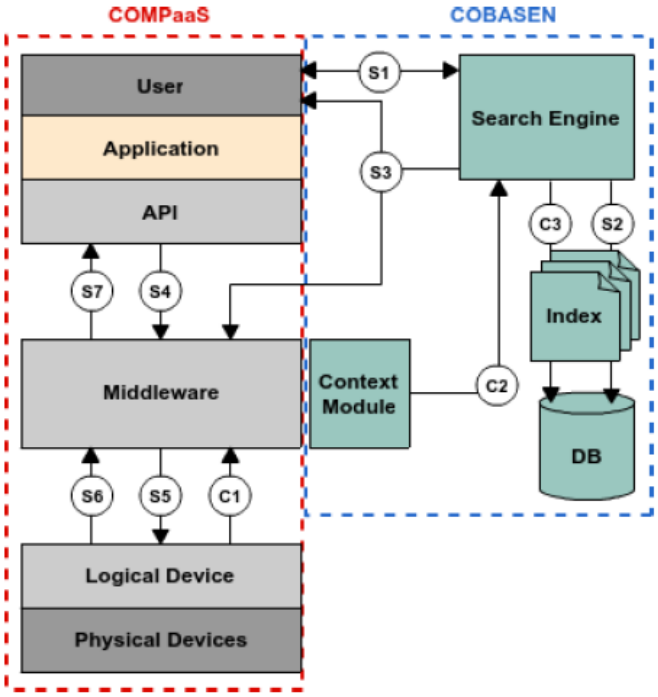


Figure 4.6.: Overview of COBASEN architecture working together with COMPaaS. C1 represents the gathering context information from physical devices through a middleware. C2: the context information is sent to a search engine, which indexes the context data and store in a database (C3). When a user searches for a device (S1), the search engine looks for relevance devices (S2). After the user selects the desirable device, the specification will be sent to the middleware (S3). The user can subscribe to the device via the middleware (S4). The communication between devices and the middleware are alternating between S5 and S6 until it is done. Finally, the user gets a report from the device (S7).

OpenIoT [SKH+] presents an open source middleware for IoT. The core of OpenIoT is the W3C Semantic Sensor Networks (SSN) ontology, which provides a standard model for representing physical and virtual sensors. It offers a zero-programming application development environment (see Figure 4.7). Although it requires no programming skill to use the mash-up tool, it provides not a very intuitive interface for non-IT savvy users. Moreover, the process of resource registration and discovery is ambiguous.

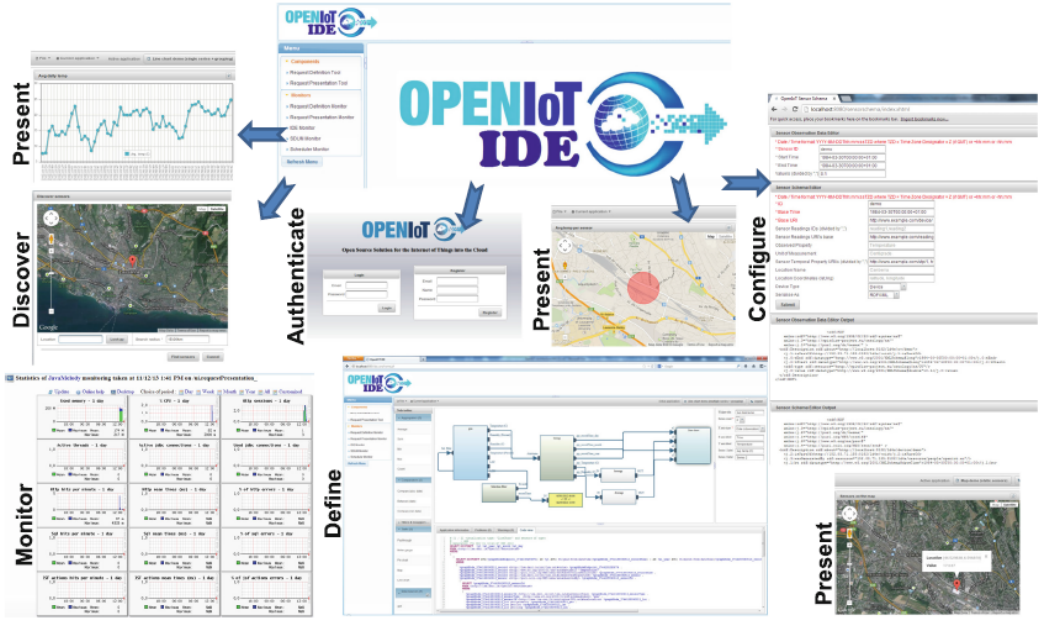


Figure 4.7.: Overview of the OpenIoT Integrated Development Environment (OpenIoT IDE).

CASCOM (Context Aware Sensor Configuration Model) [PV16] proposes a technique to enhance the IoT resource discovery. As shown in Figure 4.8, CASCOM fits in the Reasoning Engine module. It assists users to search for physical devices via GUI. When no matched resource can be found, CASCOM will compose combinations of resources that serve user's requirements. In this way, a resource mash-up can be achieved without human interaction. However, it does not explicitly mention how physical devices are discovered for the first time. This could be a task for the lower layer (Context and Semantic Discovery layer). Moreover, CASCOM does not support the existing ontologies and description languages.

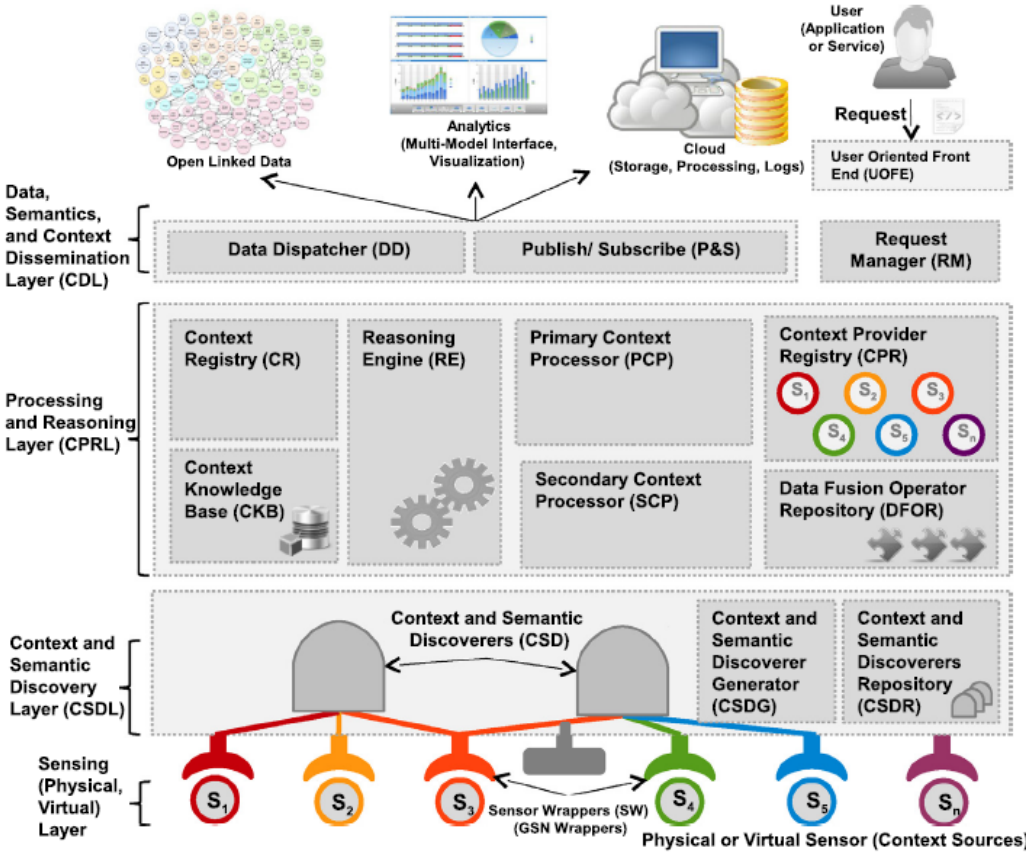


Figure 4.8.: CASCOM works as the Reasoning Engine (RE) block in the IoT architecture [PZCG12].

[WC16] proposes a centralized resource discovery system based on EPCglobal[1] (Global Electronic Product Code) [EPC]. By assigning a globally unique ID to every single device, everything will be automatically registered to the Internet. [WC16] does not provide a device mash-up solution. It rather focuses on the context-aware resource discovery which utilizes user context such as current location or user's interest. Whenever interesting devices are detected or located in accessible range, a user will receive a notification and will be asked for accessing permission (see prototype screenshot in Figure 4.9).

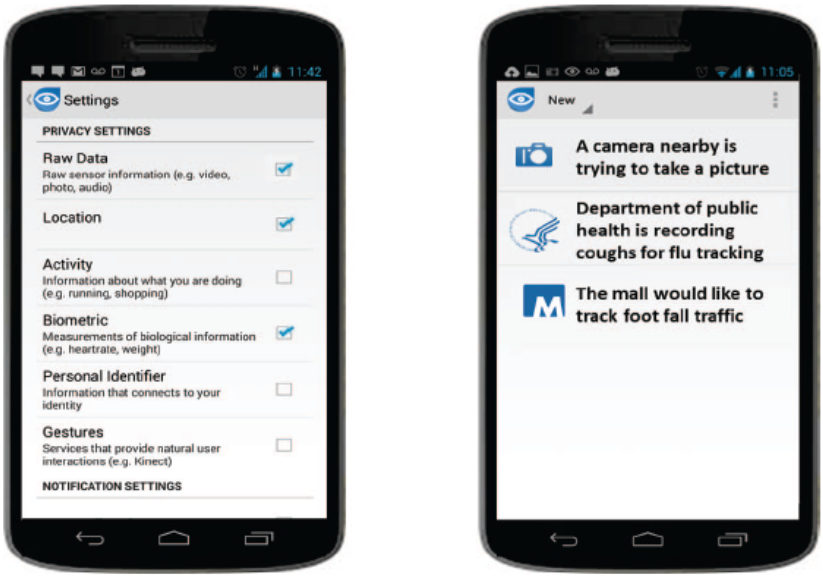


Figure 4.9.: Privacy Notification Application for IoT [WC16]. Left: Preferences settings to filter the notification. Right: Privacy notifications.

[1]EPCglobal is an initiative by GS1 and GS1 US organizations to develop RFID standards and regulations.

4.1.3. IoT tools for customizing and sharing workflow

To facilitate all users in accessing and customizing applications, a shareable workflow concept has been applied, e.g. in SHIWA [SHI] and myExperiment [DRGS09].

SHIWA supports a creation of workflow from sensors connected via SCI-BUS [SCI] gateway. It also presents simulation and analysis interfaces. Although the project aims mainly for the scientific application, we can learn from this work in the aspect of hardware scalability and interoperability.

myExperiment focuses on sharing workflows as exemplified in Figure 4.10. But it still lacks a user-friendly modeling tool and provides no detail on how to discover resources.

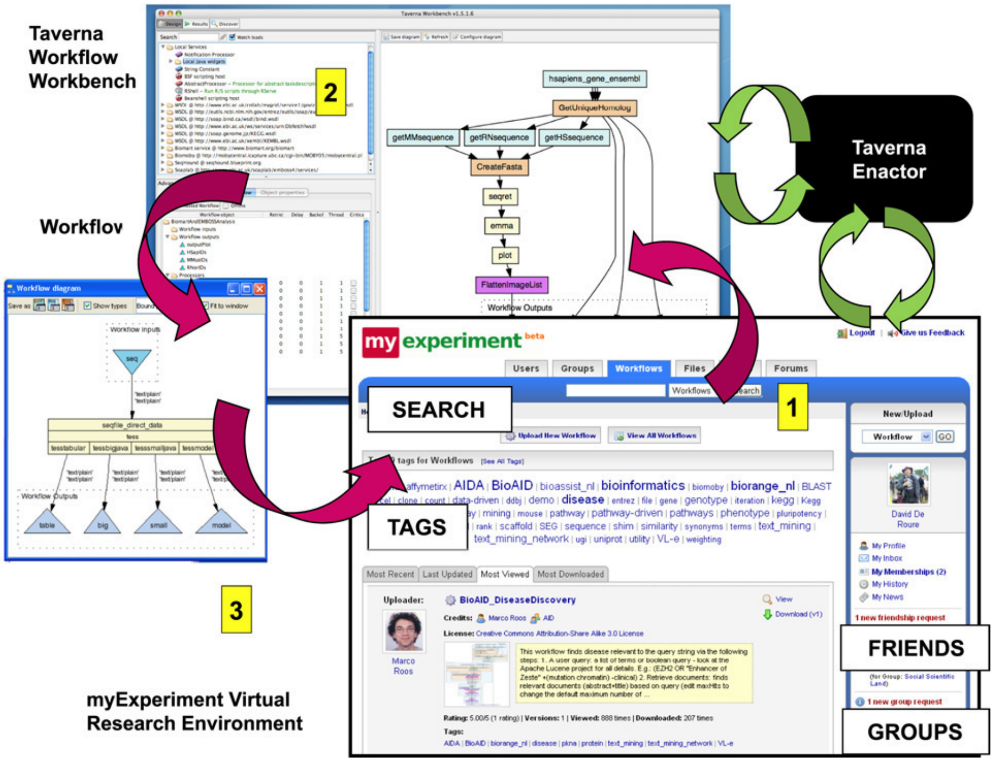


Figure 4.10.: Screenshots of myExperiment [DRGS09] shows how a scientist (1) finds a workflow, (2) executes and edits it in Taverna, and (3) uploads a new version.

4.1.4. Commercial tools

In addition to scientific research projects, there are several business solutions, such as IFTTT [IFT], Evrythng [Evr], Xively [Xiv], and the list keeps going on.

IFTTT (If This Then That) [IFT] presents a concept of mashing up sensors and services in a very user-friendly fashion. Figure 4.11 depicts the screenshots from IFTTT's dashboard. Basic applets (equivalent to *scenarios* in MERCURY), which are provided by IFTTT's developers, are ready to be used by end-users. In the meantime, expert users can create applets and share them with other users. However, IFTTT does not provide an interface to monitor the status of activated applets, which is equivalent to the execution UI in MERCURY's requirement.

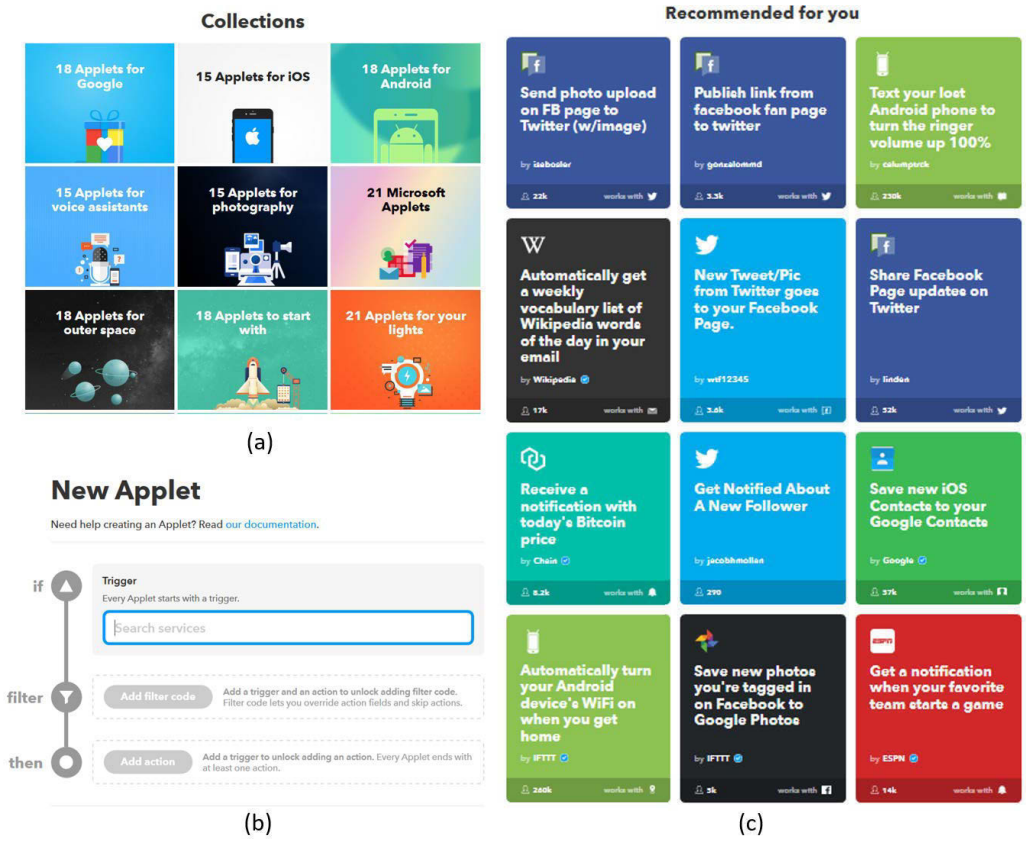


Figure 4.11.: Screenshots of IFTTT dashboard and workflow [IFT]. (a) Applets collections. (b) GUI for creating a new applet. (c) Recommendation of applets.

Evrythng [Evr], IoT Platform for Smart Consumer Products, focuses on integrating actual sensors to the cloud. It can handle arbitrary resources, though requiring programming and hardware knowledge. Nevertheless, this work does not provide a tool for creating a scenario. Evrythng is more suitable for monitoring sensor's values rather than creating an

application for IoT. However, the information retrieved from sensors can be further utilized by other platforms to trigger an action.

Similar to Evrythng, Xively [Xiv] (Business Solutions for the Internet of Thing) focuses on keeping track of real-time sensing information (see Figure 4.12). Nevertheless, it is still ambiguous how this information can be used as an event trigger.

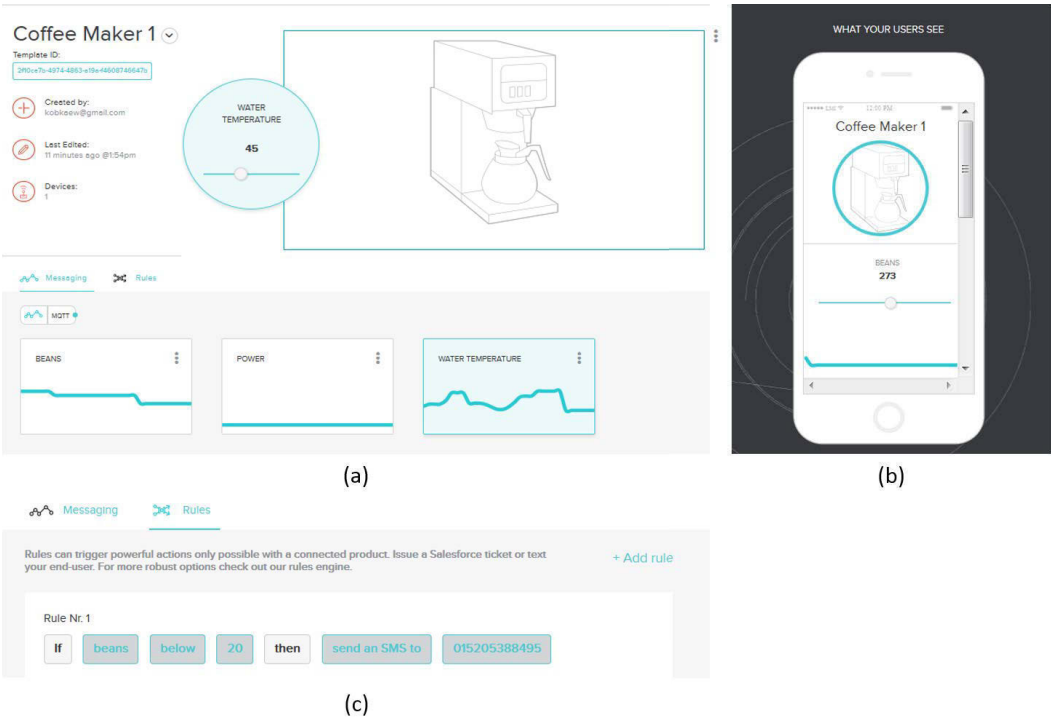


Figure 4.12.: Xively dashboard: (a and b) resources monitoring UI on desktop and mobile device respectively, and (c) rule settings for notification [Xiv].

So far, the works mentioned above have not yet described clearly how end users can discover a particular resource out of the myriad of devices available on the Internet, or how to integrate such a resource in the system. These questions shall be the principal focus of this thesis.

4.2 Resource Middleware

> The middleware is a software layer or a set of sub-layers interposed between the technological and the application levels. Its feature of hiding the details of different technologies is fundamental to exempt the programmer from issues that are not directly pertinent to her/his focus, which is the development of the specific application enabled by the IoT infrastructures. [AIM10]

Following the suggestions from [RMJPC16], a middleware should have the following characteristics: programming abstraction, interoperable, service-based, adaptive, context-aware, autonomous, and distributed. Many sensor gateways which possess such characteristics such as Linked Steam Middleware [LpNQP11], CSB-UCC [LD13], SCI-BUS [SCI] or GSN [PZC+12] can manage the connection between heterogeneous physical devices and a web-based application.

Linked Steam Middleware (LSM) [LpNQP11] is a platform to integrate IoT sensors with the Semantic Web. It provides wrappers for collecting and publishing data from IoT resources, a UI for annotating and visualizing data, and SPARQLendpoint for querying data. LSM also offers Mashup composer and Linked Sensor Explorer. However, the registration process and SPARQLendpoint are not intuitive for non-IT users.

CSB-UCC (Cloud Services Brokerage for Ubiquitous Cloud Computing) aims to handle cloud services like IaaS [2] , SaaS [3], and PaaS [4] by acting as a medium between cloud services and consumers' devices. In the meantime, SCI-BUS (SCIentific gateway Based User Support) is the core buttress for SHIWA. It has cooperated with many scientific projects such as Swiss Proteomics (Currently merged with LS[2] - Life Sciences Switzerland)[5] and German MoSGrid[6]. Nevertheless, it is still enigmatic as to how to connect actual sensors to it.

On the other hand, GSN is geared towards the creation of a wrapper for individual hardware, and thus makes them visible over the Internet. GSN servers act as communication agents between sensor networks and the Internet. A wrapper offered in GSN is a piece of Java code to acquire data from a particular type of device. It provides an abstraction layer called "Virtual Sensor" (VS) to filter and process information gained from wrappers. Also, GSN offers VS manager, Storage, Query Manager, Web Service interface and Access control. The Virtual Sensor (VS) manager is lying on top of the sensors pool layer. Above the VS manager is Storage and Query Manager which serve the Web Service interface. Access Control unit is on top of the architecture.

GSN also comes with a pre-defined VS, BridgeVS, so that users can utilize it without programming. Furthermore, a new VS processing class can be written for a complicated data filtering. Listing 4.1 exemplifies the VS definition using Bridge VS class. Apparently, this VS is both human- and machine-interpretable. For example, a room monitoring sensor consists of one camera and two temperature sensors. When the room monitoring service is called upon, it should return a picture in Jpeg format and two temperature values. Thus, editing or creating a new VS is a trivial task.

[2] Infrastructure as a Service
[3] Software as a Service
[4] Platform as a Service
[5] https://www.ls2.ch/sections/proteomics
[6] https://mosgrid.de/community

```
<virtual-sensor name="room-monitor" priority="10" protected="false" >
    <processing-class>
        <class-name>gsn.vsensor.BridgeVirtualSensor</class-name>
        <output-structure>
            <field name="image" type="binary:jpeg" />
            <field name="temp" type="int" />
        </output-structure>
    </processing-class>
    <addressing>
        <predicate key="geographical">BC143</predicate>
        <predicate key="usage">room monitoring</predicate>
        <predicate key="latitude">46.5214</predicate>
        <predicate key="longitude">6.5676</predicate>
    </addressing>
    <streams>
        <stream name="cam">
            <source name="cam" storage-size="1" >
                <address wrapper="remote">
                    <predicate key="geographical">BC143</predicate>
                    <predicate key="type">Camera</predicate>
                </address>
                <query>select * from WRAPPER</query>
            </source>
            <source name="temp1" storage-size="1m" >
                <address wrapper="remote">
                    <predicate key="type">temperature</predicate>
                    <predicate key="geographical">BC143-N</predicate>
                </address>
                <query>select AVG(temp1) as T1 from WRAPPER</query>
            </source>
            <source name="temp2" storage-size="1m" >
                <address wrapper="remote">
                    <predicate key="type">temperature</predicate>
                    <predicate key="geographical">BC143-S</predicate>
                </address>
                <query>select AVG(temp2) as T2 from WRAPPER</query>
            </source>
            <query> select  cam.picture as image, temp1.T1 as temperature
                    from  cam, temp1
                    where  temp1.T1 > 30 AND temp1.T1 = temp2.T2
            </query>
        </stream>
    </streams>
</virtual-sensor>
```

Listing 4.1: Virtual sensor definition example.

4.3 Resource Description

When a hardware is successfully connected via a middleware, it is necessary to provide a machine-interpretable description for such a resource. [MKP09] demonstrates the concept to annotate RESTful Services. The result description is used in automatic resource discovery. When a user or an application looks for a resource, a resource request with the desired capabilities of the resource must be formulated in a predefined format. The resource matching component then compares available resource descriptions with the request, and returns the resource(s) that match.

First, we consider the most straightforward web service description language, WSDL (Web Services Definition Language) [CCMW] which is a recommendation from W3C (World Wide Web Consortium)[7].

> WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly and then bound to a concrete network protocol and message format to define an endpoint[8].

From our research perspective, WSDL is the source of information that contains all necessary information, e.g. the service endpoint, address, functionality, input types, and output types. However, WSDL alone is insufficient in terms of semantic resource discovery. Thus, we need to look further for formalisms offering additional information to enhance the semantic description discovery process. The potential approaches fall into three broad categories: ontology-based, lightweight ontology-based, and non-ontological.

a. Ontology-based
   OWL-S (Web Ontology Language for Web Services) [MPM+05] and WSMO (Web Service Modeling Ontology) [RKL+05] fall in this category. These are the oldest and most mature semantic resource descriptions. However, they are heavyweight and require significant efforts and skills to create descriptions. This motivated the development of languages that fall into the second category.

b. Lightweight Ontology-based
   SAWSDL (Semantic Annotations for WSDL) [She07] and WSMOLite [FFST11], for instance, were created to augment WSDL and WSMO with relatively lightweight semantic annotations using arbitrary ontology languages.

c. Non-ontological
   The last category comprises of solutions that are not built on ontologies but use collaborative tagging techniques instead. Popular representatives of this category are tag-based descriptions like [GCPG12] and [DLY+10]. Although this technique yields a realistic result since it relies on human decisions, the biggest obstacle is the cold-start problem.

[7] https://www.w3.org/
[8] https://www.w3.org/TR/wsdl

When considering the ontology-based method, it is necessary to understand how resources are annotated semantically. [TAB09] proposes a technique to semantically annotate web services. They focus on the annotation of input and output of web services by using graph structure and chaining algorithm. Though the evaluation was applied to OWL-S descriptions, the approach is not limited to the formalism. In Chapter 5, we will explain more about the description formalisms used in this thesis.

4.4 Resource Matchers

For each of the description formalisms, several different matchers have been proposed. They differ on the algorithms and which part of the descriptions they use. [NKK10] reviews approaches categorized by description languages, which are OWL-S, WSMO, SAWSDL, and WSDL-S based. They conclude that WSMO based matchers are the most powerful in terms of performances and their holistic approach. However, in the current state, SAWSDL and OWL-S based are still the dominant approaches.

Several initiatives have evaluated the different description formalisms and matching algorithms, e.g. Web Service Challenge [KBB+09], Semantic Web Challenge [Har12], and Semantic Service Selection (S3) Contest [Klu12]. The most relevant to this work is S3 Contest, which evaluates web service matchers for OWL-S and SAWSDL. On the other hand, Web Service Challenge and Semantic Web Challenge do not publish the participants' matchers and their benchmarks. Thus, for the evaluation purpose, matchers which are reported by the S3 Contest are studied.

In Chapter 5, we investigate the different approaches used by web service matchers. Based on matchers from S3 Contest, we conclude the essential information in each description formalism. Then, the matchers that perform well are chosen for the evaluation in Chapter 11.

It is important to note that none of the existing matchers works for more than one formalism. Thus, so far, it is not possible to find a service described using OWL-S when using a SAWSDL based matcher.

4.5 Context-aware Resource Discovery

> A system is context-aware if it uses context to provide relevant information or services to the user, where relevancy depends on the user's task. [Dey01]

In MERCURY, we can retrieve such context information from the actual sensors and devices. In Chapter 6, we discuss in detail how we envision the context retrieval from available sources. We interpret this information and apply to the resource discovery so that the result would be up-to-the-minute regarding users' surroundings. First, we need to define the context model we can apply to our application.

4.5.1. Context Model

There are three prominent context models, namely, object-oriented based, spatial-based and ontology-based model [BBH+10].

a. **Object-oriented based context model** is an approach specifically designed for conceptual modeling of databases [HM10]. [HLI04] presents CML (Context Modeling Language) based on ORM (Object-Relational Mapping). As pointed by [BBH+10], CML is still immature in the aspect of hierarchical context structure. However, CML's strength lies in handling uncertainty and history of context information.

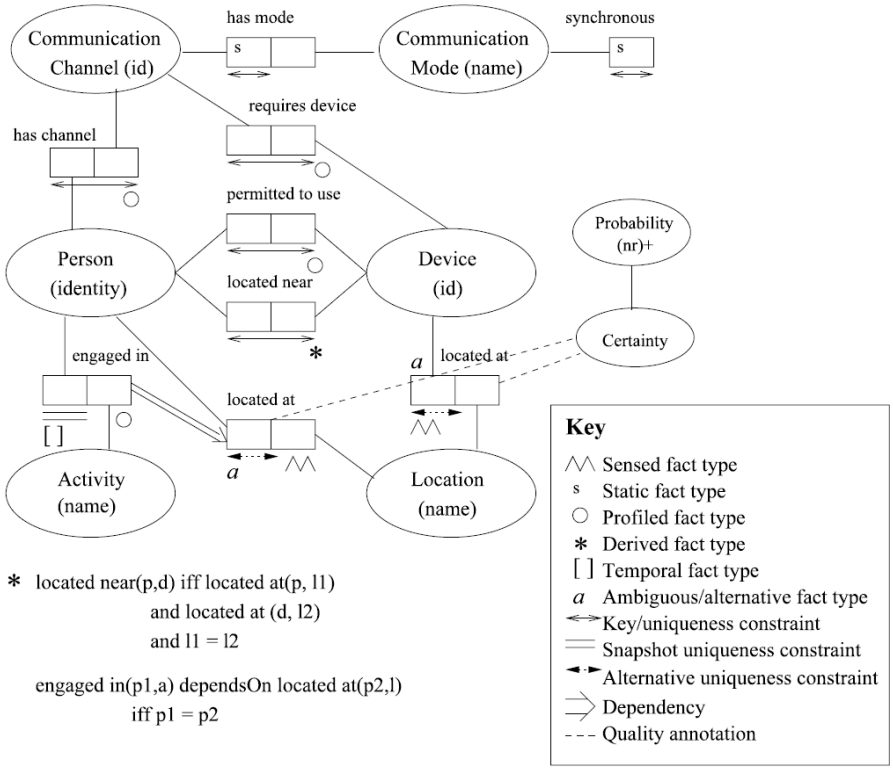


Figure 4.13.: Example of CML model [BBH+10]. Ellipses represent object types, while rectangles express relations or fact types. The Key box annotates all types defined for each fact type.

Figure 4.13 depicts the CML model. Note that CML features the ambiguous/alternative fact type. This property can help us identify the uncertain context information. CML also supports a high-level context abstraction. It detects a specific situation defined by logical expressions. For example, a device at location2 (d at l2) is defined to be nearby a person at location1 (p at l1) when both of them are detected at the same location (l1=l2).

b. **Spatial-based context model** refers to location-based context information, which is one of the three main aspects of context defined by [Dey01]. Places are spatial entities, and interaction typically requires some vicinity. Spatial context can refer to either a physical location, like geometric coordinates, or a descriptive location such as room numbers, network access point ID. The important spatial properties for reasoning are a position, range, and nearest neighbor.

   NEXUS [NM01], and Equator [MDRS05] present spatial-based models. However, the high-level context abstraction to deal with such situation recognition is not adequately supported. Moreover, spatial context takes enormous efforts to gather data and keep them up-to-date.

c. **Ontology-based context model** has its strength in the expressiveness for logical expressions and availability of reasoners. OWL-DL [HPSVH03] can describe a high abstraction situation, e.g. a business meeting event. An event is identified as a business meeting when it contains at least two participants who are employees of a company. Plus, the location for holding the event is a conference room inside the company's building. SWRL [HPSB$^+$04] (Semantic Web Rule Language), an extension of OWL-DL, offers even more expressive logical expressions. However, these ontological-based models are computationally expensive in reasoning. Therefore, using solely ontological-based model can lead to a performance issue.

None of these approaches adequately addresses the uncertainty of context information and the high-level context abstraction. There are many approaches to help estimate the context information from uncertain values. For example, Fuzzy Logic, Probabilistic Logic, Bayesian Networks, Hidden Markov Models, and Dempster-Shafer Theory, can be used when we have enough raw data and statistical information to calculate the probabilistic figures. The high-level context abstraction, as depicted in Figure 4.14, must be interpreted from the sensor-based context. Such complexity requires an ontological-based model to analyze the semantic meaning and realize the relationship between entities.

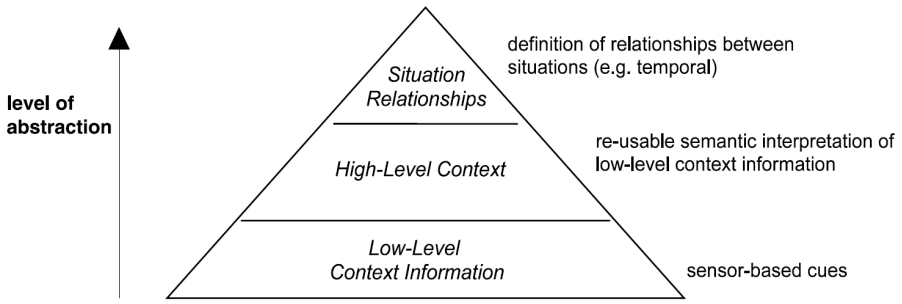


Figure 4.14.: Different layers of semantic context interpretation and abstraction [BBH$^+$10].

To get the advantages of all context models and eliminate weaknesses of each, hybrid approaches have been studied and implemented e.g. a fact-based/ontological model like CML [HLI04], and a loosely coupled markup-based/ontological model like CARE [BMR07]. These approaches can balance between the expressiveness and performance. More importantly, they also support the uncertainty of context information.

4.5.2. Context-aware Applications

Several approaches aim to include context information in a resource discovery. For example, [XZNN10] proposes the context relation model to improve the result of the resource discovery, and it can work with several description formalisms. However, this work does need users to explicitly provide the context relevant to the request.

[RLS+11] proposed the context-driven personalized resource discovery model. It focuses on utilizing context information to match with spatial and temporal information of services. This work uses a home automation scenario to demonstrate the model. Figure 4.15 visualizes the usage of the spatial and temporal context of a user to control the status of television. They also implemented a context-aware description format as exemplified in Listing 4.2. Although this description allows semantic annotation, the formalism is not yet commonly used. It should be able to handle the standard formalisms like OWL-S, WSDL, and WSMO.

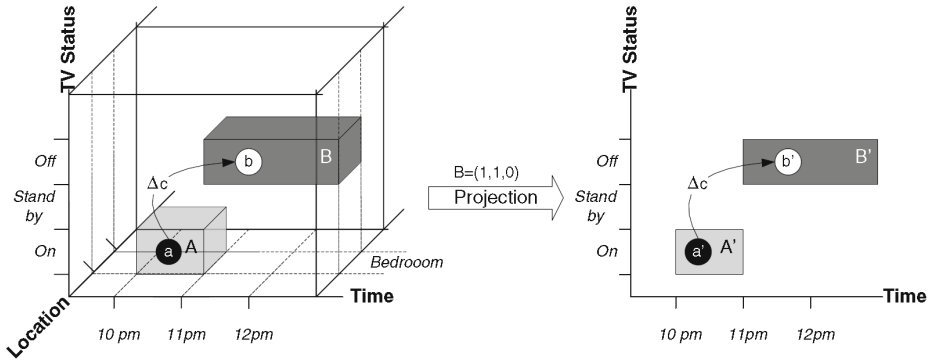


Figure 4.15.: Visualization of spatial and temporal context in a home automation application [RLS+11]. Left: TV status regarding user's location and time of the day. Right: a simplified model for TV status considering only the time of the day.

```
@Path(''coffeeMachine'')
@ContextAwareDevice (''myCoffeeMachine'')
@ContextTypes ({@ContextType=''status'',
  values = ''http://www.sm4all−project.eu/types.owl#CoffeeMachineStatus''})

public interface CoffeeMachine
{
    @POST
    @Path(''coffee'')
    @ContextAwareService (name=''Make\_coffee'',icon=''coffeeCup .jpg'')

    @Precondition ({@ContextElement(type=''status'',value=''idle'')} ,
                   {@ContextElement(type=''Location'',value=''Kitchen'')})
    @Effect ({@ContextElement(type=''status'',value=''in\_use'' )})

    void makeCoffee(String coffeeType);
    ...
}
```

Listing 4.2: Description of a context-aware coffee machine [RLS+11].

[WJ12] proposes a design to utilize context, which is derived from resources and users profile, in the resource discovery for IoT. The main focus of [WJ12] is to support uncertain and temporal context by adopting Dynamic Bayesian networks.

TRENDY [BPGO13] offers a promising solution to adapt user context awareness to the resource discovery for IoT. Figure 4.16 shows the architecture of TRENDY. A Directory Agent (DA) stores all services and contextual information such as service descriptions, location, power consumption, and response time. The selection of GL (Group Leader) and GM (Group Member) is based on context information, i.e. location. Similar to [RLS+11], TRENDY's service discovery does not conform to any standard resource description.

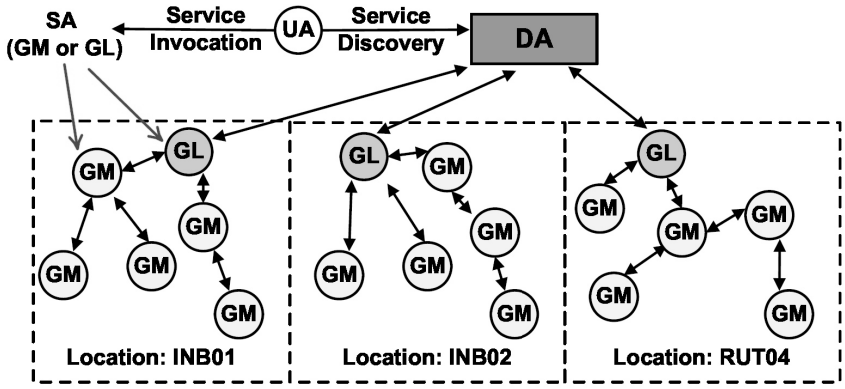


Figure 4.16.: Architecture of TRENDY [BPGO13] (DA stands for Directory Agent; SA, Service Agent; UA, User Agent; GM, Group Member (service); and GL, Group Leader).

Based on the approaches and techniques reviewed in this chapter, we can see that the thesis requirements mentioned in the previous chapter can close the gap of improvement. Also, we can use the existing works such as middlewares and resource matchers to complete our system rather than develop a new one from scratch. In the following part, an overview of the implemented solution is provided, then follow with the implementation detail of each component.

Part II.

Solution

5
Solution Overview

In the previous chapters, we presented the motivation of this thesis, introduced the resource discovery, and research questions. We reviewed the existing works which tried to solve similar issues and identified their weak points. In this chapter, we provide an overview of resource discovery, the architecture, and the principal components. These components are a context extractor, a request constructor, a request converter, and a result integrator.

The architecture of the resource discovery is introduced in Section 5.1. This architecture overview leads us to the conclusion as to which parts we need to implement and which parts we can make use of the existing resources. Also, we discuss description languages for the proof-of-concept implementation in Section 5.2, *Supported Resource Description Formalisms*. Accordingly, corresponding resource matchers will be chosen in Section 5.3, *Supported Resource Matchers*. Thus, the essential information required by these matchers can be determined. Lastly, the overview of each component in the resource discovery is introduced in Section 5.4, *Main Components*.

5.1 Resource Discovery

As mentioned in Chapter 2, *Project Background*, this thesis focuses on the resource discovery. We also pointed out in Chapter 4, *State of the Art* that the context-adaptive automatic discovery process has not been adequately supported. To provide this support, we implemented the automated resource discovery, using machine-interpretable resource descriptions and semantic annotations. We need to create a resource request, which contains the resource characteristics a user is interested in, in a predefined format and feed it to a matching component. Additionally, user profiles and environmental conditions should be taken into account to improve the matching result.

However, automatic actions can easily lead to undesired results. Users should be involved in resolving conflicts, and the resource discovery engine only suggests appropriate resources relevant in any given context. This suggestion helps the user to make decisions without being overwhelmed by irrelevant resources.

First, we need to receive free-text search keywords from a user and then create a formatted request to resolve Requirement R1 (*the resource discovery unit should be able to construct*

a free-text query message from end users into pre-defined formats). This process can be completed by the request constructor. In this step, we can include user context into the request to scope down relevant results. To achieve this, we need to define where the context should come from (R6, *the source of user context should be defined*), obtain them using the context extractor, and applied them to a search query (R7, *when user context or resource context is available, they should be applied to the search query*). Moreover, we can use the semantic annotation to improve the matching process as suggested by [MGMR02] and [QHC06].

Second, to cope with cross-formalism discovery, the formulated request should be convertible to other formalisms. The request converter handles the conversion between different formalisms. This step needs Requirement R2 (*the resource discovery should be able to interpret different description formalisms*) as a prerequisite for understanding the supported formalisms. Only after that, we can achieve Requirement R3 (*the resource discovery should handle multiple description formalisms and matchers simultaneously*).

Third, since the existing matchers are not supporting multiple formalisms, we utilize multiple resource matching engines to ensure the reliability of matching results. This idea has been supported by many works like meta-search engines reviewed in [LMS+05] and [Jad12]. Also, an ontology matching solution like [KNL13] utilizes multiple matchers in order to achieve the better result.

Finally, when multiple matchers process the same request, it is necessary to combine their results into a single list. The outcome of the discovery should be presented to users depending on the context of use as required by R12 (*the discovery result should be presented in the registration, scenario modeling, and execution processes in a way that users can apply the result instantly*). We demonstrate the usage of the resource discovery by integrating it to MERCURY in Chapter 10, *Resource Discovery Integration to MERCURY*.

5.1.1. Architecture

We presented an architecture of the resource discovery in [OEKR+14] as depicted in Figure 5.1. A request is created via the Request UI, which requires resource input, output or operation keywords from users. It is not necessary to provide all of them, at least one keyword is needed. If the keywords are given without specification of their functions, they will be considered as general keywords. A general keyword is meant to be an attribute value that can appear in any field of each resource description.

The context extractor retrieves user context and appends to the keywords provided by users. Then, all keywords will be extended with synonyms obtained from a semantic knowledge base. The keywords would be constructed into a predefined format - either OWL-S or SAWSDL by the request constructor. Supported formalisms by this work are elaborated on in Section 5.2.

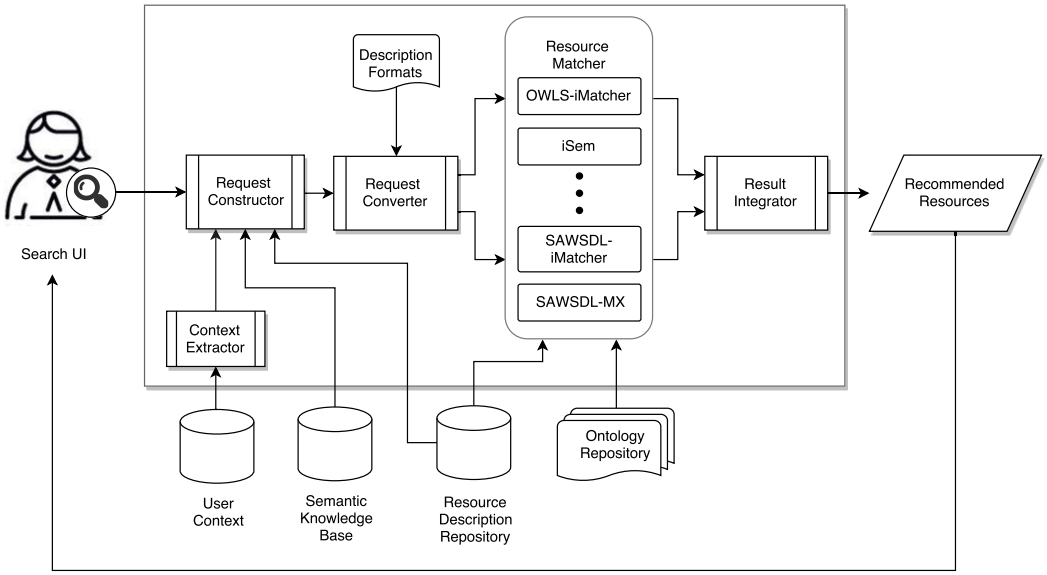


Figure 5.1.: Architecture of resource discovery.

Next, depending on resource matchers selected by a user, the request converter may need to prepare a message in supplementary formalism. For example, if the constructed format is SAWSDL, and a resource matcher is OWL-S based, the request converter should create an OWL-S request. See Section 5.3 for more explanation on the resource matchers we utilize in this thesis.

Now the request message is ready to be matched with the description in a repository. The resource description repository contains descriptions from several sources, such as registered resources in MERCURY, Programmable Web[1], or Mashery API Network[2]. Descriptions in the repository can be retrieved automatically from these sources[3]. Each resource matcher will return a different list of matching results. These lists will be merged within the result integrator, thus producing a final list of recommended resources.

The context extractor, the request constructor, the request converter, and the result integrator are four main components implemented in this work. They will be discussed as a whole again in Section 5.4.

[1] www.programmableweb.com/

[2] support.mashery.com/io-docs

[3] As the time this dissertation is conducted, there is no archive of descriptions available for download. Thus, we use an automated script to crawl the web service collections and to extract the description files one by one.

5.1.2. Initial Assumptions

In this thesis, we assume that all sensors and actuators are available as web services. This management of devices via web services can be achieved by a middleware like GSN, as explained in Chapter 4.

We also assume that the minimum information available for any web service is WSDL [CCMW] or OWL-S [MPW07] description. WSDL descriptions are necessary for the invocation of the services. Here, we assume that such description is available for each resource, but we do not make assumptions about the formalism used. MERCURY gives users the possibility to add descriptions to previously undescribed resources so that over time, the assumption that some descriptions are available can be realistic.

Also, the domain of ontology for resources needs to be defined. Because the request that contains semantically ambiguous descriptions can lead to wrong results. For instance, the term "book" can be defined as either a collection of sheets of paper bound together containing printed or written material, or an action of reserving something for future use. To eliminate the ambiguity, we assume that the installation of MERCURY will be supported by a certain business purpose, such as publishing domain or travel agency. Accordingly, the resource discovery unit can correctly assign the ontology to a request.

Now, we have noticed that it is necessary to decide on which description formalisms we will take into account. This enables us to investigate further as to which information is needed to construct the simplest yet functioning description.

5.2 Supported Resource Description Formalisms

We assume that basic descriptions like WSDL should be provided in our context. Even so, a WSDL description alone could be insufficient, since it aims mostly for human and does not contain adequate information for automated resource discovery. The AI (Artificial Intelligence) community has developed many ontology-based techniques such as OWL-S [MPW07] to solve these dilemmas. Even so, this is a rather heavyweight approach, which describes a service's capabilities in terms of its preconditions and effects, in addition to the service's interface. Although the description uses a powerful logic language, it still requires significant effort to create. Nevertheless, OWL-S is still widely used to describe and annotate web services. Additionally, WSMO [RKL$^+$05] (Web Service Modeling Ontology) formalism could be supported in future improvements. For now, we opt to study OWL-S because of its popularity and availability.

Lightweight semantic resource descriptions like SAWSDL [HS12] (Semantic Annotations for WSDL) could enhance WSDL descriptions by allowing semantic annotations using arbitrary ontologies. SAWSDL is a hybrid description that provides both syntactic and semantic discovery of services. Besides, WSMOLite [FFST11] is developed from WSMO

```
<rdf:RDF ...>
    ...
    <service:Service rdf:ID="CITY_WEATHER_SERVICE">
        <service:presents rdf:resource="#GET_WEATHER"/> ... </service:Service>
    <profile:Profile rdf:ID="GET_WEATHER">
        ...
        <profile:hasInput rdf:resource="#_CITY"/>
        <profile:hasOutput rdf:resource="#_WEATHER"/>
    </profile:Profile>
    <process:Input rdf:ID="_CITY">
        <process:parameterType>SUMO.owl#City</process:parameterType>
    </process:Input>
    <process:Output rdf:ID="_WEATHER">
        <process:parameterType>Mid-level-ontology.owl#Weather
        </process:parameterType>
    </process:Output>
    ...
</rdf:RDF>
```

Listing 5.1: Sample description in OWL-S.

to provide a lightweight ontology-based description. However, we choose to use SAWSDL in our work because it is more mature and there are more resource matchers based on SAWSDL available.

In addition, non-ontological based formalisms like [GCPG12] and [DLY+10] have no particular standard and rely totally on human interaction, e.g. tagging. The cold start problem, in particular, is the main blockage for their matching process.

Since we want to integrate arbitrary resources available over the Internet, we cannot make any assumptions about the description framework. Therefore, we try to support several different description formalisms, currently including OWL-S and SAWSDL.

Examples of OWL-S, SAWSDL1.1, and SAWSDL2.0 descriptions are shown in Listings 5.1, 5.2, and 5.3 respectively. We analyze these formalisms in more detail and conclude which parts of OWL-S and SAWSDL descriptions are necessary for the discovery process in Section 7.2, *Essential information required for resource matching*.

```
<wsdl:definitions ...>
   <wsdl:types> <xsd:schema>
         <xsd:complexType name="CityType"
         sawsdl:modelReference="SUMO.owl#City"/>
         <xsd:complexType name="WeatherType"
         sawsdl:modelReference="Mid-level-ontology.owl#Weather"/>
   </xsd:schema> </wsdl:types>
   <wsdl:message name="get_WEATHERRequest">
      <wsdl:part name="_CITY" type="CityType"/>
   </wsdl:message>
   <wsdl:message name="get_WEATHERResponse">
      <wsdl:part name="_WEATHER" type="WeatherType"/>
   </wsdl:message>
   <wsdl:portType name="CityWeatherSoap">
      <wsdl:operation name="get_WEATHER">
         <wsdl:input message="get_WEATHERRequest"/>
         <wsdl:output message="get_WEATHERResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:service name="CityWeatherService">
      <wsdl:port name="CityWeatherSoap"> ... </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

Listing 5.2: Sample description in SAWSDL1.1

```
<wsdl:description ...>
   <wsdl:documentation> ... </wsdl:documentation>
   <wsdl:types>
      <xs:schema>
         <xs:element name="_CITY"
         sawsdl:modelReference="SUMO.owl#City"/>
         <xs:element name="_WEATHER"
         sawsdl:modelReference="Mid-level-ontology.owl#Weather"/>
      </xs:schema>
   </wsdl:types>
   <wsdl:interface name = "CityWeatherSoap" >
      <wsdl:operation name="get_WEATHER"
      pattern="http://www.w3.org/ns/wsdl/in-out"
      style="http://www.w3.org/ns/wsdl/style/iri"
      wsdl:safe = "true">
         <wsdl:input messageLabel="In" element="_CITY" />
         <wsdl:output messageLabel="Out" element="_WEATHER" />
      </wsdl:operation>
   </wsdl:interface>
   <wsdl:binding ...> ... </wsdl:binding>
   <wsdl:service name="CityWeatherService"
        interface="tns:CityWeatherSoap"> ... </wsdl:service>
</wsdl:description>
```

Listing 5.3: Sample description in SAWSDL2.0

5.3 Supported Resource Matchers

To utilize existing resource descriptions and to perform resource matching, appropriate requests are needed. These requests should be composed in one of the formal languages. The crucial information of resource descriptions can be categorized into three groups: input (information needed for the resource), output (information provided by the resource), and operation (characteristics of the resource). We assist regular users to create requests in OWL-S or SAWSDL format and forward them to the resource matcher module.

As introduced in Chapter 4, we collected resource matchers from S3 Contest [Klu12] in OWL-S and SAWSDL tracks, as shown in Table 5.1. We chose matchers from S3 Contest because these matchers are provided as plugins which are easy to be deployed/undeployed and maintained. Moreover, S3 Contest also provides testing benchmarks which are judged by domain experts. Therefore, we can verify our outcome using such benchmarks.

Formalism	Matchers	Author(s)
OWL-S	OWLS-MX	[KFK05]
	OWLS-M0	[KFK05]
	OWLS-MX2	[KKF08]
	OWLS-MX3	[KK12a]
	OWLS-iMatcher	[KB08]
	OWLS-SLR Lite	[MB10]
	XSSD	[Li13]
	SPARQLent	[Sbo12]
	SeMa2	[MHB+12]
	iSeM	[KK10]
	EMMA	[GRRC12]
SAWSDL	SAWSDL-iMatcher	[WWWB11]
	iSem	[KK12b]
	SAWSDL-M0	[KKZ09a]
	SAWSDL-MX	[KKZ09a]
	SAWSDL-MX2	[KKZ09b]
	XAM4SWS-COV4SWS	[SLKS12]
	XAM4SWS-LOG4SWS	[SLES10]
	URBE	[PP09]

Table 5.1.: List of resource matchers from S3 Contest in OWL-S and SAWSDL tracks.

Let us investigate SAWSDL-M0 and iSem matchers to see how individual matcher works. We will see later that each matcher takes different criteria for judging, whether two descriptions are matched or not.

- A SAWSDL-M0 matcher matches a description without considering semantic annotations. Taking an example from Table 5.2, we have a request for a weather service

which accepts a country and city name as input and produces a weather report as output. The perfect match for the request would be offer #4 since it has the same inputs, output and operation descriptions.

- A country name can be defined as a geopolitical entity. *modelReference* attribute contains a link to an ontology which explains the relationship between a "geopolitical entity" and "country". However, when the matcher does not use an ontology-based algorithm, it will overlook offer #1 even though offer #1 is semantically matched with the request.

- iSeM Matcher can recognize offers #1 and #2 as relevant resources since it interprets the semantic annotations in descriptions.

We can see that each matcher requires different information for the matching process. Moreover, the ranking algorithm could also be different.

- For example, offers #1 and #4 seem to be perfectly matched descriptions. However, some matchers would prefer offer #4 than offer #1 because offer #4 is matched without semantic knowledge. In contrast, some matchers would perceive no difference and thus rank them alphabetically. As a result, offer #1 can get a higher rank than offer #4.

- Offer #2 is missing "city" input, while offer #3 has a mismatching operation description. If a matcher aims for matching input and output descriptions, it will find offer #3 more relevant than #2. Contrarily, offer #2 can get a higher rank than offer #3 when the matcher prioritizes the operation description over the input and output descriptions.

These are only a few examples from two resource matchers. Other matchers use different criteria for judging the match and ranking the results. Because of the various approaches, there is no certain algorithm which yields the best result for every request. Thus, utilizing different description formalisms and multiple matchers may increase the possibility of discovering all suitable resources.

So far, the initial assumptions are set, the supported formalisms are specified. We also provide an overview of resource matchers in Section 7.1, *Matcher Analysis*. An outcome of this investigation will help us determine which matchers we can use for the evaluation and which part of information inside descriptions are required. Next, we can implement the main components based on this information.

Resource Request

```
<types><schema...>
    <inputType name="_COUNTRY"/>
    <inputType name="_CITY"/>
    <outputType name="_WEATHER"/>
</schema></types>
    <operation name="get_WEATHER"/>
```

Resource Offer #1

```
<types><schema...>
    <inputType name="_GEOPOLITICAL-ENTITY"
     modelReference="portal.owl#Geopolitical-Entity"/>
    <inputType name="_CITY" modelReference="travel.owl#City"/>
    <outputType name="_WEATHER"
     modelReference ="Mid-level-ontology.owl#Weather"/>
</schema></types>
    <operation name="get_WEATHER"/>
```

Resource Offer #2

```
<types><schema...>
    <inputType name="_GEOPOLITICAL-ENTITY"
     modelReference="portal.owl#Geopolitical-Entity"/>
    <outputType name="_WEATHER"
     modelReference ="Mid-level-ontology.owl#Weather"/>
</schema></types>
    <operation name="get_WEATHER"/>
```

Resource Offer #3

```
<types><schema...>
    <inputType name="_COUNTRY"
     modelReference="portal.owl#Country"/>
    <inputType name="_CITY" modelReference="travel.owl#City"/>
    <outputType name="_ACCOMMODATION"
     modelReference="travel.owl#Accommodation"/>
</schema></types>
    <operation name="get_ACCOMMODATION"/>
```

Resource Offer #4

```
<types><schema...>
    <inputType name="_COUNTRY"
     modelReference="portal.owl#Country"/>
    <inputType name="_CITY" modelReference="travel.owl#City"/>
    <outputType name="_WEATHER"
     modelReference ="Mid-level-ontology.owl#Weather"/>
</schema></types>
    <operation name="get_WEATHER"/>
```

Table 5.2.: Simplified version of SAWSDL sample requests and offers.

5.4 Main Components

The four main building blocks of the resource discovery implemented in this thesis are a context extractor, a request constructor, a request converter, and a result integrator. The request constructor creates a standardized description from free-text keywords. Meanwhile, the context extractor can retrieve user or resource context. Then we can append this data to the request description.

Each resource may have more than one type of description. However, the users would not need to specify the standard of description for the resource they are looking for. We thus cope with all possible (and prominent) matching techniques by converting the request into various formats using the request converter. The matchers compare the request message with description files, which are already known by the resource matchers. Finally, the results returned from each matcher will be merged and sorted again by the result integrator.

5.4.1. Context Extractor

To automate the discovery process and filter down the relevant result, the context of user and resource should be taken into account. As introduced in Section 4.5, *Context-aware Resource Discovery*, we aim to use a hybrid approach of object-oriented-based, spatial-based and ontology-based context models. The first challenge is how to retrieve such information explicitly from a user. Then, we need to find out how to apply them to the resource discovery process.

We could use the user profile from the portal infrastructure's user model to obtain basic information such as address, email, and birthday. Besides, we can create a *social sensor* which analyzes the content of each user's social network statuses. These approaches can be applied depending on user's consent. In Chapter 6, *Context Extraction*, we elaborate on this module in more detail.

5.4.2. Request Constructor

This component is responsible for constructing free-text keywords into predefined formalisms. First, we need to study OWL-S, and SAWSDL (1.1, 2.0) to get the essential part of the information (see Section 7.2, *Essential information required for resource matching*). We conclude that the necessary details are input, output, and operation descriptions.

The final product from this component should have a structure like Listing 5.1 for OWL-S, 5.2 for SAWSDL1.1 or 5.3 for SAWSDL2.0 All these descriptions contain similar specifications for the same resource. They only differ in syntax and structure. Since OWL-S is usually more expressive, yet computationally expensive, SAWSDL is more preferable. Nonetheless, the default formalism should be customizable via configuration settings. In Section 8.1, *Request Constructor*, we elaborate on how the constructor creates formatted descriptions.

5.4.3. Request Converter

Although the request constructor is designed to construct a request in any supported formalism, it is more resource-effective to base the construction on a single formalism. Then, we use the request converter only when necessary. The relationship between formalisms will be summarized in Section 8.2, *Request Converter*, as well as the conversion algorithm.

5.4.4. Result Integrator

When we use several resource matchers simultaneously, the result integrator applies weighing parameters to all matchers. All similarity scores of each web service returned from all resource matchers will be used to rearrange the ranking result. Once again, the user context can play a prominent role here, in order to match the relevance of resource recommendations to the needs of users in each circumstance.

One major challenge for merging two results is a conflict in ranking. For example, the score of results from matcher "A" is $WS_A = [0.9, 0.7, 0.93, 0]$. Each element in the array represents the similarity level of each resource compared to the request. And the score of results from matcher "B" is $WS_B = [0.65, 1.0, 0.7, 0.1]$. Moreover, when matcher "B" becomes more reliable than matcher "A", the resource discovery should combine the results corresponding to this fact. The solution to this dilemma will be elaborated on in Chapter 9, *Result Integration*.

In the upcoming chapters, each main component will be elaborated on. Then, all the implemented modules are integrated and used in Chapter 10, *Resource Discovery Integration to MERCURY*. There, we explain how resource discovery can be used in the context of MERCURY. Also, the evaluation results of each module and the integrated system are discussed in Part III, *Evaluation*.

6 Context Extraction

Since our goal is to create an environment-aware resource discovery, it is important to detect the user context automatically. When the service discovery engine works together with MERCURY, we can make use of the user management model supported by Portal's infrastructure to cultivate user context. [BDH+09] and [AIM10] discuss the concept and concrete use cases for deriving semantic presence based on context from sensor-enabled social networking devices. [NGS+09], [She09], and [ASS+11] also affirm the usefulness and richness of social sensing. These studies lead us to the utilization of the social network as a social sensor. Such sensors nowadays offer location tagging, friends tagging, and timestamp information. We can analyze the neighborhood and companions of a user from them.

Furthermore, we can keep track of the user's interests and actions to create a resources recommendation. The application of this recommendation will be demonstrated again in Chapter 10, *Resource Discovery Integration to MERCURY*.

6.1 Context from User Profile

The context information is derivable from a web portal's user management model as shown in Figure 6.1, e.g. user address, telephone number, e-mail address. Users can choose to apply this context information to enhance the resource discovery process. When a user agrees to apply context information, this information will be appended to the query message.

Furthermore, the user contact model on the portal provides the social network account information, e.g. facebookSn, skypeSn, twitterSn, where *Sn* stands for 'serial number'. Based on this data, we can connect to available social networks and retrieve user context. In other words, these accounts can be used as social sensors.

Nevertheless, not every piece of information is shareable. The granularity of shareable user's detail is highly dependent on individual privacy settings, such as a posted message is set to be seen by everyone or just a group of people. By default, anyone who knows the social network id/username of a particular person can see his information. On the other hand, to restrict the publicity, the owner of the account must consent to the information sharing.

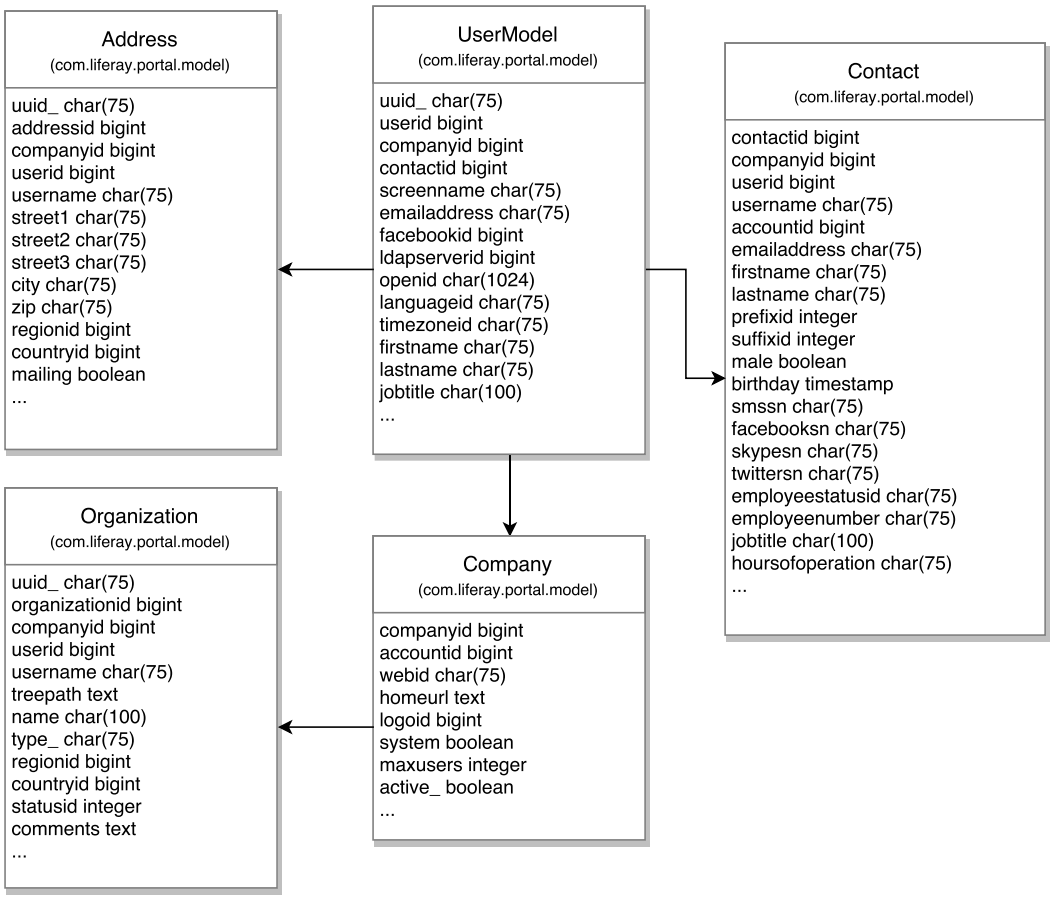


Figure 6.1.: User Model and detailed information provided by Liferay Portal.

6.2 Context from Social Sensors

Based on a social network id from the user profile, the context like the recent location or latest activity of the user can be acquired. Here, we provide two examples[1]: Twitter and Facebook. Following the concept of context of IoT in [PZCG14], the derivable contexts are recent contacts, recent places, the latest place (with a friend) and check if the user is near to a specific location. To access this personal information, an authentication process is required. For both social sensors, the context extraction method via Twitter API [Twi] and Facebook API [Fac] are quite similar. However, they differ in the authentication process.

[1]The concept and implementation in this chapter are adapted from [Sat14].

6.2.1. Context Extraction via Twitter API

The context extraction via Twitter API is straightforward, and the process flow is shown in Figure 6.2. Every session of an API call must begin with authentication. Only after the authentication is successful can the other functions be called.

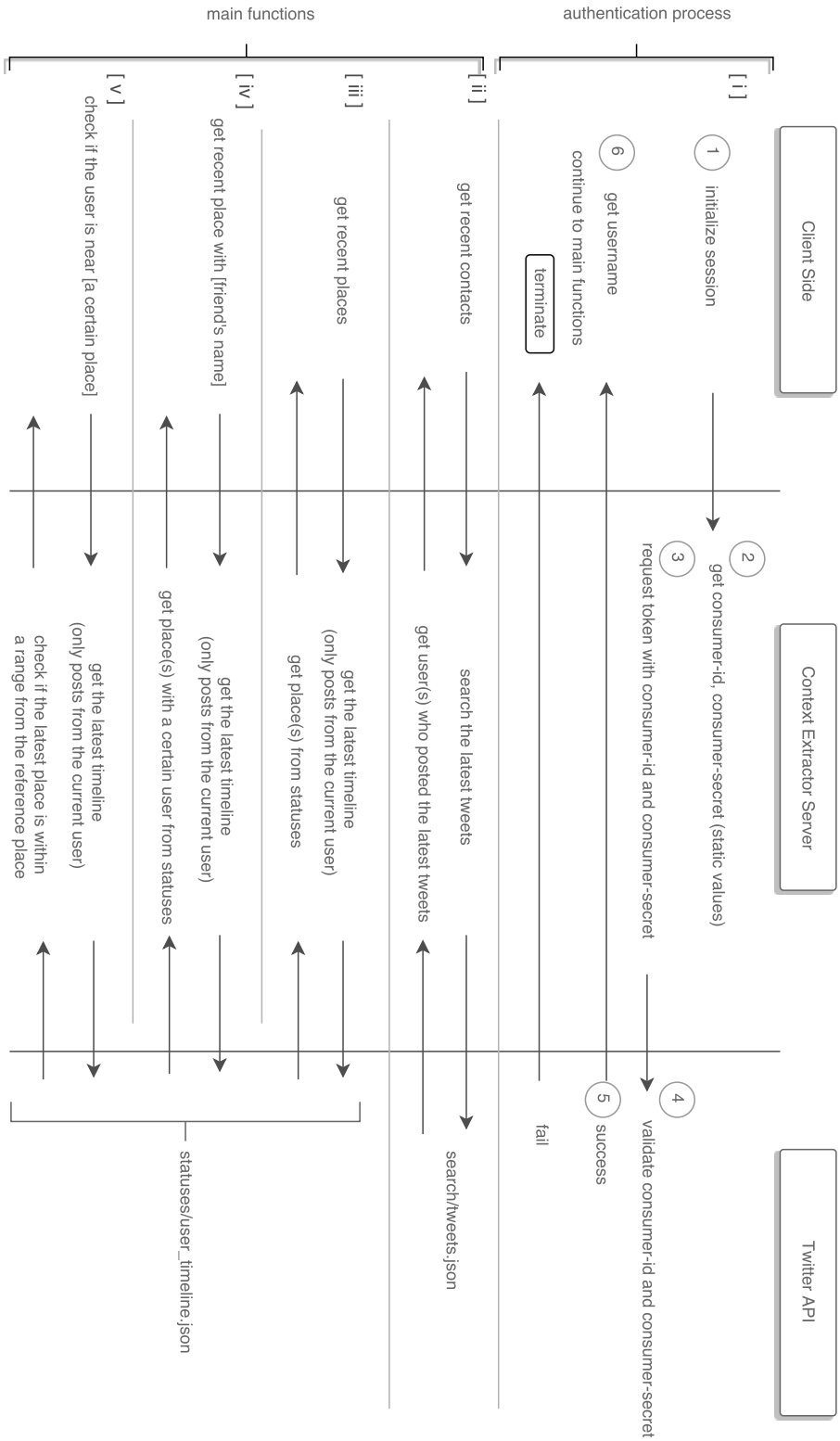


Figure 6.2.: Data flow of a social sensor by Twitter.

i. The authentication process requires only a *consumer-id* and *consumer-secret*, which is one-time generated per application (permanent). A token, which is valid for one session (temporary), will be returned to the client. A username can be retrieved from the user contact model.

   Accordingly, we can send a request to Twitter API to get recent tweets (posts by the contacts this user is following) or recent timeline (what this user tweeted).

ii. Recent contacts can be extracted from recent tweets as illustrated in Figure 6.3. The number of retrieved tweets can be determined, e.g. find contacts who tweeted from the latest 100 tweets.

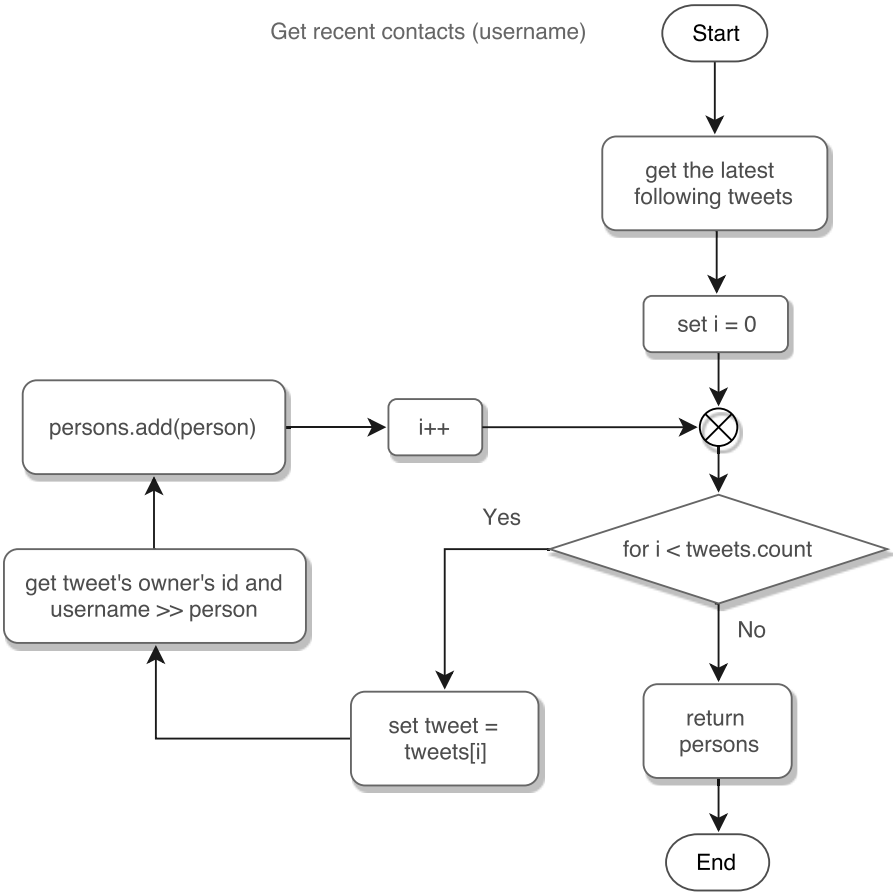


Figure 6.3.: 'Get recent contacts' function from Twitter.

iii. Recent places are retrieved from the latest timeline. The number and the maximum age of retrieving statuses are customizable. Nonetheless, the location feature can be turned off for each tweet. The status that contains no location detail is neglected. Figure 6.4 illustrates the workflow of this function. The geo-coordination returned from Twitter API comes as a bounding box value. It contains min-Latitude, max-Latitude, min-Longitude, and max-Longitude values. We assume that the exact position of a user can be roughly estimated from the center point of the bounding box. The list of recent places is sorted in chronological order (i.e. the latest place is in the first order).

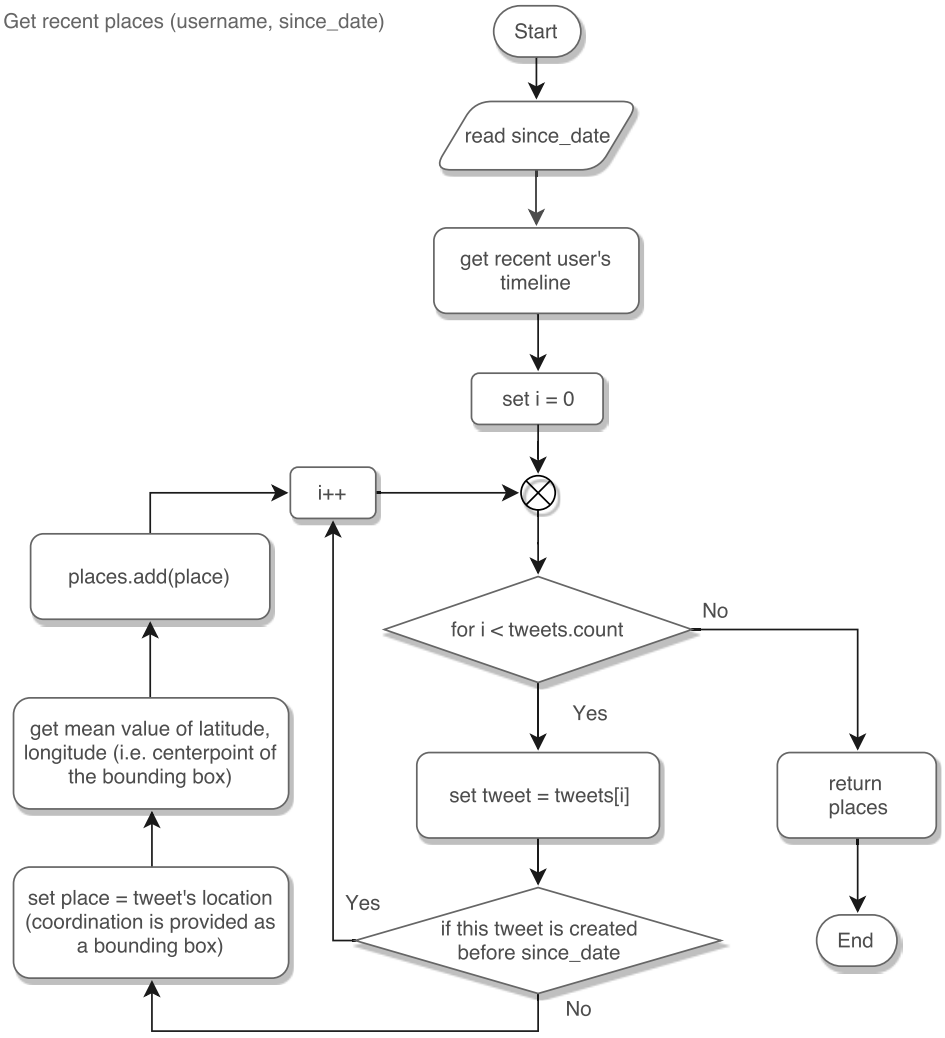


Figure 6.4.: 'Get recent places' function from Twitter.

iv. 'A recent place with another user' can be identified via a 'mention to' field. We could see the result only if the user mentioned his friend in his tweet with a location detail. Similar to the 'get recent places' function, the recent places are refined from the latest timeline. But we need an additional criterion, i.e. the user must mention to a specific user. Figure 6.5 shows the flow chart of the function.

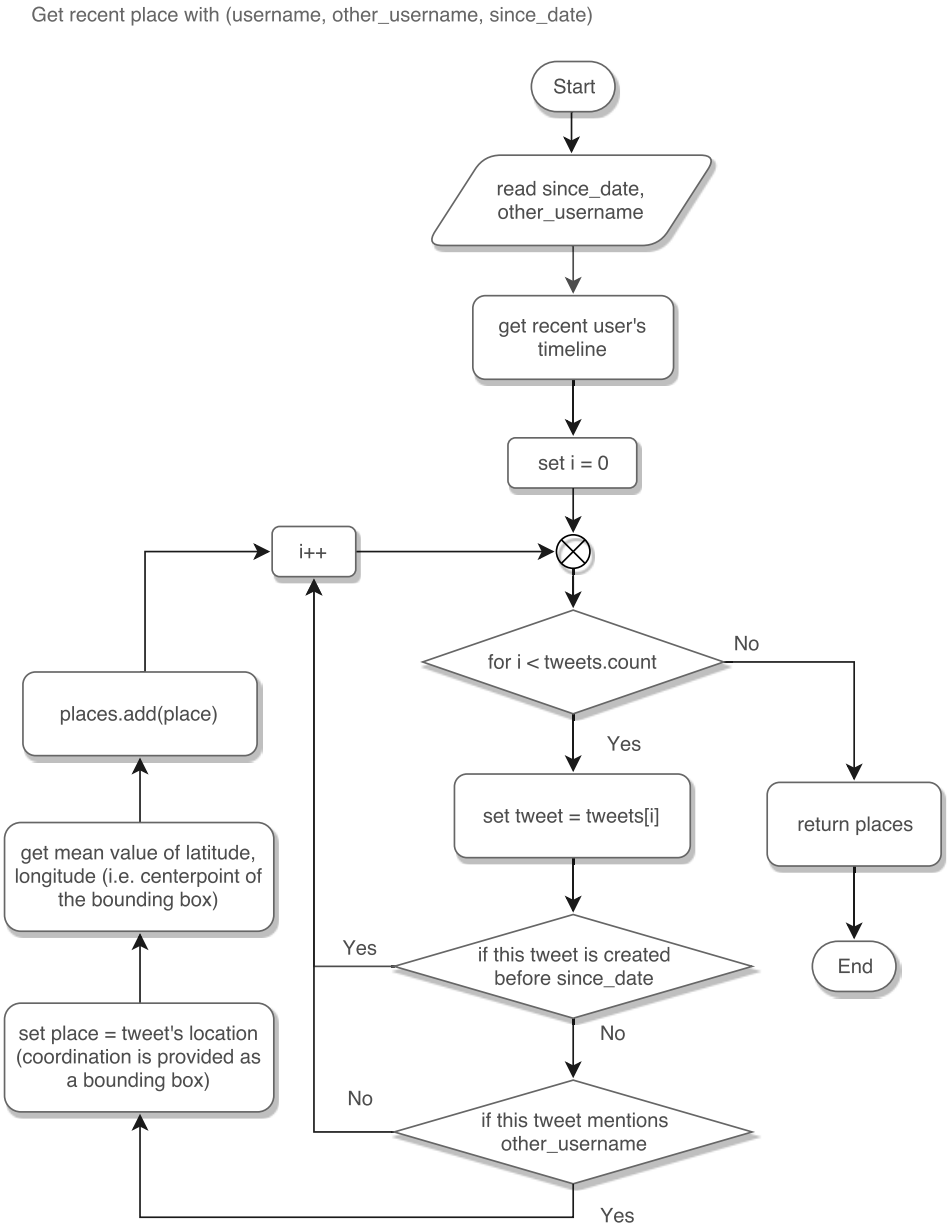


Figure 6.5.: 'Get a recent place with [another user]' function from Twitter.

v. To check if the user is near a specific site, the latest place is identified. Also, the specific location must be provided with the maximum range, defining the acceptable distance from the reference location. For example, if a user is within 2 km from home or an airport. The workflow of this function is depicted in Figure 6.6. Again, this calculation is done using the center point of geo-coordination values. For a precise outcome, the user should specify the place with a more accurate location. For instance, the name ”crescent moon beach” is preferable to ”paradise island” regarding the geographic size.

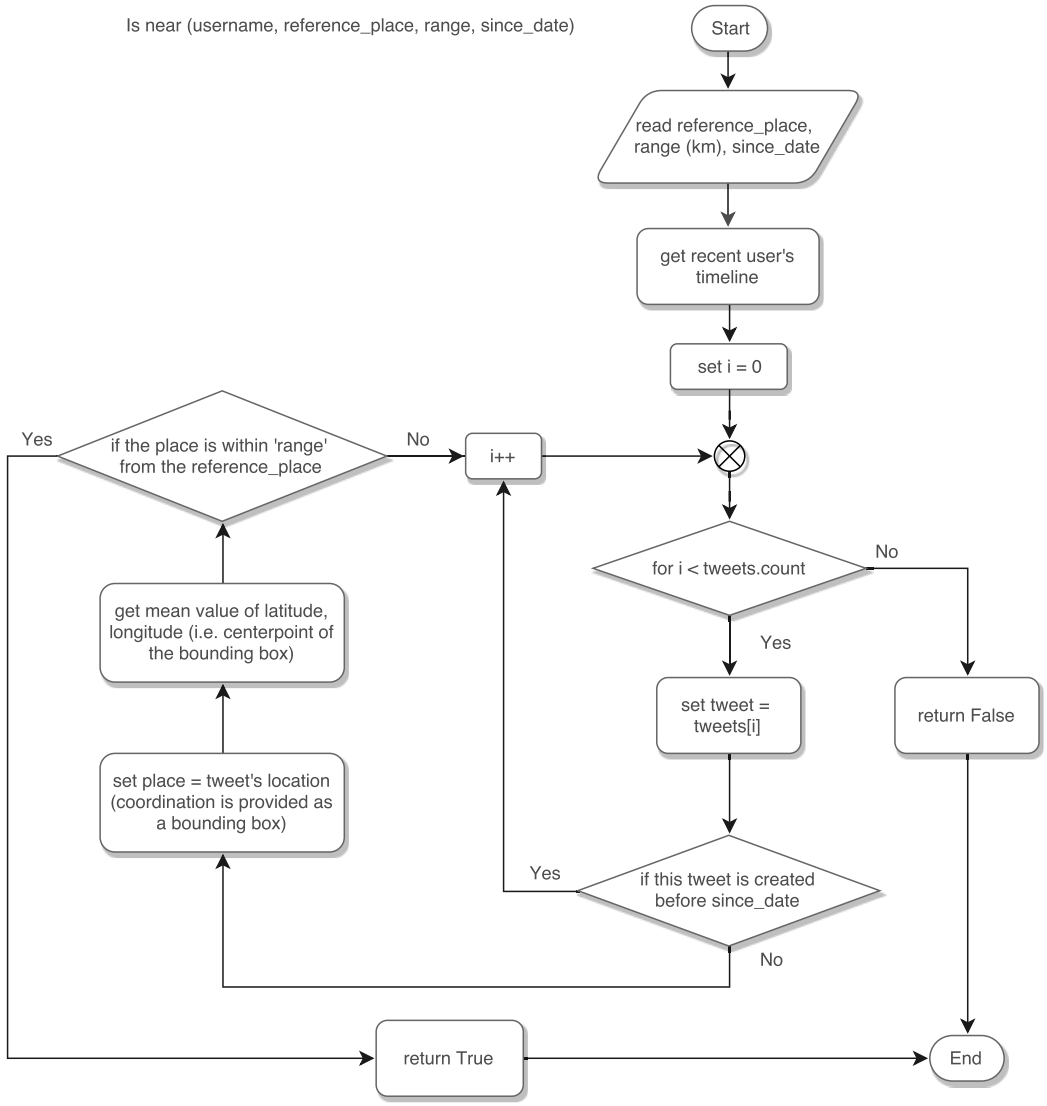


Figure 6.6.: ’Check if the user is near to [a particular location]’ function from Twitter.

It is worth noting that the retrievable context depends on the extent of the API. Thus, there are still some gaps for improvement, depending on the new features available. The context extractor for a Twitter script, implemented in Python, can be found in Listing C.1. Also, the testing script for a client application can be found in Listing C.2.

6.2.2. Context Extraction via Facebook API

The core idea of context extraction is the same for all social sensors. However, Facebook API does not allow third-party applications to retrieve user information directly. It requires every user to identify himself via a formal Facebook login process. When the login process is successful, the API returns a unique token specific to each user to the pre-defined redirect URL, not to the requester.

Nonetheless, this complication exists only in the authentication step. Afterwards, this private token can be used throughout the session.

i. Figure 6.7 depicts the authentication process. First, a third-party application needs to apply for an application id and application secret (one-time request). The API client must send this information to verify itself for every session to prevent the abusive applications to access users' information. When authentication is successful, a formal login process via a Web browser is obligatory. If the user is successfully logged in, the secret code generated from Facebook API will be pushed to the redirect URL. We could set this redirect URL to our listener server. In the meantime, the client application is waiting for this secret code from the redirect URL. When the secret code is ready, the client application will be prompted to send a request for an access token. Every access token is unique to the acquired secret code.

   Thereupon, the main extraction functions, which data flows are exemplified in Figure 6.8, can be called.

ii. To get the current user's id (whoAmI function), the access token is required by the Facebook API. This function should be called only once, the return object (user object) can then be used for the further requests.

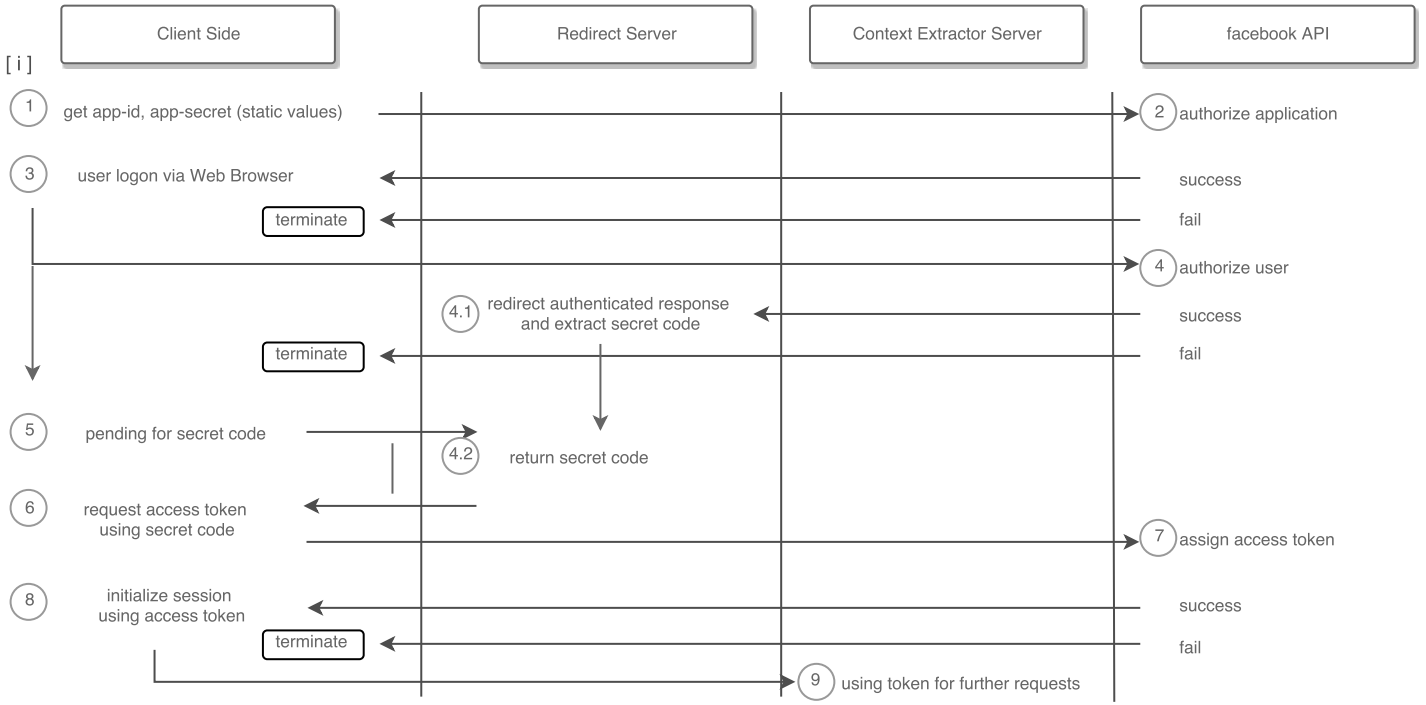


Figure 6.7.: Data flow of an authentication process for Facebook API.

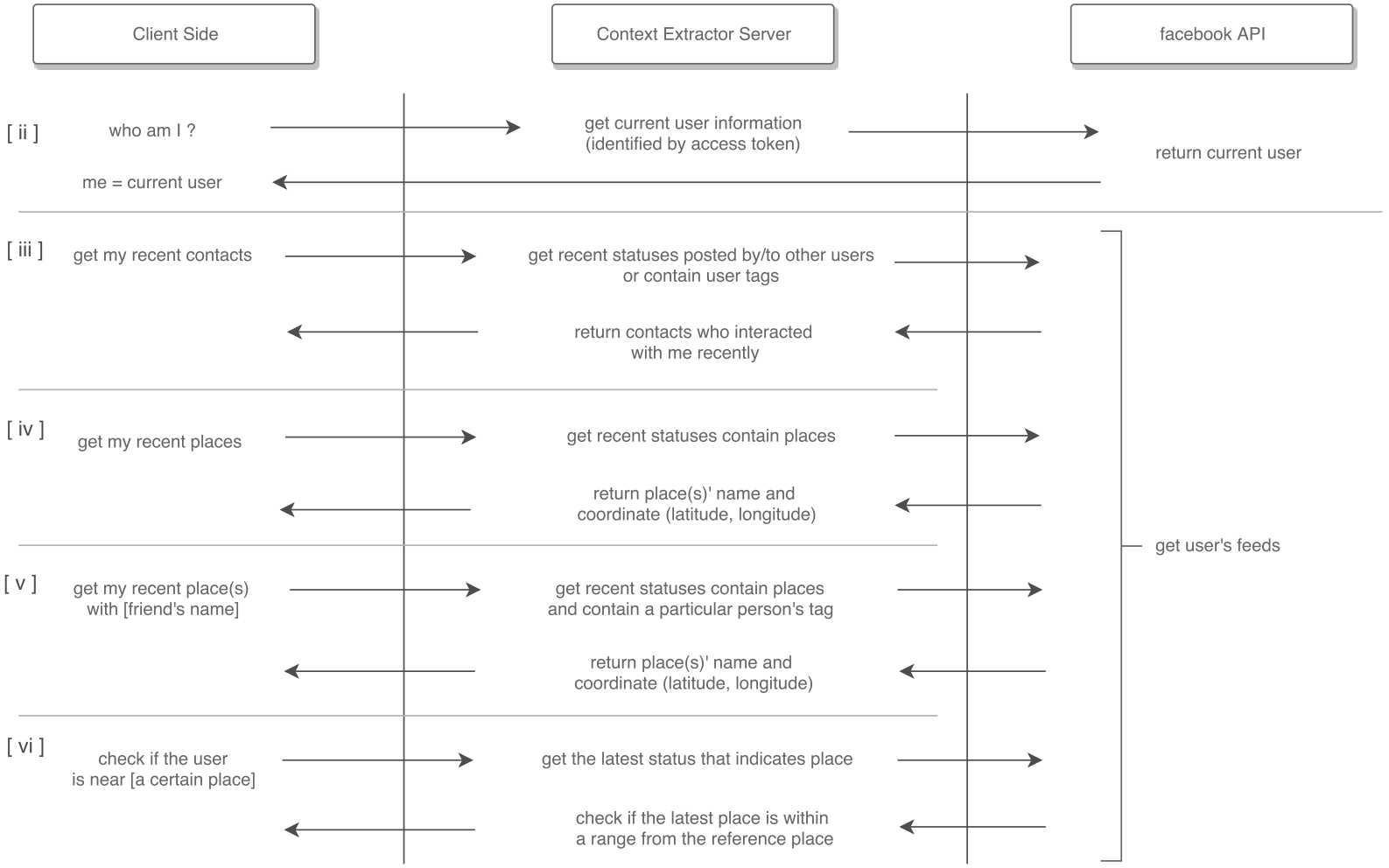


Figure 6.8.: Data flow of a social sensor by Facebook.

iii. Recent contacts are obtained from the latest status feeds, which can be a feed from any contacts. Figure 6.9 shows how to get recent contacts from three types of feeds: 1) any feed that has tags of the current user (me) 2) any feed that is directly posted to the user wall 3) any feed that the user posted on the other's wall. These interactions are considered as communication to the recent contacts.

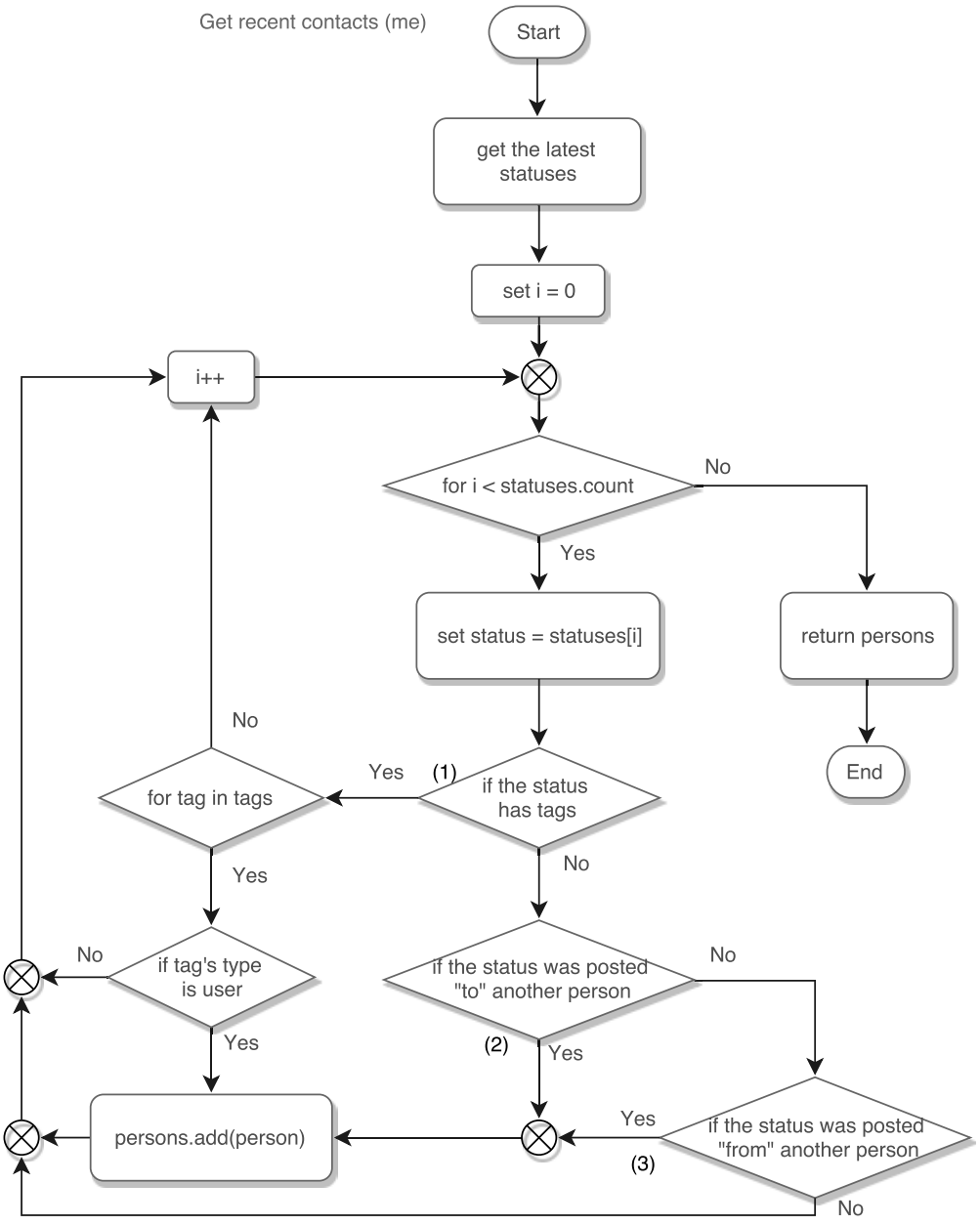


Figure 6.9.: 'Get recent contacts' function on Facebook.

iv. Recent places are also extracted from the latest status feeds, only if they contain a location information (see Figure 6.10). The returned object consists of the place's name, latitude, and longitude values.

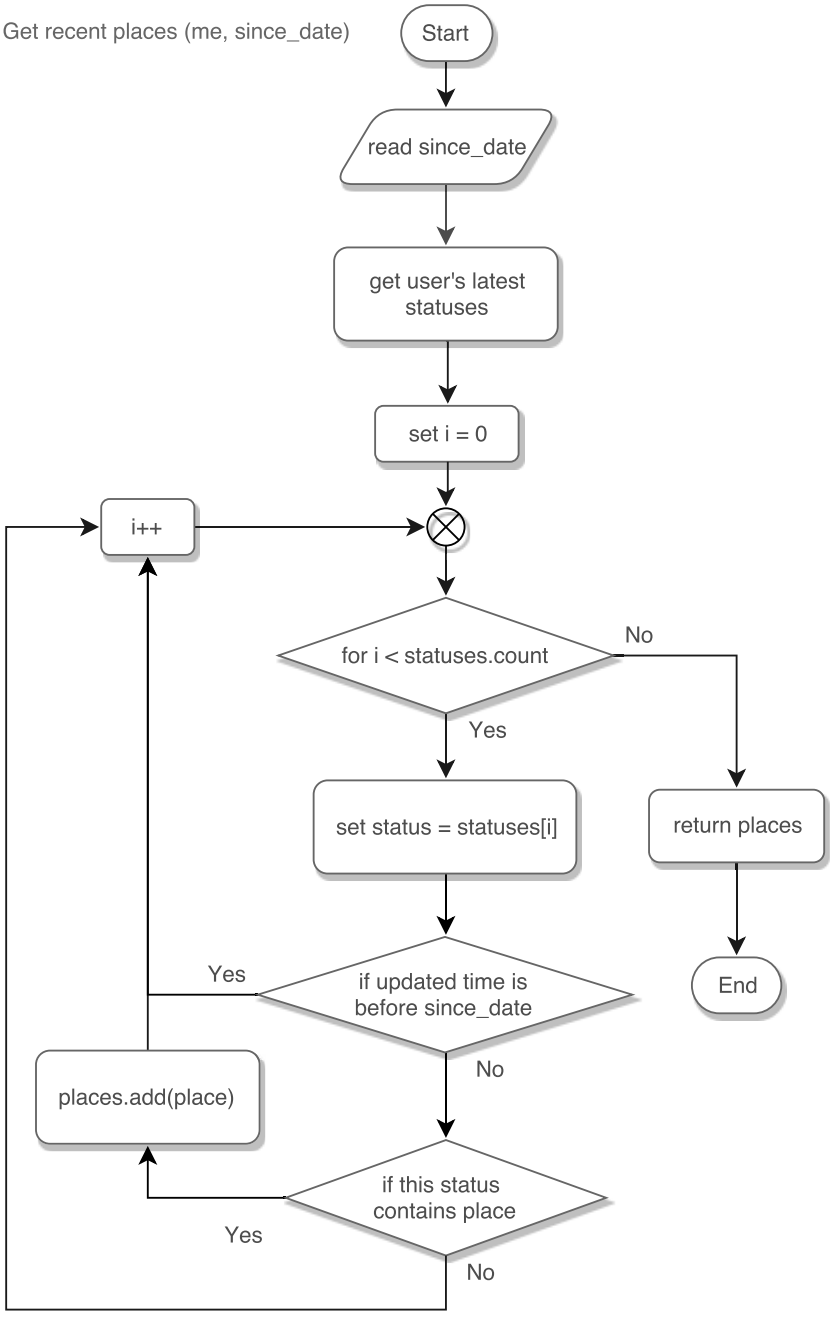


Figure 6.10.: 'Get recent places' function on Facebook.

v. A recent place with another user function is a combination of *'get recent contacts'* function and *'get recent places'* functions. Here, the other user's id is required to filter the status. The flowchart in Figure 6.11 depicts the whole process.

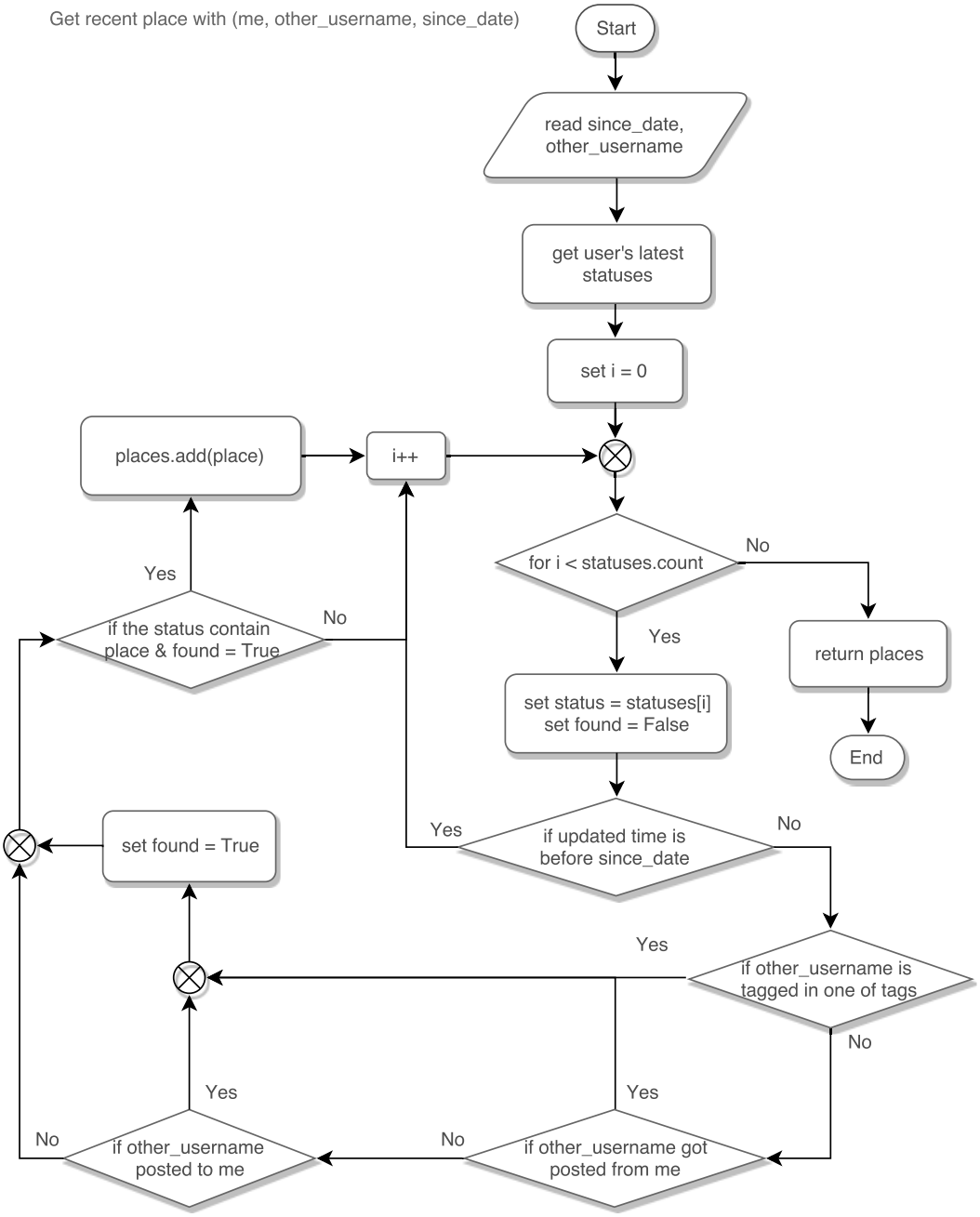


Figure 6.11.: 'Get a recent place with [another user]' function on Facebook.

vi. To check if the user is near a specific area, the location of the reference place and the threshold distance must be provided. The threshold distance defines how far the user's current (or latest) location from the reference place is in consideration. Figure 6.12 illustrates the steps to get the result.

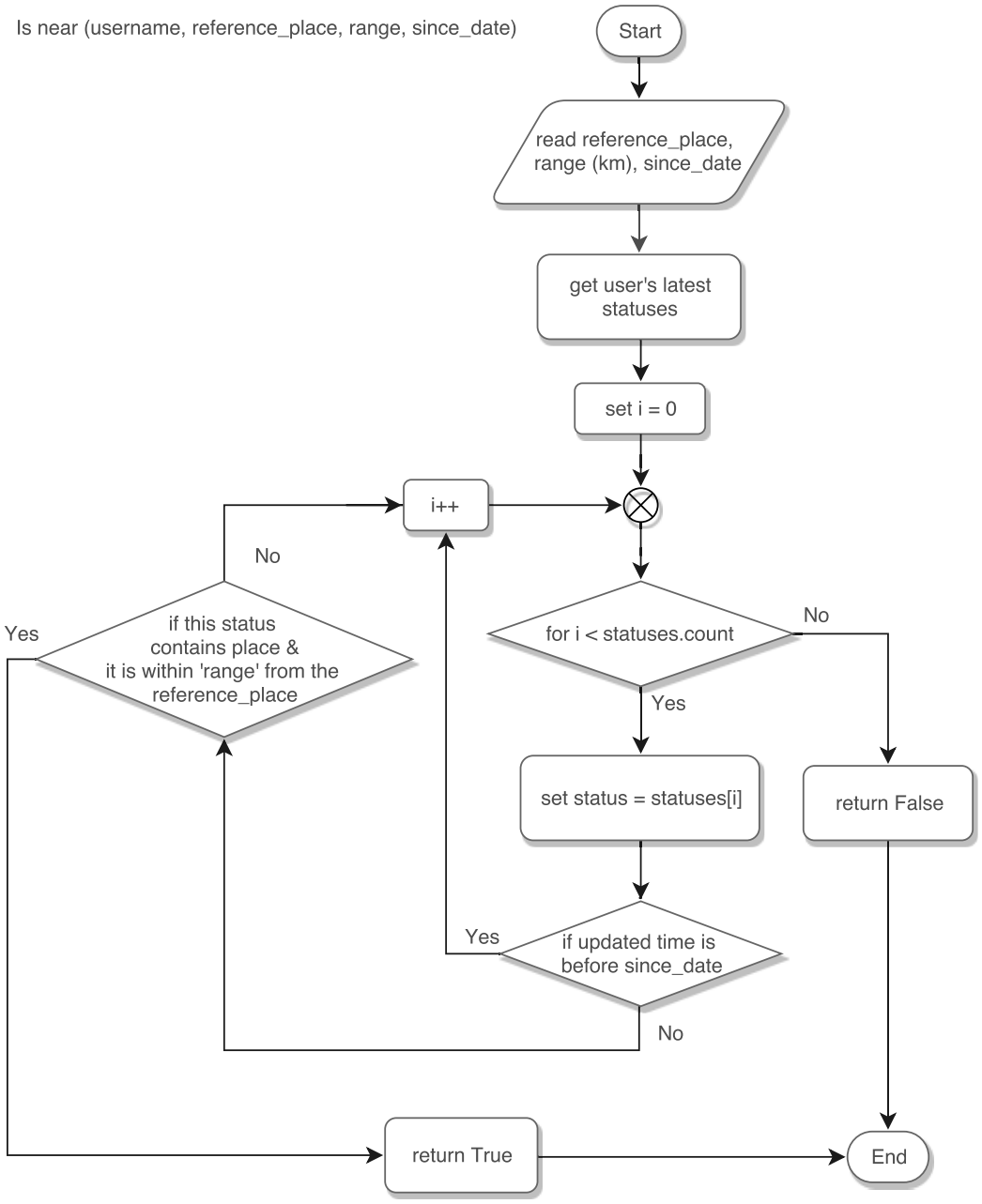


Figure 6.12.: 'Check if the user is near to [a specific location]' function on Facebook.

The executable script for the context extractor is in Listing C.3. The client script to test the functions is in Listing C.4. Lastly, the redirect server script is in Listing C.5.

Although Facebook status contains more and more information in every update (e.g. a user's mood, activities, food, music), the API still limits the information retrieval to the simple content. Therefore, when the API enhances their capabilities, we can extract even more context information from this social sensor.

6.3 Context from User Preferences and Contributions

For the registered resources, we can keep track of their frequency of use, failure occurrences, how they are used (e.g. for traveling, working out or shopping scenario), and how the community feels about them (rating). These preferences for a private resource are local to the resource owner. On the contrary, a public resource can be rated, and utilized by multiple users. Thus, its preferences depend on the community.

6.3.1. User Preferences

User preferences can be interpreted from two events: user rating, and frequency of use.

- User rating - this is an explicit preference. It provides a more qualitative value of how preferable/popular the resource is.
- Frequency of use - this is, in contrast, an implicit preference. It only indicates the quantitative information about how often this resource is being used. This measurement is suitable when we have no rating from users. However, later, this cannot differentiate if users actually 'like' the resource or are just testing it out.

6.3.2. Similar Usage

Similar usage can be considered on two levels, an atomic level and a scenario level. For example, a rain sensor is identified to be relevant to a weather forecast service because they share similar functionalities. In contrast, a weather forecast service can be related to an alarm clock in the sense that they have been used in the same scenario. We can use two criteria to identify the similarity of usage.

- Tagging is a straightforward approach. Users directly classify resources one by one. The drawback is that the resource that has never been tagged will not be discovered (cold-start problem). The automatic tagging can be applied using natural language processing (NLP), though it is error-prone.
- Semantic Description - we can use an existing description of each resource and extend it with synonyms or relevant terms. This will increase a chance of finding relevant resources and avoid a cold-start issue.

6.4 Usage

User profiles alone offer static information. We could utilize static context by applying it to a request or use it as a credential for social sensors. These social sensors can either be used as resources or as context providers. Behaving as resources, they appear to users as available sensors in the modeling process. While they act like context providers, they intrinsically provide extra information to the request in the resource discovery process.

Users' activities can also provide us context information. We can combine non-verbal context (e.g. rating, usage frequency) with textual information (tagging, semantic description) to enhance the recommendation of resources.

In the next chapter, we describe how the resource discovery processes free-text keywords into a machine-readable message. The outcome of Section 6.1, *Context from User Profile* and 6.2, *Context from Social Sensors* can be integrated with the content of the following chapter, which will simply be referred to as *user context*.

Finally, we demonstrate how to use the user context from Section 6.3, *Context from User Preferences and Contributions*, in Chapter 10, *Resource Discovery Integration to MERCURY*.

7

Request Analysis

To realize Requirement R2 (*the resource discovery should be able to interpret different description formalisms*), we study the matchers based on supported formalism we have chosen in Section 5.2, *Supported Resource Description Formalisms*.

This chapter starts with an analysis of supported matchers in Section 7.1, *Matcher Analysis*. Here, we run matchers over different requests to show that none of these matchers can perform best in every situation. Hence, we consider utilizing multiple matchers simultaneously to get the best result.

We then list the attributes that are necessary for the resource matching process. To resolve Requirement R3 (*the resource discovery should handle multiple description formalisms and matchers simultaneously*), we investigate the connection between each formalism so that we can translate one formalism to another in Section 7.2, *Essential information required for resource matching*.

7.1 Matcher Analysis

From Section 5.3, we decide to utilize the matchers from S3 contest [Klu12] in our work because of their modularity and availability of benchmarks. First, we demonstrate that each of them performs differently regarding different requests. Here, we choose two major tracks; OWL-S and SAWSDL based matchers[1]. Then, we investigate how they work and which part of the description is mandatory for them. Additionally, ElasticSearch [Ela], a prominent text-based search engine, is included to verify that we need ontology-based matchers rather than the conventional text-based search to get better results.

[1] SAWSDL based matchers are tested only with SAWSDL1.1. However, we also study how to convert SAWSDL2.0 to SAWSDL1.1 in Section 8.2, *Request Converter*. Therefore, we assume that any SAWSDL2.0 description can be converted to SAWSDL1.1, which will then be compatible with these matchers.

7.1.1. OWL-S Matchers

From OWL-S track, we study the following matchers:

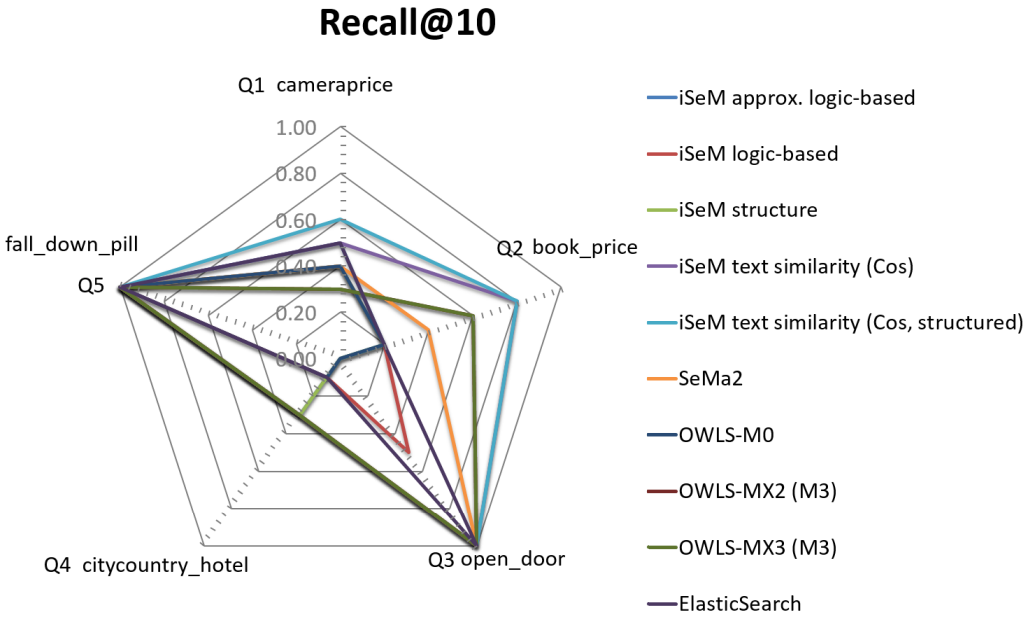
- OWLS-MX TextSim (Cos) [KFK05],
- OWLS-MX3 (Structure) [KK12a],
- OWLS-MX3 (M3) [KK12a],
- OWLS-MX2 (M3) [KK12a],
- OWLS-M0 [KFK05],
- OWLS-SLR Lite [MB10],
- OWLS-iMatcher [KB08],
- XSSD.V1 [Li13],
- SPARQLent [Sbo12],
- SeMa2 [MHB+12],
- iSeM text similarity (Cos, structured) [KK10],
- iSeM text similarity (Cos) [KK10],
- iSeM Structure [KK10],
- iSeM logic-based [KK10],
- iSeM hybrid + PE (SVM aggregation) [KK10],
- iSeM hybrid + PE + approx. (SVM aggregation) [KK10],
- iSeM approx. logic-based [KK10], and
- EMMA [GRRC12].

7.1.1.1. Matchers Performance

Nine matchers are randomly selected and tested with five OWL-S descriptions (also randomly selected). Each OWL-S matcher performs differently[2]. In terms of recall rate (see Table 7.1), *iSeM text similarity (Cos, structured)* seems to perform best. In contrast, when we consider the precision (see Table 7.2), *SeMa2* is averagely better. Also, regarding nDCG from Table 7.3, there is no best resource matcher for every query.

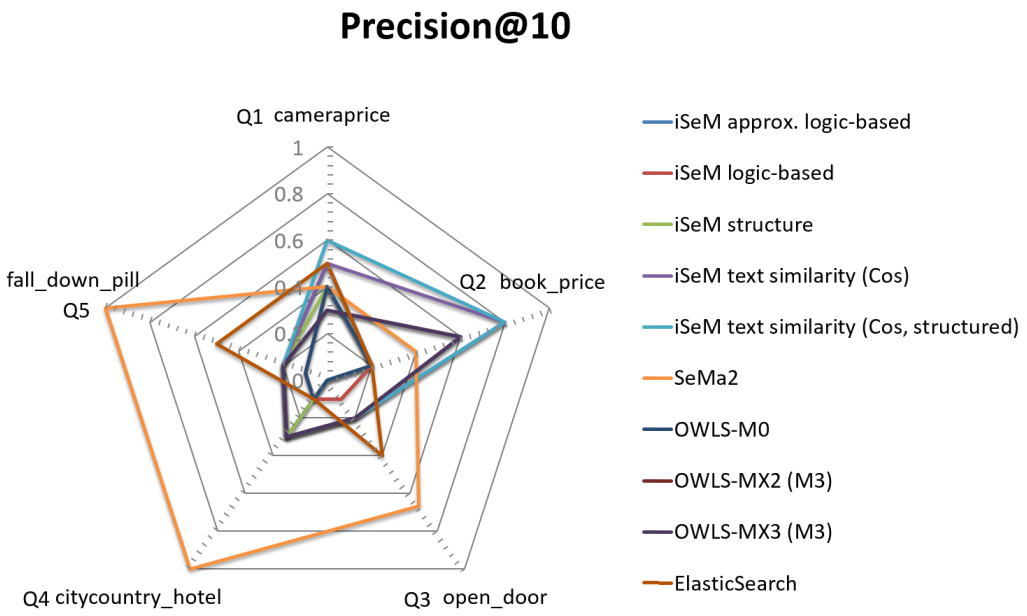
When we consider the average performance without looking at each query, the average recall, precision, and nDCG rates are inconsistent. One matcher that performs best regarding recall does not have to yield the best nDCG. For example, as summarized in Table 7.4, *iSeM text similarity (Cos, structured)* is the best in average recall. *SeMa2* is far better than the rest in average precision and F-measure rates. And *iSeM text similarity (Cos)* is slightly better than *iSeM text similarity (Cos, structured)* and *SeMa2* in terms of nDCG.

[2]We measure the quality of resource matching in this thesis using recall, precision, F-measure and nDCG rates as described in Section 11.1, *Performance measurement*.



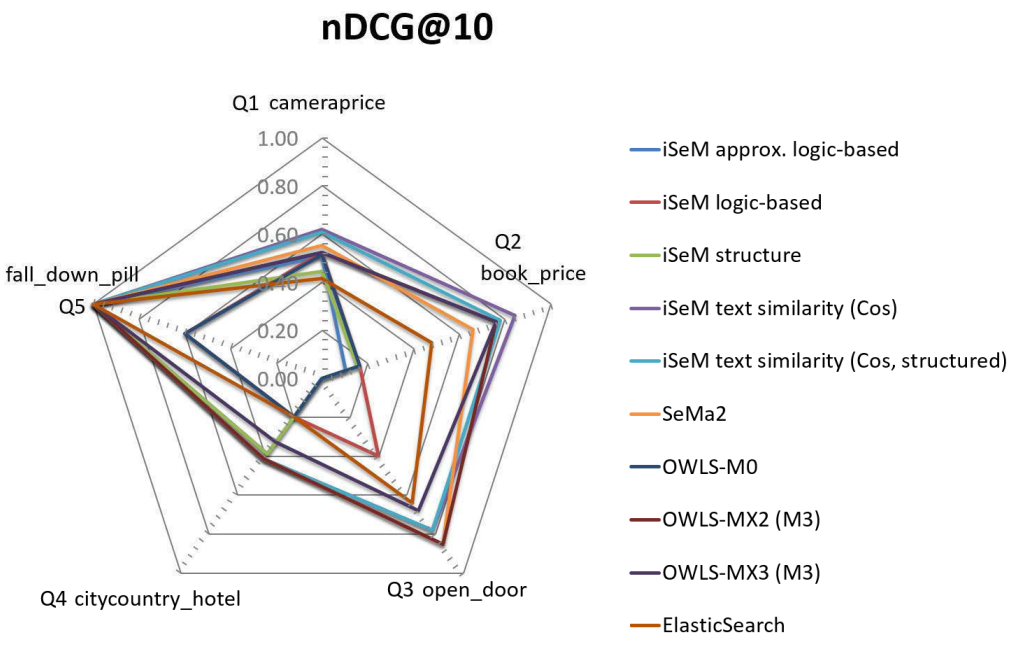
Query	Q1	Q2	Q3	Q4	Q5	Average
iSeM approx. logic	0.50	0.20	0.00	**0.30**	**1.00**	0.40
iSeM logic	0.40	0.20	0.50	0.10	**1.00**	0.44
iSeM structure	0.40	0.20	0.00	**0.30**	**1.00**	0.38
iSeM text (Cos)	0.50	**0.80**	**1.00**	**0.30**	**1.00**	0.72
iSeM text (Cos, structured)	**0.60**	**0.80**	**1.00**	**0.30**	**1.00**	**0.74**
SeMa2	0.40	0.40	**1.00**	**0.30**	**1.00**	0.62
OWLS-M0	0.40	0.20	0.00	0.10	**1.00**	0.34
OWLS-MX2	0.30	0.60	**1.00**	**0.30**	**1.00**	0.64
OWLS-MX3	0.30	0.60	**1.00**	**0.30**	**1.00**	0.64
elasticsearch	0.50	0.20	**1.00**	0.10	**1.00**	0.56

Table 7.1.: OWL-S matchers' recall rate per query.



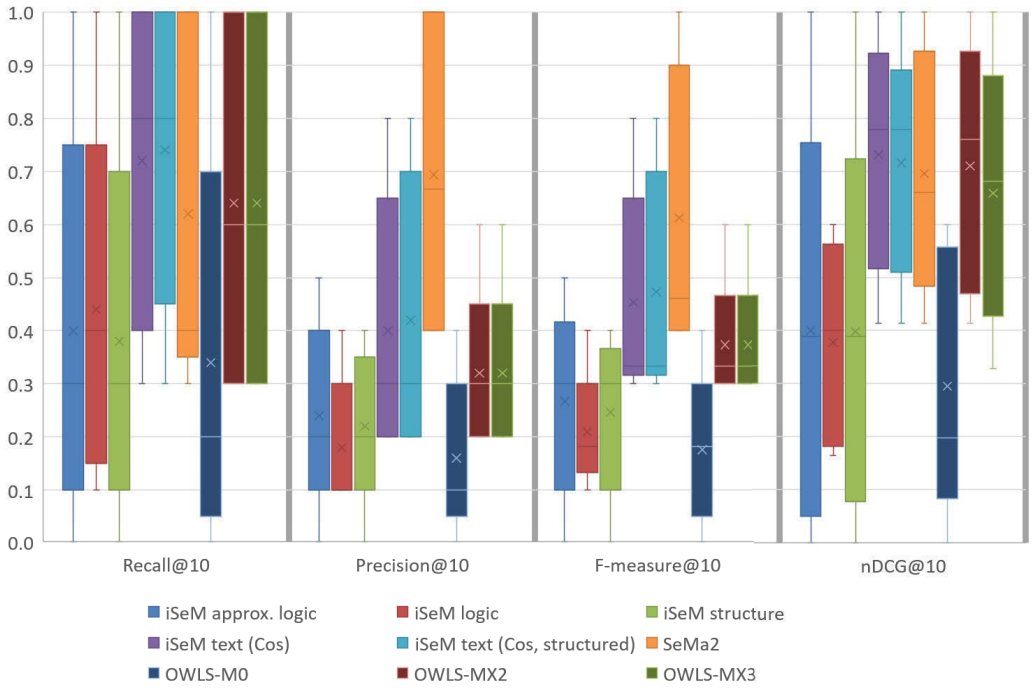
Query	Q1	Q2	Q3	Q4	Q5	Average
iSeM approx. logic	0.50	0.20	0.00	0.30	0.20	0.24
iSeM logic	0.40	0.20	0.10	0.10	0.10	0.18
iSeM structure	0.40	0.20	0.00	0.30	0.20	0.22
iSeM text (Cos)	0.50	**0.80**	0.20	0.30	0.20	0.40
iSeM text (Cos, structured)	**0.60**	**0.80**	0.20	0.30	0.20	0.42
SeMa2	0.40	0.40	**0.67**	**1.00**	**1.00**	**0.69**
OWLS-M0	0.40	0.20	0.00	0.10	0.10	0.16
OWLS-MX2	0.50	0.60	0.20	0.30	0.20	0.36
OWLS-MX3	0.30	0.60	0.20	0.30	0.20	0.32
elasticsearch	0.50	0.20	0.40	0.10	0.50	0.34

Table 7.2.: OWL-S matchers' precision rate per query.



Query	Q1	Q2	Q3	Q4	Q5	Average
iSeM approx. logic-based	0.51	0.10	0.00	0.39	**1.00**	0.40
iSeM logic-based	0.52	0.17	0.40	0.20	0.60	0.38
iSeM structure	0.45	0.16	0.00	0.39	**1.00**	0.40
iSeM text similarity (Cos)	**0.62**	**0.84**	0.78	**0.41**	**1.00**	**0.73**
iSeM text similarity (Cos, structured)	0.61	0.78	0.78	**0.41**	**1.00**	0.72
SeMa2	0.55	0.66	**0.85**	**0.41**	**1.00**	0.69
OWLS-M0	0.51	0.17	0.00	0.20	0.60	0.30
OWLS-MX2 (M3)	0.52	0.76	**0.85**	**0.41**	**1.00**	0.71
OWLS-MX3 (M3)	0.52	0.76	0.68	0.33	**1.00**	0.66
elasticsearch	0.41	0.48	0.64	0.20	**1.00**	0.55

Table 7.3.: OWL-S matchers' nDCG rate per query.



Matcher name		Recall	Precision	F-measure	nDCG
iSeM approx. logic-based	min	0.00	0.00	0.00	0.00
	mean	0.40	0.24	0.27	0.40
	max	0.50	1.00	0.50	1.00
iSeM logic-based	min	0.10	0.10	0.10	0.17
	mean	0.44	0.18	0.21	0.38
	max	0.40	1.00	0.40	0.60
iSeM structure	min	0.00	0.00	0.00	0.00
	mean	0.38	0.22	0.25	0.40
	max	0.40	1.00	0.40	1.00
iSeM text similarity (Cos)	min	0.20	0.20	0.30	0.41
	mean	0.72	0.40	0.45	**0.73**
	max	0.80	1.00	0.80	1.00
iSeM text similarity (Cos, structured)	min	0.20	0.20	0.30	0.41
	mean	**0.74**	0.42	0.47	0.72
	max	0.80	1.00	0.80	1.00
SeMa2	min	0.40	0.40	0.40	0.41
	mean	0.62	**0.69**	**0.61**	0.70
	max	1.00	1.00	1.00	1.00
OWLS-M0	min	0.00	0.00	0.00	0.00
	mean	0.34	0.16	0.18	0.30
	max	0.40	1.00	0.40	0.60
OWLS-MX2 (M3)	min	0.20	0.20	0.30	0.41
	mean	0.64	0.32	0.37	0.71
	max	0.60	1.00	0.60	1.00
OWLS-MX3 (M3)	min	0.20	0.20	0.30	0.33
	mean	0.64	0.32	0.37	0.66
	max	0.60	1.00	0.60	1.00

Table 7.4.: OWL-S matchers' average result quality.

7.1.1.2. Matchers Requirements

To test which information is used by matchers, first, we change OWL-S descriptions' filenames and verify if this change affects the matchers. Then we explore the content of those description files. In OWL-S description, the first level nodes are; *Ontology*, *Service*, *Profile*, *Atomic process*, *Input*, *Output*, *WSDL Grounding*, and *WSDL Atomic Process Grounding*. We examine them by removing each node one by one and check whether the result is affected. Table 7.5 shows the result of these investigations.

From the initial examination, some matchers return no result or abruptly stop working during the runtime without any helpful error message. We neglect these matchers in the further investigation. In this step, we have not considered the quality of the result yet.

We found that none of the OWL-S matchers takes a description's filename into account. The *Ontology*, *WSDL Grounding*, and *WSDL Atomic Process Grounding* nodes can be removed without any consequences.

Matcher name	Filename	Content								
		Ontology	**Service**	**Profile**	**Atomic process**	**Input**	**Output**	WSDL Grounding	WSDL Atomic Process Grounding	
OWLS-MX TextSim (Cos) [KFK05]	✗	✗	✓	✗	✗	✗	✗	✗	✗	
OWLS-MX3 (Structure) [KK12a]	✗	✗	✓	✗	✗	✗	✗	✗	✗	
OWLS-MX3 (M3) [KK12a]	✗	✗	✓	✓	✗	✓	✓	✗	✗	
OWLS-MX2 (M3) [KK12a]	✗	✗	✓	✓	✗	✓	✓	✗	✗	
OWLS-M0 [KFK05]	✗	✗	✓	✓	✗	✓	✓	✗	✗	
OWLS-SLR Lite [MB10]	Not working									
OWLS-iMatcher [KB08]	Not working									
XSSD.V1 [Li13]	Not working									
SPARQLent [Sbo12]	Not working									
SeMa2 [MHB+12]	✗	✗	✗	✓	✗	✓	✓	✗	✗	
iSeM text similarity (Cos, structured) [KK10]	✗	✗	✓	✓	✓	✓	✓	✗	✗	
iSeM text similarity (Cos) [KK10]	✗	✗	✓	✓	✓	✓	✓	✗	✗	
iSeM Structure [KK10]	✗	✗	✓	✓	✓	✓	✓	✗	✗	
iSeM logic-based [KK10]	✗	✗	✓	✓	✓	✓	✓	✗	✗	
iSeM hybrid + PE (SVM aggregation) [KK10]	Not working									
iSeM hybrid + PE + approx. (SVM aggregation) [KK10]	Not working									
iSeM approx. logic-based [KK10]	✗	✗	✓	✓	✓	✓	✓	✗	✗	
EMMA [GRRC12]	Not working									

✓=this node affects the results when missing, ✗=this node makes no difference when it is missing

Table 7.5.: First level elements required by OWL-S matchers.

Next, we examine attributes and child nodes of Service, Profile, Atomic Process, Input and Output nodes.

- OWL-S Service
  This node contains *ID* attribute, *Presents*, *Described By*, and *Supports* nodes. However, as shown in Table 7.6, *Described By* and *Supports* nodes make no difference for the matchers when they are missing. Consequently, we examine further in Service's *ID* and *Presents* nodes.

  In addition to the removal of node/attribute, we test using empty values and assigning wrong values to each node/attribute. This helps us understand if matchers require the node/attribute only for the syntax validation, or they use the content in the node/attribute for the matching. Table 7.7 shows that *Service:ID* attribute must exist but the value can be arbitrary. Meanwhile, *Presents* node must contain a correct value for the matching process.

- OWL-S Profile
  This node contains *ID* attribute, *Is Presented by*, *Service Name*, *Text Description*, *Has Input*, *Has Output*, and *Has Process* nodes. Table 7.6 shows that only *Has Input* and *Has Output* nodes are required by matchers. Also, Table 7.7 further indicates that the content within *Has Input* and *Has Output* nodes are taken into account for the matching process.

Matcher name	Service				Profile					
	ID	**Presents**	Described By	Supports	ID	Is Presented by	Service Name	Text Description	**Has Input**	**Has Output**	Has Process
OWLS-MX TextSim (Cos)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (Structure)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (M3)	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
OWLS-MX2 (M3)	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
OWLS-M0	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
SeMa2	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗
iSeM text similarity (Cos, structured)	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
iSeM text similarity (Cos)	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
iSeM Structure	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
iSeM logic-based	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗
iSeM approx. logic-based	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗

✓=this node affects the results when missing, ✗=this node makes no difference when it is missing

Table 7.6.: Elements inside Service and Profile nodes required by OWL-S matchers.

Matcher name	Service						Profile					
	ID			Presents:resource			HasInput:resource			HasOutput:resource		
	M	B	W	**M**	**B**	**W**	**M**	**B**	**W**	**M**	**B**	**W**
OWLS-MX TextSim (Cos)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (Structure)	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (M3)	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
OWLS-MX2 (M3)	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
OWLS-M0	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
SeMa2	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
iSeM text similarity (Cos, structured)	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
iSeM text similarity (Cos)	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
iSeM Structure	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
iSeM logic-based	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
iSeM approx. logic-based	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓

✓=has effect, ✗=has no effect

M=missing node/attribute, B=blank value, W=wrong value

Table 7.7.: Effect from elements inside Service and Profile nodes on OWL-S matchers when they are missing, contain blank value or contain wrong value.

- OWL-S Atomic Process
  Atomic Process node has *ID* attribute, *describes*, *hasInput*, and *hasOutput* child nodes. According to Table 7.8, these attribute and child nodes are removable. Therefore, we can deduce that *Atomic Process* node is required for a formalism validation, but contains no meaning.

- OWL-S Input
  From Table 7.9, we can conclude that a correct value of attribute *ID* of *Input* node is required. *ParameterType* node must contain the correct content. However, attribute *ParameterType: DataType* can be omitted. The node *label* is insignificant.

- OWL-S Output
  Similar to the OWL-S Input, a correct value of *ID* attribute, and *ParameterType* node's content are required by most of the matchers, while *label* node and *ParameterType:DataType* attribute are unnecessary for the matching process.

Matcher name	AtomicProcess				Input			Output		
	ID	describes	hasInput	hasOutput	**ID**	**paramType**	label	**ID**	**paramType**	label
OWLS-MX TextSim (Cos)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (Structure)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (M3)	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
OWLS-MX2 (M3)	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
OWLS-M0	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
SeMa2	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
iSeM text similarity (Cos, structured)	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
iSeM text similarity (Cos)	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
iSeM Structure	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
iSeM logic-based	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗
iSeM approx. logic-based	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗

✓=this node affects the results when missing, ✗=this node makes no difference when it is missing

Table 7.8.: Elements inside Atomic Process, Input and Output nodes required by OWL-S matchers.

Matcher name	Input								Output							
	ID			ParameterType: DataType			**ParameterType Content**		**ID**			ParameterType: DataType			**ParameterType Content**	
	M	**B**	**W**	M	B	W	**B**	**W**	**M**	**B**	**W**	M	B	W	**B**	**W**
OWLS-MX TextSim (Cos)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (Structure)	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OWLS-MX3 (M3)	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
OWLS-MX2 (M3)	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
OWLS-M0	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
SeMa2	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
iSeM text similarity (Cos, structured)	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
iSeM text similarity (Cos)	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
iSeM Structure	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
iSeM logic-based	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓
iSeM approx. logic-based	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓

✓=has effect, ✗=has no effect

M=missing node/attribute, B=blank value, W=wrong value

Table 7.9.: Effect from elements inside Input and Output nodes on OWL-S matchers when they are missing, contain blank value or contain wrong value.

7.1.2. SAWSDL Matchers

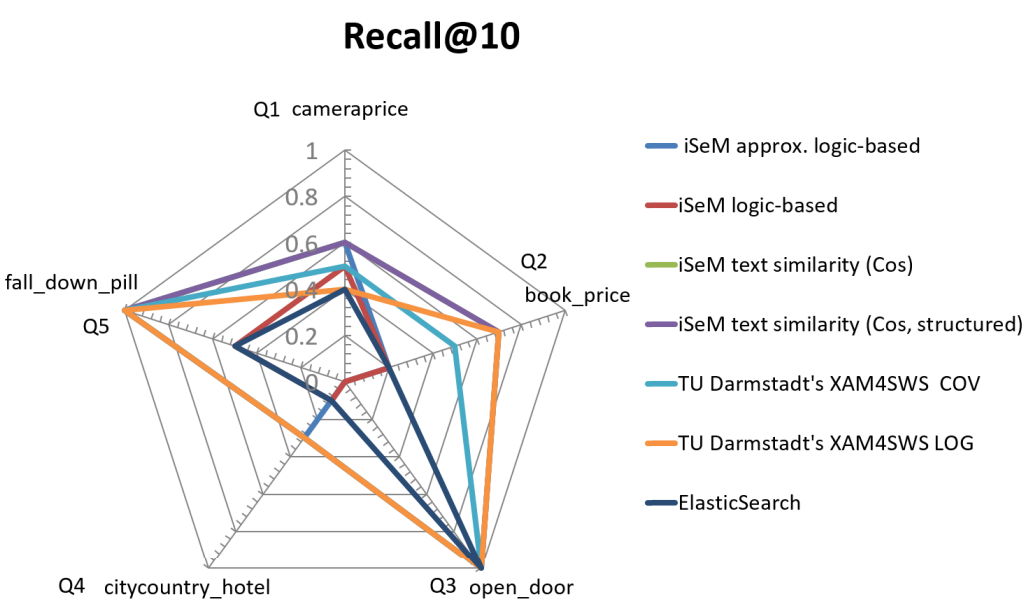
From SAWSDL track in S3 Contest, the following matchers are studied:

- iMatcher IOEuclidean [WWWB11],
- iSeM Approx. logic based [KK12b],
- iSeM Hybrid + approx [KK12b],
- iSeM Hybrid [KK12b],
- iSeM structure [KK12b],
- iSeM Logic-based [KK12b],
- iSeM SVM aggregation [KK12b],
- iSeM Text Sim. (Cos) [KK12b],
- iSeM Text Sim. (Cos +Structure) [KK12b],
- SAWSDL-M0 [KKZ09a],
- SAWSDL-MX [KKZ09a],
- SAWSDL-MX2 [KKZ09b],
- XAM4SWS-COV [SLKS12],
- XAM4SWS-LOG [SLES10], and
- URBE [PP09].

7.1.2.1. Matchers Performance

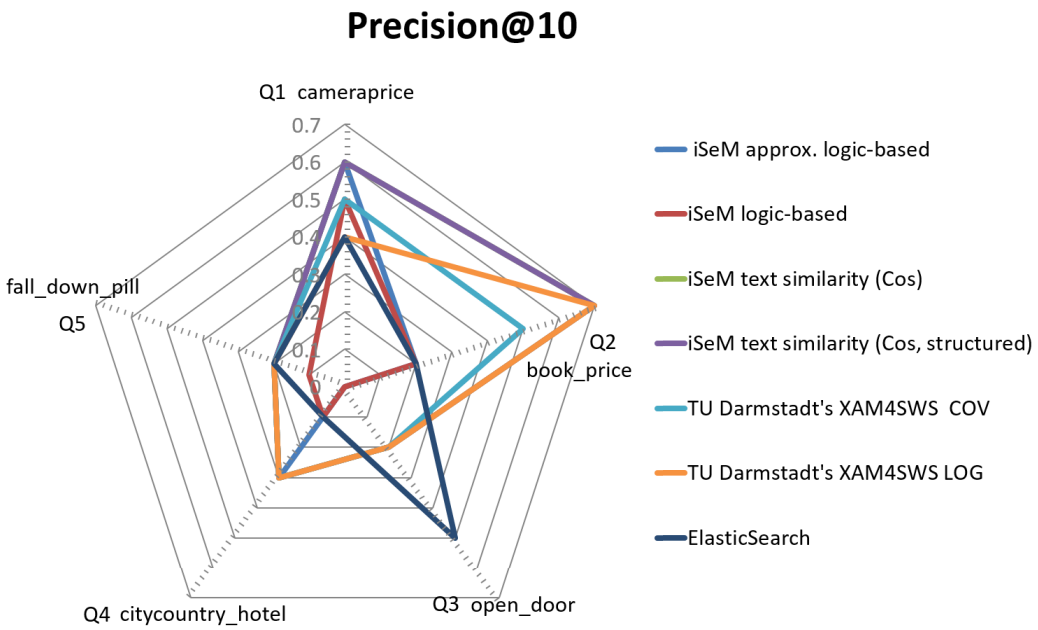
Randomly selected six matchers are tested with five SAWSDL descriptions. The same trait as OWL-S matchers is also found in SAWSDL matchers. Table 7.10, 7.11, and 7.12 illustrate recall, precision and nDCG rates of each SAWSDL matchers for different queries.

Although *iSeM text similarity (Cos, structured)* seems to perform best in general, a text-based search engine (*ElasticSearch*) or XAM4SWS-LOG matcher can perform better than it in some cases.



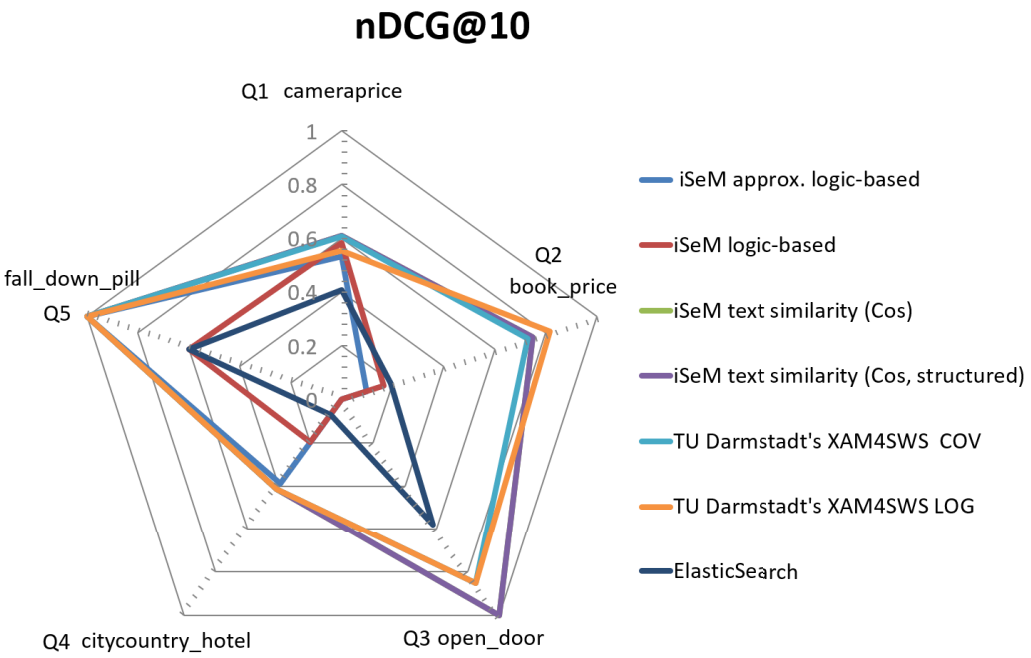
Query	Q1	Q2	Q3	Q4	Q5	Average
iSeM approx. logic-based	**0.60**	0.20	0.00	**0.30**	**1.00**	0.42
iSeM logic-based	0.50	0.20	0.00	0.10	0.50	0.26
iSeM text similarity (Cos)	**0.60**	**0.70**	**1.00**	**0.30**	**1.00**	**0.72**
iSeM text similarity (Cos, structured)	**0.60**	**0.70**	**1.00**	**0.30**	**1.00**	**0.72**
TU Darmstadt's XAM4SWS COV	0.50	0.50	**1.00**	**0.30**	**1.00**	0.66
TU Darmstadt's XAM4SWS LOG	0.40	**0.70**	**1.00**	**0.30**	**1.00**	0.68
Elasticsearch	0.40	0.20	**1.00**	0.10	0.50	0.44

Table 7.10.: SAWSDL matchers' recall rate per query.



Query	Q1	Q2	Q3	Q4	Q5	Average
iSeM approx. logic-based	**0.60**	0.20	0.00	**0.30**	**0.20**	0.26
iSeM logic-based	0.50	0.20	0.00	0.10	0.10	0.18
iSeM text similarity (Cos)	**0.60**	**0.70**	0.30	**0.30**	**0.20**	**0.42**
iSeM text similarity (Cos, structured)	**0.60**	**0.70**	0.30	**0.30**	**0.20**	**0.42**
TU Darmstadt's XAM4SWS COV	0.50	0.50	0.20	**0.30**	**0.20**	0.34
TU Darmstadt's XAM4SWS LOG	0.40	**0.70**	0.20	**0.30**	**0.20**	0.36
Elasticsearch	0.40	0.20	**0.50**	0.10	**0.20**	0.28

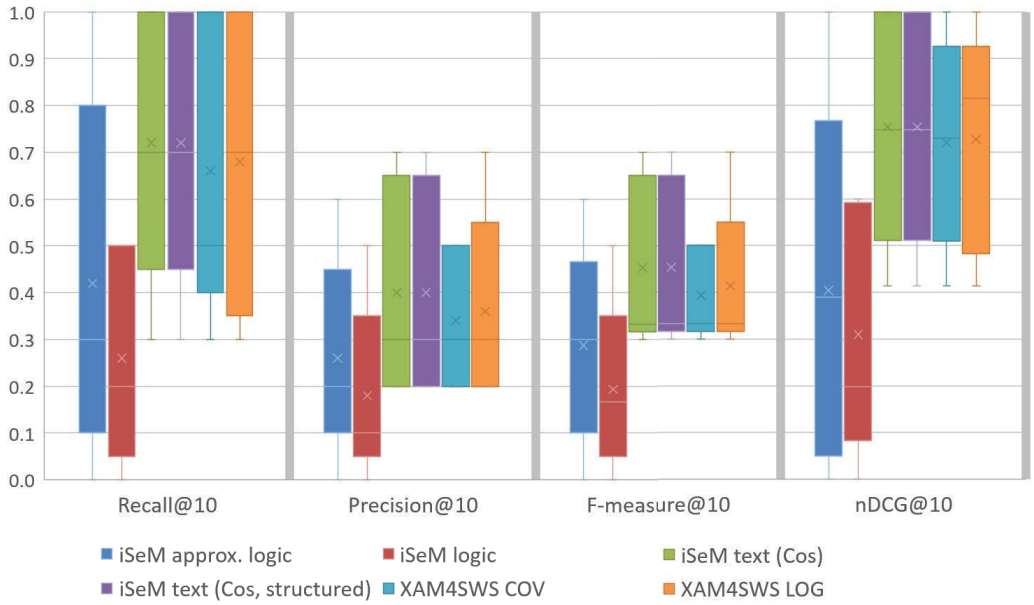
Table 7.11.: SAWSDL matchers' precision rate per query.



Query	Q1	Q2	Q3	Q4	Q5	Average
iSeM approx. logic-based	0.53	0.10	0.00	0.39	**1.00**	0.40
iSeM logic-based	0.58	0.17	0.00	0.20	0.60	0.31
iSeM text similarity (Cos)	**0.61**	0.75	**1.00**	**0.41**	**1.00**	**0.75**
iSeM text similarity (Cos, structured)	**0.61**	0.75	**1.00**	**0.41**	**1.00**	**0.75**
TU Darmstadt's XAM4SWS COV	**0.61**	0.73	0.85	**0.41**	**1.00**	0.72
TU Darmstadt's XAM4SWS LOG	0.55	**0.82**	0.85	**0.41**	**1.00**	0.73
Elasticsearch	0.41	0.19	0.58	0.07	0.60	0.37

Table 7.12.: SAWSDL matchers' nDCG rate per query.

The result from SAWSDL matchers summarized in Table 7.13 shows that *iSeM text similarity (Cos)* and *iSeM text similarity (Cos, structured)* are best perform in every perspective. However, this does not guarantee to be the best choice for every query.



Matcher name		Recall	Precision	F-measure	nDCG
iSeM approx. logic	min	0.00	0.00	0.00	0.00
	mean	0.42	0.26	0.29	0.40
	max	1.00	0.60	0.60	1.00
iSeM logic	min	0.00	0.00	0.00	0.00
	mean	0.26	0.18	0.19	0.31
	max	0.50	0.50	0.50	0.60
iSeM text (Cos)	min	0.30	0.20	0.30	0.41
	mean	0.72	0.40	0.45	0.75
	max	1.00	0.70	0.70	1.00
iSeM text (Cos, structured)	min	0.30	0.20	0.30	0.41
	mean	0.72	0.40	0.45	0.75
	max	1.00	0.70	0.70	1.00
XAM4SWS COV	min	0.30	0.20	0.30	0.41
	mean	0.66	0.34	0.39	0.72
	max	1.00	0.50	0.50	1.00
XAM4SWS LOG	min	0.30	0.20	0.30	0.41
	mean	0.68	0.36	0.41	0.73
	max	1.00	0.70	0.70	1.00

Table 7.13.: SAWSDL matchers' average result quality.

These results advocate the concept of utilizing several matchers simultaneously, which is proposed by this thesis.

7.1.2.2. Matchers Requirements

To test what are crucial information for SAWSDL matchers, first, we examine a description's filename. Matchers which cannot operate properly or returns totally wrong results will be excluded from the further investigation. As shown in Table 7.14, only iMatcher-IOEuclidean relies on a filename, not the content of the file.

Within a SAWSDL description, *Types*, *Message*, *Service*, and *PortType* nodes are considered for the resource matching (except for the iMatcher-IOEuclidean matcher). Meanwhile, *Binding* node is not necessary for matchers. In Table 7.15, we examine each attribute and child node within *Types*, *Message*, *Service*, and *PortType* nodes.

Matcher name	Filename	Content					Remarks
		Binding	**Types**	**Message**	**Service**	**PortType**	
iMatcher IOEuclidean [WWWB11]	✓	✗	✗	✗	✗	✗	
iSem Approx. logic based [KK12b]	✗	✗	✓	✓	✓	✓	
iSem Hybrid + approx [KK12b]	Not Working						
iSem Hybrid [KK12b]	Not Working						
iSem structure [KK12b]	✗	✗	✓	✓	✓	✓	Wrong ranking result
iSem Logic-based [KK12b]	✗	✗	✓	✓	✓	✓	
iSem SVM aggregation [KK12b]	Not Working						
iSem Text Sim. (Cos) [KK12b]	✗	✗	✓	✓	✓	✓	
iSem Text Sim. (Cos +Structure) [KK12b]	✗	✗	✓	✓	✓	✓	
SAWSDL-M0 [KKZ09a]	✗	✗	✗	✓	✓	✓	Wrong ranking result
SAWSDL-MX [KKZ09a]	✗	✗	✗	✓	✓	✓	Wrong ranking result
SAWSDL-MX2 [KKZ09b]	✗	✗	✗	✓	✓	✓	Wrong ranking result
XAM4SWS-COV4SWS [SLKS12]	✗	✗	✓	✓	✗	✓	
XAM4SWS-LOG4SWS [SLES10]	✗	✗	✓	✓	✗	✓	
URBE [PP09]	Not Working						

✓=this node affects the results when missing, ✗=this node makes no difference when it is missing

Table 7.14.: First level elements required by SAWSDL matchers.

Matcher name	Types			Message		Service		PortType	
	Annotation	Lifting schema	**Input**	**Output**	**Request**	**Response**	Name	Port	**Name**	**Operation**
iSem Approx. logic based	✗	✗	✗	✓	✓	✓	✗	✗	✓	✓
iSem Logic-based	✗	✗	✓	✗	✓	✓	✗	✗	✓	✓
iSem Text Sim. (Cos)	✗	✗	✓	✓	✓	✓	✗	✗	✓	✓
iSem Text Sim. (Cos +Structure)	✗	✗	✓	✓	✓	✓	✗	✗	✓	✓
XAM4SWS-COV4SWS	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓
XAM4SWS-LOG4SWS	✗	✗	✗	✗	✓	✓	✗	✗	✗	✓

✓=this node affects the results when missing, ✗=this node makes no difference when it is missing

Table 7.15.: Elements inside Types, Message, Service and PortType nodes required by SAWSDL matchers.

- SAWSDL Types
  Types node contains *Annotation*, *Lifting Schema*, *Input*, and *Output* nodes. *Annotation* and *Lifting Schema* nodes cause no effect to the matching results when they are removed. In contrast, the Input and Output nodes are required.

  Both Types:Input and Types:Output nodes have *Name* attribute, *ModelRef* attribute, and *Sequence* child node. *ModelRef* attribute contains the semantic annotation of that node.

  From Table 7.16, most matchers require *Name* and *ModelRef* attributes within *Types:Input* and *Types:Output* nodes for matching. On the other hand, *Sequence* node can be omitted.

Matcher name	Types: Input							Types: Output						
	Name			**ModelRef**			Sequence	**Name**			**ModelRef**			Sequence
	M	B	W	M	B	W	M	M	B	W	M	B	W	M
iSem Approx. logic based	✗	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗
iSem Logic-based	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗
iSem Text Sim. (Cos)	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗
iSem Text Sim. (Cos +Structure)	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✗
XAM4SWS-COV4SWS	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
XAM4SWS-LOG4SWS	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

✓=has effect, ✗=has no effect

M=missing node/attribute, B=blank value, W=wrong value

Table 7.16.: Effect from elements inside Types node on SAWSDL matchers when they are missing, contain blank value or contain wrong value.

- SAWSDL Message
  Message node specifies input and output descriptions. The input description is referred to as a request, whereas the output description is a response. Table 7.17 provides more detail in *Message:Request* and *Message:Response* nodes. Both nodes have the same structure. They consist of *Name* attribute and *Part* node. *Message:Part* node has two attributes; *PartName* and *PartType*. All matchers require *Message:Name* and *PartType* attributes to contain correct information. Whereas *PartName* attribute can have an arbitrary value.

Matcher name	Message: Request									Message: Response								
	Name			Part: Partname			Part: Parttype			Name			Part: Partname			Part: Parttype		
	M	B	W	M	B	W	M	B	W	M	B	W	M	B	W	M	B	W
iSem Approx. logic based	✓	✓	✓	✓	✗	✗	✗	✗	✗	✓	✓	✓	✓	✗	✗	✓	✓	✓
iSem Logic-based	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
iSem Text Sim. (Cos)	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
iSem Text Sim. (Cos +Structure)	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
XAM4SWS-COV4SWS	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗
XAM4SWS-LOG4SWS	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗

✓=has effect, ✗=has no effect

M=missing node/attribute, B=blank value, W=wrong value

Table 7.17.: Effect from elements inside Message:Request and Message:Response nodes on SAWSDL matchers when they are missing, contain blank value or contain wrong value.

- SAWSDL Service
  Service node contains *Name* attribute and *Port* child node. As shown in Table 7.15, the attribute and the child node are not used for the matching process. Nevertheless, as indicated in Table 7.14, *Service* node must exist in the description.

- SAWSDL PortType
  PortType:Name attribute and *PortType:Operation* node are required by matchers. As shown in Table 7.18, *PortType:Name* attribute can either be missing or contain an arbitrary value, but it cannot be left blank.

  PortType: Operation's name must exist and cannot be empty. While *Operation:Input* and *Operation:Output* nodes must have *Message* attribute linking to the name of *Message:Request* and *Message:Response* respectively.

Matcher name	PortType: Name			PortType: Operation								
				Name			Input Message			Output Message		
	M	B	W	M	B	W	M	B	W	M	B	W
iSem Approx. logic based	✗	✓	✗	✓	✓	✗	✓	✗	✗	✓	✓	✓
iSem Logic-based	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓
iSem Text Sim. (Cos)	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓
iSem Text Sim. (Cos +Structure)	✗	✓	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓
XAM4SWS-COV4SWS	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓
XAM4SWS-LOG4SWS	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓

✓=has effect, ✗=has no effect

M=missing node/attribute, B=blank value, W=wrong value

Table 7.18.: Effect from elements inside PortType node on SAWSDL matchers when they are missing, contain blank value or contain wrong value.

Now, we can deduce the essential information of resource description from the previous analysis for the purpose of resource matching.

7.2 Essential information required for resource matching

From the study of resource descriptions and their corresponding matchers, not every field of resource description is mandatory in the process of resource matching. Some pieces of information are required to validate the description format, though the contents do not affect the matching result. Optional information is omissible, and only a few data are considered in comparison algorithm.

According to our studies in Section 7.1, we came to the conclusion on OWL-S and SAWSDL1.1 description as shown in Figure 7.1 and 7.2. Additionally, we also studied SAWSDL2.0 and summarized requirements in Figure 7.3. Note that these mapping charts are the result of matchers' algorithms. They are not inherent or bound to the description formalisms themselves.

In the mapping charts, each box represents a node which connects to a parent node and children nodes. A leaf node with (A) means that it can have arbitrary values or be empty, but the element must exist in all case. (B) means this attribute cannot be left blank but can have an arbitrary value. (C) indicates that a valid content must be provided. (D) is equivalent to don't care, in other words, the element is not considered at all.

7.2.1. OWL-S Description

According to Figure 7.1, the following rules must be applied to an OWL-S description.

- A service node must contain *ID* and *Presents* nodes.
- *Service: ID* node can have an arbitrary value or can be left blank.
- *Service: Presents* node must contain an exact value that matches a value in *Profile: ID* node, though the content would not affect the matching result. This is due to the fact that *Service: Presents* is a pointer to the corresponding *Profile*.
- *Profile* node must contain *HasInput* and *HasOutput* nodes.
- *Profile:HasInput* and *Profile:HasOutput* are indicators for *Input* and *Output* nodes.
- Under the Input/Output nodes, *ParameterType:Content* attribute contains a semantic annotation which is used by semantic resource matchers.

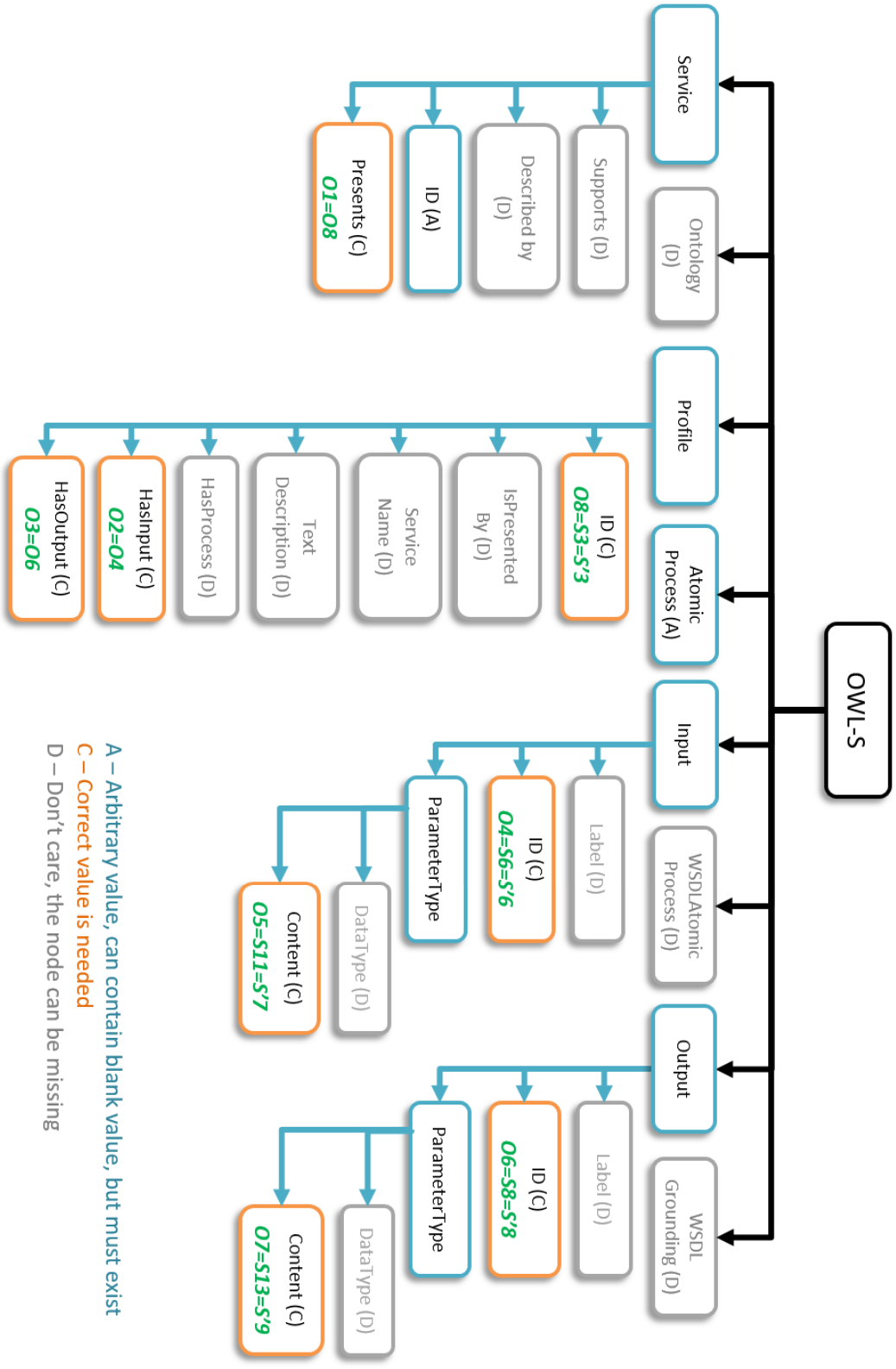


Figure 7.1.: Structure of OWL-S description with essential information for resource matching process.

7.2.2. SAWSDL1.1 Description

On the other hand, as shown in Figure 7.2, a SAWSDL1.1 description requires the following rules.

- *Service:Port* node, *PortType:Name* attribute, and *PortType:Operation:Name* attribute must exist and should not be left blank.
- *PortType:Operation:InputMsg* and *PortType:Operation:OutputMsg* nodes must be presented.
- *InputMsg* and *OutputMsg* nodes indicate *Message:Request* and *Message:Response* nodes. The value of these nodes must be correct.
- *Request* and *Response* nodes must contain *Part:PartType* and *Part:PartName* attributes.
- *PartType* attribute is a pointer to *Types:Input* or *Types:Output* node. It must contain correct information that similar to *Types:Input:Name* and *Types:Output:Name*.
- *Message:Request/Response:Part:PartName* attribute can have any value, including a blank value.
- A semantic annotation can be found in an element "ModelReference" under *Types:Input* or *Types:Output* node. These nodes must contain correct information.

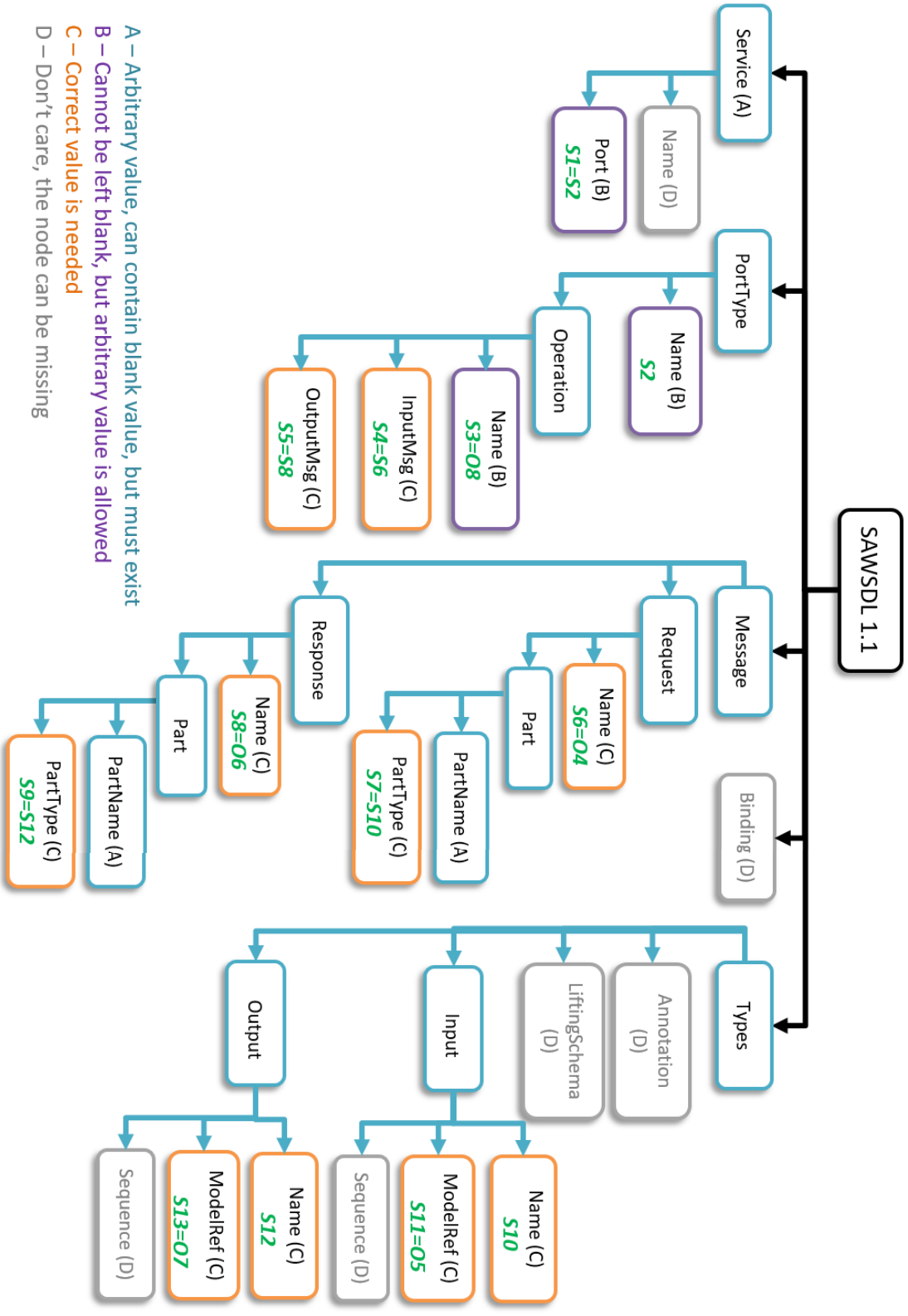


Figure 7.2.: Structure of SAWSDL1.1 description with essential information for resource matching process.

7.2.3. SAWSDL2.0 Description

A SAWSDL2.0 is similar to SAWSDL1.1, except that *PortType* becomes *Interface* and *Message* node is integrated under *Type* node. The structure of SAWSDL2.0 with the level of necessity for its elements is shown in Figure 7.3.

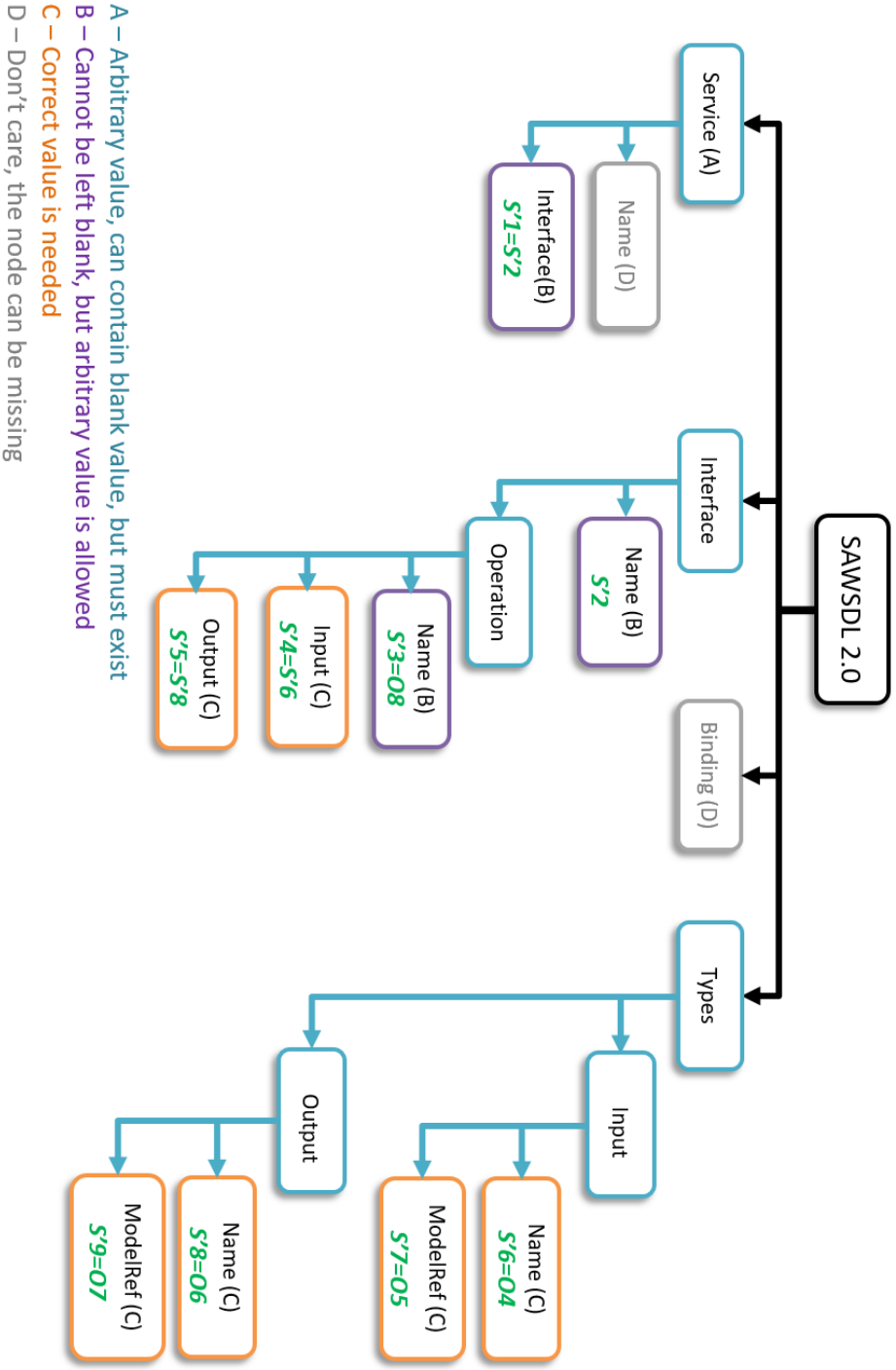


Figure 7.3.: Structure of SAWSDL2.0 description with essential information for resource matching process.

8

Request Preparation

In this chapter, we explain how to change free-text keywords into a formatted request. By applying the knowledge from Section 7.2, *Essential information required for resource matching*, we can realize the request constructor (Section 8.1). Also, by analyzing the common features of different formalisms, we implemented a converter engine as elaborated in Section 8.2. This converter can be used either to transform a newly generated description from Section 8.1 or an existing description into another formalism.

8.1 Request Constructor

Consider Ex.2 scenario: the resource that the user is looking for should accept a city name as an input, and return weather condition as an output. Optionally, the behavior (operation description) of the resource can be described as "WeatherService". Consequently, a request message can be constructed from plain texts. A user can use a search UI as a simple search by providing free text keywords according to Requirement R4 (*the discovery process should offer a basic search for resource descriptions*). The example of keywords could be ”get weather report” or ”weather service”.

In addition, we provide a UI for users to fill in keywords by functionality. This would resolve Requirement R5 (*the discovery process should offer an advanced search considering semantic annotations and syntax of keywords*). An advanced search requires keywords in particular fields, i.e. input, output, and operation description of a resource. For example, **Input**: ”city”, **Output**: ”weather”, and **Operation**: ”weather service”.

Figure 8.1 shows how the given keywords are used to construct a minimum request based on example descriptions from a repository. The sampling of descriptions over the repository is done via a text-based search API. In addition, context information from Chapter 6, *Context Extraction* can be appended to a query in this process.

Moreover, this search API can be used to filter a list of all possible relevant resource descriptions out of the whole repository. Instead of comparing our request with 100,000 descriptions (which would take a thousand times longer than processing 100 descriptions),

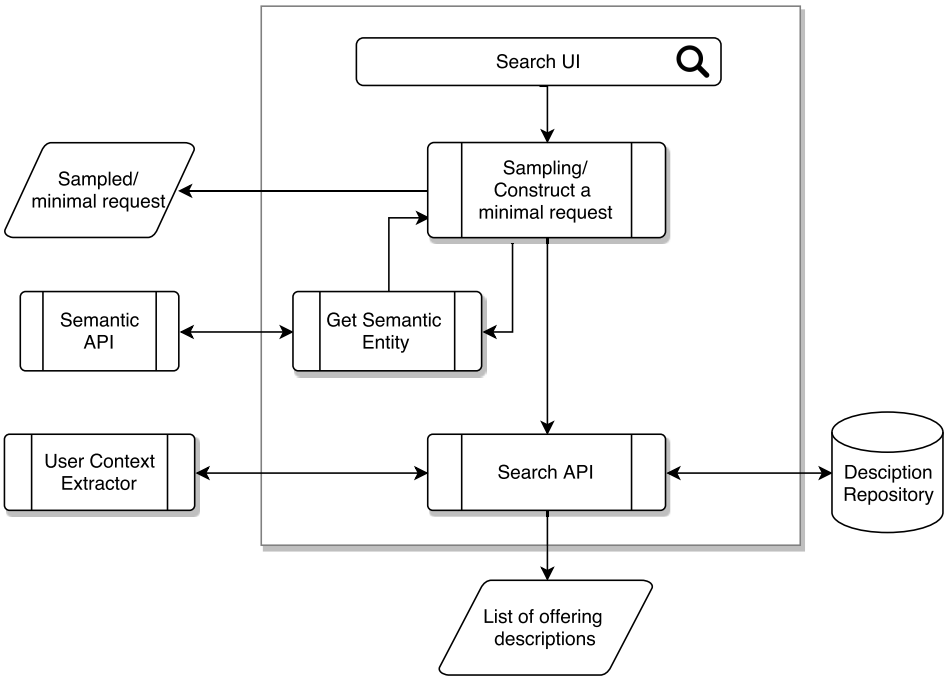


Figure 8.1.: Request constructor diagram.

it is more realistic to filter down the set of descriptions. Although this constructor can provide possible resources with semantic query expansion, the ranking algorithm is yet based on a text-based search, not semantic knowledge. Thus, this work needs to include real ontology-based search engines (i.e. resource matchers) in the service discovery.

8.1.1. Algorithm

Figure 8.2 depicts the workflow of the request constructor. First, the original keywords given by a user will be used for sample requests from the repository. For instance, a keyword ”weather service” is considered to be matched to a resource ”country weather service” and ”global weather report” in a description repository. These two supposedly matched descriptions are used further as query messages for ontology-based resource matchers.

As suggested by [MGMR02] and [QHC06], semantic terms can further improve the results of documents matching. The semantic extension can be used when the original keywords do not suffice to retrieve relevant descriptions. The semantic terms could be derived from an external Semantic Library API. With the extended terms, we look for samples in the repository again. We can get a wider range of resources like ”forecast service”.

If there is still no result, this is very unlikely to find such a resource in the repository. However, a minimal request can be created with the original keywords specified in the input, output, and operation fields.

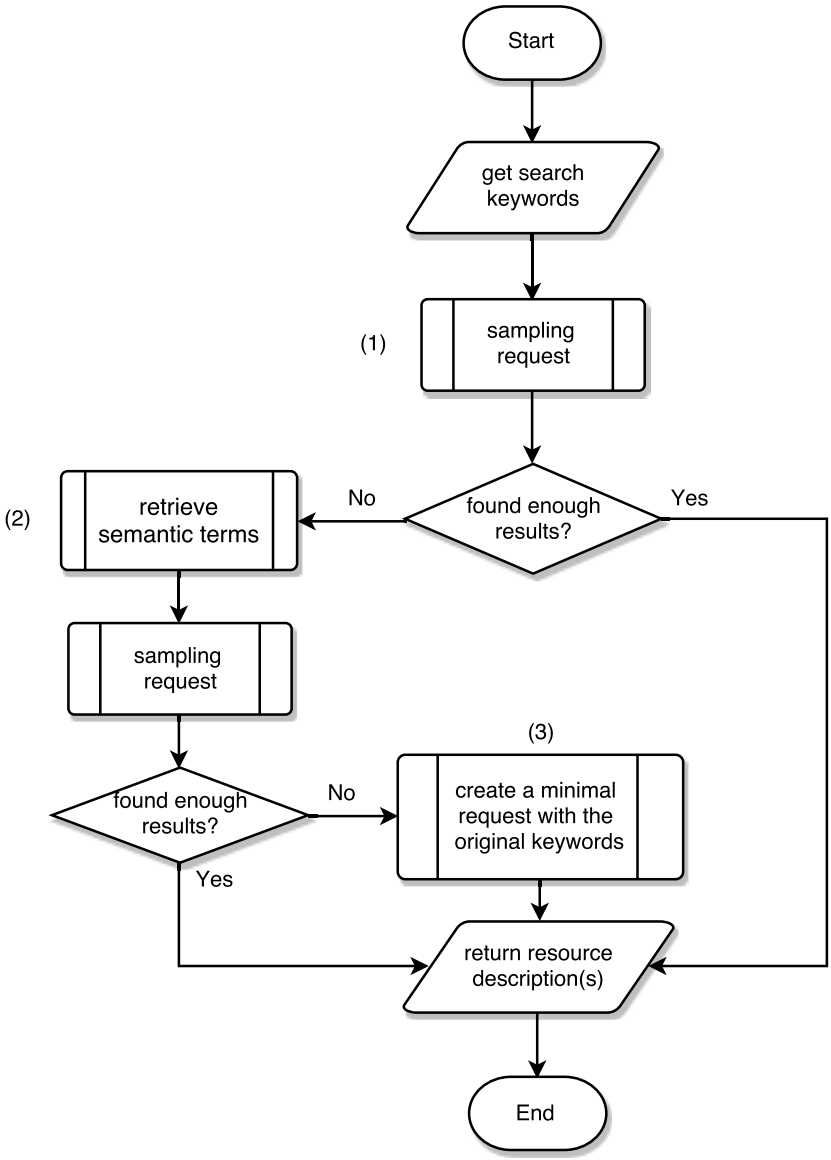


Figure 8.2.: Request constructor data flow.

8.1.1.1. Sampling relevant descriptions

A minimal description is needed by resource matchers as an input (request) for matching with descriptions in a repository (offers). Before constructing the minimal description from scratch, we observe roughly over the repository for existing descriptions that match the keywords. However, each keyword has a different priority depending on the source and functionality of it. In some cases, the keywords do not match with any existing de-

scriptions, so broader semantic terms should be included to yield the possibility of return results.

Figure 8.3 illustrates how each keyword is weighted in a query. When we want to prioritize the original keywords over the semantically extended terms, we can set the *original_booster* value to be higher than the *semantic_booster*. Nevertheless, for the first try, we do not include semantic terms yet.

Furthermore, the mode of search, either simple or advanced, also affects how the request is sampled.

a. A simple search
   We simply use an option, "match all", to search for every description that the keywords appear in.

b. An advanced search
   If the keywords are from an advanced search, the operation description will be prioritized (e.g. giving a boost factor: 3) over an input/output descriptions (e.g. giving a boost factor: 2). These factors are configurable as boosting value for operational, input and output keywords. We look for descriptions that contain the given keywords only in the specified fields.

8.1.1.2. Extending a request with semantic entities

According to the flow in Figure 8.3, when an exact match cannot provide enough samples, we use semantic terms to retrieve more relevant results. The initial keywords should have a higher priority than the semantic entities. In the query, this priority is defined as a *boost* parameter. The original keywords' boost factor and semantic entities' boost factor can be assigned separately. For example, the terms "country city weather" are provided by a user. These key terms will get semantic entities like "state land municipality condition atmospheric" which should be less important than the original terms. Therefore, we can configure the boost factor "3" to the original keywords, and "1" for the semantic keywords.

As shown in Listing 8.1, the keywords are only differentiated by their origins. The keywords are submitted for a simple search, all keywords are treated equally and will be searched in every field of descriptions.

On the other hand, in an advanced search, the keywords are parsed into three fields: input, output, and operation. As exemplified in Listing 8.2, here, we want to prioritize the operation field over the input and output fields. Thus, we set the *operation_boost* value to 3, *input_boost* and *output_boost* values to 2. The term "country" is an original keyword from the user, and also is defined as an input of the resource. Thus, its boost value is $original\_booster \times input\_boost = 6$. The term "state" is a semantic entity, which behaves as an input, so the boost parameter becomes $semantic\_booster \times input\_boost = 2$. Since "weather" is an original keyword, defined as an operation description, then it is set to $original\_booster \times operation\_boost = 9$. These keywords will be used to compare only in three specific fields, as suggested by their names.

```
{ "query":{
  "bool":{"should":[
      {"query": "country city weather",
         "boost":3},
      {"query": "state land municipality condition atmospheric",
         "boost":1}
  ]} },
 "sort":{"_score":"desc"}, "size":10
}
```

Listing 8.1: Example query for simple search using original_boost = 3, and semantic_boost = 1

```
{ "query":{
  "bool":{ "should": [
    {inputField: {"boost": 6, "value": "country"}},
    {inputField: {"boost": 2, "value": "state"}},
    {inputField: {"boost": 2, "value": "land"}},
    {inputField: {"boost": 6, "value": "city"}},
    {inputField: {"boost": 2, "value": "metropolis"}},
    {inputField: {"boost": 2, "value": "municipality"}},
    {outputField: {"boost": 6, "value": "weather"}},
    {outputField: {"boost": 2, "value": "condition"}},
    {outputField: {"boost": 2, "value": "atmospheric"}},
    {outputField: {"boost": 2, "value": "phenomenon"}},
    {operationField: {"boost": 9, "value": "get"}},
    {operationField: {"boost": 3, "value": "return"}},
    {operationField: {"boost": 9, "value": "weather"}}}
  ]}
 }, "sort":{"_score":"desc"}, "size":20
}
```

Listing 8.2: Example query for advanced search with operation_boost value = 3, input_boost = 2, output_boost = 2, original_boost = 3, and semantic_boost = 1

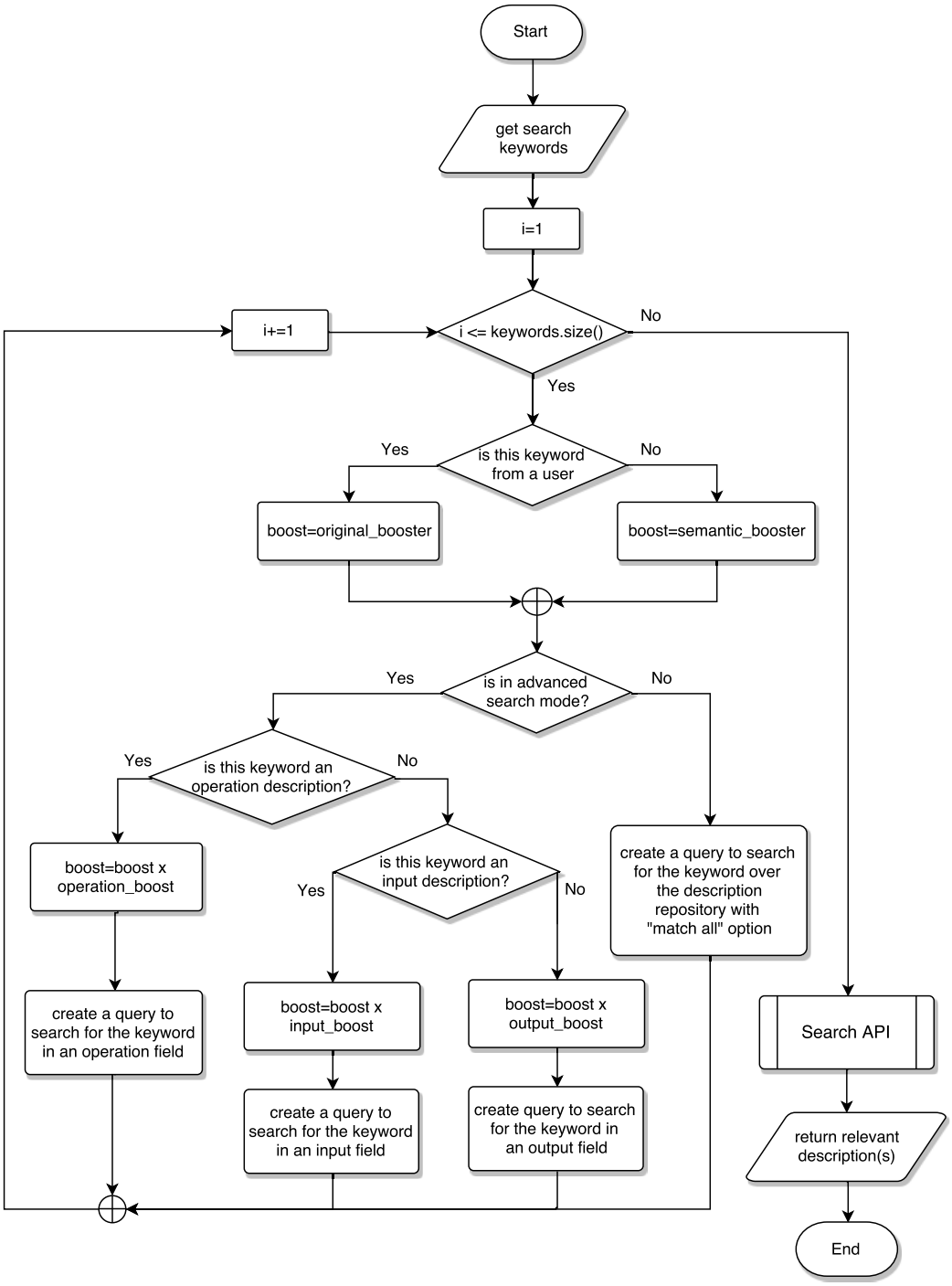


Figure 8.3.: Request sampler data flow.

8.1.1.3. Creating a minimal description from keywords

When the sampling request with semantic extension is still insufficient, we create a description from scratch. The minimum requirement is that the operation name, input name, and output name must be filled in. Table 8.1 demonstrates major properties in supported formalisms. Note that the input/output type ontology fields are optional but can highly enhance the description if they are provided.

Property	OWL-S	SAWSDL1.1	SAWSDL2.0
Operation name	RDF > profile:Profile ID	definitions > portType:Operation name	description > interface:Operation name
Input name	RDF > process:Input ID	definitions > message > part name	description > interface > operation > input element
Input type ontology	RDF > process:Input> parameterType	definitions > types > schema > complexType modelReference	description > types > schema > element modelReference
Output name	RDF > process:Output ID	definitions > message > part name	description > interface > operation > output element
Output type ontology	RDF > process:Output > parameterType	definitions > types > schema > complexType modelReference	description > types > schema > element modelReference

Table 8.1.: Main properties in OWL-S, SAWSDL1.1, and SAWSDL2.0

An operation name is defined as *Profile ID* in OWL-S. While SAWSDL1.1 refers to the operation name as *portType:name*, and *interface:name* in SAWSDL2.0. An input and an output name are described in OWL-S as *process:InputID* and *process:OutputID* respectively. In contrast, SAWSDL1.1 refers to both input and output in *message:part name*. Meanwhile, SAWSDL2.0 separately cites an input name as *interface:operation:input element*, and an output name as *interface:operation:output element*.

We can extend the description by annotating ontology to each input and output element. In OWL-S, *process:Input/Output:parameterType* node indicates the link to a corresponding ontology term, whereas SAWSDL1.1 and SAWSDL2.0 specify an ontology link as *types:schema:element:modelReference*. For instance, the term "City" with semantic annotation can be referred to its super classes within an ontology model, such as "Geopolitical Area" or "Land Area."

8.2 Request Converter

As previously mentioned, a resource can be described in any formalism. This thesis covers OWL-S, SAWSDL1.1, and SAWSDL2.0 to prove our concept. The request constructor can provide either OWL-S or SAWSDL descriptions. However, it is less likely that one resource would be described in both OWL-S and SAWSDL formats. If we focus on one formalism, it is possible that we could miss the resources those are described in another format. Thus, the request converter is developed to enable the service discovery to use both types of description formalisms.

[MPW07] summarizes how to map SAWSDL descriptions to OWL-S. In this work, we created mapping schemas for OWL-S, SAWSDL1.1, and SAWSDL2.0 as depicted in Figure 7.1, Figure 7.2 and Figure 7.3 respectively. The request converter will look for a mapping schema and convert a request from an initial formalism to the destination formalism(s). The destination formalism will be chosen regarding the type of matchers used in the next process (see Figure 5.1).

Inside SAWSDL1.1 definitions element, a resource can have several ports corresponding to different binding protocols. *Service:Name* in SAWSDL1.1 is equivalent to *Service:ID* in OWL-S. Each SAWSDL1.1 *PortType* has ”Operation(s)”, of which name is equivalent to *Profile:ID* of OWL-S. However, neither *Service:Name* nor *Operation:Name* (as well as *Service:ID* and *Profile:ID* in OWL-S) are considered in the resource matcher reasoning. Thus, we can assign any value to these fields.

Message:Request:Part:Name, of SAWSDL1.1 is equivalent to OWL-S *Input:ID*, whereas *Message:Response:Part:Name* of SAWSDL1.1 is correspondent to OWL-S *Output:ID*. The semantic annotation of inputs and outputs can be mapped from SAWSDL1.1 *Input:ModelReference* and *Output:ModelReference* to OWL-S *Input:ParameterType* and *Output:ParameterType*.

SAWSDL2.0 follows the similar trait of SAWSDL1.1, except that *PortType* is renamed to *Interface*, and *Message* node is removed. *Interface* node also contains an *Operation*. In contrast to SAWSDL1.1, this *Operation* node specifies input(s) and output(s) names directly. Thus, this reduces one step to access the semantic annotation node. The summary of necessary information and their containers are shown in Table 8.1.

Now we have conversion templates. The next question is when to apply this conversion. We classify the usage of the request converter into two ways, as we refer to as modes of conversion.

8.2.1. Modes of Conversion

The conversion process can either operate before the request is made (offline mode) or be called on demand (online mode). In the first mode, we rely on a single formalism. All descriptions in a repository will be prepared in the same language, and ready to be matched

by matchers specialized in that language. For example, if we choose to use OWL-S matchers, we can use the converter to convert all SAWSDL descriptions to OWL-S formalism before we match them to a query.

Contrarily, when we use multi-type matchers to match a query with different formalisms. This requires the online conversion. A query will be converted to different formalisms and be fed to the corresponding matchers.

8.2.1.1. Conversion on the description repository (offline mode)

We can use the request converter to prepare descriptions before the request is made. This conversion prepares all descriptions in the repository into a single formalism. This offline conversion is recommended when a certain description formalism is used. For example, if we aim toward the real-time responsive discovery, it is recommended to use the SAWSDL formalism.

We can configure our discovery engine to create an initial request in SAWSDL and deploy only SAWSDL based matchers. Therefore, it is more tangible to convert all OWL-S descriptions into SAWSDL beforehand.

On the other hand, if we aim for using OWL-S based matchers, the request constructor can be configured to generate an OWL-S request. Also, all the SAWSDL descriptions in the repository must be converted as depicted in Figure 8.4. This process should be called upon when a new description is added to a description repository.

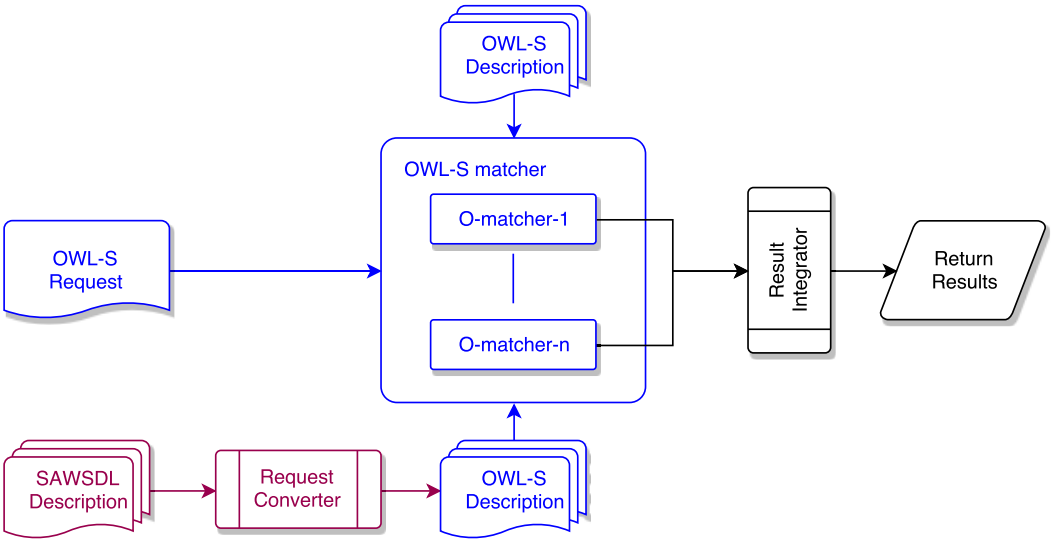


Figure 8.4.: Request converter performs in the description repository (offline mode).

8.2.1.2. Conversion on demand (online mode)

The offline conversion has disadvantages that it is based on a single formalism and takes more storage to keep different versions of descriptions. When the discovery settings are dynamic, e.g. having SAWSDL and OWL-S based matchers running simultaneously, online conversion is needed. For example, the SAWSDL request will be converted to an OWL-S formalism as shown in Figure 8.5. This approach utilizes the advantage of both OWL-S and SAWSDL matchers at once.

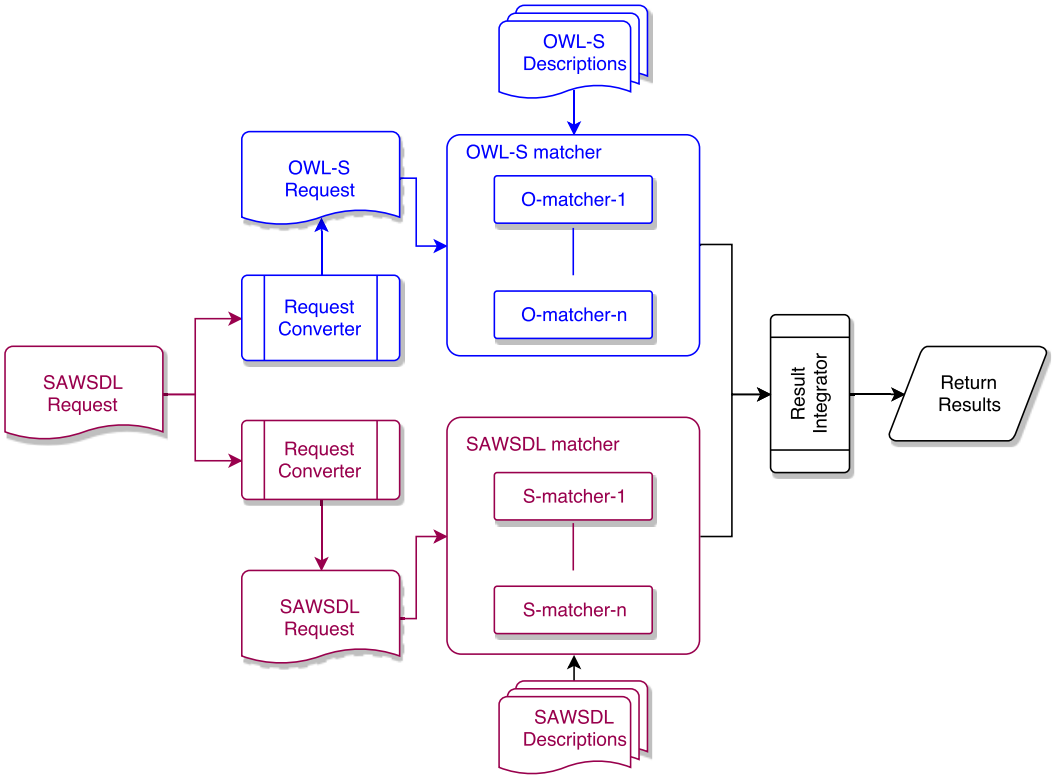


Figure 8.5.: Request converter performs on demand (online mode).

8.2.2. Conversion Algorithm

The main classes in the converter are reader and writer classes. Each formalism has reader and writer classes. The reader class is responsible for parsing and extracting essential information (see Section 7.2, *Essential information required for resource matching*) from a description, whereas the writer class is responsible for creating a description in a particular formalism with mandatory information.

A conversion from OWL-S to SAWSDL requires an OWLreader and a WSDLwriter. On the other hand, conversion from SAWSDL to OWL-S requires a WSDLreader and an OWLwriter[1].

8.2.2.1. Conversion from SAWSDL to OWL-S

This conversion requires a WSDL1.1 or WSDL2.0 reader class, and an OWL writer class. Figure 8.6 illustrates these classes and connections between them. The algorithm used for creating an OWL-S description is depicted in Figure 8.7. The first level of both OWL-S and SAWSDL is called *Service*. A description can have more than one service, and each SAWSDL service can contain several *PortTypes*. Each SAWSDL *PortType* provides information of resource's input(s) and output(s).

For OWL-S, an input/output message from SAWSDL must be stored in *Profile* node. While *Process* node maintains the semantic annotations of the inputs and outputs (if they are available in SAWSDL *ModelReference* attribute). All inputs and outputs from every *PortType* and *Service* from SAWSDL are transferred to the OWL-S description in each iteration.

[1]The WSDLreader can parse both WSDL and SAWSDL descriptions. Likewise, the OWLreader can parse OWL and OWL-S descriptions.

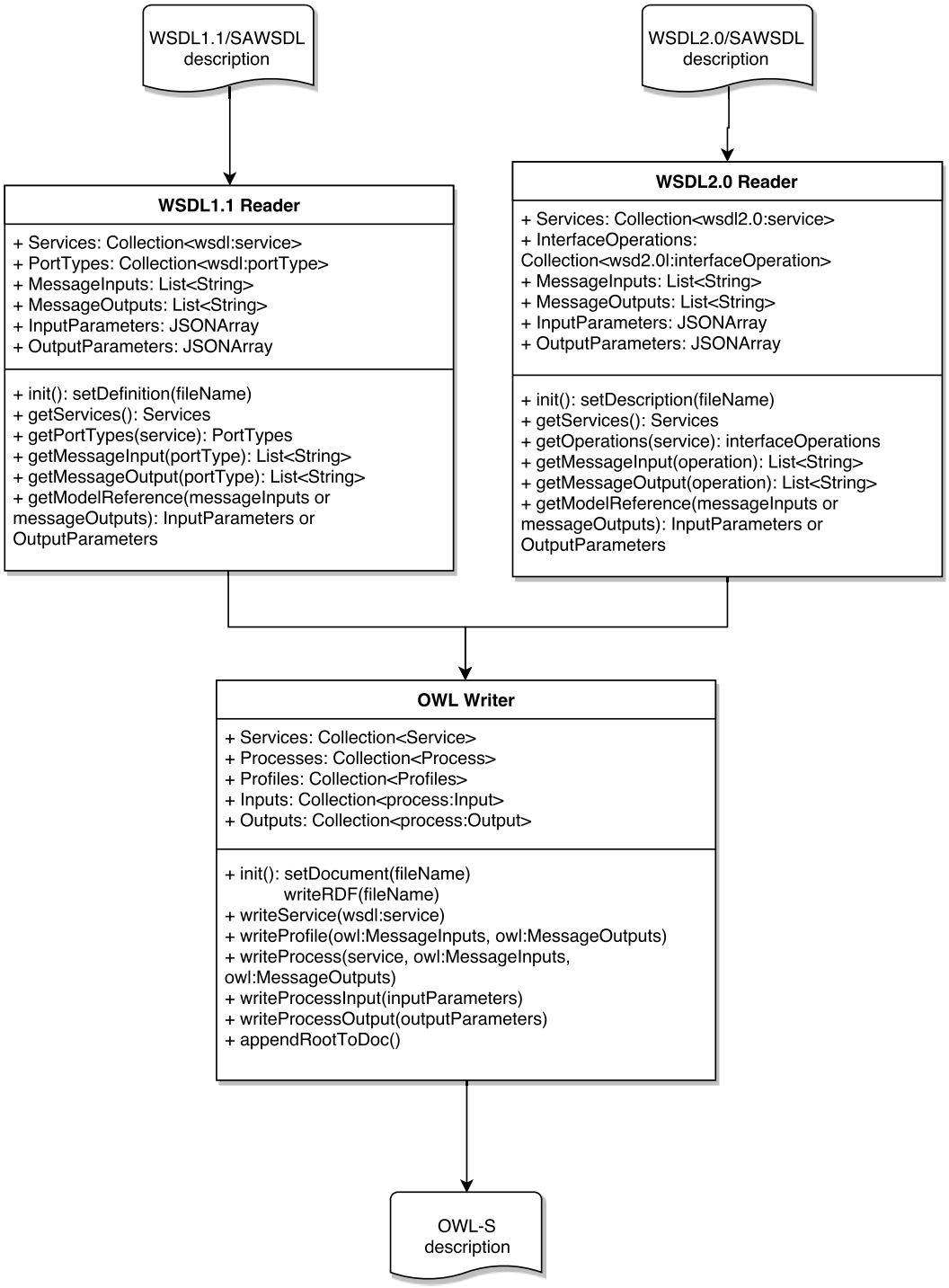


Figure 8.6.: Request converter class diagram showing data flow of SAWSDL (1.1 on the left side, and 2.0 on the right side) to OWL-S conversion.

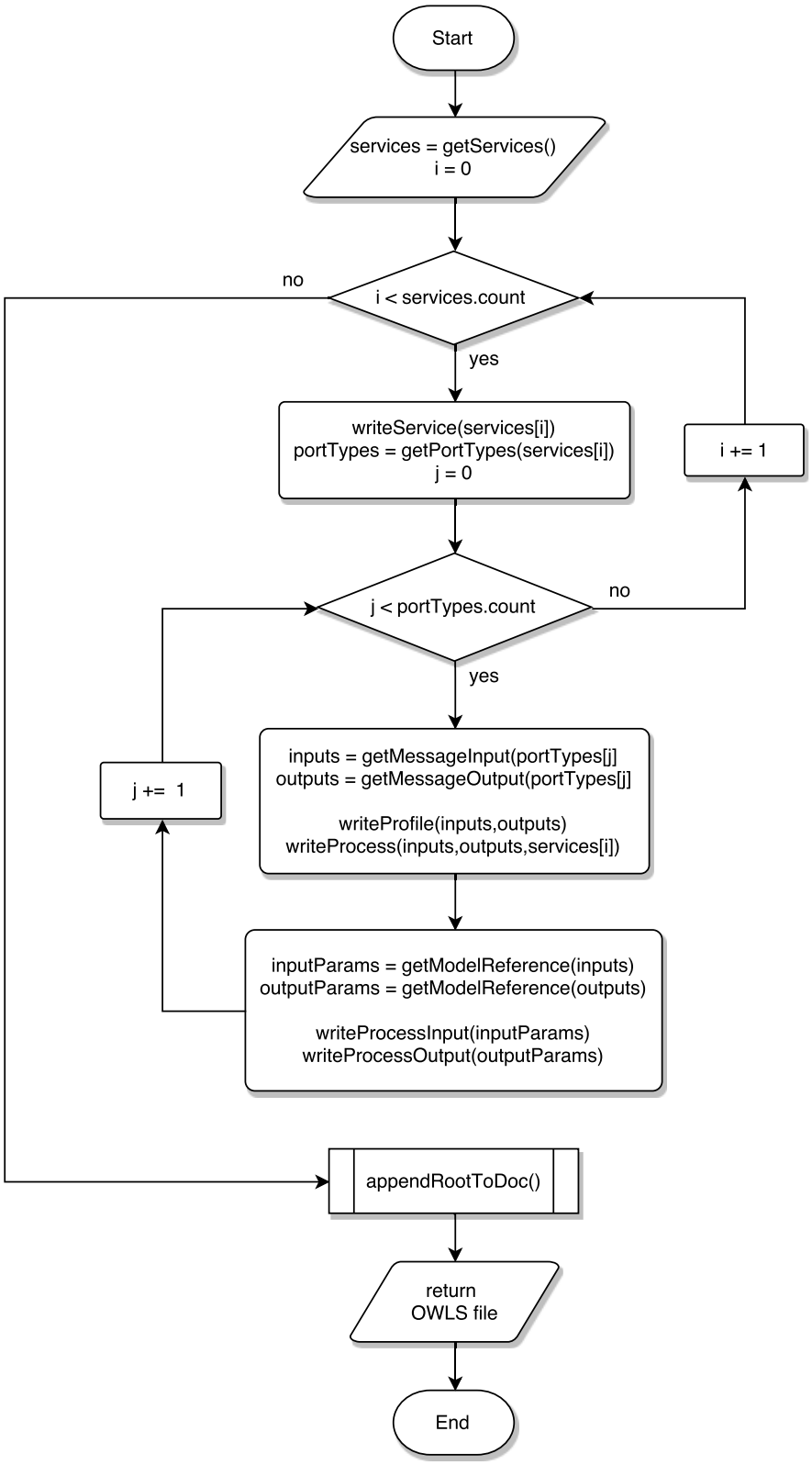


Figure 8.7.: Conversion from SAWSDL1.1 to OWL-S description flow chart.

8.2.2.2. Conversion from OWL-S to SAWSDL

This conversion requires an OWL reader class and a WSDL writer class as depicted in Figure 8.8. When an OWL-S is converted to SAWSDL, as illustrated in Figure 8.9, all *Profiles* in every *Services* are extracted. Inputs and outputs must be inserted to a SAWSDL *Types* node. *Message* nodes contain semantic annotations of the input(s) and output(s). The OWL-S *profiles* are equivalent to SAWSDL *operations*.

After all the *profiles* are extracted, the SAWSDL *PortType* will be created from OWL-S *Processes*, previously created SAWSDL *Operations*, and SAWSDL *Service:Port* attribute. When all the OWL-S services are read and written to SAWSDL formalism, we will have SAWSDL descriptions ready for a matching process.

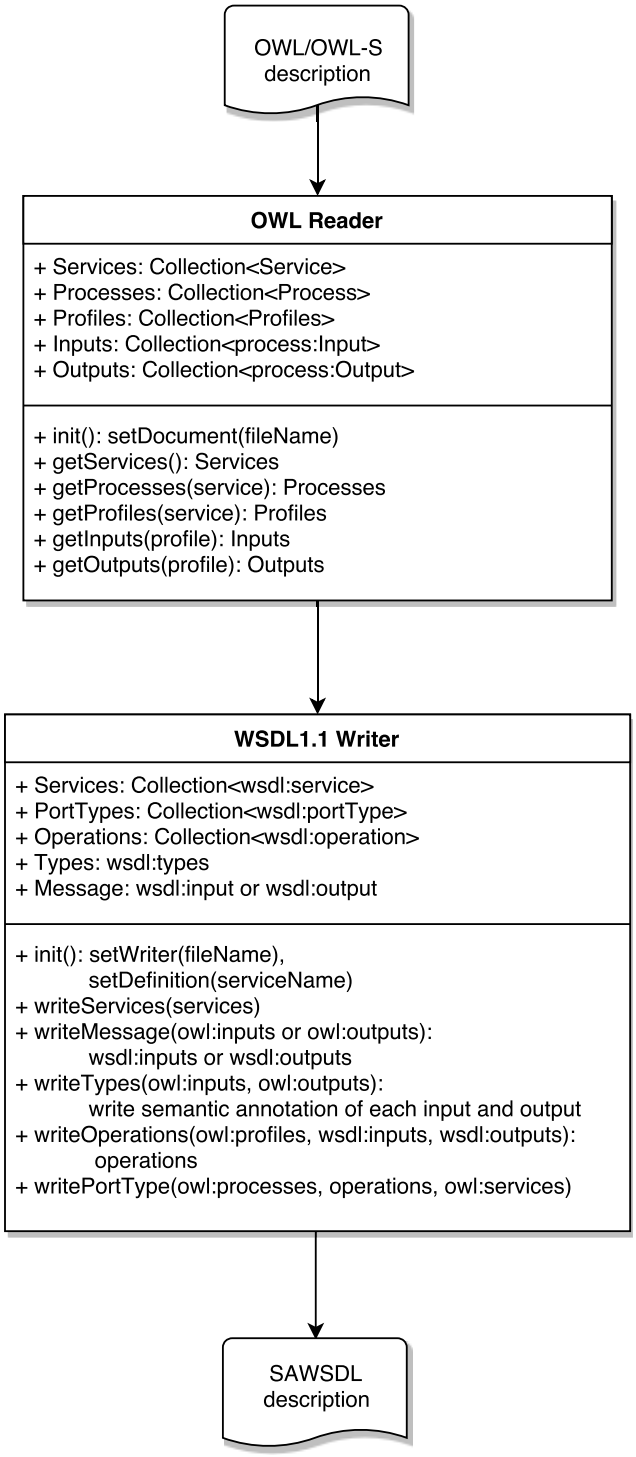


Figure 8.8.: Request converter class diagram showing data flow of OWL-S to SAWSDL1.1 conversion.

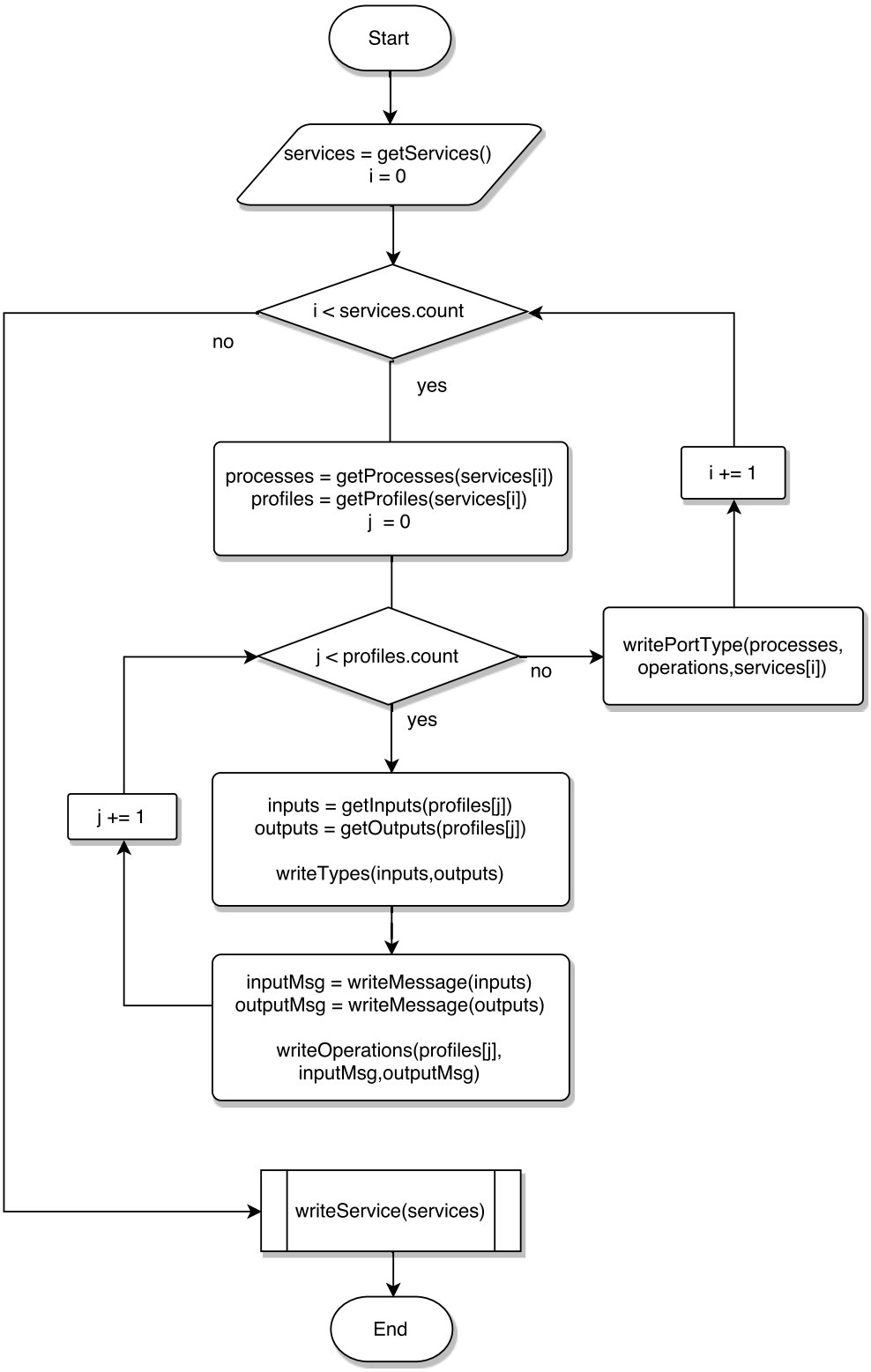


Figure 8.9.: Conversion from OWL-S to SAWSDL1.1 description flow chart.

9 Result Integration

Previously in Chapter 8, we describe our main components for preparing a request for the resource discovery. The request is propagated to resource matchers. Since using multiple matchers increases the chance of getting better results, we suppose that our resource discovery would get multiple result sets as well. However, to make use of these results, we need to combine them in the way that the rank is maintained.

The integration of results is not trivial as the results from matchers can be incongruent. Furthermore, regarding the similarity aggregation methods compared in [KNL13] and [DMD+03], using homogeneous weights on multiple matchers returns unsatisfactory results. Each matcher, therefore, should be assigned a different weight depending on its performance.

This chapter provides an overview of existing techniques for result merging and discusses their advantages and drawbacks. Then we present a new method more suitable for our purposes. We aim to resolve the conflict in the ranking between each matcher. Also, the integrator should be able to reject poor results to improve the quality of the final result.

9.1 Existing Techniques for Result Merging

[LMS+05] and [Jad12] discussed different methods of merging multiple search engines' results. These techniques can be categorized into three types: score-based, rank-based and content-based.

9.1.1. Score-based Merging Algorithm

This is the most straightforward technique. Assuming all search engines have comparable similarity scores, then all results can be merged by linear combination methods discussed in [RS03], which accumulate each item's normalized score from all search engines and reorder them to a final ranked list.

For example, given a similar query to a search engine X and a search engine Y, we get different results from them. X returns a set of results with similarity values appended to each

result: $[(r_1, 0.95), (r_2, 0.8), (r_3, 0.63), (r_4, 0.75), (r_5, 0.13)]$, where the highest similarity score of X is 1 and irrelevant results get 0 as a similarity score.

In contrast, Y assigns a similarity score to each result from a range 0 to 10. It returns a set of results for the same query: $[(r_1, 8.1), (r_2, 9.8), (r_3, 5.2), (r_4, 6.9), (r_5, 4.6)]$.

First, we normalize the similarity scores from Y to make them compatible with the result of X. Therefore, Y result = $[(r_1, 0.81), (r_2, 0.98), (r_3, 0.52), (r_4, 0.69), (r_5, 0.46)]$.

Then we sum the score of X's and Y's results. Merged result = $[(r_1, 1.76), (r_2, 1.78), (r_3, 1.15), (r_4, 1.44), (r_5, 0.59)]$.

Afterwards, we can sort the result in a descending order regarding the accumulated similarity scores and obtain the final result: $[r_2, r_1, r_4, r_3, r_5]$.

However, this approach does not take into account that different search engines differ in their reliability. Plus, not every search engine provides similarity scores to clients, as they tend to use these scores internally. Thus, this does not work well in practice.

9.1.2. Content-based Merging Algorithm

Among the content-based merging algorithms, the approaches like Search Result Records (SRRs), Top Document (TopD) and their successors are claimed to be the most effective [LMS+05].

- **SRRs** are dynamically generated HTML texts containing metadata to be displayed as search results (snippets). For each document, the similarity between the query and its title, and the similarity between the query and its snippet are computed. Then the two similarities are linearly aggregated. The weight of each search engine is computed based on the *Okapi* probabilistic model [Rob01].

  The *Okapi* model requires the information of document frequency (*df*) of each term. The *df* of the query term *t* in each search engine is approximated by the number of documents containing term *t* within their titles and snippets.

  Finally, the estimated similarity of each result is adjusted by multiplying the relative deviation of its source search engine's score to the mean of all the search engine scores.

- **TopD** algorithm uses the similarity between a query *q* and the top-ranked document (d_{ix}) returned from search engine X to estimate a score of the search engine (S_X). This score reflects how good the search engine is on the user query. The highest ranked document is the most relevant to the user query. Nevertheless, fetching the top-ranked document from its local server causes some extra network delay to the merging process. The similarity between query *q* and d_i is calculated by using the sum of the Okapi weight of each query term *t*.

However, these are rather resource-expensive since they need to download documents for analysis.

9.1.3. Rank-based Merging Algorithm

This method is straightforward and versatile. It assigns each item a score corresponding to the rank in which it appears within each search result. It neglects the original scores from the search engines, then assigns a new score to each item. This does not require document analysis, so it saves time and memory consumption. The simplest rank-based method is to consider the best rank from all search engines directly using voting systems such as Borda's Positional method or Borda count [Pac12]. Moreover, when taking the reliability of search engines into account, we can use an approach like weighted Borda-Fuse.

- **Borda's Positional** computes the Lp-Norm of the ranks in different search engines. That is, with a query q:
  Merged rank(q) = $\Sigma(Rank_1(q)^p,\ Rank_2(q)^p,\ \cdots,\ Rank_n(q)^p)^{\frac{1}{p}}$.

  This algorithm has considered the L1-Norm which is the sum of all the ranks in different search engine result lists.

- **Weighted Borda-Fuse** algorithm treats each search engine unequally. The merged results are depending on weights which represent the reliability of each search engine. These weights can be set by the users in their profiles. Thus, the score (vote) of i^{th} result from *j* search engine is:
  $V(r_{i,j}) = w_j * (\max_X(r_j) - i + 1)$.

  Where w_j is the weight of *j* search engine and r_j is the numbers of results rendered by the search engine j. Retrieved items that appear in more than one search engines receive the sum of their votes.

- **Borda count** , a voting-based data fusion method, is simplistic and efficient in terms of quality and time consumption. In Borda count, each search engine represents a voter. Each voter ranks a fixed set of candidates according to preference. The top rank gains the highest score. Consecutive ranks get lower scores. Then, all vote counts of each candidate will be collected from all electors.

  For example, as shown in Figure 9.1, three candidates: A, B, and C are voted by six electors. B is voted 1^{st} rank three times, voted 2^{nd} rank two times, and voted 3^{rd} rank once. We assign a score for the first rank as 2 (most important), the second rank as 1, and the third rank as 0 (least important). Thus, B will get the accumulated score of: $(3 \times 2) + (2 \times 1) + (1 \times 0) = 8$.

  The item which gets the highest sum of scores becomes the first rank. As a result, the Borda count should rank the example as B (1^{st}), A (2^{nd}) and C (3^{rd}).

In this work, Borda count is used for assigning scores to each item from resource matchers' results due to its efficiency and simplicity. Total scores are arranged to provide the final result and used to calculate a weight value for each resource matcher. This weight value represents how reliable each resource matcher is for a query. The following section will elaborate on these technical details.

Voter#	1	2	3	4	5	6
1st	B	B	B	A	A	C
2nd	A	A	C	B	C	B
3rd	C	C	A	C	B	A

A = (2*2) + (2*1) + (2*0) = 6

B = (3*2) + (2*1) + (1*0) = 8

C = (1*2) + (2*1) + (3*0) = 4

Figure 9.1.: Example of Borda count method.

9.2 Merging Algorithm

We initially treat all the matchers with identical weights and then merge all results all results into a single list. Following the concept presented by [MGMR02], the merged result is compared to the original results to calculate weights for each matcher. Then, the process is iterated until the best quality result is achieved.

The overview of the result integrator flow is depicted in Figure 9.2. When we are interested in top t results and use n different matchmakers, a matrix with dimension $[n \times t]$ is created from all matchmakers' results. Each element contains a resource's unique id and similarity score assigned by each matchmaker.

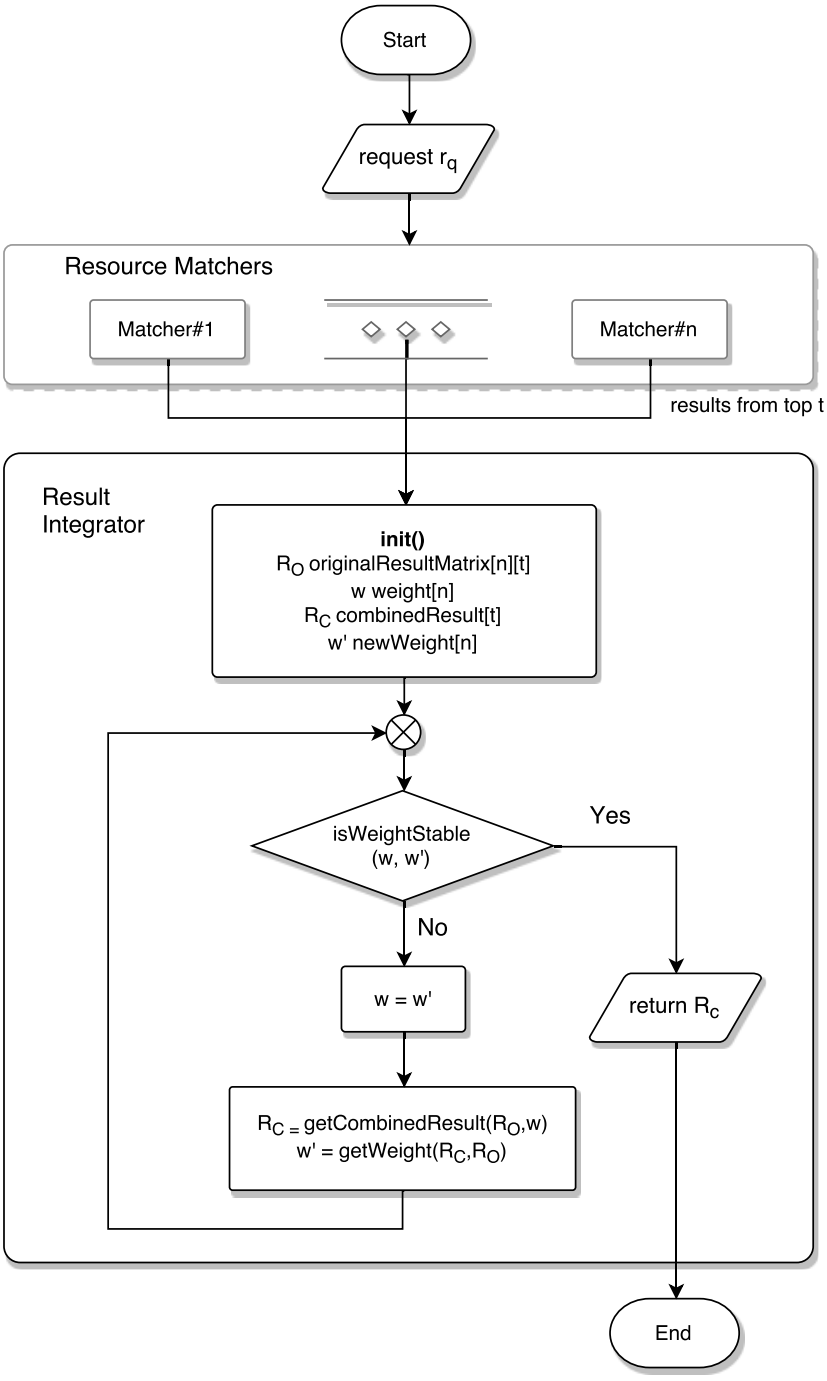


Figure 9.2.: Flow chart of the result integrator.

Though each matchmaker has a different method to calculate a similarity score, we can normalize these scores to extract the information about the difference between each rank. If the similarity score is not available, we can use Borda count technique to predict the score from ranks.

The flow starts with an initial function to set default values for calculation. These values are R_O (a collection of original result matrices), *w* (an initial weight used for defining the reliability level of each matcher), R_C (a combined result), and *w'* (weight computed from a distance between R_C and R_O).

R_C relies on the average result from all matchers. However, some matchers return inaccurate results. To eliminate the poor results, we measure a reliability weight of each matcher. A matcher gets a higher weight when it returns a result that is close to R_C. The poorest result (lowest weight) will be removed from R_O. Then, R_C will be computed again. The removal of poor results and recalculation of R_C will be repeated multiple times until the optimum result is obtained.

The condition to stop this iteration is that the matchers' weights do not change between consecutive rounds. In other words, when there is no poor matcher left to be eliminated, the weight in the current round and the previous round are indifferent (or insignificantly different). Thus, the iteration is terminated and the result R_C from this round will be used.

9.2.1. Initial function

In the init() function, shown in Figure 9.3, results from resource matchers are accumulated into one matrix, R_O, with a dimension of $[n \times t]$.

For example,

> *we use three matchers; MA, MB, and MC, and consider four top ranks of matching results. Given a request r_q, the matcher MA returns a resource α as the first rank, γ as the second rank, δ and ϵ as the third and fourth rank respectively. Thus,*
>
> $R_{MA}(r_q) = \begin{bmatrix}\alpha & \gamma & \delta & \epsilon\end{bmatrix}$,
>
> *where the leftmost element represents the highest matched item and, vice versa, the rightmost element is the least likely similar item to the request.* (Ex. 3)
>
> *The result from the matcher MB produced from the same request is:*
> $R_{MB}(r_q) = \begin{bmatrix}\beta & \gamma & \alpha & \delta\end{bmatrix}$.
>
> *The result from the matcher MC is:*
>
> $R_{MC}(r_q) = \begin{bmatrix}\alpha & \delta & \beta & \gamma\end{bmatrix}$.

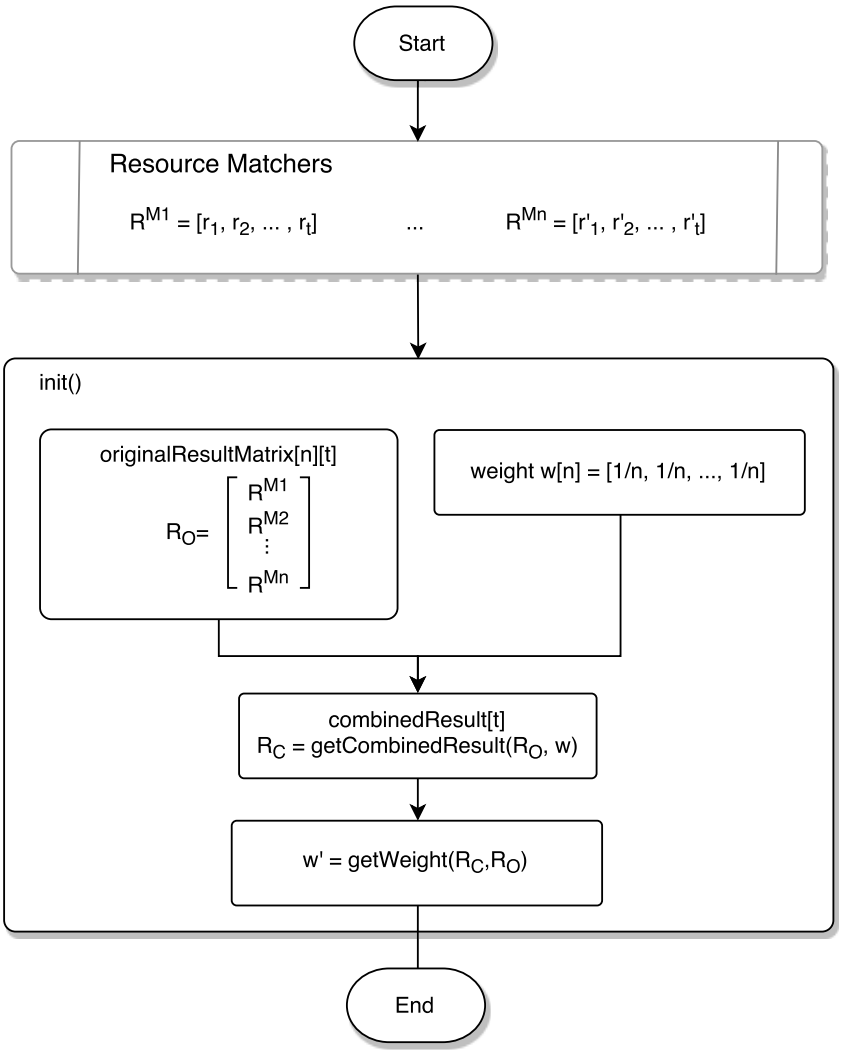


Figure 9.3.: 'init()' function of the result integrator.

We can simply combine R_{MA}, R_{MB}, and R_{MC} together into one matrix, and append a score to each element. Therefore,

$$R_O(r_q) = \begin{bmatrix} r_1^A & r_2^A & r_3^A & r_4^A \\ r_1^B & r_2^B & r_3^B & r_4^B \\ r_1^C & r_2^C & r_3^C & r_4^C \end{bmatrix},$$

where an element in the matrix (r) represents a key pair value. A resource ID is a key, and a matching score is a value. r_1^A represents the first rank result from the matcher MA. When

r_1^A has an ID α, and a score for the first rank item is 1, thus, $r_1^A = (\alpha, 1)$. The matching score is supposedly assigned by each matcher. If a matcher does not provide the score value, we can compute the score as explained in 9.2.2, *get score* function.

Accordingly, $R_O(r_q)is$ $\begin{bmatrix} (\alpha,1) & (\gamma,1) & (\delta,0.63) & (\epsilon,0.5) \\ (\beta,1) & (\gamma,1) & (\alpha,0.63) & (\delta,0.5) \\ (\alpha,1) & (\delta,1) & (\beta,0.63) & (\gamma,0.5) \end{bmatrix}$

The initial weights of all matchers are equally distributed, $w[n] = \begin{bmatrix} \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix}$.

Therefore, for Ex.3, $w = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$.

Then we can compute a combined result R_C using Borda count (see Subsection 9.2.2 for more detail). Accordingly, a new weight *w'* can be measured from Euclidean difference between R_C and R_O (see Subsection 9.2.3 for more detail).

9.2.2. Get combined result function

To combine the ranking result from all matchers meaningfully, we use a Borda count technique. Figure 9.4 depicts the whole process for this function.

An item in a result array is a key-value pair which consists of a resource ID (in most cases, this can be represented as URI) of each rank as a key, and a score of that rank as a value. The score is calculated from a *getScore* function and will be accumulated if the same ID appears in other matchers' results. If a matcher returns less than *t* items, an empty key-value pair is used to fill up the missing rank. When all results from all matchers are processed, the elements in R_C will be sorted according to score values and selected only top *t* items.

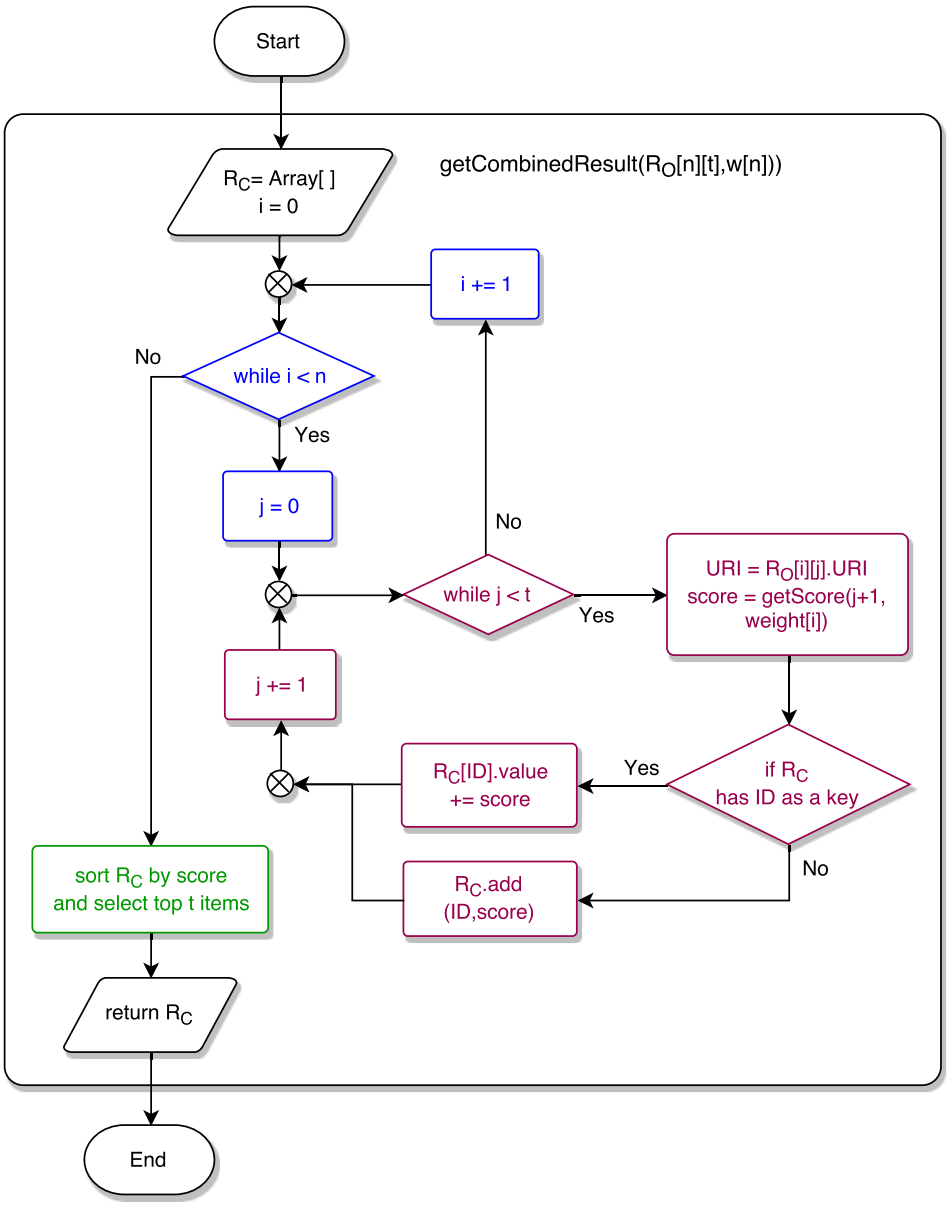


Figure 9.4.: 'getCombinedResult' function.

Get score function

As mentioned earlier, when matching scores are provided by matchers, we can normalize and use them directly. However, it is most likely that these scores are used internally and are not provided with the result. In this case, we can calculate these scores using Borda count technique. Figure 9.5 depicts the flowchart of the *getScore* function.

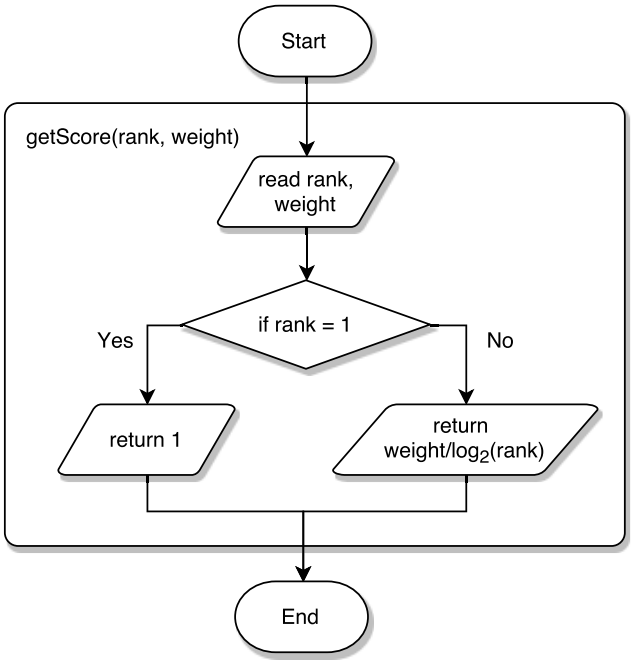


Figure 9.5.: 'getScore' function.

In Borda count, top *t* ranks from all matchmakers with the same query will be considered. Each distinctive element will be assigned a score according to the rank so that the element in the upper rank has the higher score. We use the formula according to the normalized Discounted Cumulative Gain (nDCG), i.e. $\frac{1}{\log_2 \#rank}$; or 1 where #rank is 1.

Each distinct element's score will be accumulated. For example, if a resource, α, is ranked in the first place by MA and MC, while MB ranks the resource α in the 3rd place, the total score for the resource α is $1 + \frac{1}{\log_2 3} + 1$. This score value will be multiplied by the weight value of individual matcher.

$$score = \sum_{i=1}^{n} \frac{weight[i]}{\log_2(\#rank)} \quad ; \; weight[i] \; when \; \#rank \; = \; 1 \tag{9.1}$$

Next, the total score is sorted in descending order. Then the top *t* ranks will be added to the combined result.

From Ex.3;

$R_{MA}(r_q) = \begin{bmatrix} \alpha & \gamma & \delta & \epsilon \end{bmatrix}$,

$R_{MB}(r_q) = \begin{bmatrix}\beta & \gamma & \alpha & \delta\end{bmatrix}$,

$R_{MC}(r_q) = \begin{bmatrix}\alpha & \delta & \beta & \gamma\end{bmatrix}$

and $w = \begin{bmatrix}\frac{1}{3} & \frac{1}{3} & \frac{1}{3}\end{bmatrix}$.

Every resource gets a score as follows:

- $\alpha\ score = \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot \frac{1}{log_2 3} + \frac{1}{3} \cdot 1 = 0.88$
- $\beta\ score = \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot \frac{1}{log_2 3} = 0.54$
- $\gamma\ score = \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot 1 + \frac{1}{3} \cdot \frac{1}{log_2 4} = 0.83$
- $\delta\ score = \frac{1}{3} \cdot \frac{1}{log_2 3} + \frac{1}{3} \cdot \frac{1}{log_2 4} + \frac{1}{3} \cdot 1 = 0.71$
- $\epsilon\ score = \frac{1}{3} \cdot \frac{1}{log_2 4} + \frac{1}{3} \cdot 0 + \frac{1}{3} \cdot 0 = 0.17$

(Ex. 4)

Therefore, the raw result is

$\begin{bmatrix}(\alpha, 0.88) & (\gamma, 0.83) & (\delta, 0.71) & (\epsilon, 0.17) & (\beta, 0.54)\end{bmatrix}$.

Finally, after sorting the score and choosing only top four ranks, we get

$R_C(r_q) = \begin{bmatrix}(\alpha, 0.88) & (\gamma, 0.83) & (\delta, 0.71) & (\beta, 0.54)\end{bmatrix}$.

9.2.3. Get weight function

The weight value of each resource matcher can indicate the accuracy of its result. We assume that the majority result is the most accurate result. Thus, we compare the result of each matcher with the merged result to judge if a matcher returns a good or bad result.

Figure 9.6 illustrates the workflow to get the weight value of each resource matcher. Without prior knowledge, the distance between the merged result and the original result are compared using *Euclidean distance* measurement. The bigger the difference, the lesser value the weight will become. In other words, if the merged result is closer or similar to the result from one matcher, it indicates that this matcher is more reliable than the other. Consequently, the weight of this matcher should be increased.

This weight measurement is necessary, because when we utilize multiple matchers simultaneously, some matchers may yield a very unlikely result. The low-relevance result should

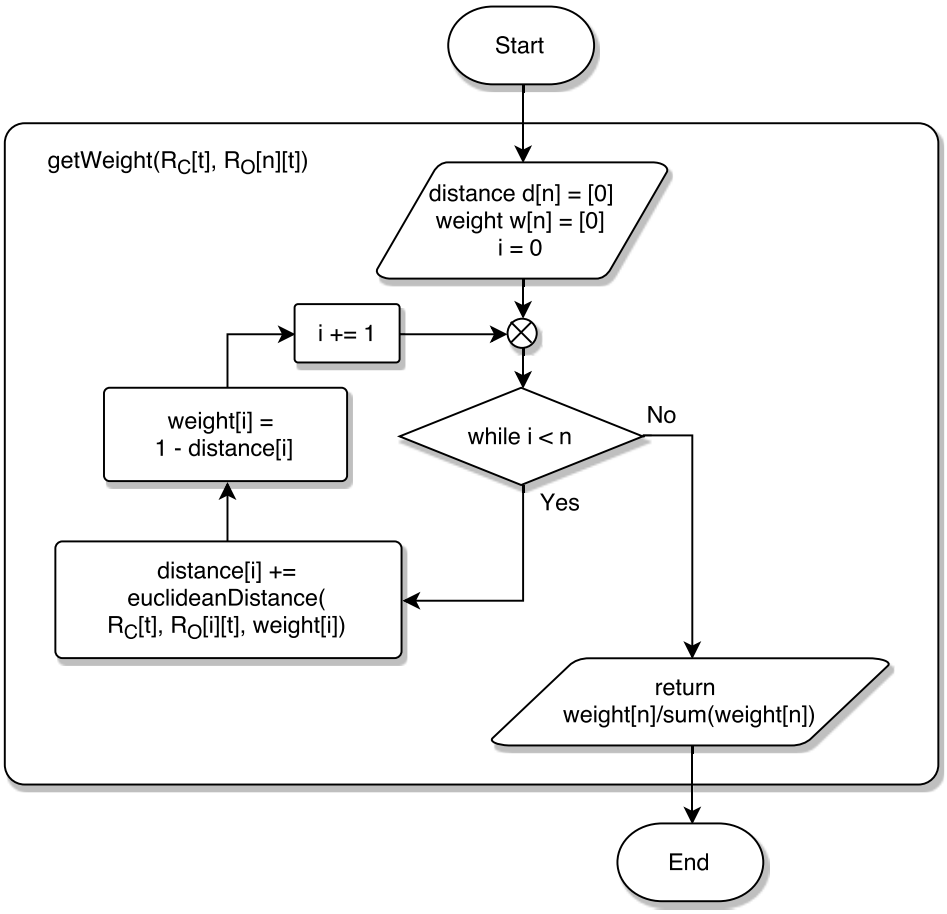


Figure 9.6.: 'getWeight' function.

be detected and removed from the combined result. A matcher is identified as a low-accuracy matcher when it receives a lower weight. The result from this matcher is then discarded or deprioritized, and the final result is recalculated.

According to Equation 9.1, weight values for all matchers will be used in the calculation of scores, resulting in the next round of R_C calculation. Given the distance, E_d, between each matcher's result and R_C, the weight is:

$$weight[M_n] = 1 - E_d[n] \tag{9.2}$$

Euclidean distance function

We can compare the original result from each matcher $R_{Mn}$to R_C and measure the distance. The Euclidean distance E_d can reflect the reliability of each matcher. First, every URI item from a single query in R_C will be searched in every R_M. If the ranks of a particular item differ from R_C to R_M, the distance will be increased. Figure 9.7 depicts a

flowchart of the measurement of the Euclidean distance between two arrays, R_C to R_O.

Note that $R_O[n] = \begin{bmatrix} R_{M1} \\ R_{M2} \\ \dots \\ R_{Mn} \end{bmatrix}$.

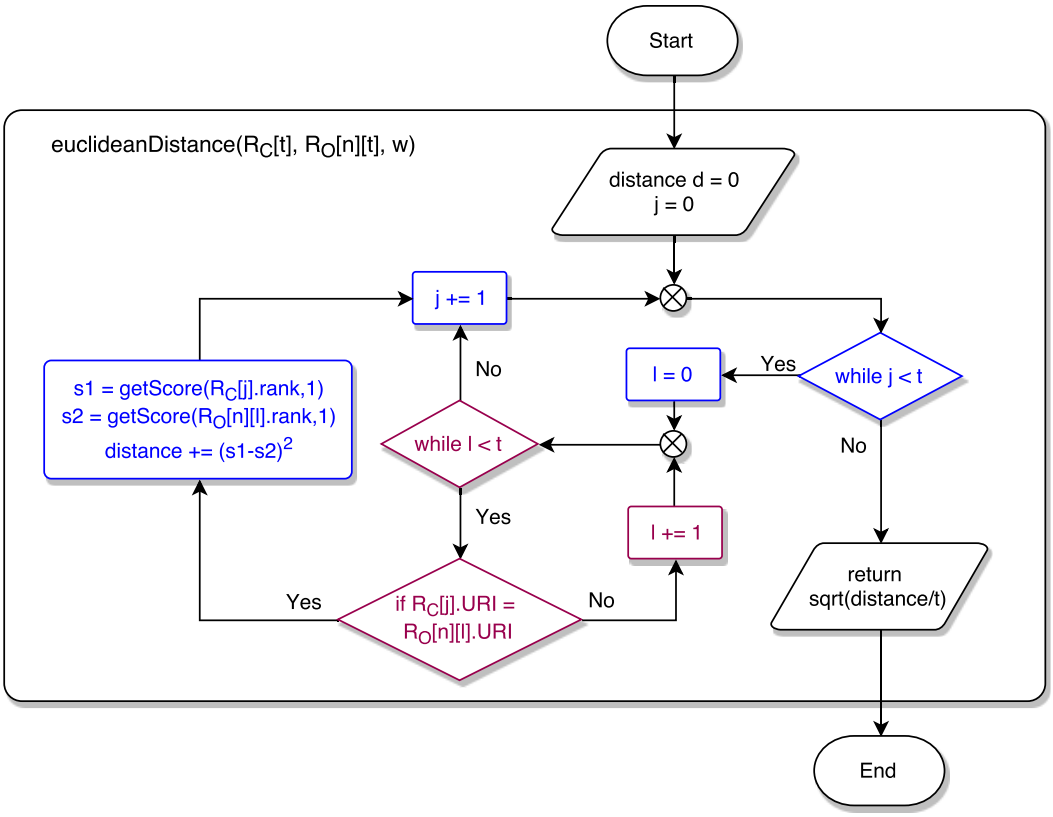


Figure 9.7.: 'euclideanDistance' function to measure the difference between the merged result and the original results.

Considering Ex.4: $R_C(r_q) = \begin{bmatrix} \alpha & \gamma & \delta & \beta \end{bmatrix}$

$R_{MA}(r_q) = \begin{bmatrix} \alpha & \gamma & \delta & \epsilon \end{bmatrix}$

Applying Equation 9.1 and use the initial weight ($\frac{1}{3}$) to compute R_C, a score of α from R_C (denoted as s1) is $\frac{1}{3}$, and a score of α from R_{MA} (denoted as s2) is $\frac{1}{3}$. Thus, the distance is:

$$d(x) = (s_1 - s_2)^2;\ s2 = 0\ when\ the\ item\ x\ does\ not\ exist\ in\ the\ result\ of\ that\ matcher. \tag{9.3}$$

Applying this similar method to all items,

$$
\begin{aligned}
&d(\alpha) = (\tfrac{1}{3} \cdot 1 - \tfrac{1}{3} \cdot 1)^2 = 0, \\
&d(\beta) = (\tfrac{1}{3} \cdot \tfrac{1}{log_2 4} - \tfrac{1}{3} \cdot 0)^2 = 0.0278, \\
&d(\gamma) = (\tfrac{1}{3} \cdot \tfrac{1}{log_2 2} - \tfrac{1}{3} \cdot \tfrac{1}{log_2 2})^2 = 0, \textit{ and} \\
&d(\delta) = (\tfrac{1}{3} \cdot \tfrac{1}{log_2 3} - \tfrac{1}{3} \cdot \tfrac{1}{log_2 3})^2 = 0.
\end{aligned}
\qquad \text{(Ex. 5)}
$$

Note that we use the initial weight when we compute weights for the first time. According to the flow in Figure 9.2, the weights are updated in every iteration and used to calculate the distance too.

Putting these results together, the distance between R_C and R_{MA} is:

$$E_d(R_C, R_{MA}) = \sqrt{\frac{\sum_{i=1}^{t} d_i}{t}} \qquad (9.4)$$

Finally, all weights will be normalized by dividing by the sum of weights.

$$normalized\ weights[M_x] = \frac{weight[M_x]}{\sum_{i=1}^{n} weight[i]};\ x\ is\ the\ number\ of\ matcher. \qquad (9.5)$$

9.2.4. Check weight change function

From here we can re-compute R_C and the matcher weight again until they are insignificantly changed. The result from the best quality round will be chosen. Instead of comparing the final result with the predefined solution, we can assume that when the weight difference between two rounds converges at 0, it indicates the stability of the weight computing and the result of this round supposedly yields the best quality.

The *isWeightStable* function, shown in Figure 9.8, compares the difference between a previous weight and a new weight from a new R_C. Not all weight will be compared, but only the maximum weight. The threshold value is configurable. The higher threshold can be set, but this would decrease the quality of the final R_C, since the iteration could stop before the best R_C is reached.

However, if poor performance matchers are not muted out when the iteration continues, the quality of the final result would not be much improved. The solution is to remove the least-weighted matcher from the getCombinedresult calculation. Finally, the criteria for the removal of matchers must be set, so that there will be some matchers left to be computed.

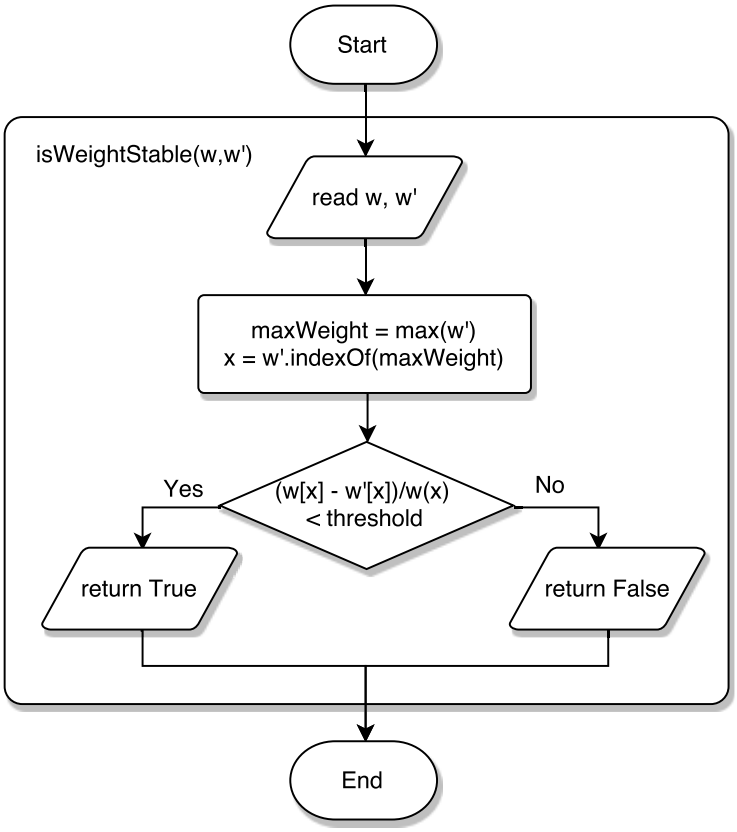


Figure 9.8.: 'isWeightStable' function to check the condition for stopping the iteration.

To stop the elimination of the poor performance matcher, we use statistic figures as a threshold value. Only the matcher that has a weight lower than;

$$threshold = mean(weights) \times \kappa \times standard\ variation(weights), \tag{9.6}$$

and the lowest weight of all will be removed. The value κ can be adjusted according to the granularity of matcher quality. The higher this value is, the more poor performance matchers will be accounted to the final result.

When the request constructor, request converter, resource matchers and result integrator are ready and put together as shown in Figure 5.1, the integration of the resource discovery is complete. In the following chapter, we show how users can use the resource discovery module via MERCURY.

10

Resource Discovery Integration to MERCURY

In previous chapters, we explained the implemented modules for the service discovery. Now we can deploy the resource discovery into MERCURY. As introduced in Chapter 2, *Project Background*, we can utilize the discovery function in the registration, scenario modeling, and execution processes. The core functions of the service discovery are implemented independently from MERCURY to fulfill Requirement R11 (*the resource discovery should operate as a standalone module*). However, the presentation of discovery result, as required by Requirement R12 (*the discovery result should be presented in the registration, scenario modeling, and execution processes in a way that users can apply the result instantly*) is specific to the context of MERCURY. The evaluation of this integrated solution will be presented in Chapter 13, *Integrated System Evaluation Results*.

10.1 Registration

There are many approaches to register a resource into MERCURY, such as retrieving local devices seen on a user's local machine by calling a script which detects all hardware drivers. A remote resource can also be registered directly by providing a URL to the resource description. Moreover, the user can simply search for a resource by free-text keywords. This work provides two different ways of discovery: a quick search and an advanced search.

- **Quick search** - The keywords provided by a user are taken as general keywords which can appear in any field of description. Meanwhile, a resource matcher is the default matcher derived from a configuration file.
- **Advanced search** - Keywords are categorized into specific fields, according to the necessary information deduced from section 7.2, *Essential information required for resource matching*. Additionally, resource matcher(s) can be chosen via a GUI.

The registration can be initiated with a resource discovery, according to the flowchart in Figure 10.1. If simple keywords are provided, the quick search function will be called.

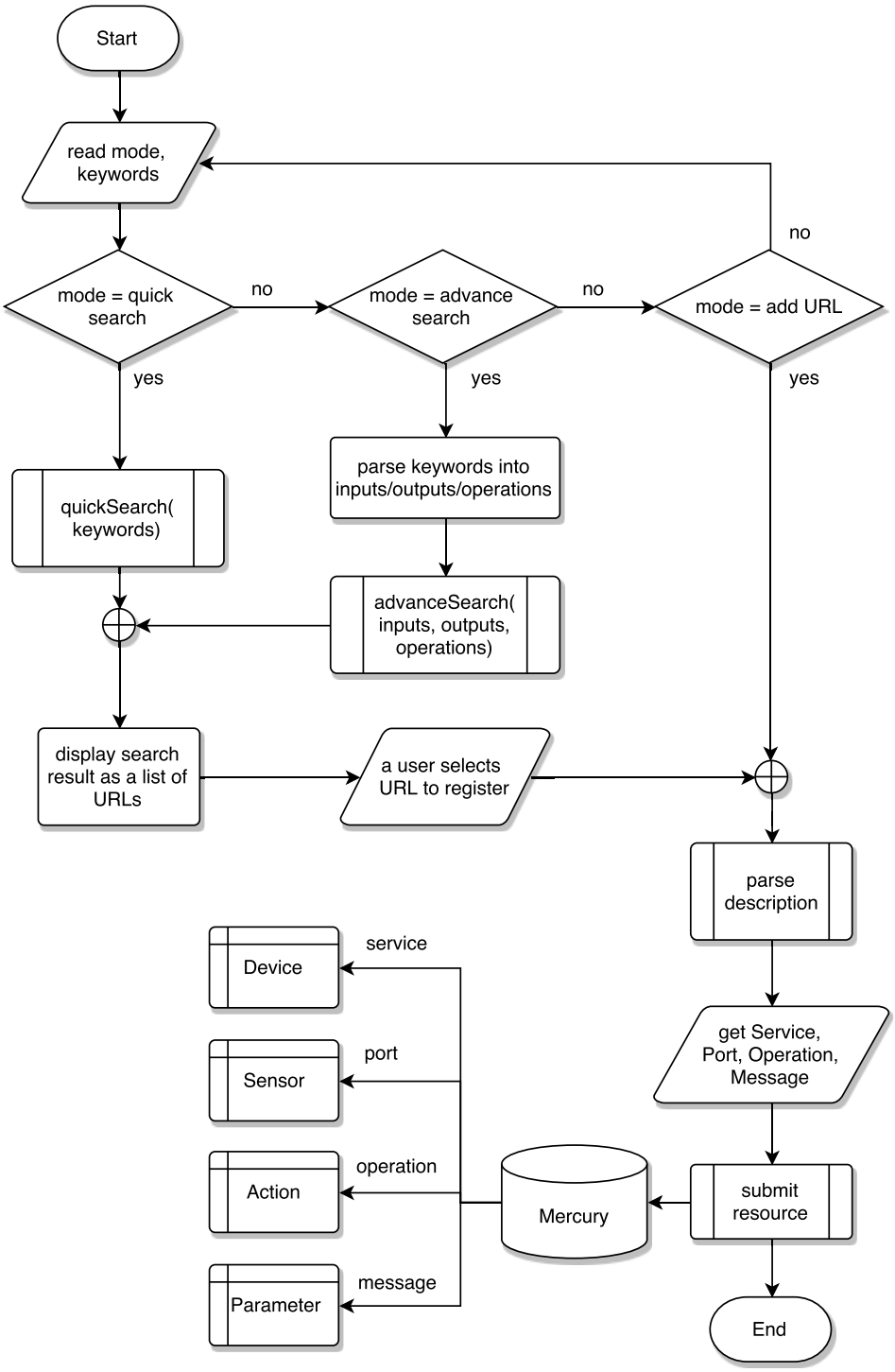


Figure 10.1.: Registration flow with the resource discovery.

On the other hand, the categorized keywords can be submitted to the advanced search function, together with the list of resource matchers. Then a user will be prompted with a list of relevant resources. When the user selects a resource, the resource's description will be parsed, and the resource's details will be stored in MERCURY's database. This allows us to achieve Requirement R8 (*the resources' descriptions should be derived, stored, and made editable by authorized users*).

Table 10.1 shows the necessary information from a SAWSDL description file required by MERCURY. A service is stored as a device, a port as a sensor (a device can have multiple sensors), an operation as an action, whereas input/output messages are stored as parameters.

Description	**Table.field**	**Remarks**
service.name	device.name	
service.description	device.description	
service.url	device.url	
	device.id (1)	auto-generated
port.name	sensor.sensor.name	
	sensor.id (2)	auto-generated
operation.name	action.name	
operation.documentation	action.annotation	
device id	action.deviceId	from (1)
sensor id	action.sensorId	from (2)
operation.inputmsg.name	parameters.name	
operation.inputmsg.type	parameters.type	
operation.outputmsg.name	parameters.name	
operation.outputmsg.type	parameters.type	
operation.msg.modelreference	parameters.annotation	
	parameters.id (3)	auto-generated
input parameters id	action.inputParameterId	from (3)
output parameters id	action.outputParameterId	from (3)

Table 10.1.: Mapping table of elements from a SAWSDL description file versus tables and fields in MERCURY's database.

In the registration process, a user can turn the implicit context feature on/off or provide the context explicitly. Additionally, both implicit and explicit context, including the original keywords, are extended with semantic annotations. This increases the chances of finding more relevant resources.

10.1.1. Applying context to the Registration process

During the registration process, MERCURY can retrieve a user profile and extract a user's location as described in Section 6.1, *Context from User Profile*. The location name (e.g. city name, country name) will be added to a query message. This query expansion can improve the resource recommendation.

For a quick search, the user's location will be simply used to compare with every field of a description, whereas in an advanced search, the location will be appended to the operation name. For example, a user looks for an accommodation service, and the user address in his profile is Germany. The recommended resources should offer a hotel service in Germany in a higher rank than a hotel service in France.

Not only can a static address from the user profile be utilized, but also the current location derived from a social sensor (as described in Section 6.2). Furthermore, an online calendar, for example, can be analyzed to predict users' interest. For instance, if a user plans on visiting the USA next week, the recommendation should offer a hotel service in the USA in a higher rank than a hotel service in Germany.

10.2 Scenario Modeling

Regarding Requirement R9 (*the modeling tool should recommend the potential resources to users by considering users' interactions*), during the modeling or execution process, the recommendation has to support real-time interaction. Unlike the registration process, which the completion and accuracy of the result are highly prioritized, the resource recommendation in the scenario modeling searches through the registered resources which must be done in real time. This recommendation also considers the semantic annotation and user's preferences to get the set of recommended resources.

Applying context to the Scenario Modeling process

Since the resources in this process are all registered and have a history of resource usage in MERCURY, we offer four recommendation modes according to Section 6.3, *Context from User Preferences and Contributions*:

- (F) - Frequently used/ Favorite by rating.
- (S) - Semantically similar description.
- (C) - Input/output compatible.
- (U) - Previously used in similar scenarios.

The discovery process starts when a user drops an item onto the modeling canvas, then the detail of the item will be analyzed and used for the recommendation. As shown in Figure 10.2, the recommendation in mode (S) and (C) will call a semantic API to get relevant terms, and these terms will be used for extending a query for relevant resources. In the meantime, the recommendation in mode (F) and (U) require a history of resources' usage for a recommendation. The recommendations from all modes will be accumulated and presented to the user with explanations why they are being suggested.

(F) and (U) recommendations are categorized as object-oriented based context. Meanwhile, ontology knowledge is required to implement (S) and (C). Thus, they are considered as ontology-based context.

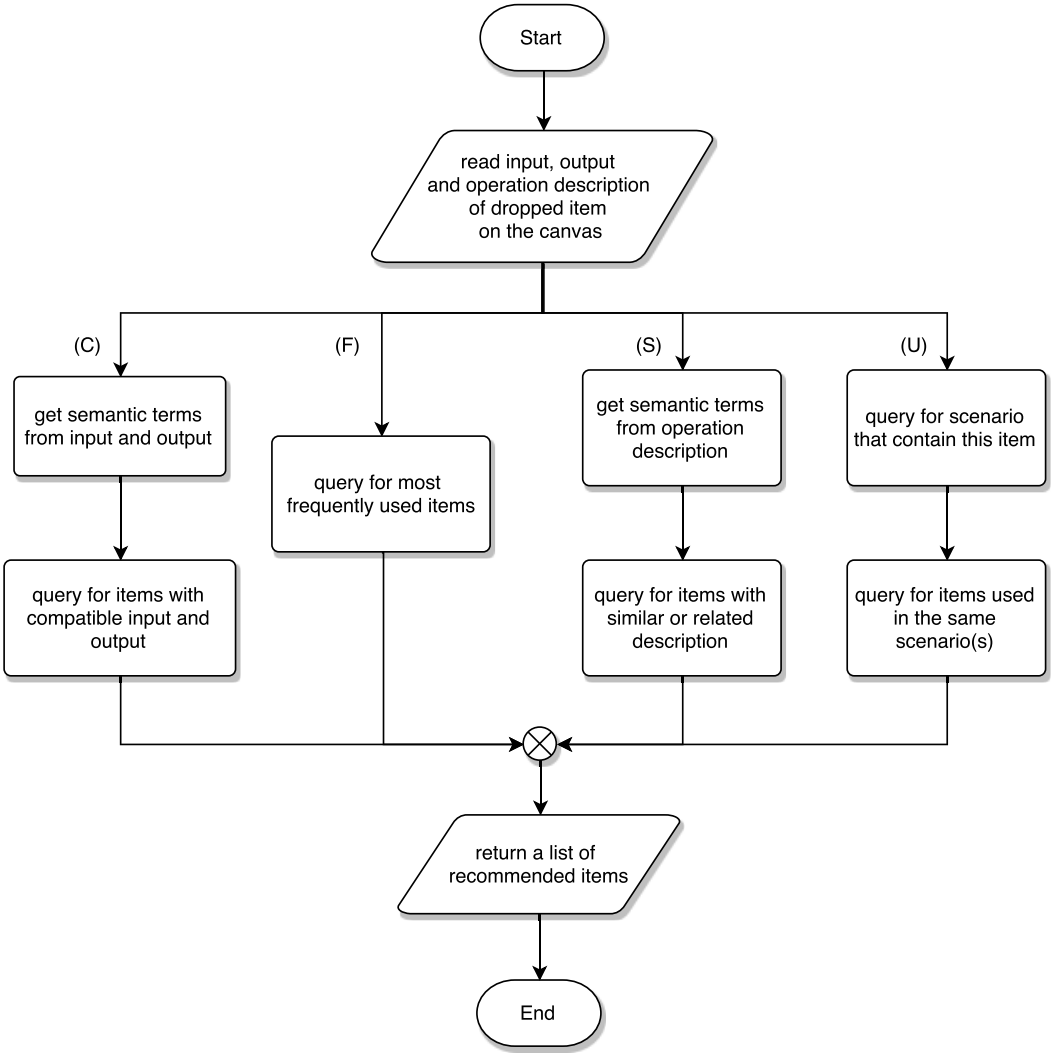


Figure 10.2.: Resource discovery flow in the scenario modeling process.

The user context information from Portal user management is also applied in these processes. However, in the scenario modeling and execution processes, there is no GUI to provide context information explicitly. Therefore, only the implicit context information can be utilized.

10.3 Execution Engine

The created scenario is translated into an executable script via a scenario translator, as depicted in Figure 10.3. A graphical representation of a scenario will be validated and converted into a script. An execution engine then executes the script and control the sensors/actuators/services via middleware.

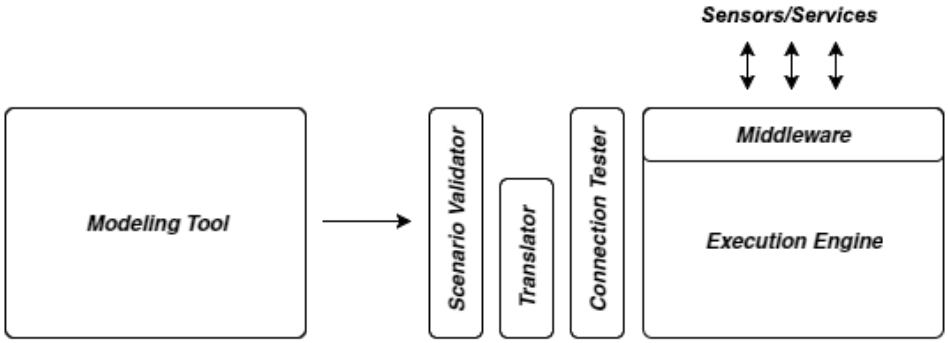


Figure 10.3.: Connection between the scenario modeling tool and the execution engine.

During the runtime, if any resource in the scenario fails to respond, then an error handler should offer a solution to a user. From Figure 10.4, the execution engine can check for a vital signal of each resource by polling them periodically. If a resource does not respond to the polling request within a period, an error handler will be called up.

The failed resource's properties will be retrieved from a database, namely, the input(s), output(s) and operation descriptions. This information will be passed to the resource discovery, thus resolving Requirement R10 (*during runtime, a placeholder item or a fail-to-respond resource should be supported by the service discovery*).

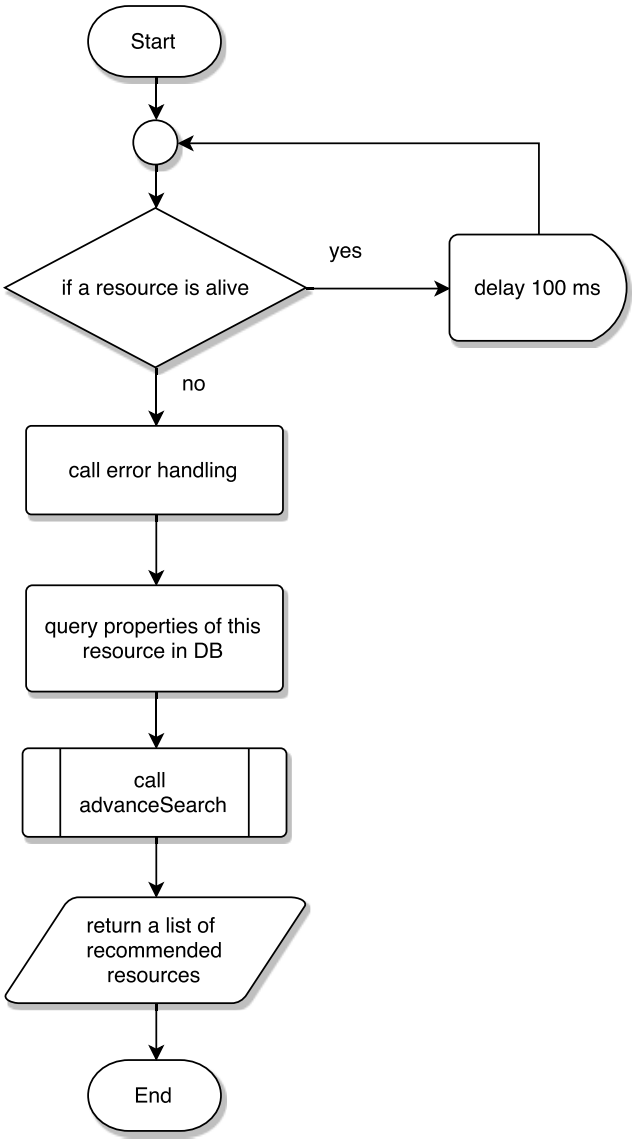


Figure 10.4.: Resource discovery flow in the scenario execution process.

Applying context to the Execution process

Similar to the Registration process, the usage of context in the Execution process is based on the user profile (Section 6.1) and a social sensor (Section 6.2). However, in this process, we emphasize on the compatibility of the input and output over the operation description. This is because the replacement of a resource should be instantly made without a redesigning of a scenario.

In this part, we elaborate on the implementation details of the resource discovery and how to use it in the context of MERCURY. Next, we evaluate each module separately to show the importance of each element. Then, we evaluate the integrated resource discovery as a whole.

Part III.

Evaluation

11 Evaluation Overview

In the preceding part, we provided the fundamental components we need to realize our goal and the implementation details of each element. To prove that our proposed solution can improve the resource discovery, we evaluate each component in Chapter 12, *Unit Test Results*. This also indicates the part which we can improve upon in future works. We then evaluate the integrated solution to observe the overall performance in Chapter 13, *Integrated System Evaluation Results*.

In this chapter, we describe the measurement methods for qualifying the result from resource discovery in Section 11.1. The method and objective of the evaluation for the request constructor, the request converter, and the result integrator are explained in Section 11.2, 11.3, and 11.4 respectively. Then, we describe the testing corpus we use for our evaluation in Section 11.5. A list of resource matchers used in this evaluation is concluded in Section 11.6.

11.1 Performance measurement

There are two types of relevances that are considered in this evaluation; binary relevance and graded relevance. For binary relevance, the result will be classified into two categories, relevant (1) or irrelevant (0). On the other hand, the graded relevance offers a finer granularity of relevance, e.g. strongly relevant (3), relevant (2), fairly relevant (1) and irrelevant (0). The binary quality measurement is calculated by precision, recall, and F-measure rates. Meanwhile, to measure the quality of rank, which requires graded relevance, a normalized Discounted Cumulative Gain (nDCG) is used.

Figure 11.1 illustrates term definitions in a search evaluation. The testing corpus comprises of relevant and irrelevant documents according to the request. Out of these corpora, all documents that are supposed to be relevant are returned to a user. Within this retrieved documents set, a group of relevant documents is defined as "True positive" (tp), while a group of irrelevant documents is "False positive" (fp). The non-retrieved documents but truly relevant are "False negative" (fn), whereas the irrelevant documents from this group are called "True negative" (tn).

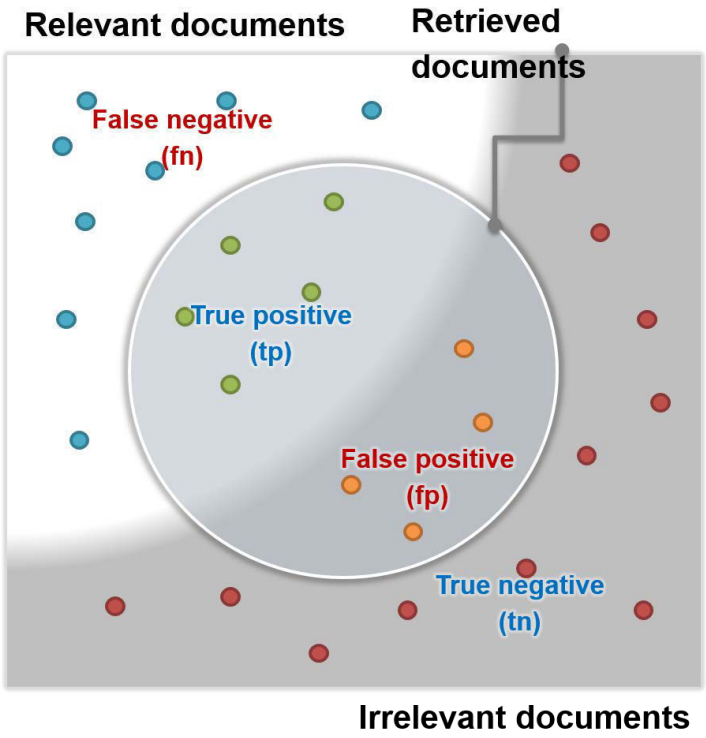


Figure 11.1.: Definition of returned relevant and irrelevant results.

In order to compare the quality of resource matchmaking results, the following measurements are used:

$$precision = \frac{|\{relevantdocuments\} \cap \{retrieveddocuments\}|}{|\{retrieveddocuments\}|} = \frac{tp}{(tp + fp)} \tag{11.1}$$

$$recall = \frac{|\{relevantdocuments\} \cap \{retrieveddocuments\}|}{|\{relevantdocuments\}|} = \frac{tp}{(tp + fn)} \tag{11.2}$$

When we consider top ten results, the recall rate can be small if the number of relevant documents is greater than ten. Therefore, we divide the original recall rate with the maximum recall number to normalize the value. For example, if there are 29 relevant documents and 9 of them are returned in a top ten list of results, the normalized recall rate will be 9/10 instead of 9/29. On the other hand, when there are five relevant documents, and two of them appear in a top ten list, the normalized recall will be 2/5 like the originally defined recall rate.

In fact, a high recall rate can be achieved by simply consider more results. In contrast, a precision rate is more likely to decrease by doing so. Thus, we measure a harmonic mean between precision and recall rates using F-measure:

$$F\text{-}measure = 2 \bullet \frac{(precision \bullet recall)}{(precision + recall)} \tag{11.3}$$

If the normalized recall explained previously is used to calculate the F-measure, we denote this term as " normalized F-measure".

The fallout rate is also common in evaluating search result.

$$fallout = \frac{fp}{(fp + tn)} \tag{11.4}$$

However, we are looking at the top few number of results. This causes tn to outnumber fp. Hence, the fallout rate bears no significance and could be negligible.

Then, we need to measure how well the matchers rank their results by using DCG.

$$DCG_P = rel_1 + \sum_{i=2}^{P} \frac{rel_i}{log_2 i};\ rel_i\ =\ the\ level\ of\ relevance\ of\ the\ item\ in\ rank\ i. \tag{11.5}$$

P is the count of observing top rank (for example, *P* is 20 when we consider top 20 items). *rel*, the level of relevance, can have the following values; 3 means strongly relevant, 2 is relevant, 1 is fairly relevant, and 0 is irrelevant.

Nevertheless, DCG alone is insufficient to compare between different queries (thus, different solutions). For this reason, we calculate nDCG (normalized DCG).

$$nDCG_P = \frac{DCG_P}{IDCG_P};\ IDCG\ =\ ideal\ DCG. \tag{11.6}$$

IDCG can be calculated from DCG where all the items are ranked correctly.

11.2 Request Constructor

Free text keywords are given as an input, and then the request constructor has to find resembling or create formatted descriptions. However, this process cannot benefit from the semantic annotation within the description yet. This step is only a preparation to construct a request message in certain formalisms, e.g. SAWSDL and OWL-S.

In this evaluation, the simple search keywords and advanced (structured) search keywords are compared to see if the specified field of search can improve the search results. Also, semantic entities are appended to a query and compared with the query without semantic entity extension. The evaluation results are shown in Section 12.1, *Evaluation of Request Constructor*.

11.3 Request Converter

Randomly selected SAWSDL and OWL-S descriptions for similar services are used as requests. The SAWSDL descriptions are converted into OWL-S formalism, and vice versa.

Afterwards, the results from the converted SAWSDL are compared with the results from the original description in SAWSDL to see how well the conversion convey the information. The same measurement is applied to OWL-S descriptions, and the evaluation results are shown in Section 12.2, *Evaluation of Request Converter*.

11.4 Result Integrator

We do not need to evaluate the service matcher part, since this was successfully done by efforts like Web Service Challenge, Semantic Web Challenge, and Semantic Service Selection (S3) Contest. Nevertheless, outcomes from the service matchers are merged by a result integrator module. Therefore, we focus on evaluating the result integrator concerning result quality compared to an individual result of a single resource matcher. The evaluation results are shown in Section 12.3, *Evaluation of Result Integrator*.

11.5 Description Collections

This thesis adopts service descriptions from S3 Contest since it provides pre-defined solution sets which are reliable and used to qualify many resource matchers. It also provides a sample set of service descriptions equally in SAWSDL and OWL-S formalisms which are widely in use. The total number of descriptions (for each formalism) is 1080. The S3 contest provides 42 service requests together with the ideal solution for each matching

task. These predefined solutions are also provided with both binary and graded evaluating results.

In Section B.2, we show the expected results for each request we use in the evaluation. For example, when we use a resource name "shoppingmall_cameraprice" as a request to search for similar resources using a matcher name "iSeM approx. logic-based", the top ten relevant results are:

- <rank id=1>pricecamera_Wallmart
- <rank id=2>cameraprice_MyShop
- <rank id=3>camerataxedprice
- <rank id=4>shoppingmall_cameraprice
- <rank id=5>SRcamera
- <rank id=6>grocerystore_teaprice
- <rank id=7>Toyotaprice_service
- <rank id=8>price_CannonCamera
- <rank id=9>KodakDigCamera_price
- <rank id=10>searchRawAddress

We can compare the above results with the expected solution listed in B.2.1.1. The resource name "shoppingmall_cameraprice" (which is identical to the request) is expected to be shown on top of the rank because it is highly relevant. Meanwhile, the results in rank 1, 2, and 5 are expected to appear after the highly relevant results. The result in rank 8, which is listed in the potentially relevant results, should appear after the highly relevant and relevant results respectively. Ideally, the other results which are not listed in the expected results should not appear in the matching result at all.

When applying the measurement explained in Section 11.1 to this example, the precision rate is 0.5 (only five results out of the top 10 are relevant). For the recall rate, if there are 25 actual relevant documents, the recall rate is 5/25 regardless of the number of top ranks considered. However, we are considering top 10 results, the normalized recall rate is $\frac{5/25}{10/25}$, where 10/25 is the ideal recall rate from top 10 results.

Mostly, the recall and precision rates do not differ much between each matcher when we consider the results at top 10 or 20 ranks. However, the ranking algorithms of each matcher make the matching quality significantly different. From the previous example, although the highly relevant results are listed in top ten, it appears in the fourth rank instead of the first rank. Even worse, the irrelevant results like ranks 6 and 7 are listed in a higher rank than the potentially relevant results. This dilemma can be resolved when we use multiple matchers.

11.6 Resource Matchers

As mentioned in Section 7.1, we studied semantic resource matchers from the S3 Contest. The matchers that are working, as listed in Table 11.1, are used in the evaluation.

Formalism	Matchers	
OWL-S	iSeM approx. logic	[KK10]
	iSeM logic	[KK10]
	iSeM structure	[KK10]
	iSeM text (Cos)	[KK10]
	iSeM text (Cos, structured)	[KK10]
	SeMa2	[MHB+12]
	OWLS-M0	[KFK05]
	OWLS-MX2	[KKF08]
	OWLS-MX3 (M3)	[KK12a]
	OWLS-MX3 (Structure)	[KK12a]
SAWSDL	iSeM approx. logic-based	[KK12b]
	iSeM logic-based	[KK12b]
	iSeM text similarity (Cos)	[KK12b]
	iSeM text similarity (Cos, structured)	[KK12b]
	iSeM SVM aggregation	[KK12b]
	SAWSDL-M0	[KKZ09a]
	SAWSDL-MX	[KKZ09a]
	SAWSDL-MX2	[KKZ09b]
	XAM4SWS-LOG4	[SLES10]
	XAM4SWS-COV	[SLKS12]

Table 11.1.: List of resource matchers and algorithms from S3 Contest used for the evaluation.

Finally, all the components are assembled and evaluated together to check the overall performance in Chapter 13, *Integrated System Evaluation Results*.

12 Unit Test Results

In this chapter, we provide the evaluation results of the individual building blocks in our solution. In Section 12.1 we evaluate the request constructor and demonstrate that constructed requests are not only compatible with resource matchers, but they also contain sufficient data for a matching process.

Next, we evaluate the request converter in Section 12.2. The converted descriptions are fed to resource matchers, using an actual description in the destination formalism as the baseline. Our goal is to translate a description from one formalism (e.g. SAWSDL) to another formalism (e.g. OWL-S) without degrading the matching result's quality.

We use the resource matchers listed in Table 11.1 in these evaluations. The returned results from all matchers are merged using the result integrator. Then, we evaluate the result integrator in Section 12.3 comparing to an individual resource matcher.

12.1 Evaluation of Request Constructor

As presented in Section 8.1, the request constructor samples similar existing descriptions or formulates a minimal request from free text keywords. First, we evaluate the simple search which compares all keywords in description files regardless of the position of words. If the sampling method cannot return any relevant description, a minimal request will be created. The settings and results of the basic search evaluation are described in 12.1.1.

In an advanced search, the user can specify the functionality of each keyword, i.e. operation, input or output. We show in 12.1.2 how the separation of the simple keywords into three categories can help us to gain more accurate results.

Additionally, we expand the query messages with semantic terms to retrieve more relevant results. In 12.1.3, we evaluate how well the semantic terms can improve the query results.

We compute the recall, precision, F-measure, and nDCG values at top ten ranks from the returned result. Note that the recall rate can be higher when we consider more ranks, such as top 20 or top 50 results. In fact, the precision rate usually increases inversely to the recall rate. In this module, we are interested only if the constructed request can fulfill the requirements of resource matchers, compared to the existing formatted request. Thus, only the top ten results of the evaluation will be observed.

12.1.1. Simple Search

To measure the quality of the constructed request, we imitate 30 descriptions (chosen randomly) from the test collections out of the 42 descriptions with pre-defined solutions. The simple plain text keywords we used to construct each request message are listed in Table 12.1, column "Simple Keywords".

Table 12.1.: List of keywords used for evaluating the request constructor with the corresponding descriptions.

Corresponding description	ID	Simple Keywords	Structured Keywords		
			Input	Output	Operation
camera price	d1.1	camera price	camera	price	camera price
	d1.2	shoppingmall camera	shoppingmall camera	-	shoppingmall camera
	d1.3	shoppingmall camera price	shoppingmall camera	price	shoppingmall camera
book price	d2.1	book price	book	price	-
	d2.2	novel price	novel	price	-
	d2.3	get novel price	novel	price	get novel price
open door	d3.1	door	door	-	get door
	d3.2	open door	-	door	get open door
citycountry hotel	d4.1	hotel	-	hotel	get hotel
	d4.2	city hotel	city	hotel	get hotel
	d4.3	city country hotel	country city	hotel	get hotel
	d4.4	region hotel	region	hotel	-
	d4.5	accomodation city	city	accommodation	-
fall down pill	d5.1	pill	-	pill	-
	d5.2	fall down pill	-	pill	fall down pill
Grocery Store Food Service	d6.1	food	-	food	get food
	d6.2	grocerystore food	grocerystore	food	get food
	d6.3	shop food	shop	food	get food
	d6.4	grocerystore	grocerystore	food	-
Researcher address	d7.1	researcher address	researcher	address	get address
	d7.2	employee information address	employee	information	find address
	d7.3	scientist university address	scientist university	address	address
hospital investigating	d8.1	hospital	hospital	-	investigating
	d8.2	hospital investigating	hospital	investigating	get investigating
	d8.3	hospital diagnose	hospital	diagnose	-
	d8.4	medical checkup	medical	checkup	-
	d8.5	medical biopsy	medical	biopsy	biopsy
comedy film	d9.1	film title	title	film	-
	d9.2	comedy film		comedy	get comedy film
	d9.3	media title	title	media	
	d9.4	comedy movie	-	comedy movie	get movie
surfing destination	d10.1	surf beach	surfing	beach	
	d10.2	sufing destination	surfing	destination	get destination
	d10.3	surfing location	surfing	location	get location
maxprice cola	d11.1	price cola	cola	-	get price
	d11.2	max price cola	cola	max price	-
get Altitude Above Sea Level Of Location	d12.1	long lat altitude	lat long	altitude	get altitude
	d12.2	lat long height	lat long	height	
	d12.3	elevation coordinates	coordinates	-	get elevation

Continued on next page

Table 12.1 – continued from previous page

Corresponding description	ID	Simple Keywords	Structured Keywords		
			Input	Output	Operation
	d12.4	altitude of location	location	-	get altitude
mile To Kilometer Converter	d13.1	convert mile to km	mile	km	convert
	d13.2	length converter	length unit	length unit	converter
	d13.3	convert mi to km	mi	km	convert
publication-number publication	d14.1	get publication	-	-	get publication
	d14.2	get publication by number	publication number	publication	get publication by number
	d14.3	publishing number	number	publishing	-
	d14.4	publication number	number	publication	-
get Sunset Sunrise Time Of Location	d15.1	sunset sunrise time	-	sunset sunrise time	get time
	d15.2	sunset sunrise time location	location	sunset sunrise time	get time
	d15.3	day night location	location	day night	-
	d15.4	coordinates dusk dawn times	coordinates	dusk dawn	get times
dvdplayer mp3player price	d16.1	dvdplayer price	dvdplayer	price	get price
	d16.2	dvdplayer mp3player price	dvd mp3 player	price	-
	d16.3	dvd mp3player price	dvd mp3 player	price	get price
get Distance Between Cities Worldwide	d17.1	distance between cities	cities	distance	get distance between cities
	d17.2	get distance cities	cities	distance	get distance
	d17.3	location distance	location	-	location distance
geographical-region map	d18.1	geographical map	-	map	get geographic map
	d18.2	geographical region map	geographical region	map	-
car price	d19.1	get car price	car	price	get car price
	d19.2	auto price	-	price	get auto price
title videomedia	d20.1	title video media	title	video	get video
	d20.2	get video media title	title	video	get media
	d20.3	get video media by title	title	video media	-
country skilled occupation	d21.1	skilledoccupation country	country	skilled occupation	get occupation country
	d21.2	country profession	country	profession	-
Recommended Price Coffee Whiskey	d22.1	coffee whiskey price	coffee whiskey	-	get price
	d22.2	recommended coffee whiskey price	recommended coffee whiskey	-	get price
	d22.3	irishcoffee price	irishcoffee	price	-
	d22.4	price suggested coffee whiskey	coffee whiskey	price	suggested price coffee
1person bicycle car price	d23.1	bicycle price	bicycle	price	get price
	d23.2	car price	car	price	get price
	d23.3	auto price	-	price	get auto price
	d23.4	bicycle person price	bicycle person	price	-
novel author	d24.1	book author	book	-	get author
	d24.2	get novel author	novel	author	get author
	d24.3	story author	story	author	-
	d24.4	book title writer	book title	writer	-
prepared food price	d25.1	preparedfood	preparedfood	-	-
	d25.2	prepared food price	prepared food	price	get food price
	d25.3	processed food price	processed food	-	get price

Continued on next page

Table 12.1 – continued from previous page

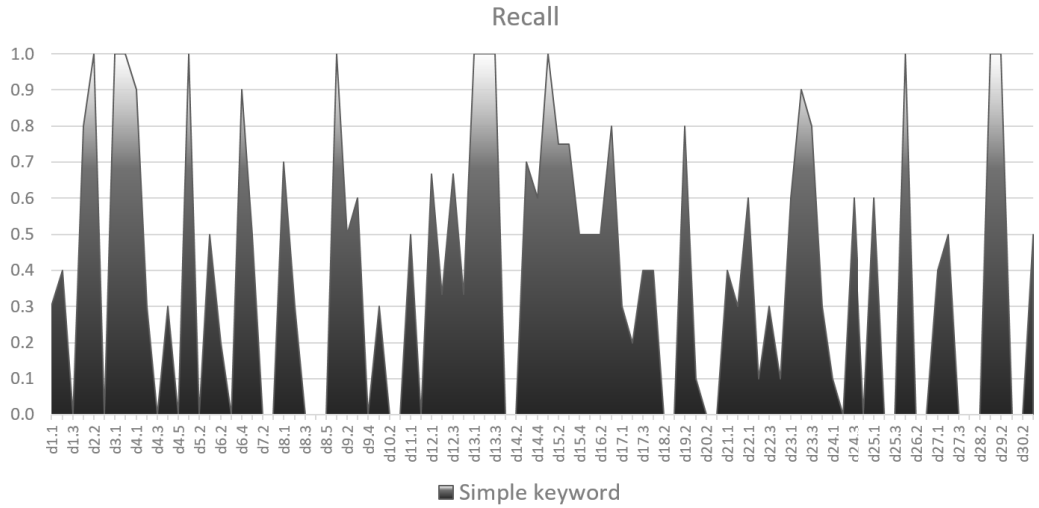
Corresponding description	ID	Simple Keywords	Structured Keywords		
			Input	Output	Operation
government degree scholarship	d26.1	scholarship	-	scholarship	-
	d26.2	find government scholarship	government degree	scholarship	get scholarship
	d26.3	government degree funding	government degree	fund	get funding
get Location Of City State	d27.1	city location	city	location	get location
	d27.2	city state location	city state	location	get location
Geopolitical-entity Weather Process	d28.1	get weather city	city	weather	get weather
	d28.2	geopolitical entity weather	geopolitical entity	weather	get weather
	d28.3	geopolitical entity weather process	geopolitical entity	weather process	get weather
lock door	d29.1	lock door	-	door	get lock door
	d29.2	close door	-	close door	get close door
Government Missile Funding	d30.1	government missile funding	missile government	funding	find funding
	d30.2	government weapon funding	government	weapon funding	-
	d30.3	weapon funding	-	weapon funding	get funding

Table 12.1.: List of keywords used for evaluating the request constructor with the corresponding descriptions.

The constructed requests created from these keywords are compared with the corresponding description (see solutions in B.2.1). For instance, when we look for a service "book price" (denoted as d2), keywords like, "book price" (denoted as d2.1), "novel price" (d2.2), or "get novel price" (d2.3) can be used to construct a minimal request that is then fed to resource matchers.

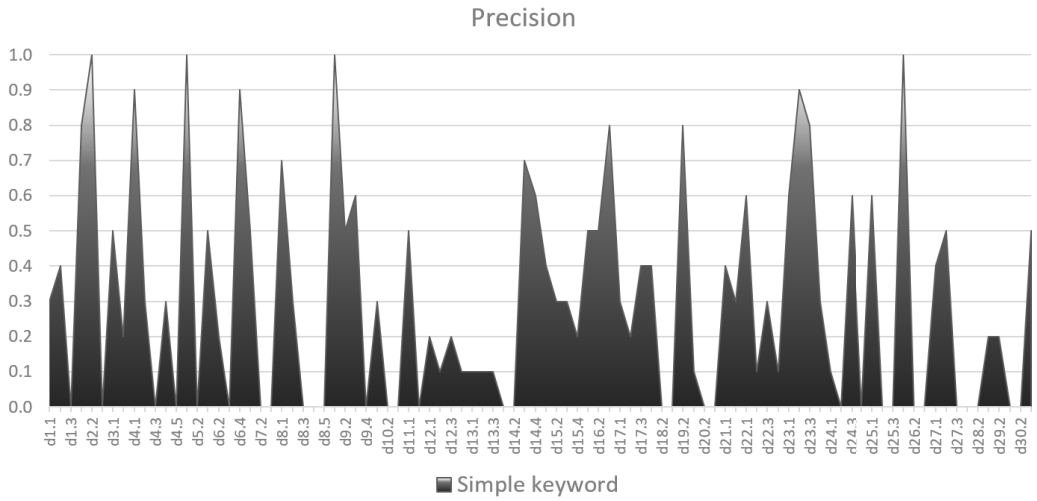
By using the given keywords to search in a resource description repository, we measure the recall, precision, F-measure, and nDCG rates as shown in Table 12.2, 12.3, 12.4, and 12.5 respectively. From these results, the simple search performs differently with different queries. For example, with a corresponding description d13, the simple search achieves 1.0 recall rate from all three combinations of keywords. On the other hand, for a description d15, the simple search can achieve 0.75 average recall rate, while the minimum value is 0.5, and the maximum value is 1.

Although the simple search yields acceptable recall rate on average, the precision rate is unsatisfactory. Thus, we improve the quality of the request constructor by introducing the search with structured keywords (or "advanced search").



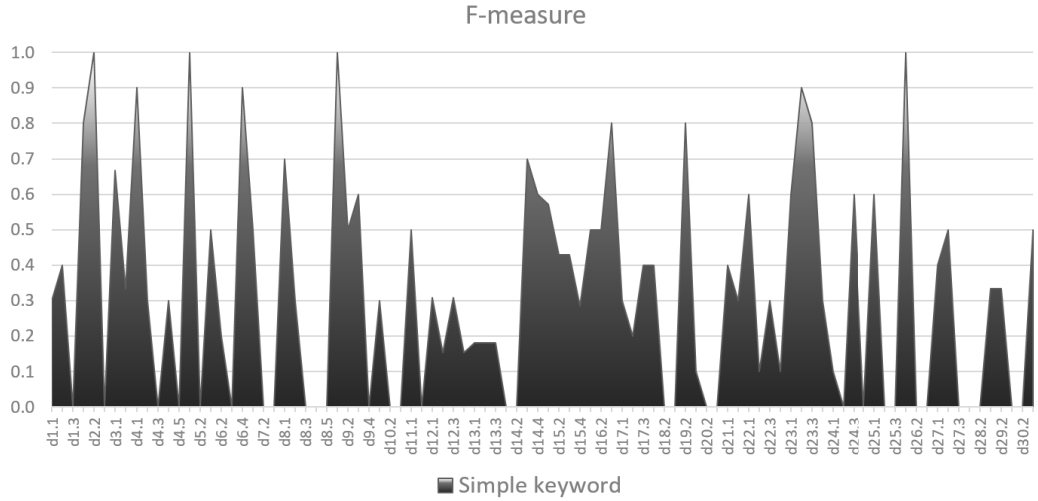
Recall	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
mean	0.23	0.60	1.00	0.30	0.50	0.40	0.17	0.20	0.53	0.10
max	0.40	1.00	1.00	0.90	1.00	0.90	0.50	0.70	1.00	0.30
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.00	0.33	1.00	0.00	0.50	0.50	0.20	0.00	0.00	0.00
mean	0.25	0.50	1.00	0.33	0.75	0.60	0.30	0.20	0.40	0.03
max	0.50	0.67	1.00	0.70	1.00	0.80	0.40	0.40	0.80	0.10
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.30	0.10	0.30	0.00	0.00	0.00	0.40	0.00	1.00	0.00
mean	0.35	0.28	0.65	0.18	0.20	0.33	0.45	0.00	1.00	0.17
max	0.40	0.60	0.90	0.60	0.60	1.00	0.50	0.00	1.00	0.50

Table 12.2.: Recall rate from the request constructor per query using simple keywords.



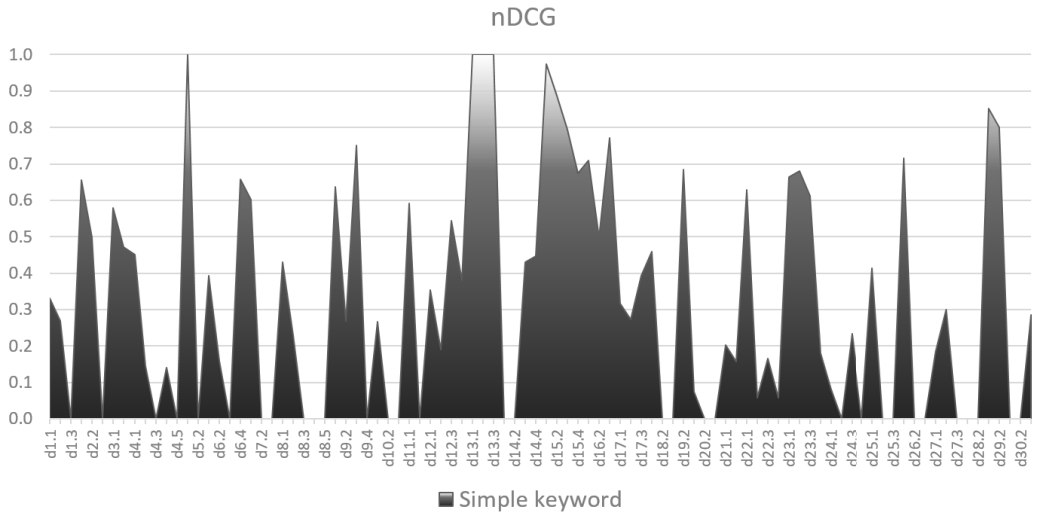
Precision	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.00	0.00	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.00
mean	0.23	0.60	0.35	0.30	0.50	0.40	0.17	0.20	0.53	0.10
max	0.40	1.00	0.50	0.90	1.00	0.90	0.50	0.70	1.00	0.30
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.00	0.10	0.10	0.00	0.20	0.50	0.20	0.00	0.00	0.00
mean	0.25	0.15	0.10	0.33	0.30	0.60	0.30	0.20	0.40	0.03
max	0.50	0.20	0.10	0.70	0.40	0.80	0.40	0.40	0.80	0.10
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.30	0.10	0.30	0.00	0.00	0.00	0.40	0.00	0.20	0.00
mean	0.35	0.28	0.65	0.18	0.20	0.33	0.45	0.00	0.20	0.17
max	0.40	0.60	0.90	0.60	0.60	1.00	0.50	0.00	0.20	0.50

Table 12.3.: Precision rate from the request constructor per query using simple keywords.



F-measure	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.00	0.00	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
mean	0.23	0.60	0.50	0.30	0.50	0.40	0.17	0.20	0.53	0.10
max	0.40	1.00	0.67	0.90	1.00	0.90	0.50	0.70	1.00	0.30
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.00	0.15	0.18	0.00	0.29	0.50	0.20	0.00	0.00	0.00
mean	0.25	0.23	0.18	0.33	0.43	0.60	0.30	0.20	0.40	0.03
max	0.50	0.31	0.18	0.70	0.57	0.80	0.40	0.40	0.80	0.10
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.30	0.10	0.30	0.00	0.00	0.00	0.40	0.00	0.33	0.00
mean	0.35	0.28	0.65	0.18	0.20	0.33	0.45	0.00	0.33	0.17
max	0.40	0.60	0.90	0.60	0.60	1.00	0.50	0.00	0.33	0.50

Table 12.4.: F-measure from the request constructor per query using simple keywords.



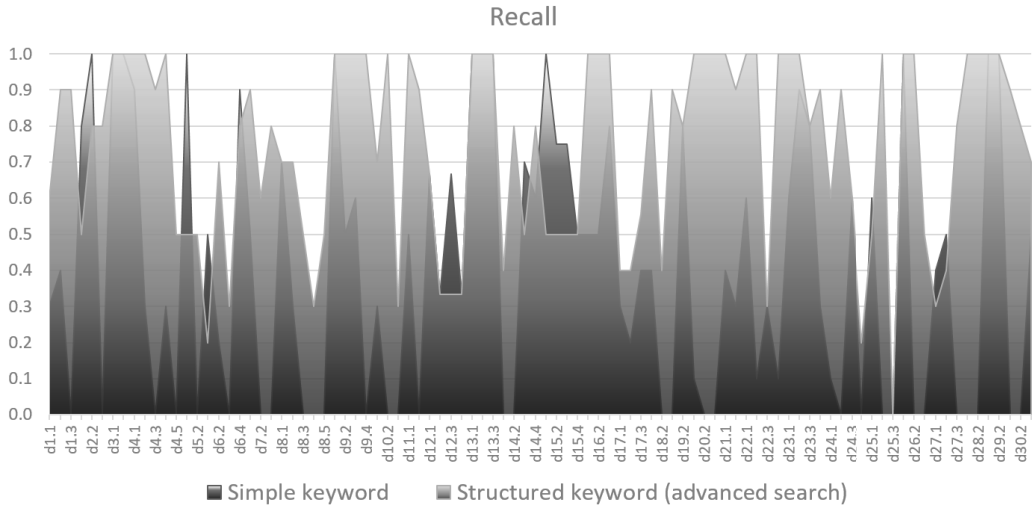
nDCG	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.00	0.00	0.47	0.00	0.00	0.00	0.00	0.00	0.00	0.00
mean	0.20	0.38	0.53	0.15	0.50	0.30	0.20	0.13	0.41	0.09
max	0.33	0.66	0.58	0.45	1.00	0.66	0.60	0.43	0.75	0.27
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.00	0.19	1.00	0.00	0.67	0.50	0.27	0.00	0.00	0.00
mean	0.30	0.37	1.00	0.22	0.83	0.66	0.33	0.23	0.34	0.02
max	0.59	0.54	1.00	0.45	0.97	0.77	0.39	0.46	0.69	0.07
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.16	0.06	0.18	0.00	0.00	0.00	0.18	0.00	0.80	0.00
mean	0.18	0.23	0.53	0.08	0.14	0.24	0.24	0.00	0.83	0.10
max	0.20	0.63	0.68	0.23	0.41	0.72	0.30	0.00	0.85	0.29

Table 12.5.: nDCG from the request constructor per query using simple keywords.

12.1.2. Advanced Search

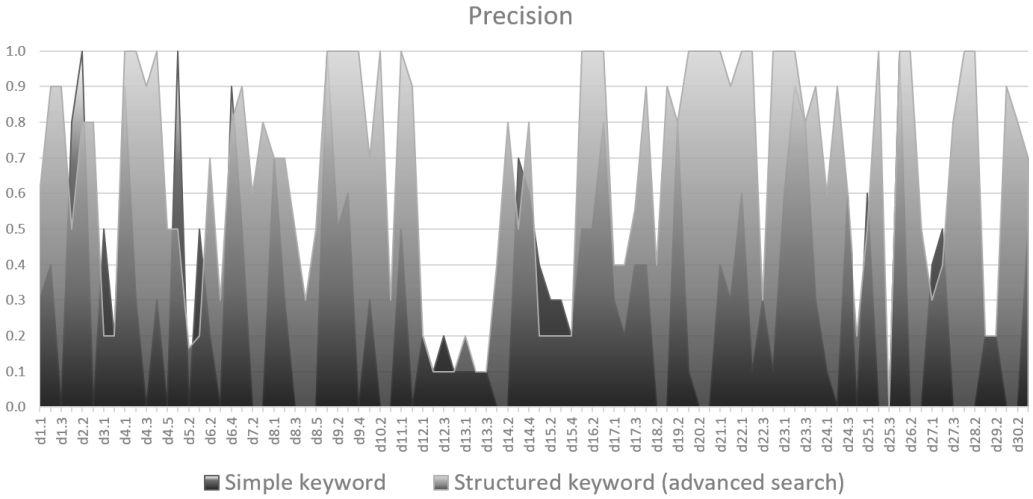
A user can specify the keywords by their functionality, these are called "structured keywords". We provide three basic types of keywords, operation, input, and output descriptions. Applying the same evaluation environment as the simple search, the advanced search is evaluated. We extend the simple keywords in Table 12.1 by putting them into three different categories as shown beside each set of simple keywords. For example, to gain resources corresponding to a description named "book price" (d2), the term "book price" (d2.1) is separated to an input "book" and an output "price".

The recall, precision, F-measure, and nDCG rates measured from the advanced search are shown in Table 12.6, 12.7, 12.8, and 12.9 respectively. On average, an advanced search yields better result quality than a simple search. Some exceptions, such as query d5 ("pill"), has better recall, precision, and nDCG when we use the simple search. This is because the actual relevant results of this query are two, but only one of these results contain the term "pill" in an output field. Therefore, the simple search can detect the keyword in both of the descriptions files, while the advanced search returns only one of them. On the other hand, if a description contains the word "pill" but has no relevance to the targeted description, the simple search could perform worse than the advanced search.



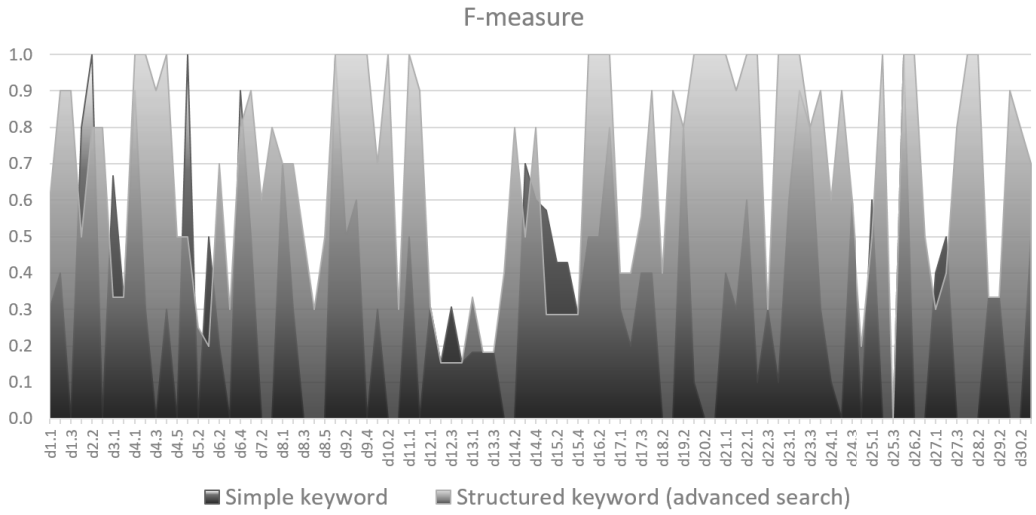
Recall	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.60	0.50	1.00	0.50	0.50	0.20	0.60	0.30	1.00	0.30
mean	0.80	0.70	1.00	0.88	0.50	0.50	0.77	0.54	1.00	0.67
max	0.90	0.80	1.00	1.00	0.50	0.80	0.90	0.70	1.00	1.00
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.90	0.33	1.00	0.40	0.50	1.00	0.40	0.40	0.80	1.00
mean	0.95	0.42	1.00	0.62	0.50	1.00	0.45	0.65	0.85	1.00
max	1.00	0.67	1.00	0.80	0.50	1.00	0.56	0.90	0.90	1.00
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.90	0.30	0.80	0.20	0.00	0.50	0.30	0.80	1.00	0.70
mean	0.95	0.83	0.93	0.58	0.50	0.83	0.35	0.93	1.00	0.80
max	1.00	1.00	1.00	0.90	1.00	1.00	0.40	1.00	1.00	0.90

Table 12.6.: Recall rate from the request constructor per query using structured keywords.



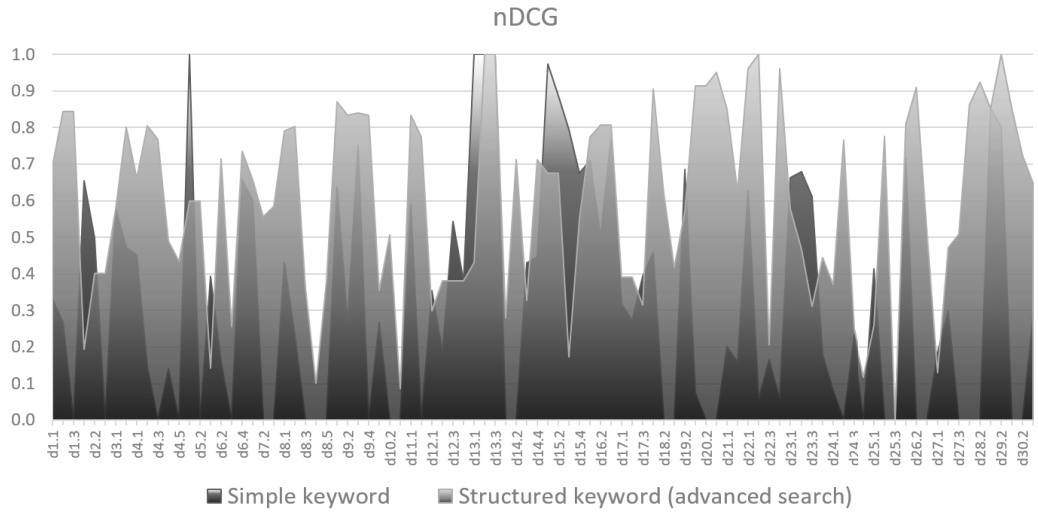
Precision	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.60	0.50	0.20	0.50	0.17	0.20	0.60	0.30	1.00	0.30
mean	0.80	0.70	0.20	0.88	0.33	0.50	0.77	0.54	1.00	0.67
max	0.90	0.80	0.20	1.00	0.50	0.80	0.90	0.70	1.00	1.00
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.90	0.10	0.10	0.40	0.20	1.00	0.40	0.40	0.80	1.00
mean	0.95	0.13	0.13	0.63	0.20	1.00	0.45	0.65	0.85	1.00
max	1.00	0.20	0.20	0.80	0.20	1.00	0.56	0.90	0.90	1.00
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.90	0.30	0.80	0.20	0.00	0.50	0.30	0.80	0.20	0.70
mean	0.95	0.83	0.93	0.58	0.50	0.83	0.35	0.93	0.20	0.80
max	1.00	1.00	1.00	0.90	1.00	1.00	0.40	1.00	0.20	0.90

Table 12.7.: Precision rate from the request constructor per query using structured keywords.



F-measure	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.60	0.50	0.33	0.50	0.25	0.20	0.60	0.30	1.00	0.30
mean	0.80	0.70	0.33	0.88	0.38	0.50	0.77	0.54	1.00	0.67
max	0.90	0.80	0.33	1.00	0.50	0.80	0.90	0.70	1.00	1.00
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.90	0.15	0.18	0.40	0.29	1.00	0.40	0.40	0.80	1.00
mean	0.95	0.19	0.23	0.62	0.29	1.00	0.45	0.65	0.85	1.00
max	1.00	0.31	0.33	0.80	0.29	1.00	0.56	0.90	0.90	1.00
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.90	0.30	0.80	0.20	0.00	0.50	0.30	0.80	0.33	0.70
mean	0.95	0.83	0.93	0.58	0.50	0.83	0.35	0.93	0.33	0.80
max	1.00	1.00	1.00	0.90	1.00	1.00	0.40	1.00	0.33	0.90

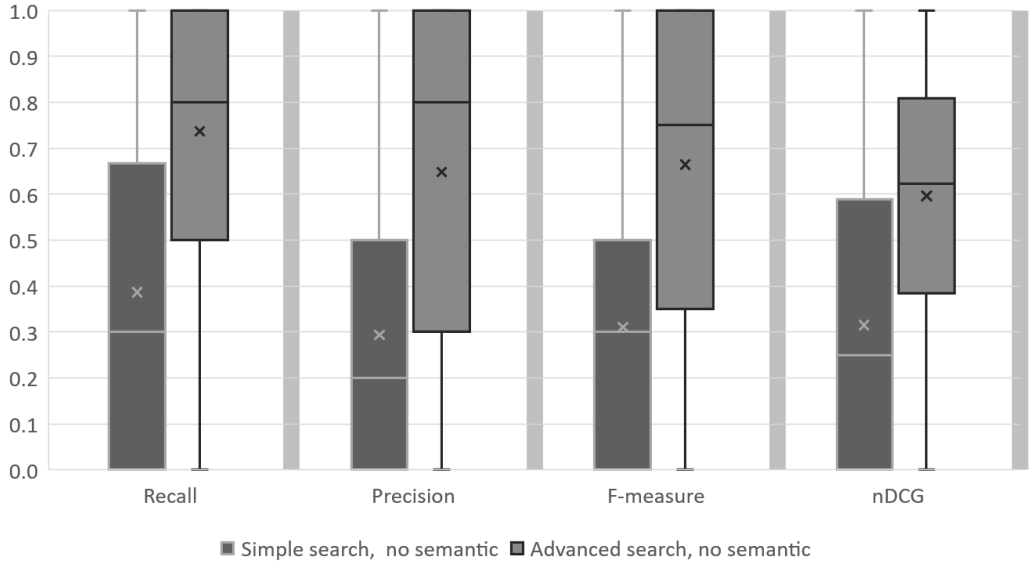
Table 12.8.: F-measure from the request constructor per query using structured keywords.



nDCG	Query ID									
	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
min	0.70	0.19	0.58	0.44	0.60	0.14	0.56	0.10	0.83	0.09
mean	0.79	0.33	0.69	0.63	0.60	0.46	0.60	0.49	0.84	0.32
max	0.84	0.40	0.80	0.81	0.60	0.73	0.66	0.80	0.87	0.51
	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
min	0.77	0.30	0.43	0.28	0.17	0.77	0.32	0.62	0.42	0.91
mean	0.80	0.36	0.81	0.51	0.52	0.80	0.37	0.76	0.49	0.93
max	0.83	0.38	1.00	0.71	0.67	0.81	0.39	0.91	0.57	0.95
	d21	d22	d23	d24	d25	d26	d27	d28	d29	d30
min	0.63	0.21	0.31	0.12	0.00	0.51	0.13	0.51	0.85	0.65
mean	0.74	0.78	0.45	0.38	0.34	0.74	0.30	0.77	0.93	0.74
max	0.85	1.00	0.58	0.77	0.78	0.91	0.47	0.92	1.00	0.85

Table 12.9.: nDCG from the request constructor per query using structured keywords.

To summarize, the simple and advanced searches are compared in Table 12.10. The result shows that using the structured keywords via an advanced search can significantly improve the overall quality of results. This happens because the simple search looks for the keywords without any specification of the position within the description. When we use the simple search to find keywords that contain common terms, there is a high tendency that we will get a low precision result. In contrast, the advanced search focuses on the function of each keyword. Therefore, if these query terms do not appear in the specified fields, we simply neglect that description.



Keywords type		Recall	Precision	F-measure	nDCG
Simple	min	0.00	0.00	0.00	0.00
	mean	0.39	0.29	0.31	0.32
	max	1.00	1.00	1.00	1.00
Structured	min	0.00	0.00	0.00	0.00
	mean	0.74	0.65	0.66	0.60
	max	1.00	1.00	1.00	1.00

Table 12.10.: Request constructor result comparison between using simple keywords and structured keywords (advanced search).

12.1.3. Semantic Search

Besides using the advanced search, we also enhance the request message with semantic terms. For this evaluation, we retrieve semantic terms from Watson KMI semantic API [1], and Altervista Thesaurus API [2]. We expect the semantic extension to increase the recall rate in the simple search.

We use the simple and structured keywords from Table 12.1 to retrieve the semantic terms. For example, the search with the keyword "book" can get extended terms like:

- publication, record, collection

- script, play script, composition, dramatic works

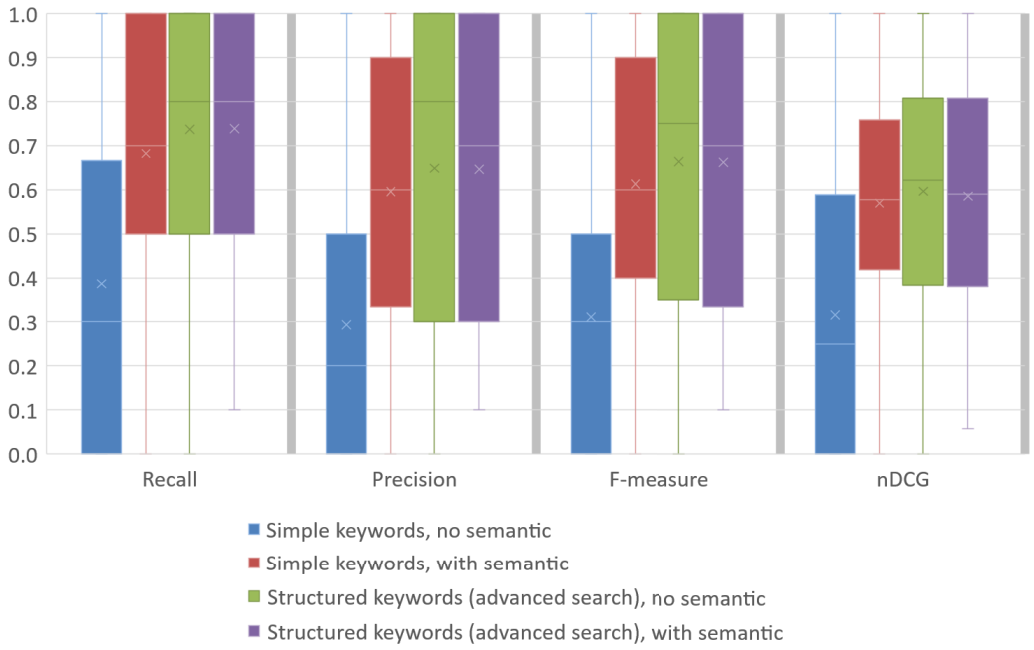
- reserve, hold, request, schedule, register.[3]

The keyword "price", for instance, has the semantic terms like cost, value, worth.

As summarized in Table 12.11, the search result from simple keywords yields better recall, precision, and nDCG when we apply semantic entities. Contrarily, semantic entities are not very helpful when using an advanced search. The extended semantic terms could have introduced more irrelevant results, thus increasing the recall rate but decreasing the precision. Therefore, semantic expansion should be provided for the simple search. On the other hand, when using the structured keywords, the semantic expansion can be excluded.

Note that this evaluation results are from the minimal request constructed from a plain text search. If we compare this result to those from full described OWL-S and SAWSDL descriptions, the result presented here can be inferior.

[1] http://watson.kmi.open.ac.uk/API/entity/keyword/?q=[keyword]
[2] http://thesaurus.altervista.org/thesaurus/v1
[3] The term "book" can be misleading, referring to the action of reserving.



Keywords type		Recall	Precision	F-measure	nDCG
Simple keywords, no semantic	min	0.00	0.00	0.00	0.00
	mean	0.39	0.29	0.31	0.32
	max	1.00	1.00	1.00	1.00
Simple keywords, with semantic	min	0.00	0.00	0.00	0.00
	mean	0.68	0.59	0.61	0.57
	max	1.00	1.00	1.00	1.00
Structured keywords, no semantic	min	0.00	0.00	0.00	0.00
	mean	0.74	0.65	0.66	0.60
	max	1.00	1.00	1.00	1.00
Structured keywords, with semantic	min	0.10	0.10	0.10	0.06
	mean	0.74	0.65	0.66	0.59
	max	1.00	1.00	1.00	1.00

Table 12.11.: Request constructor result comparison between using semantic expansion and no semantic expansion for simple and advanced searches.

12.2 Evaluation of Request Converter

The request converter elaborated in Section 8.2 transforms a description from an existing description in another format, not from plain text keywords. This converter is needed when searching over different description formalisms simultaneously.

The request converter is evaluated by converting existing descriptions into another formalism. Afterwards, the original and the converted descriptions are fed to semantic resource matchers, and the results of matching will be compared to each other. We convert 42 OWL-S descriptions (listed in Appendix B.2.1) to SAWSDL and vice versa. These resource descriptions are available in both OWL-S and SAWSDL formats so that the conversion results can be compared.

First, we test the conversion from SAWSDL to OWL-S. We expect a marginal difference in matching result's quality between converted OWL-S and original OWL-S descriptions which describe the same resource. Likewise, descriptions in OWL-S are converted into SAWSDL formalism and compared with original SAWSDL descriptions.

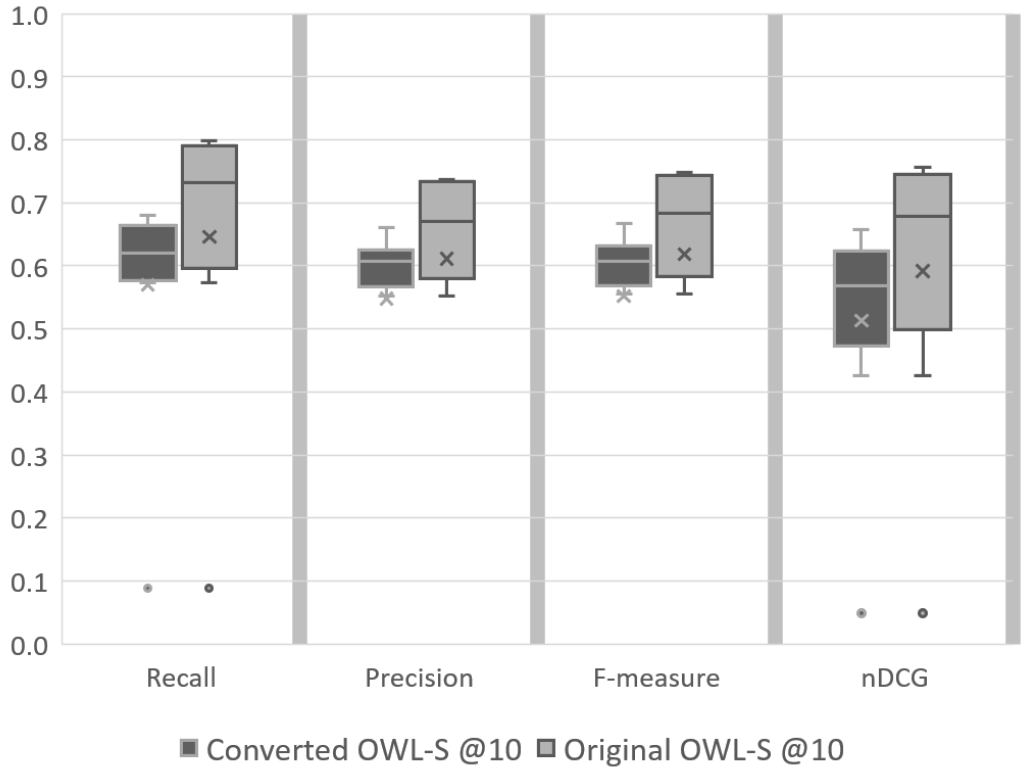
12.2.1. SAWSDL to OWL-S Matching Result

The examples of a source SAWSDL description and a converted OWL-S description can be found in Listing A.1 and A.2 respectively. 42 converted OWL-S and the corresponding OWL-S descriptions are fed to the semantic matchers which are:

- (MO1) iSeM logic-based [KK10],
- (MO2) iSeM approx. logic-based [KK10],
- (MO3) iSeM structure [KK10],
- (MO4) iSeM text similarity (Cos) [KK10],
- (MO5) iSeM text similarity (Cos, structured) [KK10],
- (MO6) SeMa2 [MHB+12],
- (MO7) OWLS-M0 [KFK05],
- (MO8) OWLS-MX2 (M3) [KKF08], and
- (MO9) OWLS-MX3 (M3) [KK12a].

Results shown in Table 12.12 indicate that the converted descriptions have lower quality than the corresponding descriptions. The summary of the evaluation is discussed as follows:

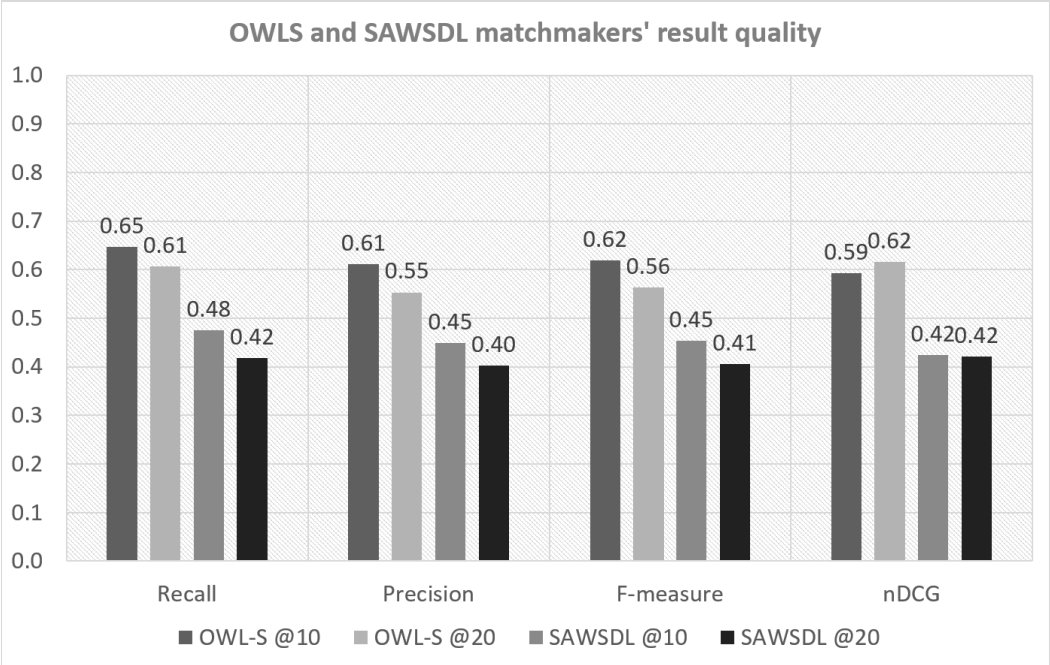
1. The converted descriptions tend to perform worse than the corresponding descriptions. This shows that the original SAWSDL description might not be as complete as the corresponding OWL-S description.



Matcher	Recall		Precision		F-measure		nDCG	
	converted	original	converted	original	converted	original	converted	original
MO1	0.58	0.62	0.58	0.61	0.58	0.61	0.52	0.57
MO2	0.09	0.09	0.09	0.09	0.09	0.09	0.05	0.05
MO3	0.57	0.57	0.55	0.55	0.56	0.56	0.43	0.43
MO4	0.65	0.80	0.61	0.74	0.62	0.75	0.60	0.74
MO5	0.62	0.73	0.58	0.67	0.59	0.68	0.57	0.68
MO6	0.68	0.76	0.66	0.74	0.67	0.74	0.66	0.72
MO7	0.61	0.66	0.61	0.65	0.61	0.65	0.56	0.64
MO8	0.66	0.79	0.62	0.73	0.63	0.74	0.62	0.75
MO9	0.66	0.79	0.62	0.73	0.63	0.74	0.63	0.76
Min	0.09	0.09	0.09	0.09	0.09	0.09	0.05	0.05
Mean	0.57	0.65	0.55	0.61	0.55	0.62	0.51	0.59
Max	0.68	0.80	0.66	0.74	0.67	0.75	0.99	0.76

Table 12.12.: Resource matching result when using converted OWL-S descriptions and corresponding OWL-S descriptions.

2. We found that the SAWSDL descriptions and OWL-S descriptions that are used to describe the same resource do not contain the same amount of information. For example, a service name "get Altitude Above Sea Level of Location" is defined in



Formalism	Recall	Precision	F-measure	nDCG
OWL-S @10	0.6465	0.6109	0.6183	0.5921
OWL-S @20	0.6065	0.5533	0.5630	0.6164
SAWSDL @10	0.4756	0.4487	0.4536	0.4238
SAWSDL @20	0.4175	0.4021	0.4062	0.4212

Table 12.13.: Comparison of quality between OWL-S and SAWSDL matchers.

SAWSDL with "coordinates" as an input, while it is defined in OWL-S with "latitude" and "longitude" as inputs.

3. We also examine the quality of results between the top 10 ranks and top 20 ranks as shown in Table 12.13. We consider using the top 10 ranks because this offers higher recall, precision, and F-measure rates. Although the top 20 ranking performs marginally better in terms of ranking, the top 10 ranking is still more practical to handle.

4. We compare the results from SAWSDL and OWL-S matchers using the same 42 requests. Table 12.13 shows that, on average, the OWLS matchers can produce a better quality of results. Therefore, when we compare converted OWL-S and converted SAWSDL descriptions, it is very likely that the result from using OWL-S descriptions with OWL-S resource matchers can outperform SAWSDL descriptions.

12.2.2. OWL-S to SAWSDL Matching Result

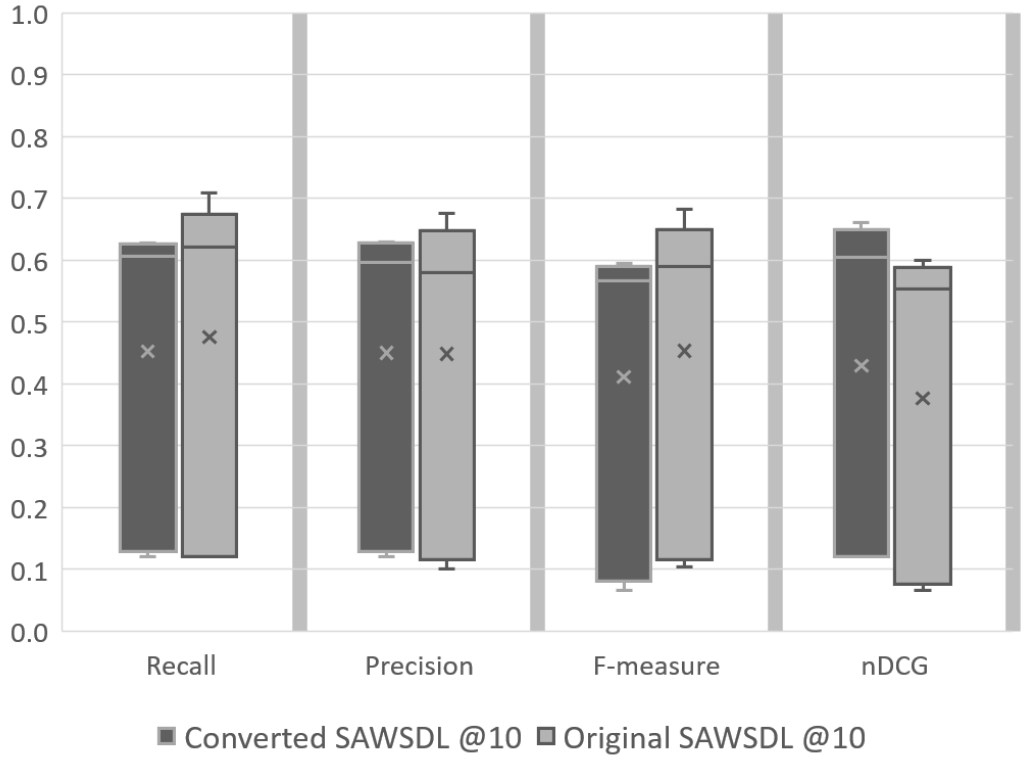
Here, we convert 42 OWL-S descriptions (as listed Appendix B.2.1) into SAWSDL. Like the previous evaluation, we compare the search results from the converted descriptions and the corresponding descriptions. The converted SAWSDL and the corresponding SAWSDL descriptions are fed to the following semantic matchers:

- (MS1) iSeM logic-based [KK12b],
- (MS2) iSeM approx. logic-based.xml [KK12b],
- (MS3) iSeM text similarity (Cos) [KK12b],
- (MS4) iSeM text similarity (Cos, structured) [KK12b],
- (MS5) SAWSDL-MX TextSim (eJAC) [KKZ09a], and
- (MS6) XAM4SWS-LOG [SLES10].

The examples of OWL-S description and SAWSDL description converted from it can be found in Appendix A, Listing A.3 and A.4 respectively. The evaluation result is shown in Table 12.14. The result qualities of converted descriptions and their corresponding descriptions are slightly different. Nevertheless, the converted descriptions provide a better ranking.

1. The converted SAWSDL descriptions provide worse results than the OWL-S origins. This is because of the intrinsic performance of matchers described previously in Table 12.13.
2. The converted SAWSDL descriptions yield slightly worse results than their corresponding SAWSDL descriptions. The declination in quality happens because the converter cannot transfer all substantial information from OWL-S to SAWSDL formalism. However, when we look back at the result from 12.2.1, we can see that these original descriptions are not identical, thus, causing the nonequivalent in the result.
3. According to the result and the conclusion we made in 12.2.1, we consider using the top 10 ranks.

There is no significant difference between original SAWSDL descriptions and the converted descriptions. However, the converted OWL-S descriptions yield worse matching results than the corresponding OWL-S descriptions. This is due to three factors. First is the intrinsic performance of the matchers of each formalism. The second is the fact that the original OWL-S and SAWSDL descriptions that are used to describe the same resource contain different amount of information. Lastly, there is information loss during the conversion, which indicates that the request conversion still has a gap for improvement.



Matcher	Recall		Precision		F-measure		nDCG	
	converted	original	converted	original	converted	original	converted	original
MS1	0.62	0.64	0.62	0.64	0.62	0.64	0.56	0.58
MS2	0.12	0.12	0.12	0.12	0.12	0.12	0.07	0.07
MS3	0.64	0.61	0.63	0.55	0.63	0.56	0.60	0.55
MS4	0.64	0.66	0.63	0.61	0.63	0.62	0.59	0.60
MS5	0.13	0.12	0.13	0.10	0.13	0.10	0.09	0.08
MS6	0.59	0.71	0.57	0.68	0.58	0.68	0.54	0.66
Min	0.12	0.12	0.12	0.10	0.07	0.10	0.12	0.07
Mean	0.45	0.48	0.45	0.45	0.41	0.45	0.45	0.42
Max	0.63	0.71	0.63	0.68	0.60	0.68	0.66	0.66

Table 12.14.: Resource matching result when using converted SAWSDL descriptions and corresponding OWL-S descriptions.

12.3 Evaluation of Result Integrator

The result integrator accumulates results from multiple matchers and repeatedly readjusts the result until it obtains the best result. The quality of the result from each iteration is compared with individual service matchers.

As mentioned in 9.2.4, *Check weight change function*, we need to set a threshold value to find the exit loop condition. If this threshold value is too high, the iteration will end before we can get the best result. If the threshold value is too low, the iteration condition might never be met, causing an infinite loop. According to Equation 9.6, $threshold = mean \times \kappa \times standard\ variation$, the threshold value is calculated from the mean, the variance of matchers' weights and the κ variable, which is set to 0.2 in this evaluation.

We use 42 descriptions from the test collections in B.2.1. For the sake of brevity, we provide (randomly selected) 30 solutions out of 42. The complete solutions can be found in [Klu12]. Similar to the previous evaluations, we use the resource matchers from Table 11.1.

We simulate the situation when an offline conversion is used. In this case, we assume that all descriptions are available (or prepared) in one single formalism, either OWL-S or SAWSDL. Randomly selected 9 OWL-S matchers consume 42 OWL-S requests and produce the list of matched resources out of 1080 descriptions. Likewise, 6 SAWSDL matchers (randomly selected) process 42 SAWSDL requests (the same resources as OWL-S) and find the matched resources from 1080 descriptions.

Additionally, we simulate the situation when an online conversion is applied. This means our resources can be described in one formalism or another. Thus, we simulate a setting where we use both OWL-S and SAWSDL based matchers to match over different description formalisms. The results from each type of matchers should be different because all resources are described in one or another formalism. The result integrator handles combining the results from different formalisms.

12.3.1. Result integration using OWL-S matchers

OWL-S matchers used in this evaluation are:

- (MO1) iSeM approx. logic-based [KK10]
- (MO2) iSeM logic-based [KK10]
- (MO3) iSeM text similarity (Cos) [KK10]
- (MO4) iSeM text similarity (Cos, structured) [KK10]
- (MO5) OWLS-MX TextSim (Cos) [KFK05]

- (MO6) OWLS-M0 [KFK05]
- (MO7) OWLS-MX2 (M3) [KK12a]
- (MO8) OWLS-MX3 (M3) [KK12a]
- (MO9) OWLS-MX3 (Structure) [KK12a].

After running 9 OWL-S matchers over the 42 requests, average recall, precision, F-measure, and nDCG rates at top 10 ranks from every OWL-S matchers are shown in Figure 12.1, 12.2, 12.3, and 12.4 respectively.

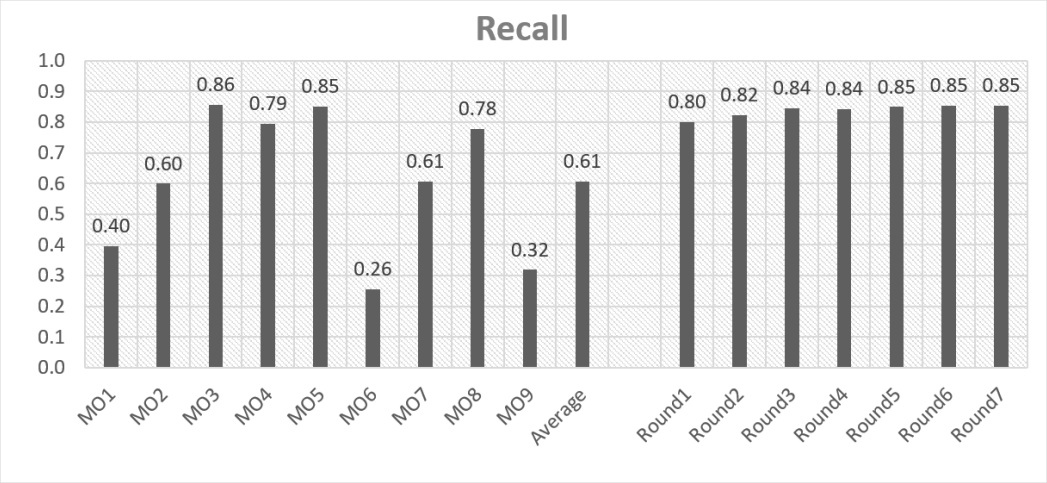


Figure 12.1.: Recall rate of each OWL-S matcher compared with the result integrator.

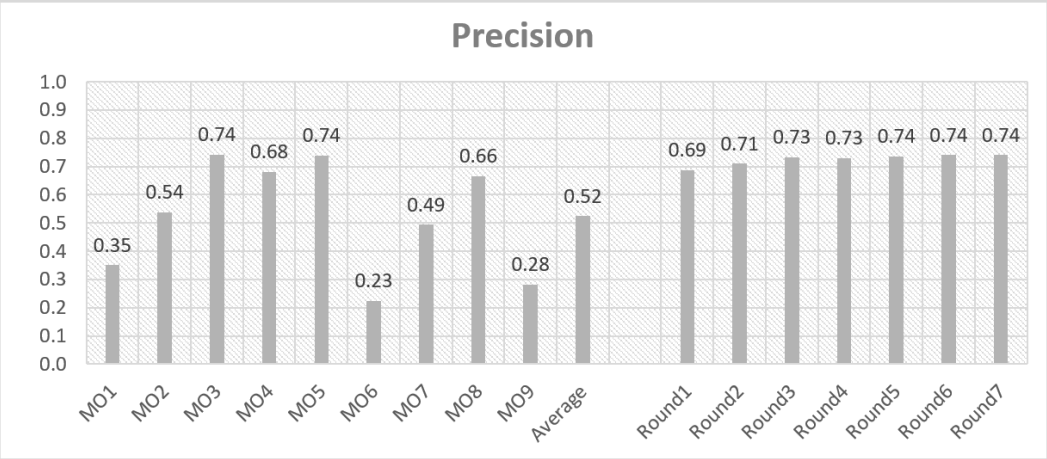


Figure 12.2.: Precision rate of each OWL-S matcher compared with the result integrator.

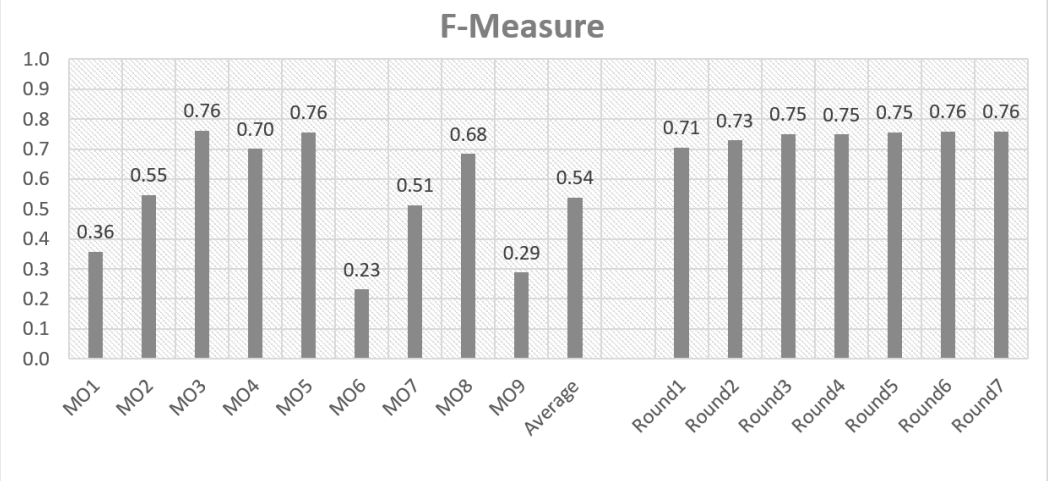


Figure 12.3.: F-measure of each OWL-S matcher compared with the result integrator.

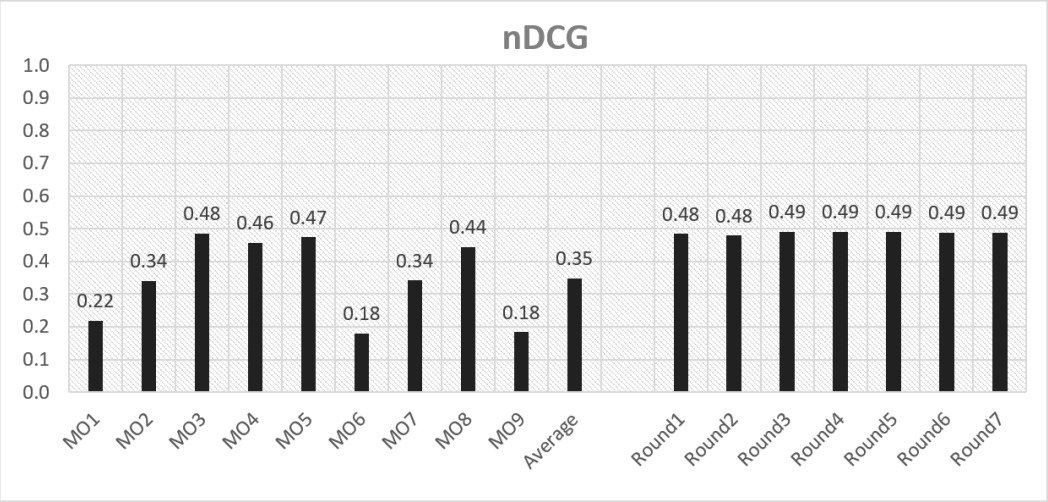


Figure 12.4.: nDCG of each OWL-S matcher compared with the result integrator.

OWL-S matchers	Recall	Precision	F-measure	nDCG
MO1	0.3962	0.3506	0.3568	0.2191
MO2	0.5995	0.5368	0.5465	0.3400
MO3	**0.8561**	**0.7424**	**0.7615**	**0.4846**
MO4	0.7948	0.6818	0.7006	0.4578
MO5	0.8506	0.7381	0.7566	0.4736
MO6	0.2564	0.2251	0.2315	0.1789
MO7	0.6067	0.4935	0.5124	0.3418
MO8	0.7777	0.6645	0.6834	0.4441
MO9	0.3182	0.2814	0.2881	0.1835
Average	0.6062	0.5238	0.5375	0.3470
Round1	0.7996	0.6861	0.7051	0.4842
Round2	0.8234	0.7100	0.7289	0.4790
Round3	0.8448	0.7316	0.7505	0.4904
Round4	0.8426	0.7294	0.7483	0.4895
Round5	0.8491	0.7359	0.7548	0.4907
Round6	0.8537	0.7403	0.7592	0.4877
Round7	**0.8537**	**0.7403**	**0.7592**	**0.4882**

Table 12.15.: Result integrator's quality from Figure 12.1, 12.2, 12.3, and 12.4 in each iteration compared to each OWL-S matcher. The results listed as Round [number] refer to the results from the result integration after each iteration.

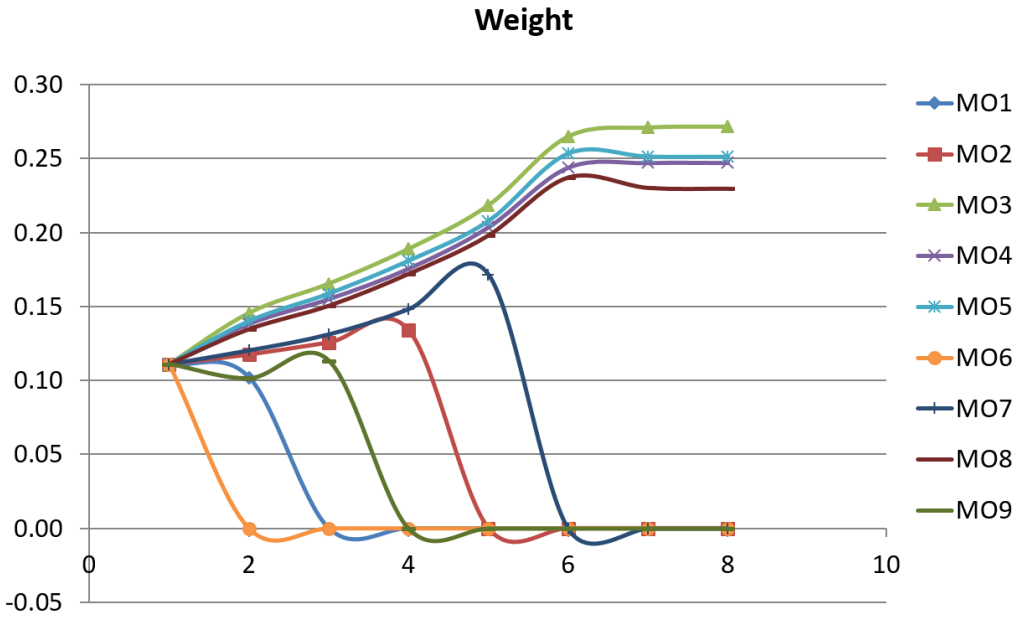
1. Table 12.15 summarizes the quality from each matcher and the result integrator. The final result from the seventh round, has higher nDCG rate than the best performance matcher, *MO3*. On the other hand, the best performance matcher yields slightly better recall, precision, and F-measure rates than the final result. The number of the selected round for the final result is not always seven. This number depends on the request, matchers' results, and the threshold value.

2. The weight measurement in Table 12.16 shows that matchers' weights became stable after the seventh round. Thus, the iteration stopped in the seventh round, which supposedly provides the best result.

3. Moreover, the nDCG of the fifth round is marginally better than the seventh round. We can increase the threshold level to force the result integrator to stop earlier. Nevertheless, we see no significant difference in these figures. Therefore, the result from the seventh round is plausible and acceptable as the best result.

4. When we look closely into the weights comparison, we can see that *MO1* performs marginally better than *MO9*. However, *MO1* was removed from the calculation before *MO9*. The reason is that the result integrator also considers the threshold condition while considering the poorest performance matcher. When the threshold is set to a lower value, the sequence of dropping matcher will change as well.

5. In this experiment, five matchers (MO1, MO2, MO6, MO7, and MO9) were dropped out after some iterations due to their poor performances. The decision to remove which matcher in each iteration is based on the performance of that matcher on each request. Therefore, there is no matcher that can always be abandoned or included.



Weight of matcher in round	1	2	3	4	5	6	7	8
MO1	0.111	**0.102**	**0.000**	0.000	0.000	0.000	0.000	0.000
MO2	0.111	0.118	0.126	0.134	0.000	0.000	0.000	0.000
MO3	0.111	0.146	0.165	0.189	0.218	0.265	0.271	0.272
MO4	0.111	0.138	0.155	0.176	0.204	0.244	0.247	0.247
MO5	0.111	0.140	0.159	0.181	0.208	0.253	0.251	0.251
MO6	0.111	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MO7	0.111	0.120	0.131	0.148	0.172	0.000	0.000	0.000
MO8	0.111	0.135	0.151	0.172	0.198	0.237	0.230	0.230
MO9	0.111	**0.101**	**0.113**	**0.000**	0.000	0.000	0.000	0.000

Table 12.16.: Weight value of OWL-S matchers calculated from each iteration.

12.3.2. Result integration using SAWSDL matchers

SAWSDL matchers used in this evaluation are:

- (MS1) iSem Approx. logic based [KK12b],
- (MS2) iSem Logic-based [KK12b],
- (MS3) iSem Text Sim. (Cos) [KK12b],
- (MS4) iSem Text Sim. (Cos +Structure) [KK12b],
- (MS5) XAM4SWS-COV [SLKS12], and
- (MS6) XAM4SWS-LOG [SLES10].

Using 6 SAWSDL matchers matching over the 42 requests, we measure average recall, precision, F-measure, and nDCG rates (at top ten ranks) as shown in Figure 12.5, 12.6, 12.7, and 12.8 respectively.

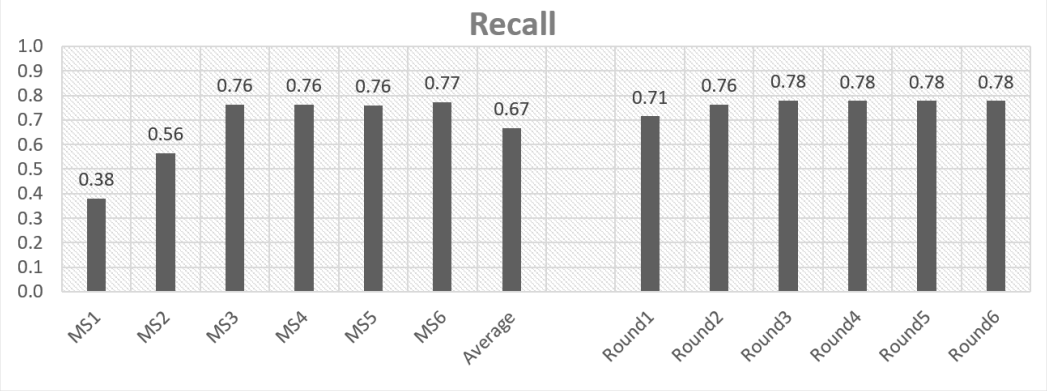


Figure 12.5.: Recall rate of each SAWSDL matcher compared with the result integrator.

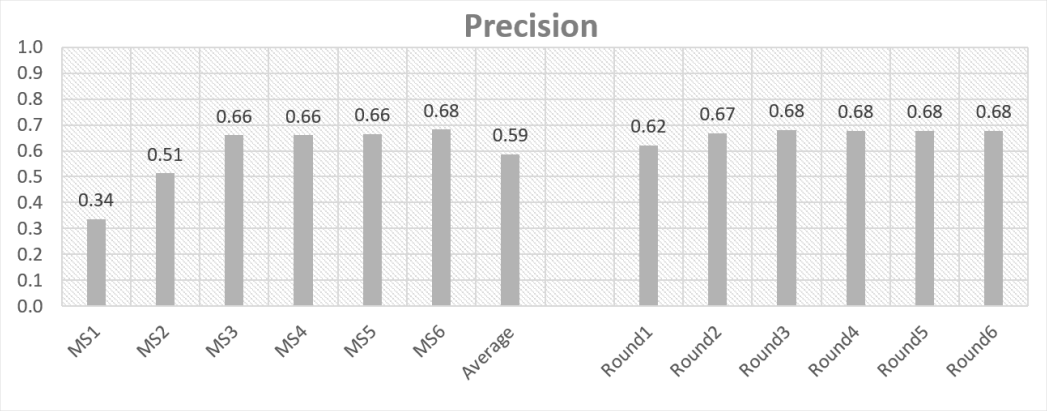


Figure 12.6.: Precision rate of each SAWSDL matcher compared with the result integrator.

1. Table 12.17 summarizes the quality of result from each matcher and the result integrator. The condition to stop the iteration is met in the sixth round. The result yields better recall rate than the best result from *MS6* matcher. However, the precision, F-measure, and nDCG rates are slightly lower than the best matcher. This could happen when all the matchers return different results which are not consistent with each other.

2. The weights of all matchers are calculated and shown in Table 12.18. In this experiment, the criteria to select the round that the weights become stable is met in the sixth round.

3. Two poor performance matchers (MS1 and MS2) are dropped out.

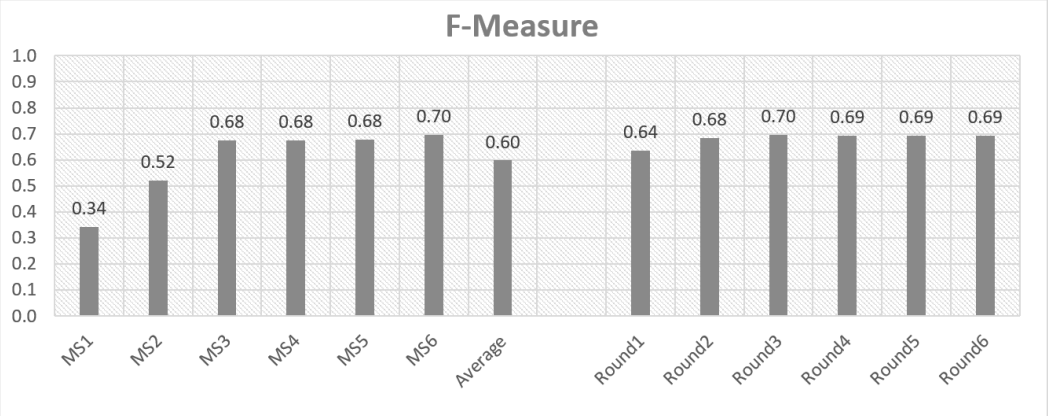


Figure 12.7.: F-measure of each SAWSDL matcher compared with the result integrator.

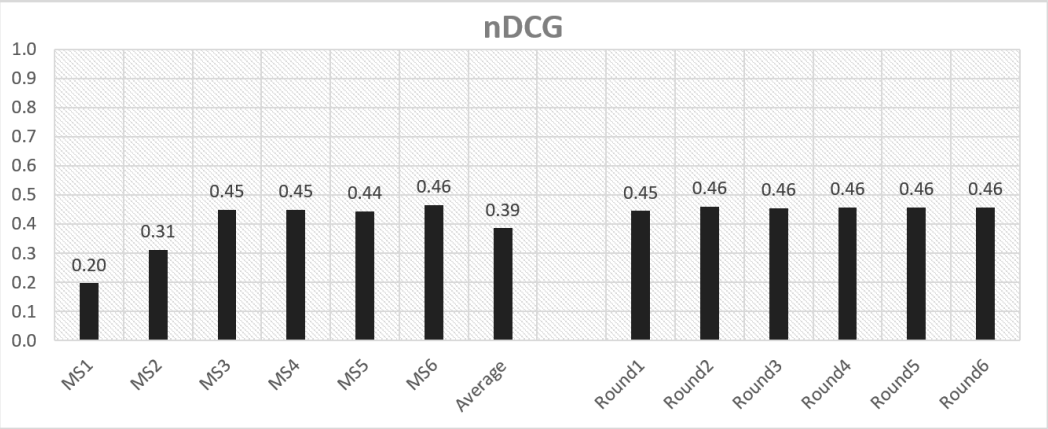
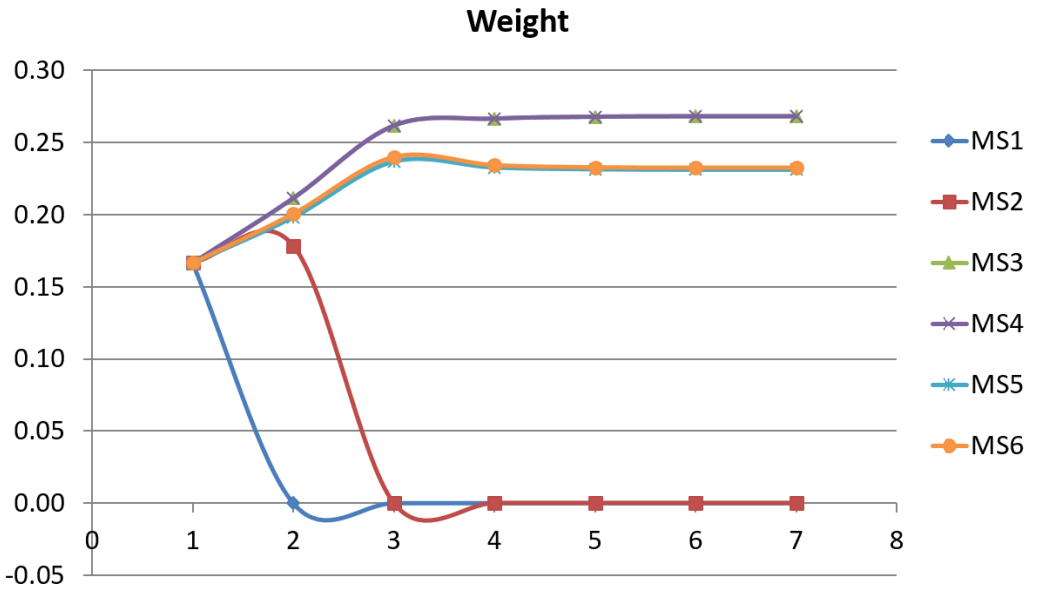


Figure 12.8.: nDCG of each SAWSDL matcher compared with the result integrator.

SAWSDL matchers	Recall	Precision	F-measure	nDCG
MS1	0.3806	0.3355	0.3414	0.1966
MS2	0.5641	0.5130	0.5202	0.3119
MS3	0.7620	0.6602	0.6760	0.4481
MS4	0.7620	0.6602	0.6760	0.4481
MS5	0.7586	0.6645	0.6789	0.4438
MS6	**0.7719**	**0.6818**	**0.6951**	**0.4648**
Average	0.6665	0.5859	0.5979	0.3856
Round1	0.7149	0.6212	0.6354	0.4451
Round2	0.7627	0.6688	0.6831	0.4589
Round3	0.7793	0.6797	0.6951	0.4553
Round4	0.7771	0.6775	0.6930	0.4560
Round5	0.7771	0.6775	0.6930	0.4565
Round6	**0.7771**	**0.6775**	**0.6930**	**0.4565**

Table 12.17.: Result integrator's quality from Figures 12.5, 12.6, 12.7, and 12.8 in each iteration compared to each SAWSDL matcher.



Weight of matcher in round	1	2	3	4	5	6	7
MS1	0.167	0.000	0.000	0.000	0.000	0.000	0.000
MS2	0.167	0.178	0.000	0.000	0.000	0.000	0.000
MS3	0.167	0.211	0.262	0.266	0.268	0.268	0.268
MS4	0.167	0.211	0.262	0.266	0.268	0.268	0.268
MS5	0.167	0.198	0.237	0.233	0.232	0.231	0.231
MS6	0.167	0.201	0.240	0.234	0.233	0.233	0.233

Table 12.18.: Weight value of SAWSDL matchers calculated from each iteration.

12.3.3. Result integration using OWL-S and SAWSDL matchers

The result integration can cope with different formalism-based matchers. In order to evaluate this functionality, we separate resource descriptions in a repository into two groups. The first group contains only OWL-S descriptions, and the second contains only SAWSDL descriptions. OWL-S and SAWSDL matchers used in this evaluation are chosen randomly:

- (M1) OWLS-iSeM text similarity (Cos) [KK10],
- (M2) OWLS-iSeM text similarity (Cos, structured) [KK10],
- (M3) OWLS-iSeM approx. logic-based [KK10],
- (M4) OWLS-iSeM logic-based [KK10],
- (M5) OWLS-SeMa2 [MHB+12],
- (M6) OWLS-M0 [KFK05],
- (M7) OWLS-MX2 (M3) [KKF08],
- (M8) OWLS-MX3 (M3) [KK12a],
- (M9) XAM4SWS-LOG4SWS [SLES10],
- (M10) SAWSDL-iSeM approx. logic-based [KK12b],
- (M11) SAWSDL-iSeM logic-based [KK12b],
- (M12) SAWSDL-iSeM text similarity (Cos) [KK12b],
- (M13) SAWSDL-iSeM text similarity (Cos, structured) [KK12b], and
- (M14) SAWSDL-MX TextSim (eJAC) [KKZ09a].

Using these matchers to match 42 requests (in both OWL-S and SAWSDL formalisms) with 540 SAWSDLS and 540 OWL-S descriptions (randomly separated from 1080 descriptions), average recall, precision, F-measure, and nDCG rates from every matchers are shown in Figure 12.9, 12.10, 12.11, and 12.12 respectively.

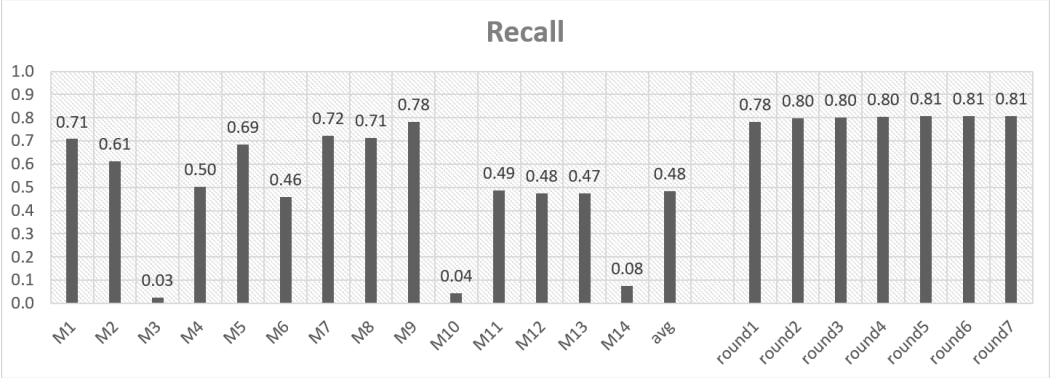


Figure 12.9.: Recall rate of OWL-S and SAWSDL matchers compared with the result integrator.

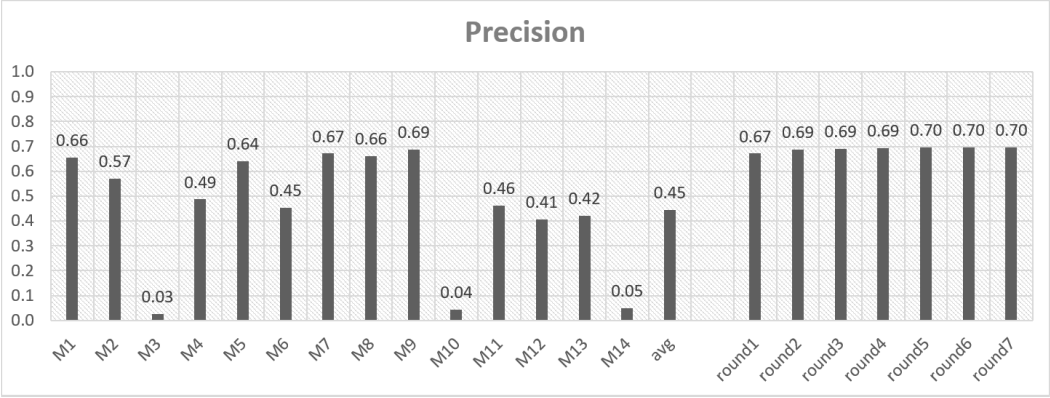


Figure 12.10.: Precision rate of OWL-S and SAWSDL matchers compared with the result integrator.

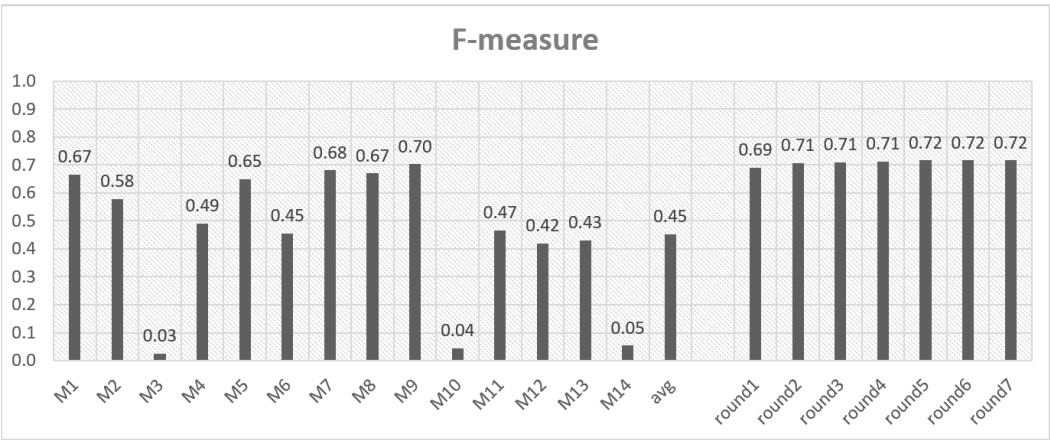


Figure 12.11.: F-measure of OWL-S and SAWSDL matchers compared with the result integrator.

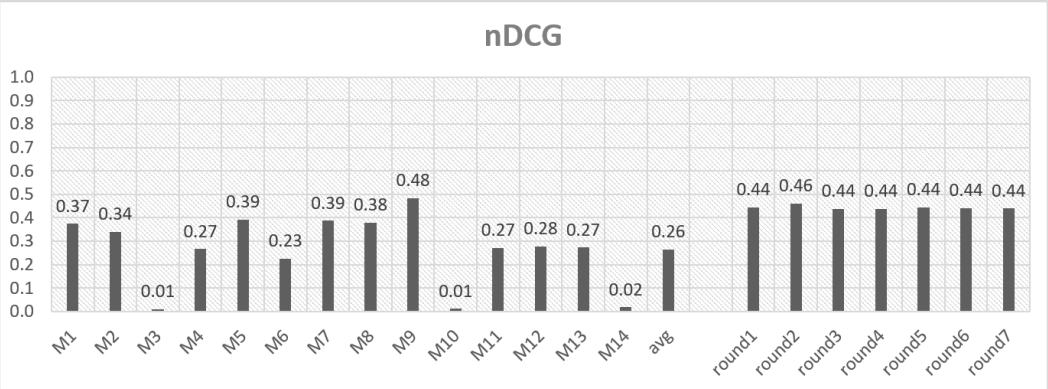


Figure 12.12.: nDCG of OWL-S and SAWSDL matchers compared with the result integrator.

Matchers	Recall	Precision	F-measure	nDCG
M1	0.7086	0.6558	0.6653	0.3739
M2	0.6120	0.5693	0.5771	0.3395
M3	0.0260	0.0260	0.0260	0.0094
M4	0.5025	0.4870	0.4897	0.2662
M5	0.6854	0.6407	0.6479	0.3923
M6	0.4582	0.4524	0.4536	0.2251
M7	0.7235	0.6710	0.6803	0.3872
M8	0.7127	0.6602	0.6695	0.3783
M9	**0.7822**	**0.6883**	**0.7026**	**0.4822**
M10	0.0433	0.0433	0.0433	0.0117
M11	0.4865	0.4610	0.4654	0.2703
M12	0.4752	0.4069	0.4191	0.2770
M13	0.4746	0.4221	0.4306	0.2731
M14	0.0750	0.0498	0.0540	0.0181
Average	0.4833	0.4453	0.4518	0.2646
Round1	0.7820	0.6710	0.6902	0.4440
Round2	0.7972	0.6861	0.7053	0.4605
Round3	0.8015	0.6905	0.7097	0.4373
Round4	0.8039	0.6926	0.7119	0.4379
Round5	0.8082	0.6970	0.7163	0.4446
Round6	0.8082	0.6970	0.7163	0.4404
Round7	**0.8082**	**0.6970**	**0.7163**	**0.4392**

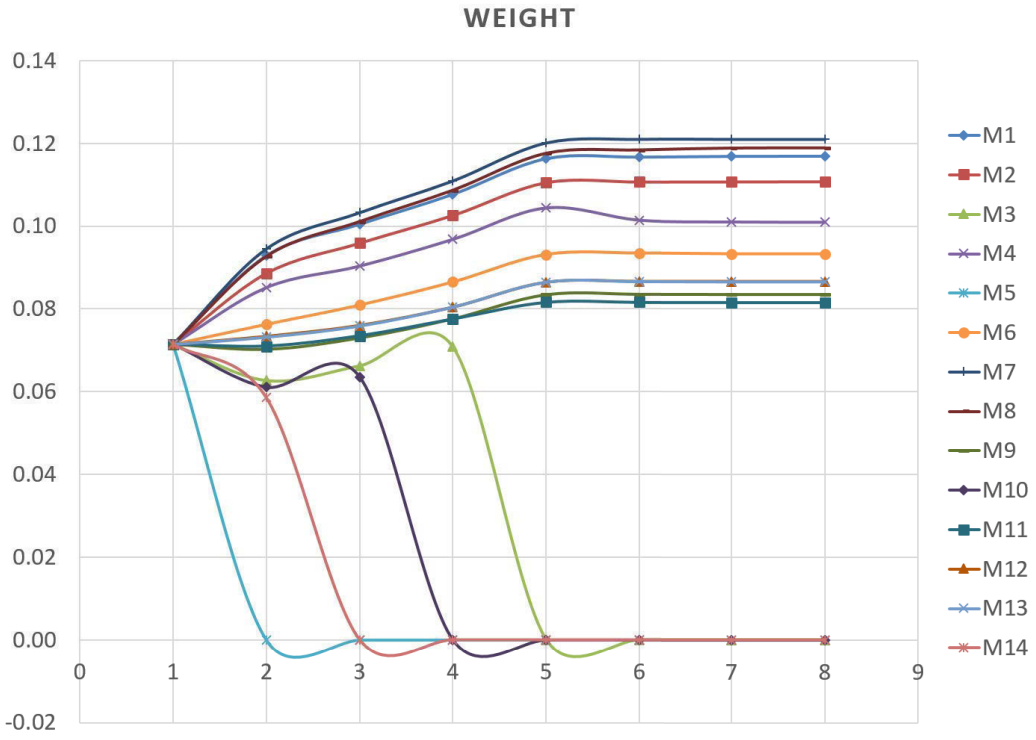
Table 12.19.: Result integrator's quality from Figures 12.9, 12.10, 12.11, and 12.12 in each iteration compared to each matcher.

1. As summarized in Table 12.19, we found that the integrated result yields better recall, precision, and F-measure rates than the best matcher, *M9*, which is a SAWSDL based matcher. This is because some returned relevant resources are described in OWL-S language.

2. In contrast, the nDCG rate of our method cannot achieve the best quality. OWL-S matchers and SAWSDL matchers have different methods of ranking their results, but we treat them equally. Therefore, the ranking of integrated results might be biased by the majority of matchers.

3. The weights of all matchers are calculated and shown in Table 12.20. The criteria to select the round that weights become stable is met in the seventh round. Again, the number of the selected round can be changed, depending on the evaluation settings.

4. Four poor performance matchers (M3, M5, M10, and M14) are dropped. Two of them are OWL-S based matchers, and the other two are SAWSDL based matchers. Matchers M3, M10, and M14 perform critically poorer than others. Matcher M5 is removed first, even though it performs better than many matchers. This happens because M5 returns the most distinguishing result that is inconsistent with the majority of matchers.



Weight of matcher in round	1	2	3	4	5	6	7	8
M1	0.071	0.093	0.100	0.108	0.116	0.117	0.117	0.117
M2	0.071	0.089	0.096	0.103	0.111	0.111	0.111	0.111
M3	0.071	0.063	0.066	0.071	0.000	0.000	0.000	0.000
M4	0.071	0.085	0.090	0.097	0.104	0.101	0.101	0.101
M5	0.071	0.000	0.000	0.000	0.000	0.000	0.000	0.000
M6	0.071	0.076	0.081	0.086	0.093	0.093	0.093	0.093
M7	0.071	0.095	0.103	0.111	0.120	0.121	0.121	0.121
M8	0.071	0.093	0.101	0.109	0.118	0.118	0.119	0.119
M9	0.071	0.070	0.073	0.078	0.083	0.084	0.084	0.083
M10	0.071	0.061	0.064	0.000	0.000	0.000	0.000	0.000
M11	0.071	0.071	0.073	0.078	0.082	0.082	0.082	0.082
M12	0.071	0.073	0.076	0.080	0.086	0.087	0.087	0.087
M13	0.071	0.073	0.076	0.080	0.086	0.087	0.087	0.087
M14	0.071	0.059	0.000	0.000	0.000	0.000	0.000	0.000

Table 12.20.: Weight value of multi-type matchers calculated from each iteration.

12.3.4. Tuning up the result integrator

From the previous results, we demonstrate that the result integrator can capture good results and combine them into a single list. Additionally, in the circumstance that resources are described in one formalism or another formalism, not both, we can retrieve better results than the best performance matcher. The algorithm we use in the result integrator is not dependent on the description formalism. Thus, it can be applied to any other description languages.

There are some factors we can adjust to get a better performance such as the number of considering ranks, the threshold value for removing matchers, and the minimum number of matchers. These values should be tuned to fit the requirements. For example, if we increase the number of ranks from 10 to 20, the recall rate would increase. However, the precision rate would drop in exchange. The threshold value can be very small to ensure the best final result, but it could take longer processing time. Moreover, if the minimum number of matchers is too high, we might end up unable to remove any matchers even though they perform poorly.

Previously, we evaluated the result integrator using 14 matchers. We run the evaluation with a different number of matchers to check if the result integrator is still able to detect the best result.

We apply 42 requests from B.2.1 to 6 matchers, which are:

- (M1) OWLS-iSeM text similarity (Cos),

- (M7) OWLS-MX2 (M3),

- (M8) OWLS-MX3 (M3),

- (M9) SAWSDL-XAM4SWS-LOG,

- (M13) SAWSDL-iSeM text similarity (Cos, structured), and

- (M14) SAWSDL-SAWSDL-MX TextSim (eJAC).

The summarized result in Figure 12.13 shows that the result integrator can imitate the best result. However, the recall, precision, and the F-measure rates are slightly worse than the result from the best matcher. The cross-formalism result integrator cannot perform well enough when using few matchers. Therefore, the more matchers we include, the better result we gain. Nevertheless, when we compare our solution to the average performance, the result is very satisfactory.

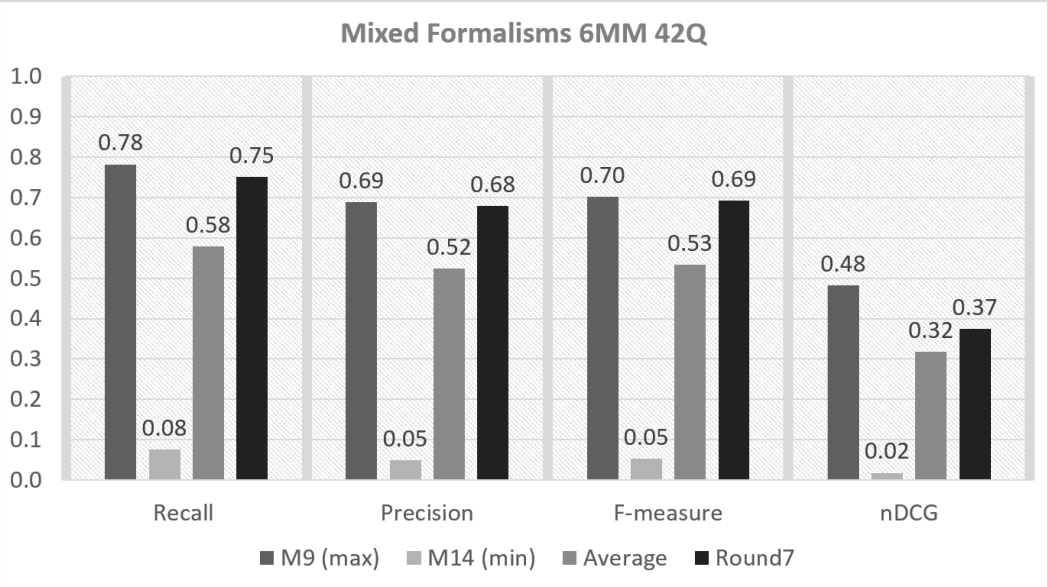


Figure 12.13.: Comparison of results from the best performance matcher, poorest performance matcher, average performance of 6 matchers, and the request integrator running over 42 requests.

12.3.5. Summary of Result Integrator

Looking back at the OWL-S matching result in Table 12.15, our approach boosted the overall quality from the average matchers by more than 40%. However, the result of our approach cannot always overcome the best matcher's result. When comparing to the best matcher, the recall, precision, and F-measure decrease by less than 0.3%, while nDCG increases by 0.7% This trade-off is acceptable since the decrease in quality is insignificant.

A SAWSDL matching result in Table 12.17 shows that results from our approach are more relevant than the average matchers by more than 15%. Comparing to the best matcher, the recall rate improves by 0.7%, while the precision decreases by 0.6%, and the nDCG is worse by 2%.

One distinctive feature of the result integrator is the ability to handle results from multiple formalisms. The merged result outperforms the best matcher when all resources are not commonly described in one formalism, which is more likely to happen in a real-world scenario.

We are confident that the result integrator can yield as good a result as the best matcher on each request. The drawback of this result integrator is that it requires more than one good matcher to perform well. If there is only one good matcher and the other matchers perform poorly but consistent with each other, our result integrator will trust the majority results and thus produce a wrong result. However, we believe that this marginal decrease in quality is negligible, especially when we have no ground truth to compare with.

13

Integrated System Evaluation Results

When we connect the request construction, the request converter, the resource matchers, and the result integrator as explained in Chapter 5, *Solution Overview*, we will have a complete resource discovery engine.

We focus on the quality of the discovered resources list, e.g. how many relevant results are returned and how they are ranked. Also, the time performance is a major concern. We discuss what can be the blockage of the process and how to improve it. Finally, we place the resource discovery into the context of MERCURY and demonstrate how MERCURY's users could benefit from this work.

13.1 Overall Quality Performance

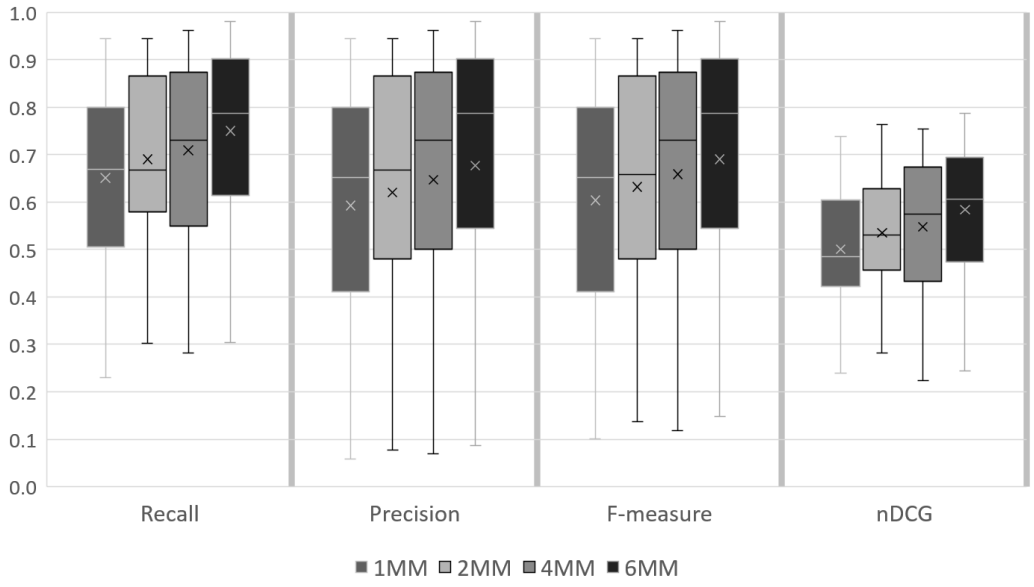
As previously shown in Section 7.1, *Matcher Analysis*, the quality of the result depends on matchers' performance on each query. This evaluation randomly selects one, two, four and six SAWSDL matchers to be evaluated. The settings for matchers are shown in Table 13.1. Similar to the evaluation settings in Table 12.1, 188 keywords with 30 target descriptions are used in this evaluation.

Settings	Matchers
1 Matcher	M1 - (OWL-S) iSeM logic-based M2 - (OWL-S) iSeM approx. logic-based M3 - (OWL-S) iSeM text similarity (Cos) M4 - (OWL-S) iSeM text similarity (Cos, structured) M5 - OWLS-MX2 (M3) M6 - OWLS-M0 M7 - (SAWSDL)iSeM logic-based M8 - (SAWSDL)iSeM approx. logic-based M9 - (SAWSDL)iSeM text similarity (Cos) M10 - (SAWSDL)iSeM text similarity (Cos, structured) M11 - (SAWSDL)iSeM structure M12 - SAWSDL-MX TextSim (eJAC)
2 Matchers	M7+M12 M1+M4 M9+M2 M5+M12 M8+M10 M2+M5
4 Matchers	M7+M8+M9+M10 M2+M3+M4+M5 M2+M5+M9+M12 M3+M6+M7+M10
6 Matchers	M7+M8+M9+M10+M11+M12 M1+M4+M3+M2+M6+M5 M1+M3+M5+M10+M8+M12 M4+M5+M6+M7+M9+M10

Table 13.1.: Sets of matchers used for the integrated system evaluation.

Table 13.2 shows that the average recall, precision, F-measure, and nDCG rates improve with the increasing number of matchers. The minimum recall, precision, F-measure, and nDCG rates from using four matchers are less than those from using two matchers. This happens due to the chance of getting poor performance matchers in four matchers being higher than two matchers. We depend on the results of the majority of matchers, without any prior knowledge of matchers or expected results. Thus, including more matchers can become useful when most of the matchers are reliable.

Note that the result from one matcher presented here is not equivalent to the result from a resource matcher alone. In the resource discovery, we have the request converter to create a request from plain text keywords, while resource matchers accept only a formatted request.



Number of matchers		1	2	4	6
Recall	min	0.2292	0.3021	0.2813	0.3047
	mean	0.6512	0.6900	0.7096	0.7501
	max	0.9438	0.9438	0.9625	0.9813
Precision	min	0.0583	0.0778	0.0701	0.0868
	mean	0.5930	0.6200	0.6467	0.6767
	max	0.9438	0.9438	0.9625	0.9813
F-measure	min	0.1015	0.1367	0.1183	0.1486
	mean	0.6037	0.6322	0.6590	0.6902
	max	0.9438	0.9438	0.9625	0.9813
nDCG	min	0.2400	0.2822	0.2235	0.2444
	mean	0.5006	0.5350	0.5482	0.5844
	max	0.7384	0.7631	0.7540	0.7870

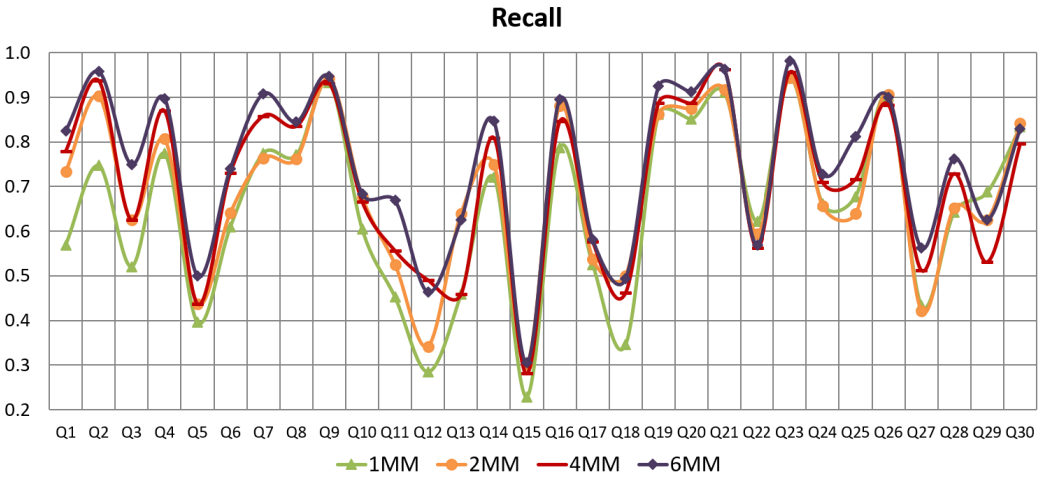
Table 13.2.: Quality measurement of the resource discovery comparing between using one, two, four, and six matchers.

Recall, precision, F-measure, and nDCG rates for each query are shown in Table 13.3, 13.4, 13.5, and 13.6 respectively. We group the queries for each target descriptions into 30 sets. Each set is referred to as "Q" following with the number of the corresponding description. For example, the description for a resource "camera price" should be discovered by using keywords d1.1, d1.2, and d1.3 (refer to Table 12.1). In total, three simple queries and three structured queries are grouped and referred to as "Q1".

Although the average performance is satisfactory, when we consider the result per query, we can see that the multiple matchers solution still fails in some cases. For example, with request Q29, one matcher performs better than using multiple matchers. This happens when a few matchers yield the correct result. In this extreme case, less than half of matchers return the right answer. Moreover, there are two relevant results for this query, so the recall rate can only be 0, 0.5 or 1. Any combination of multiple matchers cannot give the correct answer because it seems like an incorrect solution to the integrator. This indicates the weakness of this approach when the majority of matchers return incorrect results, the result integrator still trusts them.

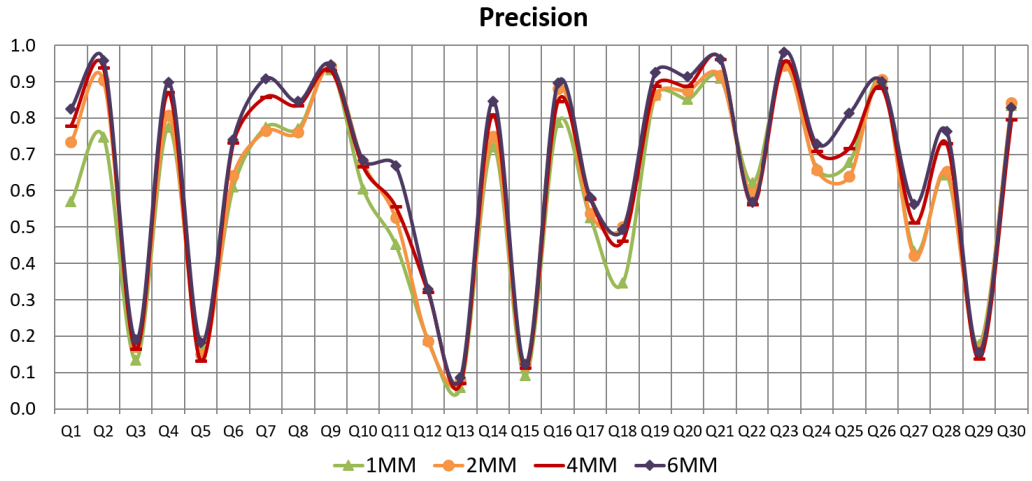
When using six matchers, it is possible that the best performance matcher can be removed or suppressed because the other matchers return the same incorrect result. Nevertheless, as illustrated in Table 13.2, the multiple matchers can averagely perform better than the single matcher approach. As previously discussed, we do not know which matcher can perform best in each circumstance. Therefore, using multiple matchers technique can increase the chance of getting a better result.

Remarks: the context extraction from Chapter 6 has not been evaluated in this work yet. This part shall be evaluated in the future work since we need to find the outcome of this evaluation first. Then we can use this work as a baseline for further improvements using context information.



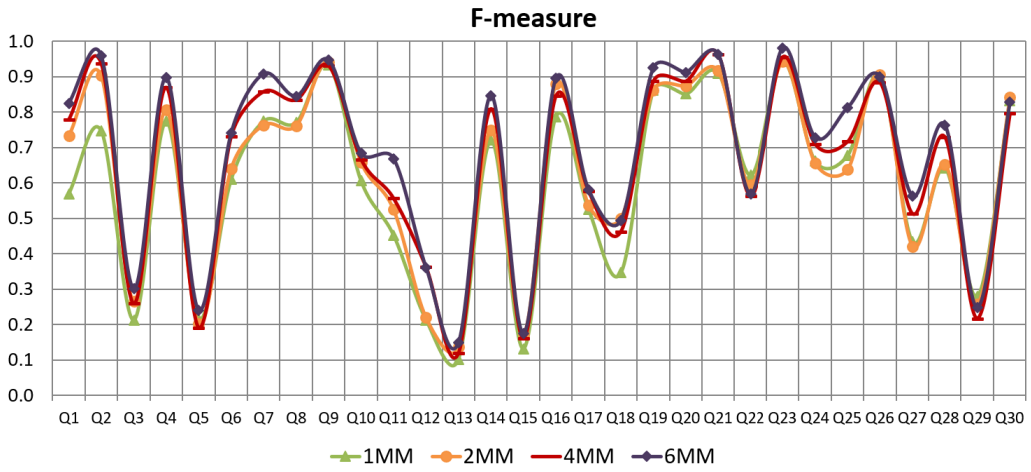
#Matchers	Query ID									
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1	0.57	0.75	0.52	0.77	0.40	0.61	0.78	0.77	0.93	0.61
2	0.73	0.90	0.63	0.81	0.44	0.64	0.76	0.76	0.94	0.68
4	0.78	0.94	0.63	0.87	0.44	0.73	0.86	0.84	0.93	0.67
6	0.82	0.96	0.75	0.90	0.50	0.74	0.91	0.85	0.95	0.68
	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
1	0.45	0.28	0.46	0.72	0.23	0.79	0.53	0.35	0.86	0.85
2	0.53	0.34	0.64	0.75	0.30	0.88	0.54	0.50	0.86	0.88
4	0.56	0.49	0.46	0.81	0.28	0.85	0.58	0.46	0.89	0.89
6	0.67	0.46	0.63	0.85	0.30	0.90	0.58	0.49	0.93	0.91
	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30
1	0.91	0.62	0.94	0.66	0.68	0.90	0.43	0.64	0.69	0.83
2	0.92	0.59	0.94	0.66	0.64	0.91	0.42	0.65	0.63	0.84
4	0.96	0.56	0.96	0.71	0.72	0.88	0.51	0.73	0.53	0.80
6	0.96	0.57	0.98	0.73	0.81	0.90	0.56	0.76	0.63	0.83

Table 13.3.: Recall rate of the resource discovery comparing between using one, two, four, and six matchers.



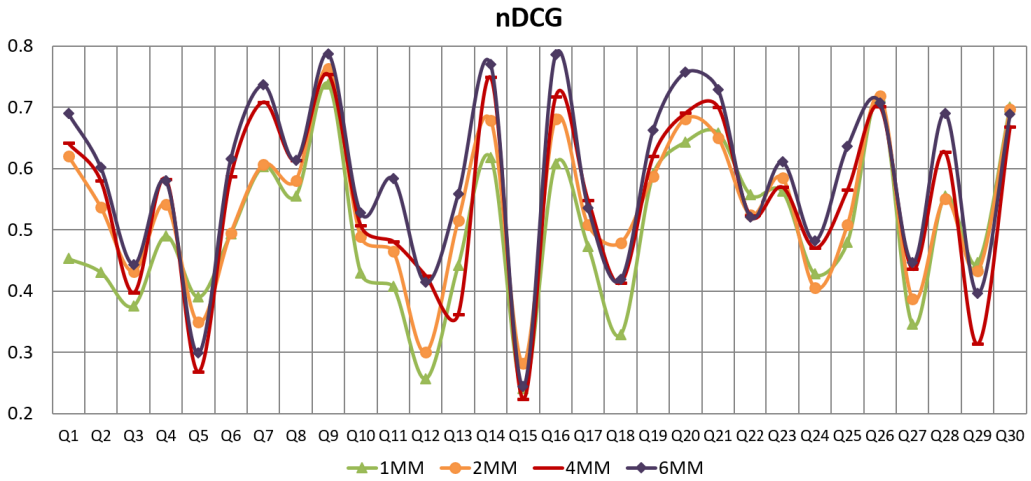
#Matchers	Query ID									
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1	0.57	0.75	0.13	0.77	0.18	0.61	0.78	0.77	0.93	0.61
2	0.73	0.90	0.17	0.81	0.15	0.64	0.76	0.76	0.94	0.68
4	0.78	0.94	0.16	0.87	0.13	0.73	0.86	0.84	0.93	0.67
6	0.83	0.96	0.19	0.90	0.18	0.74	0.91	0.85	0.95	0.68
	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
1	0.45	0.19	0.06	0.72	0.09	0.79	0.53	0.35	0.86	0.85
2	0.53	0.19	0.08	0.75	0.12	0.88	0.54	0.50	0.86	0.88
4	0.56	0.32	0.07	0.81	0.11	0.85	0.58	0.46	0.89	0.89
6	0.67	0.33	0.09	0.85	0.12	0.90	0.58	0.49	0.93	0.91
	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30
1	0.91	0.62	0.94	0.66	0.68	0.90	0.43	0.64	0.18	0.83
2	0.92	0.59	0.94	0.66	0.64	0.91	0.42	0.65	0.16	0.84
4	0.96	0.56	0.96	0.71	0.72	0.88	0.51	0.73	0.14	0.80
6	0.96	0.57	0.98	0.73	0.81	0.90	0.56	0.76	0.16	0.83

Table 13.4.: Precision rate of the resource discovery comparing between using one, two, four, and six matchers.



#Matchers	Query ID									
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1	0.57	0.75	0.21	0.77	0.22	0.61	0.78	0.77	0.93	0.61
2	0.73	0.90	0.27	0.81	0.20	0.64	0.76	0.76	0.94	0.66
4	0.78	0.94	0.26	0.87	0.19	0.73	0.86	0.84	0.93	0.67
6	0.83	0.96	0.30	0.90	0.24	0.74	0.91	0.85	0.95	0.68
	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
1	0.45	0.21	0.10	0.72	0.13	0.79	0.53	0.35	0.86	0.85
2	0.53	0.22	0.14	0.75	0.17	0.88	0.54	0.50	0.86	0.88
4	0.56	0.36	0.12	0.81	0.16	0.85	0.58	0.46	0.89	0.89
6	0.67	0.36	0.15	0.85	0.17	0.90	0.58	0.49	0.93	0.91
	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30
1	0.91	0.62	0.94	0.66	0.68	0.90	0.43	0.64	0.28	0.83
2	0.92	0.59	0.94	0.66	0.64	0.91	0.42	0.65	0.26	0.84
4	0.96	0.56	0.96	0.71	0.72	0.88	0.51	0.73	0.22	0.80
6	0.96	0.57	0.98	0.73	0.81	0.90	0.56	0.76	0.25	0.83

Table 13.5.: F-measure of the resource discovery comparing between using one, two, four, and six matchers.



#Matchers	Query ID									
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
1	0.45	0.43	0.38	0.49	0.39	0.49	0.60	0.56	0.74	0.43
2	0.62	0.54	0.43	0.54	0.35	0.49	0.61	0.58	0.76	0.49
4	0.64	0.58	0.40	0.58	0.27	0.59	0.71	0.61	0.75	0.51
6	0.69	0.60	0.44	0.58	0.30	0.62	0.74	0.61	0.79	0.53
	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
1	0.41	0.26	0.44	0.62	0.24	0.61	0.47	0.33	0.59	0.64
2	0.46	0.30	0.52	0.68	0.28	0.68	0.51	0.48	0.59	0.68
4	0.48	0.43	0.36	0.75	0.22	0.72	0.55	0.41	0.62	0.69
6	0.58	0.41	0.56	0.77	0.24	0.79	0.54	0.42	0.66	0.76
	Q21	Q22	Q23	Q24	Q25	Q26	Q27	Q28	Q29	Q30
1	0.66	0.56	0.56	0.43	0.48	0.72	0.35	0.56	0.45	0.70
2	0.65	0.52	0.58	0.41	0.51	0.72	0.39	0.55	0.43	0.70
4	0.70	0.52	0.57	0.47	0.57	0.70	0.44	0.63	0.31	0.67
6	0.73	0.52	0.61	0.48	0.64	0.71	0.45	0.69	0.40	0.69

Table 13.6.: nDCG of the resource discovery comparing between using one, two, four, and six matchers.

13.2 Overall Time Consumption

In practice, the quality of result alone might not suffice. We demonstrated that the more matchers are involved, the better result we gain. However, many good performance matchers take a long time to process the request. They could take several minutes, and this is impractical when we want to have an on-the-go recommendation. Hence, we conduct the overall time consumption evaluation to verify what could be our optimal solution.

This evaluation was made on a machine with Intel Core i5-2520M CPU 2.5 GHz and 8 GB RAM, Windows 7 64-bit OS. Note that the performance can still be tuned up with a dedicated server with more powerful capacities. We use the same test settings as in Section 13.1, comparing between using a single matcher, two matchers, four matchers, and six matchers. The matchers settings and the keywords used for testing can be found in Table 13.1 and Table 12.1 respectively.

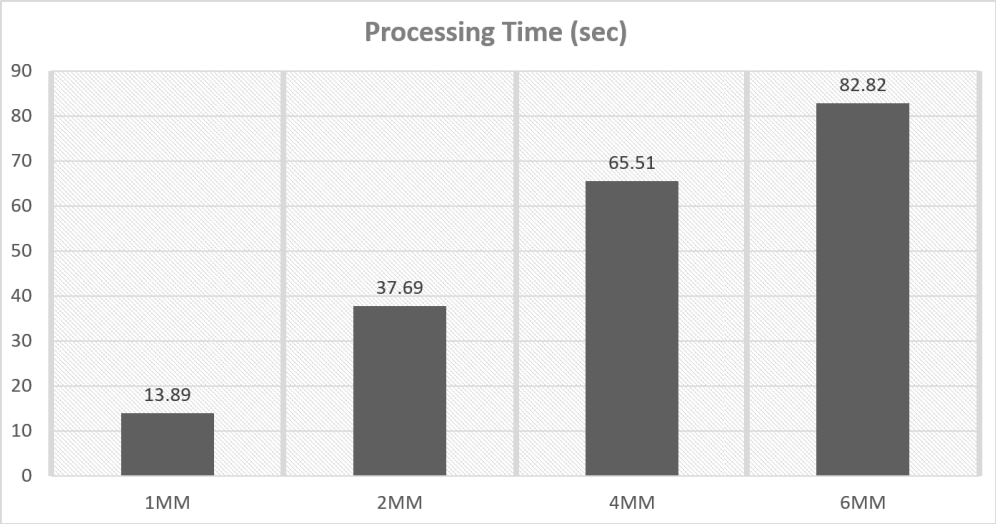
The overall time consumption of the resource discovery is shown in Table 13.7. The request preparation spends the most time in the HTTP request to the Semantic search API. Since the HTTP request for each query term can be done simultaneously, we use a multi-thread process for calling the semantic search function. This multi-threading can drastically improve the time consumption in waiting for the server response. Although the number of requests directly affect the time spent on the semantic search, a number of matchers does not affect the time performance in this part.

All the resource matchers need to parse all descriptions as a preparation for a matching process. The main parameter that affects this process is the number of descriptions in the repository, whereas the number of matchers and requests are irrelevant to the time spent on this process.

The most time-consuming process is the matching process. Since we use the multi-threading process in this step, the time consumption depends on the slowest matcher. It is not always the case, but the tendency is that the more matchers involved in the process, the longer processing time it takes. Therefore, depending on the user requirement, whether we need a faster or more accurate solution, this decides how many matchers should be used.

Finally, results from all matchers are merged by the result integrator. The more matchers are involved, the longer time is needed for the merging process because of the amount of processing data. Nevertheless, we can neglect the difference in this part because most of the processing time has been spent in the matching process.

This time processing result in running matchers can be compared with V-Doc [QHC06], which deals directly with the ontology matching. V-Doc was evaluated on Intel Pentium4, 2.4 GHz, 512 M memory, Windows XP. It takes 7 minutes to complete 51 requests. To compare this figure with our approach, we alter the number of requests to see how the total processing time changes. We measure the total time used to process one request, five requests and ten requests by one matcher (1MM/1Q, 1MM/5Q, 1MM/10Q), two requests



#Matchers / Processing time (sec)	1	2	4	6
Request Preparation	0.67	0.75	0.68	0.97
Read TestCollection	0.01	0.01	0.01	0.01
Run Matcher	**13.12**	**36.82**	**64.67**	**81.66**
Merge Ranks	0.10	0.11	0.15	0.18
Total	13.89	37.69	65.51	82.82

Table 13.7.: Comparison of time consumption of the resource discovery between using one, two, four and six matchers.

and five requests by two matchers (2MM/2Q, 2MM/5Q), ten requests by four matchers (4MM/10Q), and ten requests by six matchers (6MM/10Q).

From Figure 13.1, we can see that the number of requests has a minor influence on the total processing time compared to the number of matchers. The resource matchers have a lot of head cost to read all the descriptions in a description repository before matching, making little difference whether we submit one query or fifty queries in one process.

If 51 requests are made at once, the processing time for one matcher should not be more than 20 seconds. On the other hand, if we submit one request at a time, it can take up to 11.8 minutes. These processing times can even be longer when using multiple matchers in exchange for higher quality results. Moreover, compared to an approach that considers synonyms from WordNet running with the same settings, it takes up to four hours[QHC06]. We cannot compare the quality of these works to our results directly because the evaluations were done on different datasets. However, this shows the trend that the ontology-based matching usually takes minutes to complete, not to mention the semantic extension.

Currently, our approach provides satisfactory results in terms of quality. However, the time performance should be improved and we cannot rely solely on the ontology-based match-

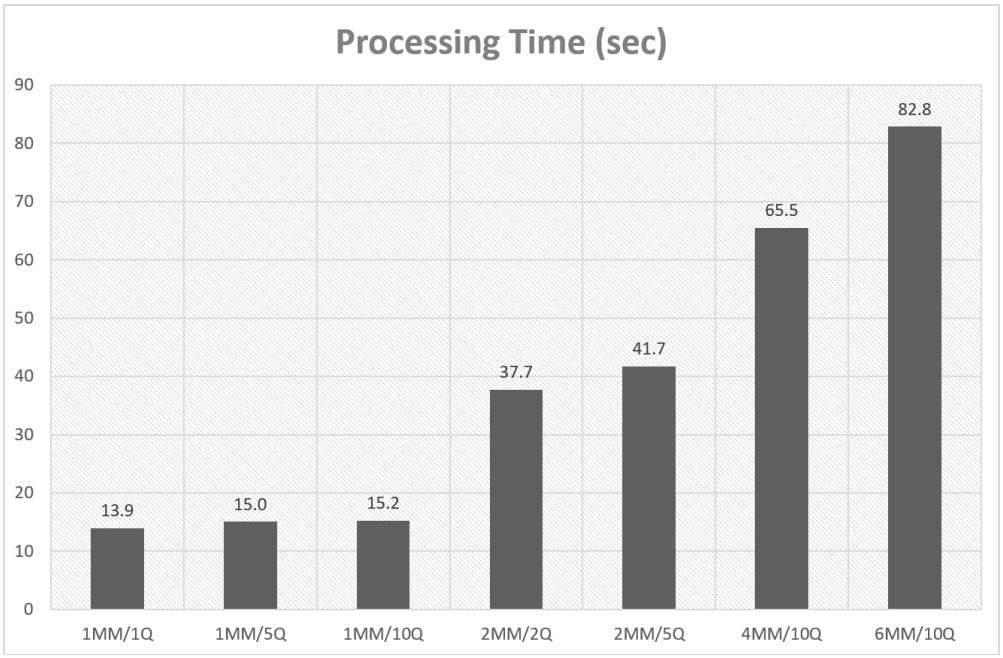


Figure 13.1.: Time consumption of the resource discovery when using one, two, four, and six matchers with varied number of requests.

ing. In the future work, we might need to consider a hybrid approach with a compromised quality of results. Nevertheless, the current approach is ready to be deployed within the context of MERCURY.

13.3 Resource Discovery in MERCURY

For a modularity and reusability required by Requirement R11, the completed resource discovery was implemented as a standalone web service. According to a workflow in MERCURY in Figure 13.2, there are three use cases of resource discovery, i.e. the registration, the scenario modeling, and the scenario execution processes. Hence, a GUI should be designed to assist a user as required by Requirement R12 (*the discovery result should be presented in the registration, scenario modeling, and execution processes in a way that users can apply the result instantly*).

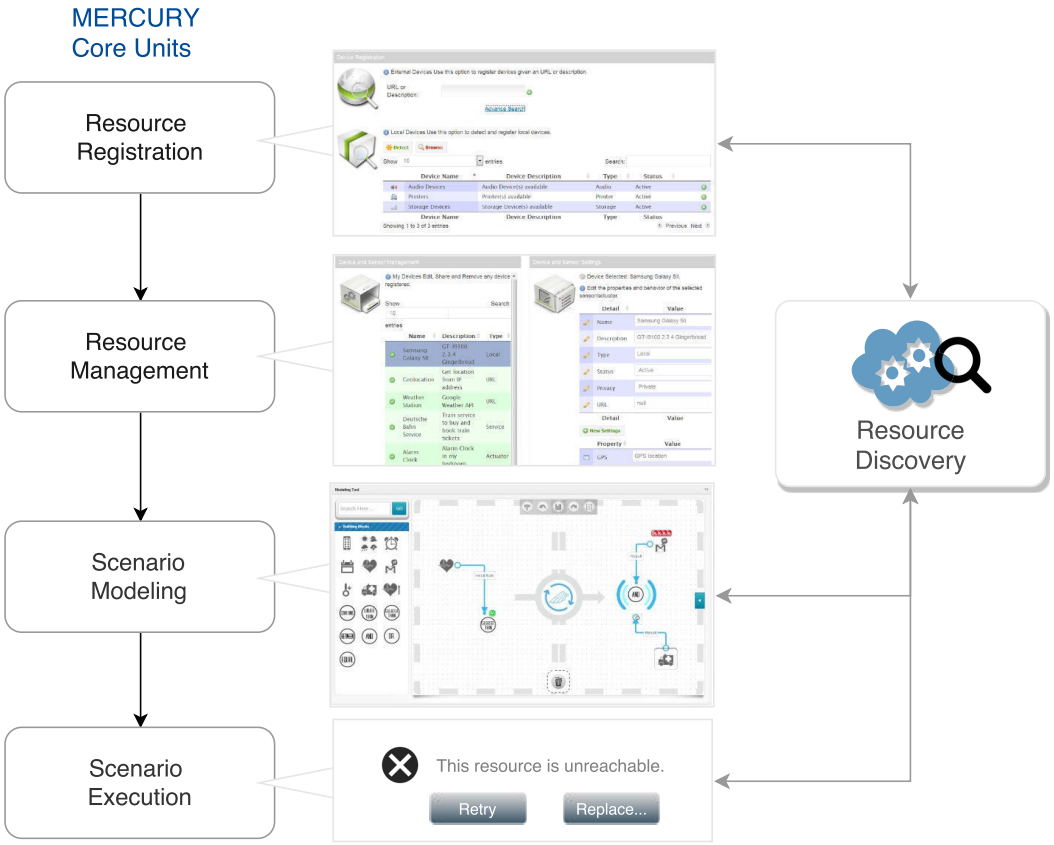


Figure 13.2.: Summary of the resource discovery roles in MERCURY.

This evaluation presents two graphical interfaces, the resource discovery in registration and scenario modeling processes. Since the execution GUI is still under development, this part of the evaluation will be enlisted in the future work. All units are deployed as individual portlets on Liferay Portal. While the resource discovery unit runs as a Java-based web application (tested on Apache Tomcat 7 and 8.5).

13.3.1. Resource Discovery in Registration

In the registration GUI, users can:

a. view local resources,

b. directly register a resource using its URI, or

c. search for a resource by free text keywords.

By default, a basic search mode is presented as depicted in Figure 13.3. The return result, which is a list of resource IDs, is presented to a user intuitively and waits to be added further.

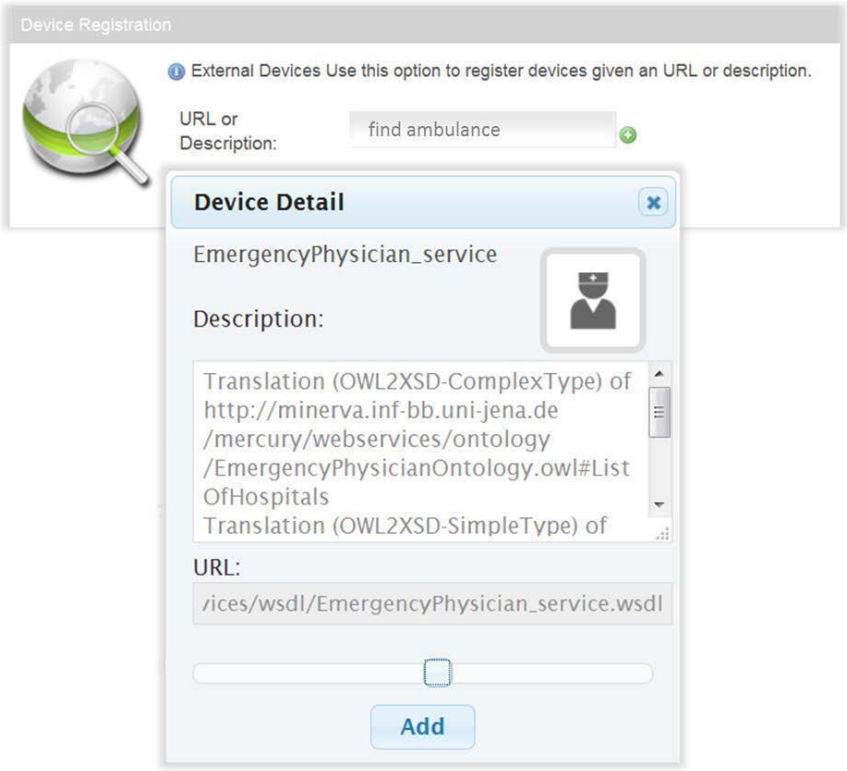


Figure 13.3.: Screenshot of the resource recommendation in MERCURY's registration process.

Additionally, an advanced search is accessible under an input box, as depicted in Figure 13.4. By providing separate keywords, a user can get more accurate search results. At this step, the user's current location can be implicitly appended to the query as explained in Section 6.1, *Context from User Profile*, and Section 6.2, *Context from Social Sensors*. With the same concept, other contexts can be applied in the same fashion. However, such an automatic context retrieval should be transparent and approved by the owner of the context.

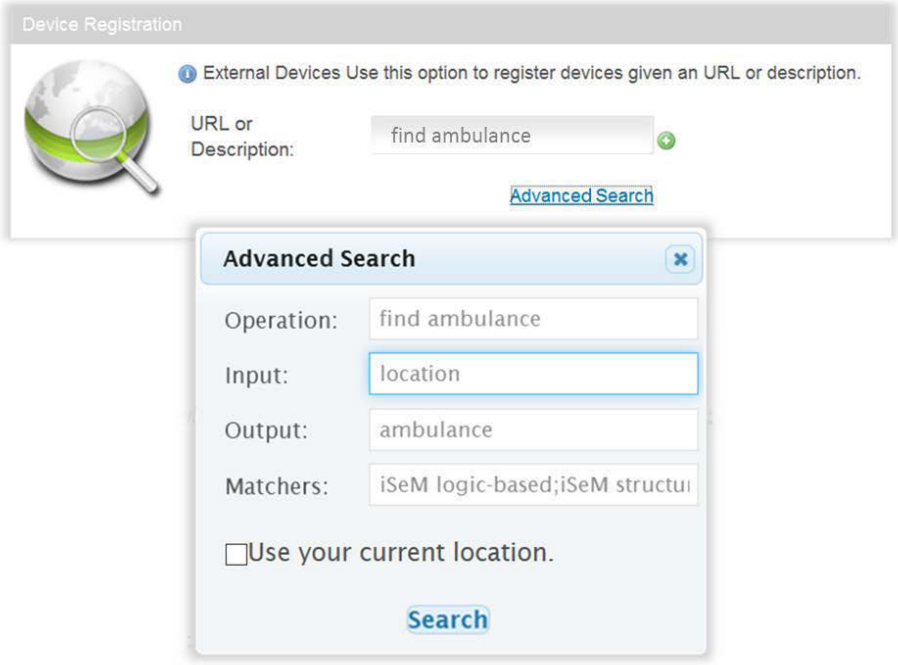


Figure 13.4.: Screenshot of the advanced search in MERCURY's registration process.

13.3.2. Resource Discovery in Scenario Modeling

To help a user find resources in the scenario modeling, more context should be involved. Previously, we considered mostly static context. Here, with user interactions, we can utilize more of dynamic context. In the current stage, the context presented in Section 6.3, *Context from User Preferences and Contributions*, is used to determine the most frequently used item and the item that has been used together in similar scenarios (see Figure 13.5(a) and (d)). These recommendations can be fine-tuned by using the global preferences (recommendations from other users) or personal preferences (from the current user).

Moreover, the semantic descriptions are also used to determine the semantically similar item, and the input/output ontology will be used to find compatible items as exemplified in Figure 13.5 (b) and (c). These recommendations rely on users' interactions and how well resources are described. In a cold start, we can utilize the user's location information to retrieve a list of nearby resources.

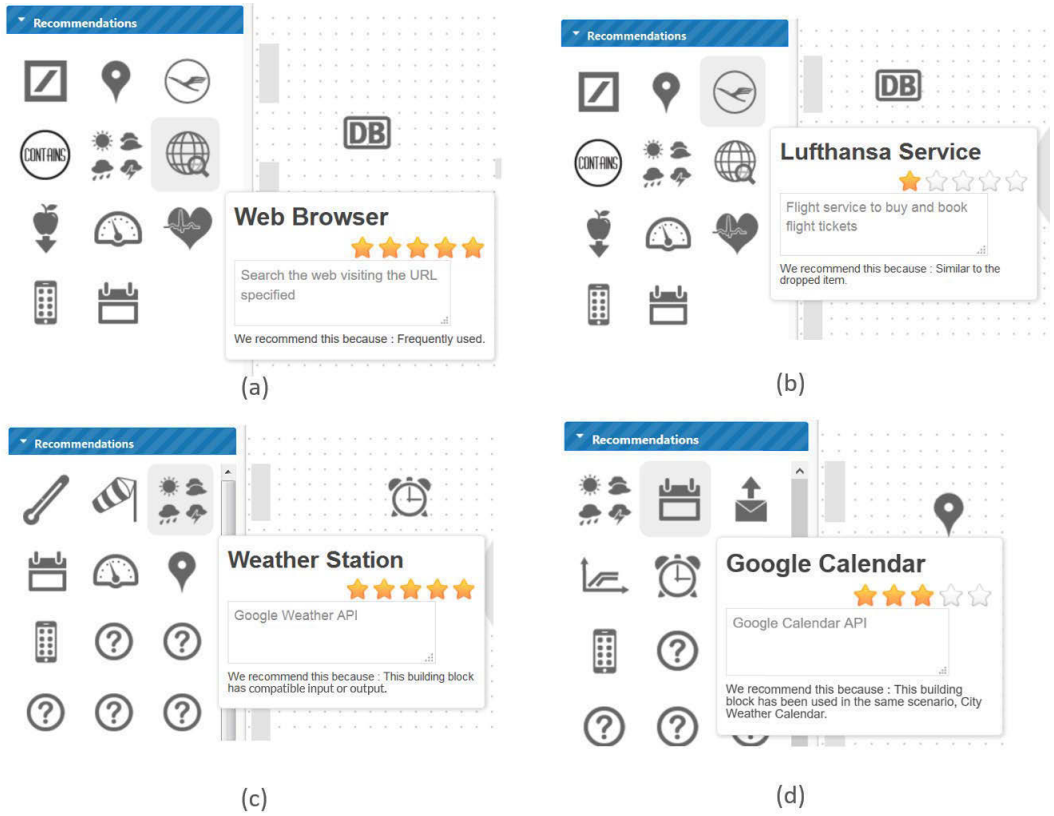


Figure 13.5.: Screenshot of the resource recommendation in MERCURY's scenario modeling process with reasons: (a) this item is frequently used,
(b) is semantically equivalent to the selected item,
(c) has compatible input/output to the selected item, and
(d) is previously used in the similar scenario.

Part IV.

Conclusion

14 Summary

This thesis is initiated as a part of a project called MERCURY which is motivated by the urge of using IoT in everyday life. We implemented the core components as portlets since we envision the multi-purpose advantage of the portal-based application. Nevertheless, some components are also available as standalone services, such as the scenario modeling, the execution engine, and the resource discovery. We have reviewed the existing and related works in Chapter 4, *State of the Art*. The studies show that even though there are several attempts to realize the same goal as MERCURY, there is still a lack of either a user-friendly UI or a support of resource discovery. Therefore, we developed a context-adaptive resource discovery to fill in the gap.

This thesis focuses on the implementation and evaluation of the resource discovery, especially in the dynamic environment, where resources are described in arbitrary formalisms. Moreover, the availability of semantic annotations within a resource description allows us to enhance the discovery process. Additionally, due to the abundance of smart sensors we carry around these days, user context derived from them becomes more up-to-date and thus more useful.

In this chapter, we summarize all main components in the resource discovery and the requirements they have achieved. Finally, we orchestrate all components into a standalone service and integrate them into MERCURY.

14.1 Resource Discovery Main Components

We present the main components developed in this thesis - the context extractor, the request constructor, the request converter, and the result integrator - in Part II. According to the proposed architecture in Chapter 5, *Solution Overview*, all the components are assembled and identified with requirements they have achieved as shown in Figure 14.1.

14.1.1. Context Extractor

To resolve Requirement **R6** (*the source of user context should be defined*), we define the source of context in this work and how we retrieve it. We utilize the context from Section 6.1, *Context from User Profile*, and 6.2, *Context from Social Sensors*. These contexts

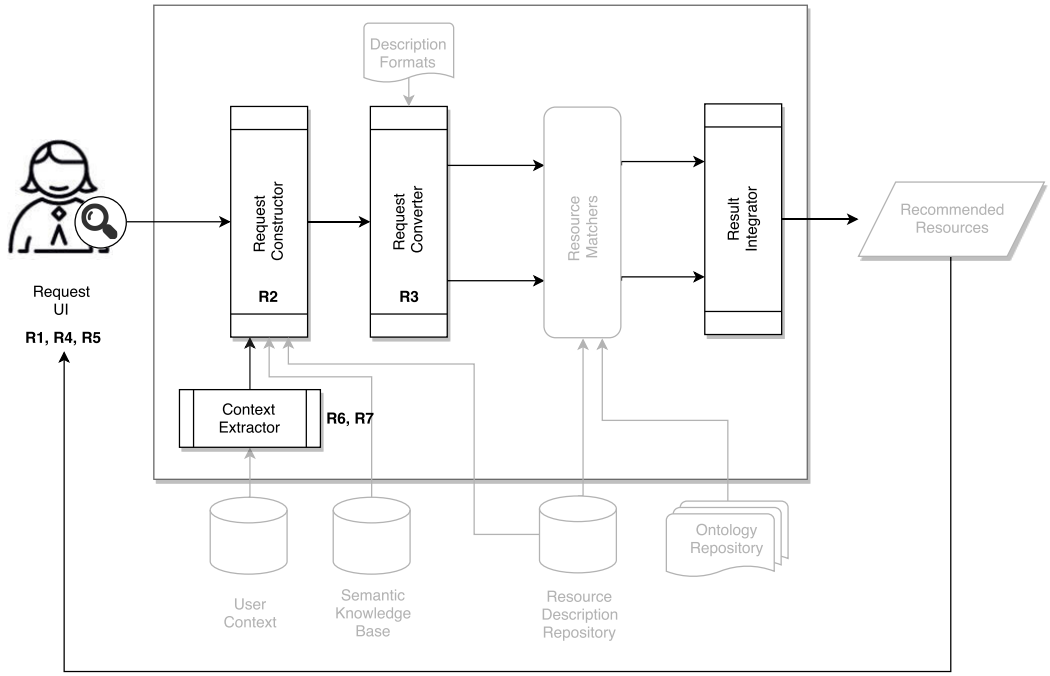


Figure 14.1.: Implemented parts of the resource discovery.

are applied to the resource discovery process while preparing the request in the resource registering, scenario modeling and scenario executing processes in MERCURY. Thus, Requirement **R7** (*when user context or resource context is available, they should be applied to the search query*) is resolved. The context from 6.3, *Context from User Preferences and Contributions*, is also applied to the scenario modeling process.

14.1.2. Request Analysis

To create a usable description, the necessary information for each formalism has been defined in Chapter 7, *Request Analysis*. We studied the OWL-S and SAWSDL based matchers to find a pattern they have in common for resource matching. We deduced the essential information, which are; input, output, and operation descriptions of each resource, and applied this knowledge to the construction and conversion of a request in Chapter 8, *Request Preparation*. This analysis also realizes Requirement **R2** (*the resource discovery should be able to interpret different description formalisms*).

14.1.3. Request Constructor

The request constructor is implemented to create a formatted description from the free-text keyword(s), which is consumable by the matchers. Thus, Requirement **R1** (*the resource*

discovery unit should be able to construct a free-text query message from end users into pre-defined formats) is fulfilled here. We offer a basic search (Requirement **R4**) and an advanced search (Requirement **R5**) to users.

The basic search looks for the given term in every part of descriptions, regardless of the objective of that part. The advanced search, on the other hand, separates the keyword into three categories; input, output, and operation descriptions. This specific search also allows us to utilize the semantic annotation of keywords and enhance the search result.

The evaluation outcome from the request constructor is very satisfactory. This result is crucial since the output from this step is used further in the rest of the work. Also, the concept of structured keywords is presented as an advanced search.

It is worth noting that when a user inputs simple keywords, the semantic extension can improve the quality of search. In contrast, using structured keywords obtains the best recall, precision, F-measure, and nDCG when no semantic terms are available. This is because the advanced search is designed to explore in the particular fields. Adding more semantic keywords can widen the scope of the search result, thus lessening the accuracy. Therefore, a semantic query expansion is applied according to the type of input keywords.

14.1.4. Request Converter

At this point, we can create a minimal description, either in OWL-S or SAWSDL formalism. We need the request converter to create a description in another formalism from the mapping model presented in Section 7.2, *Essential information required for resource matching*, to serve Requirement **R3** (*the resource discovery should handle multiple description formalisms and matchers simultaneously*).

As of now, our solution supports the conversion from OWL-S/OWL to SAWSDL/WSDL1.1, SAWSDL/WSDL1.1 to OWL-S/OWL and SAWSDL/WSDL2.0 to OWL-S/OWL. We implemented the OWL-S/OWL to SAWSDL/WSDL2.0 conversion, but it has not yet been evaluated because the resource matchers used in this research do not support WSDL2.0.

For the mode of conversion, an offline mode requires the conversion when a new description is added to the description repository. On the other hand, an online mode needs the conversion every time a request is made.

The converted descriptions yield slightly worse results than the corresponding descriptions. We conclude that the quality of results drops because the original descriptions are not equivalent to the corresponding descriptions and there could be some information loss during the conversion. Moreover, according to the evaluation results of the resource matchers, OWL-S matchers perform better than SAWSDL matchers. Thus, this indicates that OWL-S descriptions averagely yield better results. Nonetheless, OWL-S based matchers usually consume a lot more processing time and power than SAWSDL based matchers.

14.1.5. Resource Matching and Result Integration

As demonstrated in Section 7.1, *Matcher Analysis*, we learned that no single matcher could perform best in every circumstance. To maximize the quality of results, we decided to utilize multiple matchers. Hence, we need a result integrator to consolidate the different results from different matchers.

The more matchers involved in the discovery, the higher the quality of results would be. However, the higher accuracy of the final result is a trade-off with the time performance. Therefore, an optimum number of matchers must be determined. The evaluation results show that the proposed method can yield the best result from the best-performed matcher. Moreover, it drastically outperforms the average result from all matchers.

In Part III, we presented the evaluation method and results. The request constructor, the request converter, resource matchers and the result integrator were evaluated separately so that we can measure their qualities. Then we integrated them and evaluated to measure the overall performance to find the gap of improvement.

14.2 Integration with MERCURY

According to MERCURY architecture presented in Chapter 2, we have implemented most of it, as shown in Figure 14.2. The resource discovery operates as a standalone web service as required by Requirement **R11**. This enables the modularity and reusability of the search engine. When deploying the resource discovery to MERCURY, it can be easily applied to the resource registration, scenario modeling and scenario execution modules. Only the GUI designs are needed in the implementation of each part.

On top of the usage of the resource discovery service, the infrastructure of portal technology allows us to make use of the user and environmental context. This work aims to exploit three types of context, the spatial-based, ontology-based and object-oriented based context.

- The user management within the portal and the social sensors explained in Chapter 6, *Context Extraction*, provides spatial-based context. This spatial information can be appended to a query message in the request constructing process.
- The ontology-based context is applicable when the descriptions contain semantic annotation, which is used in the resource matching process.
- The object-oriented context can be retrieved from users' interactions within MERCURY. This is applied to the scenario modeling process. The recommendation is determined upon the resource that a user selected.

The resource discovery can be used within MERCURY in the following processes;

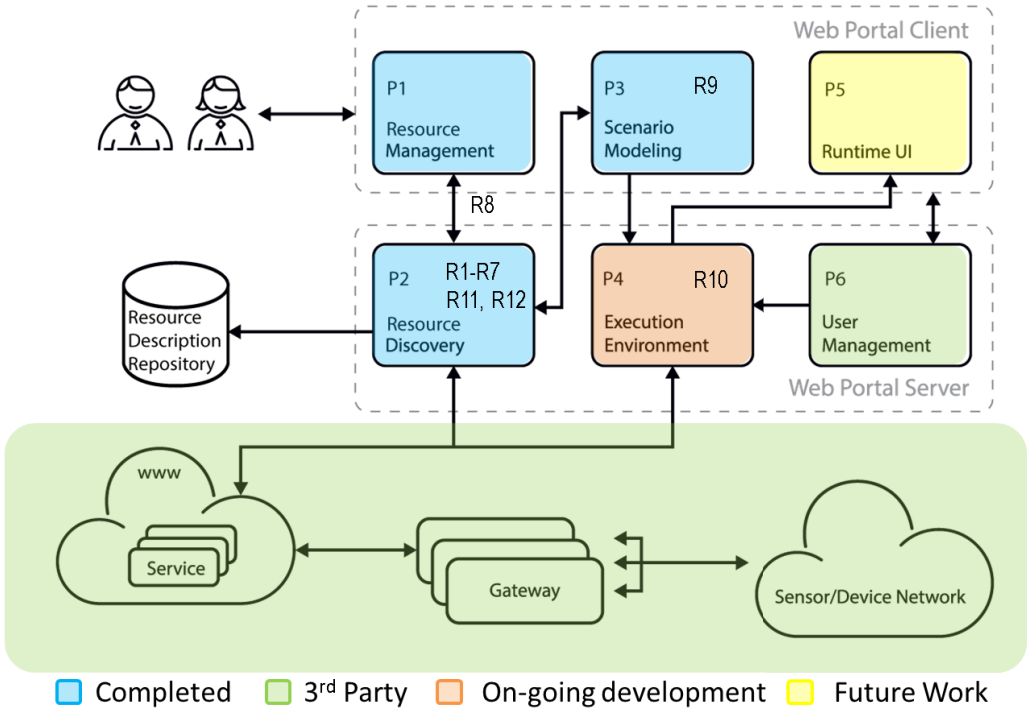


Figure 14.2.: Summary of implemented parts in MERCURY.

- The registration of resources (resolving Requirement **R8**)
- The scenario modeling from registered resources (resolving Requirement **R9**)
- The scenario execution (resolving Requirement **R10**).

The implicit context can be derived from the Portal's user management model, social sensors or the history of usage. Such information must be approved by a user before it can be applied. The explicit context can also be provided as free-text keywords.

Finally, the result of the resource discovery is presented as a resource recommendation throughout the MERCURY workflow and thus fulfills Requirement **R12**. All requirements are summarized again in Table 14.1.

Requirement	Description	Status
P1	We need a resource registration and management module.	Completed
P2	We need a resource discovery module.	Completed
R1	The resource discovery unit should be able to construct a free-text query message from end users into pre-defined formats.	Completed
R2	The resource discovery should be able to interpret different description formalisms.	Completed
R3	The resource discovery should handle multiple description formalisms and matchers simultaneously.	Completed
R4	The discovery process should offer a basic search for resource descriptions.	Completed
R5	The discovery process should offer an advanced search considering semantic annotations and syntax of keywords.	Completed
R6	The source of user context should be defined.	Completed
R7	When user context or resource context is available, they should be applied to the search query.	Completed
R8	The resources' descriptions should be derived, stored, and made editable by authorized users.	Completed
R11	The resource discovery should operate as a standalone module.	Completed
R12	The discovery result should be presented in the registration, scenario modeling, and execution processes in a way that users can apply the result instantly.	Completed
P3	We need a scenario modeling interface and model translator.	Completed
R9	The modeling tool should recommend the potential resources to users by considering users' interactions.	Completed
P4	We need an execution environment.	On-going
R10	During runtime, a placeholder item or a fail-to-respond resource should be supported by the service discovery.	Completed
P5	We need a runtime UI.	Future Work
P6	We need a user management module and need to extract user context from it.	Completed

Table 14.1.: Summary of requirements and their statuses.

15
Future Plan

In this thesis, we use two major description formalisms, OWL-S and SAWSDL as a proof of concept. There are more formalisms, such as WADL (Web Application Description Language), WSMO (Web Service Modeling Ontology) or tagged-based formalisms, like [GCPG12] and [DLY+10], which can be supported by the resource discovery. Also, due to the emergence of Big data and cloud services, structural descriptions in JSON and XML formats are being considered to be included in our future work.

Although the context extraction is defined and implemented in this thesis, the evaluation of quality improvement has not been conducted yet. The extracted context can be evaluated using the outcome of this thesis as a baseline. Furthermore, we can enhance the quality of the discovery result by coping with the dynamic context detected by sensors. For instance, a GPS sensor can be used instead of the location from the user model for a more precise result. The context of social networking other than Facebook and Twitter can also be used. Currently, we deal with only explicit context, e.g. tagged location, friends, and timestamps of each user's status. By utilizing a deep learning technology, we can make the sentimental analysis from the textual content of the user's status/message. Additionally, we could analyze the attached images to obtain context information.

The evaluation part of the resource discovery in the execution process is still pending for the completion of the execution engine, but it could be evaluated shortly. So far, the resources we have connected to MERCURY are mainly web services and Arduino[1] sensors. For the scalability issue, a sensor gateway or generic middleware could be a solution for MERCURY.

One important issue of the current approach is the time performance. Although our proposed technique provides a satisfactory quality of resource discovery, the processing time is not yet practical for a real-time recommendation. Each resource matcher can be adjusted to perform faster, but this has to be done manually. Therefore, we are still looking for a more sustainable solution in order to improve the time performance.

We have a vision that the developed concept of resource discovery is not limited to MERCURY, but can demonstrate an enhancement of IoT domain applications. Each developed

[1] https://www.arduino.cc/

module can be reused separately in any other application involving ontology matching, search result merging, sensors/actuators discovery, resource recommendation, resource description conversion and users' context derivation via IoT.

In the big picture, MERCURY, assisted with the resource discovery, can become a powerful tool to maintain the resiliency of an execution of scenario, to ease the process of a scenario modeling, and to be adaptive to users' environment beyond the heterogeneous description formalisms.

From a business point of view, companies can utilize MERCURY to attract their customers, and thus increase their revenues. For individuals, MERCURY can offer an easy way to create an application to improve one's lifestyle. And finally, this platform offers a collaborative sensors and services development where everyone can contribute to.

List of scientific publications

1. Kobkaew Opasjumruskit, Birgitta König-Ries, and Jesús Expósito. Dynamic Strategies for Query Constructing and Rank Merging from Multiple Search Engines. In *Service Oriented and Cloud Computing*, volume 9306 of *LNCS*, chapter 10. Springer, 2015

2. Kobkaew Opasjumruskit, Jesús Expósito, Birgitta König-Ries, Andreas Nauerz, and Martin Welsch. Service Discovery with Personal Awareness in Smart Environments. In *Creating Personal, Social, and Urban Awareness through Pervasive Computing.*, pages 86–107. IGI Global, 2014

3. Kobkaew Opasjumruskit, Jesús Expósito, Birgitta König-Ries, Andreas Nauerz, and Martin Welsch. MERCURY: User Centric Device and Service Processing : Demo paper. In *19th International In workshop on Personalization and Recommendation on the Web and Beyond held at Mensch and Computer*, Konstanz, Germany, sep 2012

4. Birgitta König-Ries, Kobkaew Opasjumruskit, Andreas Nauerz, and Martin Welsch. MERCURY : User Centric Device & Service Processing. In *MKWI 2012*, Braunschweig, Germany, February 2012

5. Kobkaew Opasjumruskit. Towards Leveraging Semantic Web Service Technology for Personalized, Adaptive Automatic Ubiquitous Sensors Discovery in Context of the Internet of Things. *University Halle-Wittenberg Institute of Computer Science*, page 48

References

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[Ard] Arduino. `https://www.arduino.cc/`. Accessed: 2017-08-06.

[ASS+11] Raian Ali, Carlos Solis, Mazeiar Salehie, Inah Omoronyia, Bashar Nuseibeh, and Walid Maalej. Social Sensing: When Users Become Monitors. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 476–479, New York, NY, USA, 2011. ACM.

[ATdMH15] Leonardo Albernaz Amaral, Ramão Tiago Tiburski, Everton de Matos, and Fabiano Hessel. Cooperative Middleware Platform As a Service for Internet of Things Applications. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, pages 488–493, New York, NY, USA, 2015. ACM.

[BBH+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A Survey of Context Modelling and Reasoning Techniques. *Pervasive Mobile Computing*, 6(2):161–180, April 2010.

[BBM] Martin Bauer, Mathieu Boussard, and Stefan Meissner. *Interactions*.

[BDH+09] John G. Breslin, Stefan Decker, Manfred Hauswirth, Gearoid Hynes, Danh Le Phuoc, Alexandre Passant, Axel Polleres, Cornelius Rabsch, and Vinny Reynolds. Integrating Social Networks and Sensor Networks. In *Proceedings on the W3C Workshop on the Future of Social Networking*, 2009.

[BMR07] Claudio Bettini, Dario Maggiorini, and Daniele Riboni. Distributed Context Monitoring for the Adaptation of Continuous Services. *World Wide Web*, 10(4):503–528, 2007.

[BPGO13] Talal Ashraf Butt, Iain Phillips, Lin Guan, and George Oikonomou. Adaptive and Context-Aware Service Discovery for the Internet of Things. In Sergey Balandin, Sergey Andreev, and Yevgeni Koucheryavy, editors, *Internet of Things, Smart Spaces, and Next Generation Networking*, volume 8121 of *Lecture Notes in Computer Science*, pages 36–47. Springer Berlin Heidelberg, 2013.

[BS16] P. Barnaghi and A. Sheth. On Searching the Internet of Things: Requirements and Challenges. *IEEE Intelligent Systems*, 31(6):71–75, Nov 2016.

[CCMW] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL). `http://www.w3.org/TR/wsdl`. Accessed: 2017-08-05.

[Dey01] Anind K. Dey. Understanding and Using Context. *Personal Ubiquitous Computing*, 5(1):4–7, January 2001.

[DLY+10] Zhaoyun Ding, Deng Lei, Jia Yan, Zhou Bin, and An Lun. A Web Service Discovery Method Based on Tag. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, pages 404–408, Feb 2010.

[DMD+03] AnHai Doan, Jayant Madhavan, Robin Dhamankar, Pedro Domingos, and Alon Halevy. Learning to Match Ontologies on the Semantic Web. *The VLDB Journal*, 12(4):303–319, November 2003.

[DRGS09] David De Roure, Carole Goble, and Robert Stevens. The Design and Realisation of the myExperiment Virtual Research Environment for Social Sharing of Workflows. *Future Generation Computer Systems*, 25(5):561–567, May 2009.

[Ela] Elasticsearch: Search & Analyze Data in Real Time. `https://www.elastic.co/products/elasticsearch`. Accessed: 2017-08-05.

[EPC] EPCglobal | GS1. `https://www.gs1.org/epcglobal`. Accessed: 2017-08-06.

[Evr] EVRYTHNG IoT Platform for Smart Consumer Products. `http://www.evrythng.com`. Accessed: 2017-07-31.

[Fac] Graph API : facebook for developers. `https://developers.facebook.com/docs/graph-api`. Accessed: 2017-08-05.

[FFST11] Dieter Fensel, FedericoMichele Facca, Elena Simperl, and Ioan Toma. Lightweight Semantic Web Service Descriptions. In *Semantic Web Services*, pages 279–295. Springer Berlin Heidelberg, 2011.

[GCPG12] Maciej Gawinecki, Giacomo Cabri, Marcin Paprzycki, and Maria Ganzha. Evaluation of Structured Collaborative Tagging for Web Service Matchmaking. In Brian Blake, Liliana Cabral, Birgitta König-Ries, Ulrich Küster, and David Martin, editors, *Semantic Web Services*, pages 173–189. Springer Berlin Heidelberg, 2012.

[GRRC12] José María García, David Ruiz, and Antonio Ruiz-Cortés. Improving semantic web services discovery using sparql-based repository filtering. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:12–24, Dec 2012.

[GTW10] Dominique Guinard, Vlad Trifa, and Erik Wilde. Architecting a Mashable Open World Wide Web of Things. Technical report, Institute for Pervasive Computing ETH Zurich, SAP Research CEC Zurich, School of Information UC Berkeley, 2010.

[Gui09] Dominique Guinard. Towards the web of things: Web mashups for embedded devices. In *In MEM 2009 in Proceedings of WWW 2009. ACM*, 2009.

[GZI11] Bin Guo, Daqing Zhang, and Michita Imai. Toward a Cooperative Programming Framework for Context-aware Applications. *Personal Ubiquitous Computing*, 15(3):221–233, March 2011.

[Har12] A. & Maynard D. Harth. Semantic Web Challenge. `http://challenge.semanticweb.org`, 2012. Accessed: 2017-08-05.

[HLI04] K. Henricksen, S. Livingstone, and J. Indulska. Towards a hybrid approach to context modeling, reasoning and interoperation. In J. Indulska and D. De Roure, editors, *Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 54–61. The University of Southampton, September 2004. ISBN: 85432 813 0.

[HM10] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. The Morgan Kaufmann Series in Data Management Systems. Elsevier Science, 2010.

[HPSB+04] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML, 2004.

[HPSVH03] Ian Horrocks, Peter F Patel-Schneider, and Frank Van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1):7–26, 2003.

[HS12] G.C. Hobold and F. Siqueira. Discovery of Semantic Web Services Compositions Based on SAWSDL Annotations. In *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pages 280–287, June 2012.

[IFT] IFTTT - Make your work flow. `https://ifttt.com`. Accessed: 2017-07-31.

[Jad12] Hossein Jadidoleslamy. Search Result Merging and Ranking Strategies in Meta-Search Engines: A Survey. *International Journal of Computer Science Issues*, 2012.

[KB08] Christoph Kiefer and Abraham Bernstein. The Creation and Evaluation of iSPARQL Strategies for Matchmaking. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications*, volume 5021 of *Lecture Notes in Computer Science*, pages 463–477. Springer Berlin Heidelberg, 2008.

[KBB+09] S. Kona, A. Bansal, M. B. Blake, S. Bleul, and T. Weise. WSC-2009: A Quality of Service-Oriented Web Services Challenge. In *2009 IEEE Conference on Commerce and Enterprise Computing*, pages 487–490, July 2009.

[KFK05] Matthias Klusch, Benedikt Fries, and Mahboob Khalid. OWLS-MX: Hybrid owl-s service matchmaking. In *Proceedings of 1st International AAAI Fall Symposium on Agents and the Semantic Web*, 2005.

[KK10] M. Klusch and P. Kapahnke. isem: Approximated reasoning for adaptive hybrid selection of semantic services. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pages 184–191, Sept 2010.

[KK12a] Matthias Klusch and Patrick Kapahnke. Adaptive signature-based semantic selection of services with OWLS-MX3. *Multiagent and Grid Systems*, 8(1):69–82, 2012.

[KK12b] Matthias Klusch and Patrick Kapahnke. The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection . *Web Semantics: Science, Services and Agents on the World Wide Web*, 15:1 – 14, 2012.

[KKF08] M. Klusch, P. Kapahnke, and B. Fries. Hybrid Semantic Web Service Retrieval: A Case Study with OWLS-MX. In *Semantic Computing, 2008 IEEE International Conference on*, pages 323–330, Aug 2008.

[KKZ09a] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer. In *Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications*, ESWC 2009 Heraklion, pages 550–564, Berlin, Heidelberg, 2009. Springer-Verlag.

[KKZ09b] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. SAWSDL-MX2: A Machine-Learning Approach for Integrating Semantic Web Service Matchmaking Variants. In Ernesto Damiani, Rong Chang, and Jia Zhang, editors, *2009 IEEE International Conference on Web Services. IEEE International Conference on Web Services (ICWS-2009), 7th, July 6-10, Los Angeles,, CA, USA*, pages 335–342. IEEE Press, 2009.

[KLD12] M. Kovatsch, M. Lanter, and S. Duquennoy. Actinium: A RESTful runtime container for scriptable Internet of Things applications. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 135–142, Oct 2012.

[Klu12] M. Klusch. Semantic Service Selection (S3) contest. `http://www-ags.dfki.uni-sb.de/~klusch/s3/index.html`, 2012. Accessed: 2017-08-05.

[KNL13] Mohammad Mehdi Keikha, Mohammad Ali Nematbakhsh, and Behrouz Tork Ladani. Structural Weights in Ontology Matching. *CoRR*, abs/1311.3800, 2013.

[KNLZ07] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. SenseWeb: An Infrastructure for Shared Sensing. *IEEE MultiMedia*, 14(4):8–13, October 2007.

[KRONW12] Birgitta König-Ries, Kobkaew Opasjumruskit, Andreas Nauerz, and Martin Welsch. MERCURY : User Centric Device & Service Processing. In *MKWI 2012*, Braunschweig, Germany, February 2012.

[LD13] Richard K. Lomotey and Ralph Deters. CSB-UCC: Cloud Services Brokerage for Ubiquitous Cloud Computing. In *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*, MEDES '13, pages 100–107, New York, NY, USA, 2013. ACM.

[LdMT+15] W. T. Lunardi, E. de Matos, R. Tiburski, L. A. Amaral, S. Marczak, and F. Hessel. Context-based search engine for industrial IoT: Discovery, search, selection, and usage of devices. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sept 2015.

[Li13] Jing Li. A Fast Semantic Web Services Matchmaker for OWL-S Services. *Journal of Networks*, 8(5), 2013.

[LMS+05] Yiyao Lu, Weiyi Meng, Liangcai Shu, Clement Yu, and King-Lup Liu. Evaluation of Result Merging Strategies for Metasearch Engines. In AnneH.H. Ngu, Masaru Kitsuregawa, ErichJ. Neuhold, Jen-Yao Chung, and QuanZ. Sheng, editors, *Web Information Systems Engineering WISE 2005*, volume 3806 of *Lecture Notes in Computer Science*, pages 53–66. Springer Berlin Heidelberg, 2005.

[LpH09] Danh Le-phuoc and Manfred Hauswirth. Linked open data in sensor data mashups, 2009.

[LpNQP11] Danh Le-phuoc, Hoan Nguyen, Mau Quoc, and Josiane Xavier Parreira. The Linked Sensor Middleware - Connecting the real world and the Semantic Web, 2011.

[MB10] Georgios Meditskos and N. Bassiliades. Structural and Role-Oriented Web Service Discovery with Taxonomies in OWL-S. *Knowledge and Data Engineering, IEEE Transactions on*, 22(2):278–290, Feb 2010.

[MDRS05] Ian Millard, David De Roure, and Nigel Shadbolt. *Contextually Aware Information Delivery in Pervasive Computing Environments*, pages 189–197. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[MF10] Friedemann Mattern and Christian Floerkemeier. From the Internet of Computers to the Internet of Things. In Kai Sachs, Ilia Petrov, and Pablo Guerrero, editors, *From Active Data Management to Event-based Systems and More*, chapter From the Internet of Computers to the Internet of Things, pages 242–259. Springer-Verlag, Berlin, Heidelberg, 2010.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema

Matching. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 117–, Washington, DC, USA, 2002. IEEE Computer Society.

[MHB+12] N. Masuch, B. Hirsch, M. Burkhardt, A. Heßler, and S. Albayrak. SeMa2: A Hybrid Semantic Service Matching Approach. In Brian Blake, Liliana Cabral, Birgitta König-Ries, Ulrich Küster, and David Martin, editors, *Semantic Web Services*, pages 35–47. Springer Berlin Heidelberg, 2012.

[MKP09] Maria Maleshkova, Jacek Kopecký, and Carlos Pedrinaci. Adapting SAWSDL for Semantic Annotations of RESTful Services. In *Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK 2009*, OTM '09, pages 917–926, Berlin, Heidelberg, 2009. Springer-Verlag.

[MPM+05] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In Jorge Cardoso and Amit Sheth, editors, *Semantic Web Services and Web Process Composition*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. Springer Berlin Heidelberg, 2005.

[MPW07] David Martin, Massimo Paolucci, and Matthias Wagner. Bringing Semantic Annotations to Web Services: OWL-S from the SAWSDL Perspective. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 340–352, Berlin, Heidelberg, 2007. Springer-Verlag.

[NGS+09] Meenakshi Nagarajan, Karthik Gomadam, Amit P. Sheth, Ajith Ranabahu, Raghava Mutharaju, and Ashutosh Jadhav. *Spatio-Temporal-Thematic Analysis of Citizen Sensor Data: Challenges and Experiences*, pages 539–553. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Nin] Ninja Blocks. `https://ninjablocks.com/`. Accessed: 2017-08-06.

[NKK10] Le Duy Ngan, M. Kirchberg, and R. Kanagasabai. Review of Semantic Web Service Discovery Methods. In *Services (SERVICES-1), 2010 6th World Congress on*, pages 176–177, July 2010.

[NLZ07] Suman Nath, Jie Liu, and Feng Zhao. SensorMap for Wide-Area Sensor Webs. *IEEE Computer*, 40:90–93, January 2007.

[NM01] Daniela Nicklas and Bernhard Mitschang. The NEXUS Augmented World Model: An Extensible Approach for Mobile, Spatially Aware Applications. In *OOIS*, 2001.

[OEKR+12] Kobkaew Opasjumruskit, Jesús Expósito, Birgitta König-Ries, Andreas Nauerz, and Martin Welsch. MERCURY: User Centric Device and Service Processing : Demo paper. In *19th International In workshop on Personalization and Recommendation on the Web and Beyond held at Mensch and Computer*, Konstanz, Germany, sep 2012.

[OEKR+14] Kobkaew Opasjumruskit, Jesús Expósito, Birgitta König-Ries, Andreas Nauerz, and Martin Welsch. Service Discovery with Personal Awareness in Smart Environments. In *Creating Personal, Social, and Urban Awareness through Pervasive Computing.*, pages 86–107. IGI Global, 2014.

[OKRE15] Kobkaew Opasjumruskit, Birgitta König-Ries, and Jesús Expósito. Dynamic Strategies for Query Constructing and Rank Merging from Multiple Search Engines. In *Service Oriented and Cloud Computing*, volume 9306 of *LNCS*, chapter 10. Springer, 2015.

[Opa] Kobkaew Opasjumruskit. Towards Leveraging Semantic Web Service Technology for Personalized, Adaptive Automatic Ubiquitous Sensors Discovery in Context of the Internet of Things. *University Halle-Wittenberg Institute of Computer Science*, page 48.

[Pac12] Eric Pacuit. Voting Methods. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Winter 2012 edition, 2012.

[Phu09] Danh Le Phuoc. SensorMasher - publishing and building mashup of sensor data. In *5th International Conference on Semantic Systems, Graz, Austria, September 2-4, 2009. Proceedings*, 2009.

[PP09] Pierluigi Plebani and Barbara Pernici. URBE: Web Service Retrieval Based on Similarity Evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1629–1642, 2009.

[PV16] Charith Perera and Athanasios V. Vasilakos. A Knowledge-based Resource Discovery for Internet of Things. *Knowledge-Based Systems*, 109(C):122–136, October 2016.

[PZC+12] Charith Perera, Arkady Zaslavsky, Peter Christen, Ali Salehi, and Dimitrios Georgakopoulos. Connecting Mobile Things to Global Sensor Network Middleware Using System-generated Wrappers. In *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '12, pages 23–30, New York, NY, USA, 2012. ACM.

[PZCG12] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. CA4IOT: Context Awareness for Internet of Things. In *Proceedings of the 2012 IEEE International Conference on Green Computing and Communications*, GREENCOM '12, pages 775–782, Washington, DC, USA, 2012. IEEE Computer Society.

[PZCG13] Charith Perera, Arkady B. Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Sensing as a Service Model for Smart Cities Supported by Internet of Things. *CoRR*, abs/1307.8198, 2013.

[PZCG14] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys Tutorials*, 16(1):414–454, First 2014.

[QHC06] Yuzhong Qu, Wei Hu, and Gong Cheng. Constructing Virtual Documents for Ontology Matching. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 23–31, New York, NY, USA, 2006. ACM.

[Ras] Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. `https://www.raspberrypi.org/`. Accessed: 2017-08-06.

[RKL+05] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, January 2005.

[RLS+11] Katharina Rasch, Fei Li, Sanjin Sehic, Rassul Ayani, and Schahram Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14(4):295–319, 2011.

[RMJPC16] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. Middleware for Internet of Things: A Survey. *IEEE Internet of Things Journal*, 3(1):70–95, Feb 2016.

[Rob01] Stephen E. Robertson. Evaluation in Information Retrieval. In *Proceedings of the Third European Summer-School on Lectures on Information Retrieval-Revised Lectures*, ESSIR '00, pages 81–92, London, UK, UK, 2001. Springer-Verlag.

[RS03] M. Elena Renda and Umberto Straccia. Web Metasearch: Rank vs. Score Based Rank Aggregation Methods. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, SAC '03, pages 841–846, New York, NY, USA, 2003. ACM.

[Sat14] Florian Sattler. Social media context analysis for automatic semantic service discovery. Bachelor thesis, Friedrich Schiller University of Jena, Department of Mathematics and Computer Science, September 2014.

[Sbo12] Marco Luca Sbodio. SPARQLent: A SPARQL Based Intelligent Agent Performing Service Matchmaking. In Brian Blake, Liliana Cabral, Birgitta König-Ries, Ulrich Küster, and David Martin, editors, *Semantic Web Services*, pages 83–105. Springer Berlin Heidelberg, 2012.

[SCI] SCIentific gateway Based User Support. http://www.sci-bus.eu/. Accessed: 2017-08-05.

[SGFW10] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé, editors. *Vision and Challenges for Realising the Internet of Things*. Publications Office of the European Union, Luxembourg, 2010.

[She07] Amit Sheth. Beyond SAWSDL: A Game Plan for Broader Adoption of Semantic Web Services, 2007.

[She09] A. Sheth. Citizen Sensing, Social Signals, and Enriching Human Experience. *IEEE Internet Computing*, 13(4):87–92, July 2009.

[SHI] SHIWA Simulation Platform. http://www.shiwa-workflow.eu. Accessed: 2017-07-28.

[Sho] Shodan: the world's first search engine for internet-connected devices. https://www.shodan.io/. Accessed: 2017-08-06.

[SKH+] John Soldatos, Nikos Kefalakis, Manfred Hauswirth, Martin Serrano, Jean-Paul Calbimonte, Mehdi Riahi, Karl Aberer, Prem Prakash Jayaraman, Arkady Zaslavsky, Ivana Podnar Žarko, Lea Skorin-Kapov, and Reinhard Herzog. *OpenIoT: Open Source Internet-of-Things in the Cloud, bookTitle=Interoperability and Open-Source Solutions for the Internet of Things: International Workshop, FP7 OpenIoT Project, Held in Conjunction with SoftCOM 2014, Split, Croatia, September 18, 2014, Invited Papers, year=2015, publisher=vSpringer International Publishing*, pages 13–25. Cham.

[SLES10] S. Schulte, U. Lampe, Julian Eckert, and R. Steinmetz. LOG4SWS.KOM: Self-Adapting Semantic Web Service Discovery for SAWSDL. In *Services (SERVICES-1), 2010 6th World Congress on*, pages 511–518, July 2010.

[SLKS12] Stefan Schulte, Ulrich Lampe, Matthias Klusch, and Ralf Steinmetz. COV4SWS.KOM: Information Quality-Aware Matchmaking for Semantic Services. In Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti, editors, *The Semantic Web: Research and Applications*, volume 7295 of *Lecture Notes in Computer Science*, pages 499–513. Springer Berlin Heidelberg, 2012.

[Sta] Statista - Number of IoT connected devices worldwide from 2015 to 2025. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. Accessed: 2017-07-31.

[TAB09] Hassina Nacer Talantikite, Djamil Aissani, and Nacer Boudjlida. Semantic annotations for web services discovery and composition. *Computer Standards & Interfaces*, 31(6):1108 – 1117, 2009.

[Thi] Thingful: A Search Engine for the Internet of Things. https://thingful.net/. Accessed: 2017-08-06.

[Twi] REST APIs | Twitter Developers. https://dev.twitter.com/rest/public. Accessed: 2017-08-05.

[WC16] E. Wang and R. Chow. What can I do here? IoT service discovery in smart cities. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6, March 2016.

[WJ12] Qiang Wei and Zhi Jin. Service Discovery for Internet of Things: a context-awareness perspective. In *Proceedings of the Fourth Asia-Pacific Symposium on Internetware, Internetware 2012, QingDao, China, October 30-31, 2012*, pages 25:1–25:6, 2012.

[WWWB11] Dengping Wei, Ting Wang, Ji Wang, and Abraham Bernstein. SAWSDL-iMatcher: A Customizable and Effective Semantic Web Service Matchmaker. *Web Semantics*, 9(4):402–417, December 2011.

[Xiv] IoT Platform for Connected Devices. https://www.xively.com. Accessed: 2017-07-31.

[XZNN10] Hua Xiao, Ying Zou, J. Ng, and L. Nigul. An Approach for Context-Aware Service Discovery and Recommendation. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 163–170, July 2010.

[ZPG13] Arkady B. Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. Sensing as a Service and Big Data. *CoRR*, abs/1301.0159, 2013.

Appendices

A

Appendix A

Listing A.1: Original SAWSDL.

```
<wsdl:definitions name="CityWeather"
targetNamespace="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather"
xmlns="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather-impl"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:intf="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
  <wsdl:types>
    <xsd:schema version="OWLS2WSDL Wed Sep 22 14:33:46 CEST 2010"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:annotation>
        <xsd:documentation source="Translation (OWL2XSD-ComplexType)
        of http://127.0.0.1/ontology/SUMO.owl#City"/>
        <xsd:documentation source="Translation (OWL2XSD-ComplexType)
        of http://127.0.0.1/ontology/Mid-level-ontology.owl#Weather"/>
      </xsd:annotation>
      <xsd:element name="Weather"
      sawsdl:liftingSchemaMapping="http://127.0.0.1/services/
          liftingSchemaMappings/
          city_weatherfront_service_Weather_liftingSchemaMapping.xslt"
          type="WeatherType"/>
      <xsd:element name="City"
      sawsdl:liftingSchemaMapping="http://127.0.0.1/services/
          liftingSchemaMappings/
          city_weatherfront_service_City_liftingSchemaMapping.xslt"  type=
          "CityType"/>
      <xsd:complexType name="TemperatureMeasure"
      sawsdl:modelReference="http://127.0.0.1/ontology/SUMO.owl#
          TemperatureMeasure">
        <xsd:sequence>
          <xsd:element name="lessThanOrEqualTo"
          type="xsd:anyURI"/>
          <xsd:element name="lessThan" type="xsd:anyURI"/>
          <xsd:element name="greaterThanOrEqualTo"
          type="xsd:anyURI"/>
          <xsd:element name="ReciprocalFn" type="xsd:anyURI"/>
          <xsd:element name="greaterThan" type="xsd:anyURI"/>
          <xsd:element name="RoundFn" type="xsd:anyURI"/>
          <xsd:element name="MagnitudeFn" type="xsd:anyURI"/>
        </xsd:sequence>
```

```
39        </xsd:complexType>
40        <xsd:complexType name="CityType"
41        sawsdl:modelReference="http://127.0.0.1/ontology/SUMO.owl#City">
42          <xsd:sequence>
43            <xsd:element name="capitalCity" type="GeopoliticalArea"/>
44            <xsd:element name="cityAddress" type="Address"/>
45            <xsd:element name="subRegion" type="Region"/>
46            <xsd:element name="climateTypeInArea" type="ClimateZone"/>
47            <xsd:element name="financialAsset" type="Object"/>
48            <xsd:element name="leader" type="Human"/>
49            <xsd:element name="economyType" type="EconomicAttribute"/>
50            <xsd:element name="editor" type="Text"/>
51            <xsd:element name="authors" type="Text"/>
52            <xsd:element name="WealthFn" type="CurrencyMeasure"/>
53            <xsd:element name="PropertyFn" type="Set"/>
54            <xsd:element name="ExecutiveBranchFn" type="Organization"/>
55            <xsd:element name="customer" type="Corporation"/>
56            <xsd:element name="governmentType" type="FormOfGovernment"/>
57            <xsd:element name="leaderPosition" type="Position"/>
58            <xsd:element name="fiscalYearPeriod" type="TimeInterval"/>
59            <xsd:element name="JudiciaryFn" type="JudicialOrganization"/>
60            <xsd:element name="dependentGeopoliticalArea" type="
                  GeopoliticalArea"/>
61            <xsd:element name="totalGDP" type="CurrencyMeasure"/>
62            <xsd:element name="CitizenryFn" type="GroupOfPeople"/>
63            <xsd:element name="GovernmentFn" type="Government"/>
64            <xsd:element name="currencyType" type="CurrencyMeasure"/>
65            <xsd:element name="legalSystemType" type="LegalSystemAttribute"
                  />
66            <xsd:element name="primaryGeopoliticalSubdivision" type="
                  GeopoliticalArea"/>
67            <xsd:element name="LegislatureFn" type="LegislativeOrganization"
                  />
68            <xsd:element name="industryOfArea" type="Physical"/>
69          </xsd:sequence>
70        </xsd:complexType>
71        <xsd:complexType name="WeatherType"
72        sawsdl:modelReference="http://127.0.0.1/ontology/Mid-level-ontology.
              owl#Weather">
73          <xsd:sequence>
74            <xsd:element name="subProcess" type="Process"/>
75            <xsd:element name="frequency" type="TimeDuration"/>
76            <xsd:element name="realization" type="Proposition"/>
77            <xsd:element name="result" type="Entity"/>
78            <xsd:element name="causesSubclass" type="Process"/>
79            <xsd:element name="inhibits" type="Process"/>
80            <xsd:element name="hasSkill" type="Agent"/>
81            <xsd:element name="precondition" type="Process"/>
82            <xsd:element name="causes" type="Process"/>
83            <xsd:element name="direction" type="DirectionalAttribute"/>
84            <xsd:element name="destination" type="Entity"/>
85            <xsd:element name="origin" type="Object"/>
86            <xsd:element name="targetInAttack" type="Object"/>
87            <xsd:element name="prevents" type="Process"/>
88            <xsd:element name="path" type="Object"/>
89          </xsd:sequence>
90        </xsd:complexType>
91      </xsd:schema>
92    </wsdl:types>
93    <wsdl:message name="get_WEATHERRequest">
94      <wsdl:part name="_CITY" type="CityType">
95      </wsdl:part>
```

```
  </wsdl:message>
  <wsdl:message name="get_WEATHERResponse">
    <wsdl:part name="_WEATHER" type="WeatherType">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="CityWeatherSoap">
    <wsdl:operation name="get_WEATHER">
      <wsdl:input message="get_WEATHERRequest">
      </wsdl:input>
      <wsdl:output message="get_WEATHERResponse">
      </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CityWeatherSoapBinding" type="CityWeatherSoap">
    <wsdlsoap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="get_WEATHER">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input>
        <wsdlsoap:body use="encoded"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather"/>
      </wsdl:input>
      <wsdl:output>
        <wsdlsoap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CityWeatherService">
    <wsdl:port name="CityWeatherSoap" binding="CityWeatherSoapBinding">
      <wsdlsoap:address
      location="http://127.0.0.1/services/sawsdl_wsdl11/CityWeather"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Listing A.1: Original SAWSDL.

Listing A.2: Converted OWL-S from a SAWSDL description.

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:base="http://127.0.0.1/converted/1.1/city_weather_service.owls"
xmlns:grounding="http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:process="http://www.daml.org/services/owl-s/1.1/Process.owl#"
xmlns:profile="http://www.daml.org/services/owl-s/1.1/Profile.owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:service="http://www.daml.org/services/owl-s/1.1/Service.owl#">
  <service:Service rdf:ID="CityWeatherService">
    <service:presents rdf:resource="#PROFILE0"/>
    <service:describedBy rdf:resource="#PROCESS0"/>
  </service:Service>
  <profile:Profile rdf:ID="PROFILE0">
    <profile:hasInput rdf:resource="#CityType"/>
    <profile:hasOutput rdf:resource="#WeatherType"/>
  </profile:Profile>
  <process:AtomicProcess rdf:ID="PROCESS0">
    <service:describes rdf:resource="#CityWeatherService"/>
    <process:hasInput rdf:resource="#CityType"/>
```

```
21      <process:hasOutput rdf:resource="#WeatherType"/>
22    </process:AtomicProcess>
23    <process:Input rdf:ID="CityType">
24      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#
            anyURI">
25      http://minerva.inf-bb.uni-jena.de/mercury/webservices/ontology/SUMO.owl
            #City
26      </process:parameterType>
27      <rdfs:label/>
28    </process:Input>
29    <process:Output rdf:ID="WeatherType">
30      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#
            anyURI">
31      http://minerva.inf-bb.uni-jena.de/mercury/webservices/ontology/Mid-
            level-ontology.owl#Weather
32      </process:parameterType>
33      <rdfs:label/>
34    </process:Output>
35  </rdf:RDF>
```

Listing A.2: Converted OWL-S from a SAWSDL description.

Listing A.3: Original OWL-S.

```
1   <rdf:RDF xmlns:owl = "http://www.w3.org/2002/07/owl#"
2   xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
3   xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4   xmlns:service = "http://www.daml.org/services/owl-s/1.1/Service.owl#"
5   xmlns:process = "http://www.daml.org/services/owl-s/1.1/Process.owl#"
6   xmlns:profile = "http://www.daml.org/services/owl-s/1.1/Profile.owl#"
7   xmlns:grounding = "http://www.daml.org/services/owl-s/1.1/Grounding.owl#"
8   xml:base = "http://127.0.0.1/services/1.1/city_weather_service.owls">
9     <owl:Ontology rdf:about="">
10      <owl:imports rdf:resource="http://127.0.0.1/ontology/Service.owl" />
11      <owl:imports rdf:resource="http://127.0.0.1/ontology/Process.owl" />
12      <owl:imports rdf:resource="http://127.0.0.1/ontology/Profile.owl" />
13      <owl:imports rdf:resource="http://127.0.0.1/ontology/Grounding.owl" />
14      <owl:imports rdf:resource="http://127.0.0.1/ontology/SUMO.owl" />
15      <owl:imports rdf:resource="http://127.0.0.1/ontology/Mid-level-ontology
            .owl" />
16    </owl:Ontology>
17    <service:Service rdf:ID="CITY_WEATHER_SERVICE">
18      <service:presents rdf:resource="#CITY_WEATHER_PROFILE"/>
19      <service:describedBy rdf:resource="#CITY_WEATHER_PROCESS"/>
20      <service:supports rdf:resource="#CITY_WEATHER_GROUNDING"/>
21    </service:Service>
22    <profile:Profile rdf:ID="CITY_WEATHER_PROFILE">
23      <service:isPresentedBy rdf:resource="#CITY_WEATHER_SERVICE"/>
24      <profile:serviceName xml:lang="en">CityWeatherService</profile:
            serviceName>
25      <profile:textDescription xml:lang="en">
26      This service returns current weather of a given city.
27      </profile:textDescription>
28      <profile:hasInput rdf:resource="#_CITY"/>
29      <profile:hasOutput rdf:resource="#_WEATHER"/>
30      <profile:has_process rdf:resource="CITY_WEATHER_PROCESS" />
31    </profile:Profile>
32    <!--<process:ProcessModel rdf:ID="CITY_WEATHER_PROCESS_MODEL">
33    <service:describes rdf:resource="#CITY_WEATHER_SERVICE"/>
34    <process:hasProcess rdf:resource="#CITY_WEATHER_PROCESS"/>
35    </process:ProcessModel>-->
36    <process:AtomicProcess rdf:ID="CITY_WEATHER_PROCESS">
```

```
37      <service:describes rdf:resource="#CITY_WEATHER_SERVICE"/>
38      <process:hasInput rdf:resource="#_CITY"/>
39      <process:hasOutput rdf:resource="#_WEATHER"/>
40    </process:AtomicProcess>
41    <process:Input rdf:ID="_CITY">
42      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#
            anyURI">
43      http://127.0.0.1/ontology/SUMO.owl#City
44      </process:parameterType>
45      <rdfs:label/>
46    </process:Input>
47    <process:Output rdf:ID="_WEATHER">
48      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#
            anyURI">
49      http://127.0.0.1/ontology/Mid-level-ontology.owl#Weather
50      </process:parameterType>
51      <rdfs:label/>
52    </process:Output>
53    <grounding:WsdlGrounding rdf:ID="CITY_WEATHER_GROUNDING">
54      <service:supportedBy rdf:resource="#CITY_WEATHER_SERVICE"/>
55      <grounding:hasAtomicProcessGrounding>
56        <grounding:WsdlAtomicProcessGrounding rdf:ID="
              CITY_WEATHER_AtomicProcessGrounding"/>
57      </grounding:hasAtomicProcessGrounding>
58    </grounding:WsdlGrounding>
59    <grounding:WsdlAtomicProcessGrounding rdf:about="#
          CITY_WEATHER_AtomicProcessGrounding">
60      <grounding:wsdlDocument rdf:datatype="http://www.w3.org/2001/XMLSchema#
            anyURI">
61      http://127.0.0.1/wsdl/CityWeather.wsdl
62      </grounding:wsdlDocument>
63      <grounding:owlsProcess rdf:resource="#CITY_WEATHER_PROCESS"/>
64      <grounding:wsdlOperation>
65        <grounding:WsdlOperationRef>
66          <grounding:operation
67          rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
68          http://127.0.0.1/wsdl/CityWeather#get_WEATHER
69          </grounding:operation>
70          <grounding:portType
71          rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
72          http://127.0.0.1/wsdl/CityWeather#CityWeatherSoap
73          </grounding:portType>
74        </grounding:WsdlOperationRef>
75      </grounding:wsdlOperation>
76      <grounding:wsdlInputMessage rdf:datatype="http://www.w3.org/2001/
            XMLSchema#anyURI">
77      http://127.0.0.1/wsdl/CityWeather#get_WEATHERRequest
78      </grounding:wsdlInputMessage>
79      <grounding:wsdlOutputMessage rdf:datatype="http://www.w3.org/2001/
            XMLSchema#anyURI">
80      http://127.0.0.1/wsdl/CityWeather#get_WEATHERResponse
81      </grounding:wsdlOutputMessage>
82      <grounding:wsdlInput>
83        <grounding:WsdlInputMessageMap>
84          <grounding:owlsParameter rdf:resource="#_CITY"/>
85          <grounding:wsdlMessagePart
86          rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
87          http://127.0.0.1/wsdl/CityWeather#_CITY</grounding:wsdlMessagePart
                >
88          <grounding:xsltTransformationString>
89          None (XSL)
90          </grounding:xsltTransformationString>
```

```
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
        <grounding:owlsParameter rdf:resource="#_WEATHER"/>
        <grounding:wsdlMessagePart
        rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
        http://127.0.0.1/wsdl/CityWeather#_WEATHER
        </grounding:wsdlMessagePart>
        <grounding:xsltTransformationString>
        None (XSL)
        </grounding:xsltTransformationString>
      </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Listing A.3: Original OWL-S.

Listing A.4: Converted SAWSDL from an OWL-S description.

```
<wsdl:definitions name="city_weather_service"
targetNamespace="http://127.0.0.1/services/sawsdl_wsdl11/
    city_weather_service"
xmlns="http://127.0.0.1/services/sawsdl_wsdl11/city_weather_service"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:sawsdl="http://www.w3.org/ns/sawsdl"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="_CITY"
      sawsdl:modelReference="http://minerva.inf-bb.uni-jena.de/mercury/
          webservices/ontology/SUMO.owl#City"/>
      <xsd:element name="_WEATHER"
      sawsdl:modelReference="http://minerva.inf-bb.uni-jena.de/mercury/
          webservices/ontology/Mid-level-ontology.owl#Weather"/>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="CITY_WEATHER_PROFILE_output">
    <wsdl:part name="output0" type="_WEATHER">
    </wsdl:part>
  </wsdl:message>
  <wsdl:message name="CITY_WEATHER_PROFILE_input">
    <wsdl:part name="input0" type="_CITY">
    </wsdl:part>
  </wsdl:message>
  <wsdl:portType name="CITY_WEATHER_PROCESS">
    <wsdl:operation name="CITY_WEATHER_PROFILE">
      <wsdl:input message="CITY_WEATHER_PROFILE_input">
      </wsdl:input>
      <wsdl:output message="CITY_WEATHER_PROFILE_output">
      </wsdl:output>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:service name="CITY_WEATHER_SERVICE">
    <wsdl:port name="CITY_WEATHER_PROCESS">
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Listing A.4: Converted SAWSDL from an OWL-S description.

B

Appendix B

B.1 Matchers directory

In order to add or remove matchers flexibly, the matchers must be placed in a specific hierarchy as illustrated in Figure B.1. The plugin and descriptionCollections folders should retain the subdirectory structure. While the plugin, descriptionCollections and results folders themselves are renamable or removable to different directories, but their location must be specified in a config.properties file.

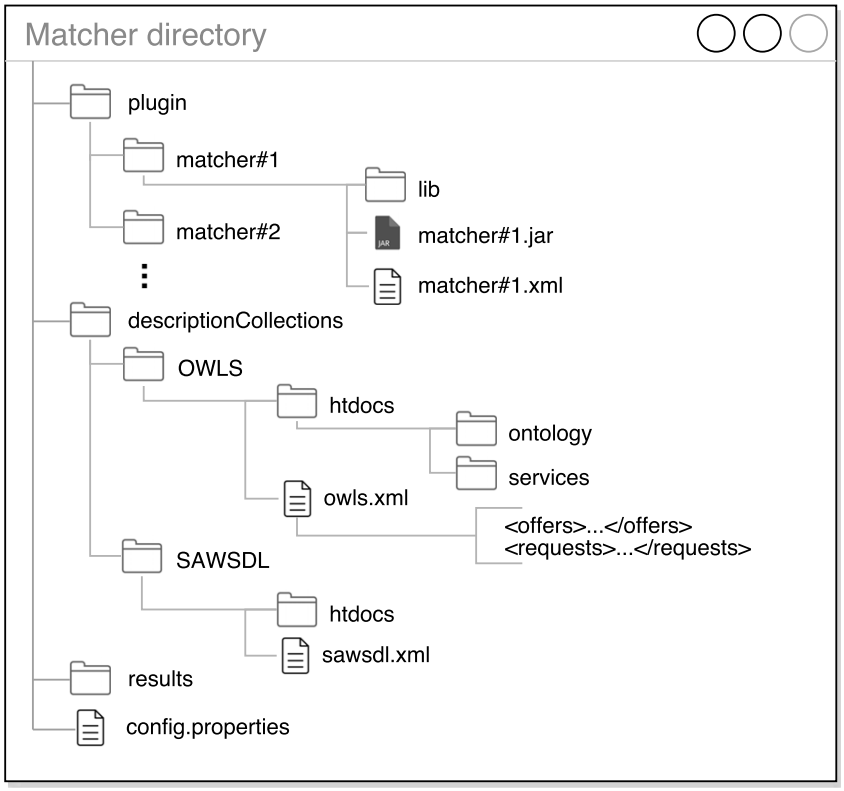


Figure B.1.: Directory structure for resource matchers.

B.2 Test Collections

This thesis adopts resource descriptions from S3 Contest. It provides a sample set of resource descriptions equally in SAWSDL and OWL-S formalisms which are widely in use. The total number of descriptions (for each formalism) is 1080. Out of 1080 descriptions, there are 42 resource descriptions which are provided with the ideal solution for each matching task. These predefined solutions also provide both binary and graded evaluating results. To clarify the evaluation method, the solutions of the requests used for the evaluation are listed here.

B.2.1. Testing requests and solutions

The solution for each request is given in a binary value and graded value. For a binary solution of each request, when a description has value 1, this identifies that the description is relevant to the request. If a description has value 0, or doesn't appear in the solution, this means the description is not relevant to the request.

On the other hand, the graded value solution yields more meaningful results. There are four types of graded value, 3 indicates a highly relevant description, 2 is a relevant description, 1 is a potentially relevant description, and 0 (or not present) means an irrelevant description.

Out of 1080 descriptions, there are 42 descriptions with predefined solutions.

1. shoppingmall_cameraprice
2. book_price
3. open_door
4. citycountry_hotel
5. fall_down_pill
6. grocerystore_food
7. researcher-in-academia_address
8. title_comedyfilm
9. surfing_destination
10. hospital_investigating
11. maxprice_cola
12. getAltitudeAboveSeaLevelOfLocation
13. mileToKilometerConverter
14. publication-number_publication
15. getSunsetSunriseTimeOfLocation
16. dvdplayermp3player_price
17. getDistanceBetweenCitiesWorldwide
18. geographical-region_map
19. car_price
20. title_videomedia
21. country_skilledoccupation
22. recommendedprice_coffeewhiskey
23. 1personbicyclecar_price
24. novel_author
25. preparedfood_price
26. governmentdegree_scholarship
27. getLocationOfCityState
28. geopolitical-entity_weatherprocess

29. lock_door
30. governmentmissile_funding
31. userscience-fiction-novel_price
32. bookpersoncreditcardaccount
33. surfinghiking_destination
34. surfingorganization_destination
35. university_lecturer-in-academia
36. bookpersoncreditcardaccount_price
37. EBookOrder1
38. getLocationOfUSCity
39. geocodeUSAddress
40. getMapOfUSAddress
41. getZipcodeForUSCity
42. getLocationOfUSZipcode

All the descriptions and solutions can be found in [Klu12]. For clarification, we provide the first 30 requests from 42 descriptions, which are randomly chosen for evaluations.

B.2.1.1. Solutions for shoppingmall_cameraprice

This request is expected to have the following matching results:

Highly relevant results

- shoppingmall_cameraprice
- shoppingmall_purchaseableitemprice
- shoppingmall_calendar-datepricecamera

Relevant results

- SRcamera
- pricecamera_Wallmart
- cameraprice_MyShop
- shoppingmall_slrprice
- PhillipDigCamera_price
- shoppingmall_compactprice
- mercantileorganization_slrprice
- shoppingmall_pricedigitalanalog
- shoppingmall_compact-taxedprice
- mercantileorganization_compact-price
- shoppingmall_analogprice-calendar-date
- shoppingmall_price-cellphone-with-camera
- shoppingmall_digital-slrprice-calendar-date
- shoppingmall_price-purchaseable-item-range

Potentially relevant results

- CCP
- price_CannonCamera
- retailstore_slrprice
- retailstore_slrtaxedprice

- retailstore_compactprice
- user_price_ShoppingStatus
- shoppingmall_maxpricedigital-video
- digitalstandardpriceprice_MediaMarkt

B.2.1.2. Solutions for book_price

This request is expected to have the following matching results:

Highly relevant results

- BookPrice
- book_price
- book_authorprice
- book_reviewprice
- monograph_price
- printedmaterial_price
- book_taxedpriceprice
- book_Cheapestprice
- book_pricereviewbook
- book_authorprice_Novel
- bookpersonOptional_price

Relevant results

- novel_price
- userbook_price
- novel_authorprice
- author_bookprice
- book_taxedprice
- novelperson_price
- bookperson_price
- encyclopedia_price
- author_publicationprice
- short-story_authorprice
- book_pricesizebook-type
- book_recommendedprice
- romanticnovel_authorprice
- sciencefictionbookuser_price
- sciencefictionbook_authorprice
- science-fiction-novel_priceauthor
- book_recommended-priceindollar
- objectpersoncredit-account_price
- science-fiction-novel_authorprice
- bookperson-creditcard-account_price
- monograph_recommended-priceineuro
- book_recommended-price_Registered-User
- printed-material-person-creditcard-account_price

Potentially relevant results

- book_author
- DeoSFN_price
- book_authortext
- SFNovelReview
- monographperson
- author_novelprice
- author_monographprice
- book_author_EncSS

- novel_authorbook-type
- book_authorbook-type
- novel_authortaxedprice
- SFNRecommendedPrice
- novel_authormaxprice
- author_booktaxedprice
- fantansynoveluser_price
- userRomanticnovel_price
- sciencefictionbook_author
- short-story_authormaxprice
- author_monographmaxprice
- short-story_authorbook-type
- short-story_authortaxedprice
- sciencefictionbook_publisher
- book__ShoppingCartservice
- romanticnovel_authormaxprice
- userscience-fiction-novel_price
- author_sciencefictionbookprice
- romanticnovel_authorbook-type
- novel_authorrecommendedprice
- romanticnovel_authortaxedprice
- author_bookrecommendedprice
- userscience-fiction-novel_Relprice
- sciencefictionbook_authormaxprice
- sciencefictionbook_authorbook-type
- userscience-fiction-novel_price_Best
- science-fiction-novel_authormaxprice
- sciencefictionbook_authortaxedprice
- short-story_authorrecommendedprice
- science-fiction-novel_authortaxedprice
- science-fiction-novel_authorbook-type
- Tizonbook_recommendedpriceindollar
- romanticnovel_author-recommendedprice
- sciencefictionbook_author-recommended-price
- bookpersoncreditcard-account_taxedfree-price
- science-fiction-novel_author-recommended-price
- bookpersoncreditcard-account_recommended-price

B.2.1.3. Solutions for open_door

This request is expected to have the following matching results:

Highly relevant results

- open_door

Relevant results

- unlock_door

B.2.1.4. Solutions for citycountry_hotel

This request is expected to have the following matching results:

Highly relevant results

- city_hotel
- countrycity_hotel
- citycountry_hotel
- city_accommodation
- countrycity_sportshotel
- postal-address city_hotel
- time-measure countrycity_hotel
- time-measure geopolitical-entity city-hotel

Relevant results

- village_hotel
- country_hotel
- countryvillage_hotel
- durationcountrycity_hotel
- citycountry_accommodation
- durationgeopolitical-entitycity_hotel
- geographical-region_bed-and-breakfast
- duration-geopolitical-entity-city_ accommodation

Potentially relevant results

- city_luxuryhotel
- towncountry_hotel
- _hotel_Worldwide
- city_hotel_Saarland
- city_bedandbreakfast
- countrycity_luxuryhotel
- geopolitical-entity_hotel
- countrycapital-city_hotel
- __luxuryhotel_Heidelburg
- geographical-region_hotel
- city_hotel_Germanservice
- citycountry_destinationhotel
- countrycity_luxuryhotel_Gel
- geopolitical-entity_luxuryhotel
- geographical-region_luxuryhotel
- geographical-region_hotel_XYZ
- geopolitical-entity_accommodation
- citycountryduration__HotelReserve
- geographical-region_accomodation
- geopolitical-entity_bedandbreakfast
- geopolitical-entity recorded-video-activity-hotel
- duration-geopolitical-entity-city-bed-and-breakfast
- time-measure-geopolitical-entity-city-accommodation

B.2.1.5. Solutions for fall_down_pill

This request is expected to have the following matching results:

Highly relevant results

- fall_down_pill

Relevant results

- flip_down_slider

B.2.1.6. Solutions for grocerystore_food

This request is expected to have the following matching results:

Highly relevant results

- grocerystore_food
- retailstore_foodquality
- retailstore_foodquantity
- grocerystore_foodquantity
- mercantileorganization_food

Relevant results

- retailstore_apple
- store_preparedfood
- grocerystore_teaprice
- retailstore_preparedfood
- retailstore_butterquantity
- grocerystore_butterselling
- grocerystore_preparedfood
- grocerystore_butterquantity
- retailstore_sandwichquantity
- grocerystore_flourdoughbutter
- Available_preparedfoodquantity
- retailstore_breadorbiscuitquantity
- Required_preparedfoodquantity
- _food_HEBgroceryCompservice
- retailstore_preparedfoodquantity
- grocerystore_preparedfoodprice
- grocerystore_sandwichquantity
- grocerystore_breadorbiscuitquantity
- grocerystore_preparedfoodquantity
- food_maxpricequantity_Aldiservice
- store_preparedfood_Merchantservice
- grocerystore_fodder_AnimalFoodservice

Potentially relevant results

- drugstore_tea
- food_maxpricequantity
- wholesalestore_preparedfood

B.2.1.7. Solutions for researcher-in-academia_address

This request is expected to have the following matching results:

Highly relevant results

- person_address
- employee_address
- academic_address
- researcher_address
- researcher_address_HOM2
- educational-employee_address
- professor-in-academia_address
- researcher-in-academia_address
- researcher-in-academia_address_ZOO
- researcher-in-academia_address_TREE

Relevant results

- academic_postal-address
- researcher_postal-address
- employee_postal-address
- researcher_abstract-information
- employee_postal-address_XYZ
- educational-employee_postal-address
- research-assistant-in-academia_address
- researcher-in-academia_abstract-information
- researcher-in-academia_ publication-reference-postal-address
- researcher-in-academia_ publication-reference-postal-address

Potentially relevant results

- university_researcher
- visiting-researcher_address
- reader-in-academia_address
- research-fellow-in-academia_ publication-reference

B.2.1.8. Solutions for title_comedyfilm

This request is expected to have the following matching results:

Highly relevant results

- title_film
- title_media
- title_comedyfilm
- title_filmpricequality
- title_mediapricequality
- title_filmmaxpricequality
- title_filmtaxedpricequality
- title_mediamaxpricequality
- title_comedyfilmpricequality
- title_filmtaxfreepricequality
- title_mediataxedpricequality
- title_comedyfilm_BFservice
- title_mediataxfreepricequality
- title_comedyfilm_Megaservice
- title_comedyfilmmaxpricequality
- title_comedyfilmtaxedpricequality
- title_filmrecommendedpricequality
- title_comedyfilmtaxfreepricequality
- title_mediarecommendedpricequality
- title_comedyfilmrecommendedpricequality

Relevant results

- title_vhsdvd
- title_filmP2P
- title_actionfilm
- title_videomedia
- title_lowcomedyfilm
- title_videomediaMM
- title_filmActionComedy
- title_highcomedyfilmreport
- title_actionfilmpricequality
- title_videomediapricequality
- title_obtainablevideomedia
- title_actionfilmmaxpricequality
- title_actionfilmtaxfreepricequality
- title_actionfilmtaxedpricequality
- title_sciencefictionfilmpricequality
- title_videomediataxedpricequality
- title_videomediataxfreepricequality
- title_videomediamaxpricequality
- linguisticexpression_videomedia
- title_sciencefictionfilmmaxpricequality
- title_videomediarecommendedprice
- title_sciencefictionfilmtaxedpricequality
- title_sciencefictionfilmtaxfreepricequality
- title_actionfilmrecommendedpricequality
- title_videomediarecommendedpricequality
- title_sciencefictionfilmrecommendedpricequality

Potentially relevant results

- title_vhs
- filmDiscovery
- filmHighlyRated
- comedyfilmfantacyfilm
- comedyfilmactionfilm
- filmvideomediaDiscoveryChannel

B.2.1.9. Solutions for surfing_destination

This request is expected to have the following matching results:

Highly relevant results

- surfing_destination
- sports_destination
- activity_destination
- surfing_destination_SOH
- surfing_destination_AUS
- surfing_destination_Always

Relevant results

- activity_city
- sports_town
- activity_town
- surfing_beach

- sports_beach
- activity_beach
- activity_ruralarea
- activity_farmland
- surfing_ruralarea
- surfinghiking_city
- hikingsurfing_city
- sports_ruralarea
- surfing_farmland
- surfinghiking_town
- activity_urbanarea
- sports_nationalpark
- activity_nationalpark
- surfing_nationalpark
- surfinghiking_ruralarea
- surfinghiking_city_SF
- surfinghiking_destination
- activity_familydestination
- surfinghiking_nationalpark
- surfinghiking_destination_PF
- sportslegal-agent_destination
- surfinghiking_destination_DFG
- surfinghiking_destination_PDS
- surfinggeneric-agent_destination

Potentially relevant results

- surfingorganization_city
- countrycity_sportshotel
- surfinggeneric-agent_city
- __destination_MyOffice
- countrycity_luxuryhotel_Gel
- personcountrycity_sportshotel
- legal-agentsurfing_destination
- surfingorganization_destination
- sportsorganization_destination
- generic-agentsports_destination
- surfingorganizationperson_destination
- surfingorganization_destination_SOD
- surfingorganization_destination_Best
- surfingorganization_destination_SAAR
- generic-agentsports_destination_Sports
- learning-centred-organizationsurfing-destination

B.2.1.10. Solutions for hospital_investigating

This request is expected to have the following matching results:

Highly relevant results

- hospital_IIPsummary
- hospital_investigating
- hospital_investigatingaddress
- careorganization_investigating
- hospital_postal-addressinvestigating

Relevant results

- HDP
- HDP2
- CAD_medical
- hospital_biopsy
- hospital_experimenting
- careorganization_biopsy
- hospital_diagnosticprocess
- medicalclinic_investigating
- organization_experimenting
- _investigating_Saarservice
- organization_diagnosticprocess
- careorganization_experimenting
- hospital_diagnosticprocesscost
- _diagnosticprocessorganization
- questionhospital_diagnosticprocess
- careorganization_diagnosticprocess
- organization_diagnosticprocesscost
- hospital_diagnosticprocess_MedDiag
- hospital_diagnosticprocesstimeinterval
- hospital_diagnosticprocesstimeduration
- organization_experimentingtimeduration
- hospital_diagnosticprocesstimemeasure
- careorganization_diagnosticprocesstimeinterval
- careorganization_diagnosticprocesstimeduration
- careorganization_diagnosticprocesstimemeasure

Potentially relevant results

- hospital_predicting
- medicalclinic_biopsy
- medicalclinic_predicting
- medicalclinic_experimenting
- careorganization_predicting
- medicalclinic_investigating_Med
- medicalclinic_diagnosticprocess
- medicalclinic_diagnosticprocesstimeinterval
- medicalclinic_diagnosticprocesstimeduration
- medicalclinic_diagnosticprocesstimemeasure

B.2.1.11. Solutions for maxprice_cola

This request is expected to have the following matching results:

Highly relevant results

- maxprice_cola
- price_cola_Guddu
- qualitymaxprice_cola
- maxprice_cola_Best
- untangibleobjects_cola

Relevant results

- taxfreeprice_cola
- maxprice_drinks
- maxprice_liquid
- price_cola_Hallo
- price_cola_Hallo2
- maxprice_beercola
- maxprice_colabreadorbiscuit
- maxprice_colabreadorbiscuit_Both

Potentially relevant results

- maxprice_colabeer
- food_maxpricequantity_Aldi
- price_irishcoffeemixerycola
- maxprice_whiskeycolabeer
- food_maxpricequantity

B.2.1.12. Solutions for getAltitudeAboveSeaLevelOfLocation

This request is expected to have the following matching results:

Highly relevant results

- getAltitudeOfLocation
- getElevationFromLocation
- getAltitudeAboveSeaLevelOfLocation

B.2.1.13. Solutions for mileToKilometerConverter

This request is expected to have the following matching results:

Highly relevant results

- mileToKilometerConverter

B.2.1.14. Solutions for publication-number_publication

This request is expected to have the following matching results:

Highly relevant results

- isbn_publication
- publication-number_book_Portal
- publication-number_publicationauthor
- AcademicBookNumberOrISBNSearch
- publication-number_currencypublication

Relevant results

- isbn_book
- isbn_bookauthor
- isbn_publication
- publication-number_book
- isbn_publicationpublisher
- AcademicBookNumberSearch
- publication-number_edited-book
- publication-number_bookauthor
- academic-item-number_publication
- publication-number_bookauthorpublisher
- academic-item-number_publicationauthor

Potentially relevant results

- isbn_publicationauthor
- academic-item-number_book
- academic-item-number_bookauthor

B.2.1.15. Solutions for getSunsetSunriseTimeOfLocation

This request is expected to have the following matching results:

Highly relevant results

- getSunsetAndSunriseTime
- getSunsetSunriseTwilightTime
- getSunsetSunriseTimeOfLocation

Relevant results

- calculateSunriseTime

B.2.1.16. Solutions for dvdplayermp3player_price

This request is expected to have the following matching results:

Highly relevant results

- dvdplayermp3player_price
- dvdplayermp3player_price_R
- dvdplayermp3player_price_MD
- mp3playerdvdplayer_priceshipping
- dvdplayermp3player_pricemessage

Relevant results

- mediaplayer_price
- electricdevice_price
- mp3player_maxprice
- dvdplayermp3player_Recprice
- mp3playerportabledvdplayer_price
- dvdplayermp3player_RecpriceEuro
- mp3playerdvdplayer_Recpriceshipping
- dvdplayermp3player_Recpricemessage
- mp3playerdvdplayer_recommendedprice
- mp3playerdvdplayer_Recpriceshipping_US

Potentially relevant results

- dvdplayer_maxprice
- dvdplayer_taxedprice
- mp3player_taxedprice
- mediaplayer_maxprice
- mediaplayer_taxedprice
- cdplayermp3player_price
- user_price_ShoppingStatus
- mediaplayer_recommendedpriceineuro
- mp3playercdplayermicrowaveoven_price
- mp3playerportabledvdplayer-recommended-pricequality
- portabledvdplayermp3player-recommended-pricetaxedprice

B.2.1.17. Solutions for getDistanceBetweenCitiesWorldwide

This request is expected to have the following matching results:

Highly relevant results

- getDistanceBetweenCitiesWorldwide

Relevant results

- getDistanceBetweenPlaces
- getDistanceBetweenLocations

Potentially relevant results

- citycity_map
- country_map
- _mapGerman
- citycity_arrowfigure
- uszipcode_distance
- locationlocation_map
- usPostalCode_distance
- calculateDistanceInMiles
- locationlocation_arrowfigure
- locationlocation_map_SRI
- addressDistanceCalculator
- geographical-region_map
- geographical-region_map_Gorg
- geographical-region_mapToBerlin
- geographical-region_mapFromFrankfurt
- calculateDistanceUsingSphericalGeometry
- calculatorDistanceSphericalLawOfCosines

B.2.1.18. Solutions for geographical-region_map

This request is expected to have the following matching results:

Highly relevant results

- locationlocation_map
- geographical-region_map
- locationlocation_map_SRI
- geographical-region_map_Gorg

Relevant results

- citycity_map
- geographical-region_mapFromFrankfurt

Potentially relevant results

- country_map
- _Francemap
- _mapGerman
- staticMapsDisplay
- citycity_arrowfigure
- _mapFrankfurtBerlin
- googleStaticMapsAPI
- locationlocation_icon
- objectsMappingService
- geographical-region_map
- locationlocation_arrow-figure
- geographical-region_map-To-Berlin

B.2.1.19. Solutions for car_price

This request is expected to have the following matching results:

Highly relevant results

- car_price
- auto_price
- vehicle_price
- auto_yearprice
- machine_price
- car_pricecolor
- lenthu_rentcar
- car_yearprice
- car_pricereport
- auto_pricecolor
- car_pricequality
- autocycle_price
- autobicycle_price
- car_taxedpriceprice

Relevant results

- Toyotaprice
- car_priceauto
- RedFerrariprice
- cheapcar_price
- fastcar_yearprice
- fastcar_pricecolor
- 3wheeledcar_price
- 4wheeledcar_price
- fastcar_pricereport
- expensivecar_price
- cheapcar_yearprice
- cheapcar_pricecolor
- car_taxedpricereport
- cheapcar_pricereport

- 3WheeledAudiCarprice
- 4wheeledcaryear_price
- 3wheeledcaryear_price
- 3WheeledOpelCarPrice
- expensivecar_yearprice
- 4wheeledcar_yearprice
- car_recommendedprice
- expensivecar_pricecolor
- auto1personbicycle_price
- 4wheeledcarbicycle_price
- amount-of-moneycar_price
- 4wheeledcaryear_pricereport
- auto_recommendedpricecolor
- car_recommendedpriceineuro
- car_recommendedpriceindollar
- auto2personbicycle_taxedprice
- amount-of-money cheapcar_price
- 1personbicycle 4wheeledcar_price
- amount-of-money 4wheeledcar_price
- amount-of-money 3wheeledcar_price
- amount-of-money expensivecar_price

Potentially relevant results

- carcycle_price
- carbicycle_price
- bicyclecar_price
- bicycleauto_price
- autocycle_taxedprice
- autobicycle_taxedprice
- bicyclecar_priceyear
- carbicycle_taxedprice
- electricdevice_price
- autobicycle_maxprice
- car2personbicycle_price
- fastcar_taxedpricereport
- cyclecar_pricetaxedprice
- 1personbicyclecar_price
- auto2personbicycle_price
- fastcar_recommendedprice
- cycle1personbicycle_price
- bicycle4wheeledcar_price
- cheapcar_taxedpricereport
- cheapcar2personbicycle_price
- carbicycle_recommendedprice
- cheapcar1personbicycle_price
- autocycle_recommendedprice
- autobicycle_recommendedprice
- fastcar_recommendedpricecolor
- cyclecar_recommendedpriceineuro
- 4wheeledcar2personbicycle_price
- 2personbicycle4wheeledcar_price
- cheapcar2personbicycle_maxprice
- 4wheeledcar1personbicycle_price
- caryear_recommendedpriceineuro
- cheapcar1personbicycle_maxprice
- cheapcar_recommendedpricecolor
- 4wheeledcar2personbicycle_maxprice
- 3wheeledcaryear_recommendedprice
- 1personbicyclecar_price_Kohlservice

- Renaultyear_recommendedpriceineuro
- expensivecar_recommendedpricecolor
- 4wheeledcar1personbicycle_maxprice
- 1personbicyclecar_price_TheBestservice
- amount-of-money-cheapcar_recommendedprice
- amount-of-money-4wheeledcar_ recommended-price
- amount-of-money-3wheeledcar_ recommended-price
- amount-of-money-expensivecar_ recommended-price

B.2.1.20. Solutions for title_videomedia

This request is expected to have the following matching results:

Highly relevant results

- title_media
- title_videomedia
- title_mediapricequality
- title_filmmaxpricequality
- title_mediamaxpricequality
- title_videomediapricequality
- title_mediataxedpricequality
- title_mediataxfreepricequality
- title_videomediamaxpricequality
- linguisticexpression_videomedia
- title_videomediataxedpricequality
- title_videomediataxfreepricequality
- title_videomediarecommendedprice
- title_videomediarecommendedpricequality

Relevant results

- title_vhs
- title_vhsdvd
- title_videomediaMM
- title_highcomedyfilmreport
- title_obtainablevideomedia
- title_actionfilmtaxfreepricequality
- title_sciencefictionfilmpricequality
- title_comedyfilmtaxfreepricequality
- title_sciencefictionfilmmaxpricequality
- title_actionfilmrecommendedpricequality

Potentially relevant results

- title_film
- title_filmP2P
- title_actionfilm
- title_comedyfilm
- videomediaBBC
- videomediaSaturn
- title_lowcomedyfilm
- videomediaSmithLee

- title_filmpricequality
- title_comedyfilm_BF
- title_filmActionComedy
- title_comedyfilm_Mega
- title_filmtaxedpricequality
- title_filmtaxfreepricequality
- title_actionfilmpricequality
- title_actionfilmmaxpricequality
- title_comedyfilmmaxpricequality
- title_comedyfilmtaxedpricequality
- filmvideomediaDiscoveryChannel
- title_filmrecommendedpricequality
- title_mediarecommendedpricequality
- title_sciencefictionfilmtaxedpricequality
- title_sciencefictionfilmtaxfreepricequality
- title_comedyfilmrecommendedpricequality
- title_sciencefictionfilmrecommendedpricequality

B.2.1.21. Solutions for country_organizationskilledoccupation

This request is expected to have the following matching results:

Highly relevant results

- -country_skilledoccupation
- -citycountry_skilled-occupation
- country_skilled-occupation_jobs
- country_company-skilled-occupation
- geopolitical-entity_skilled-occupation
- country_corporation-skilled-occupation
- country_organization-skilled-occupation
- country_skilled-occupation-timeduration
- country_skilled-occupation-timemeasure
- country_skilled-occupation-time-duration_Job
- geopolitical-entity_company-skilled-occupation
- geopolitical-entity_skilled-occupation-company
- geopolitical-entity_organization-skilled-occupation
- geographical-region_company-skilled-occupation
- geopolitical-entity_corporation-skilled-occupation
- geopolitical-entity_skilled-occupation-timeduration
- geopolitical-entity_skilled-occupation-timemeasure
- geographical-region_corporation-skilled-occupation
- geographical-region_organization-skilled-occupation
- geographical-region_organization-skilled-occupation_Job
- DJob

Relevant results

- country_profession
- country_sportsposition
- country_occupationaltrade
- country_companyprofession
- geopolitical-entity_profession
- country_corporationprofession
- country_organizationprofession
- country_professiontimeduration
- country_professiontimemeasure
- country_professionfulltimeposition
- country_professionparttimeposition
- country_companyoccupationaltrade
- geopolitical-entity_occupationaltrade
- country_organizationoccupationaltrade
- country_corporationoccupationaltrade
- geopolitical-entity_companyprofession
- country_occupationaltradetimeduration
- country_occupationaltradetimemeasure
- country_skilledoccupationfulltimeposition
- geopolitical-entity_corporationprofession
- geographical-region_companyprofession
- country_occupationaltrade-fulltime-position
- geopolitical-entity_organization-profession
- geopolitical-entity_profession-timeduration
- country_skilledoccupation-parttime-position
- geopolitical-entity_profession-timemeasure
- country_occupationaltrade-parttime-position
- geographical-region_corporation-profession
- geopolitical-entity_profession-fulltime-position
- geographical-region_organization-profession
- geopolitical-entity_profession-parttime-position
- geopolitical-entity_company-occupational-trade
- municipal-unit_skilled-occupation-fulltime-position
- country_skilled-occupation-parttime-position_Job
- geopolitical-entity_corporation-occupational-trade
- municipal-unit_skilled-occupation-parttime-position
- geographical-region_company-occupational-trade
- geopolitical-entity_organization-occupational-trade
- geopolitical-entity_occupational-trade-time-measure
- geopolitical-entity_occupational-trade-time-duration
- geographical-region_corporation-occupational-trade
- geographical-region_organization-occupational-trade
- geopolitical-entity_occupationaltrade-fulltime-position
- geopolitical-entity_skilledoccupation-parttime-position
- geopolitical-entity_occupationaltrade-parttime-position

Potentially relevant results

- country_deacon
- company_profession
- medicaldoctor_UNO
- city_skilledoccupation
- skilledoccupation_BMW
- bankeraddress_CityBank
- company_skilledoccupation
- profit-organization_profession
- companycountry_skilled-occupation
- profit-organization_skilled-occupation
- government-organization_profession
- municipal-unit_profession-timeduration
- municipal-unit_profession-timemeasure
- municipal-unit_profession-fulltimeposition
- municipal-unit_profession-parttimeposition
- public-companycountry_skilled-occupation
- government-organization_skilled-occupation
- municipal-unit_skilledoccupation-timeduration
- municipal-unit_skilledoccupation-timemeasure
- municipal-unit_occupationaltrade-timeduration
- municipal-unit_occupationaltrade-fulltimeposition
- municipal-unit_occupationaltrade-parttimeposition

B.2.1.22. Solutions for recommendedprice_coffeewhiskey

This request is expected to have the following matching results:

Highly relevant results

- price_whiskeycoffee
- price_coffeewhiskey
- maxprice_coffeewhiskey
- price_coffeewhiskeyquality
- price_whiskeycoffee_Ziko
- taxfreeprice_whiskeycoffee
- price_coffeewhiskey_Thebest
- price_coffeewhiskeytimemeasure
- recommendedprice_coffeewhiskey
- hotelrecommendedprice_coffeewhiskey
- price_coffeewhiskey-quality-time-position
- recommendedprice_coffee-whiskey_Best
- recommendedpriceineuro_coffee-whiskey
- recommendedpriceindollar_whiskey-coffee
- recommendedprice_coffee-whiskey-symbolic-string
- recommendedprice_content-bearing-object-whiskey-coffee
- recommendedprice_coffee-whiskey-symbolic-string-quality
- recommendedprice_coffee-with-whiskey-content-bearing-object

Relevant results

- price_drinks
- taxfreeprice_cola
- maxprice_drinks_Hot
- recommendedprice_irishcoffeetasting

Potentially relevant results

- coffee_maxprice
- coffee_taxedprice
- maxprice_drinks
- price_coffeewithwhiskey
- food_recommendedprice
- coffee_recommendedprice
- maxprice_whiskeycolabeer
- price_irishcoffeemixerycola
- recommendedprice_irishcoffee
- preparedfood_recommendedprice

B.2.1.23. Solutions for 1personbicyclecar_price

This request is expected to have the following matching results:

Highly relevant results

- carcycle_price
- carbicycle_price
- bicyclecar_price
- bicycleauto_price
- autobicycle_price
- bicyclecar_priceyear
- 1personbicyclecar_price
- cyclecar_pricetaxedprice
- auto1personbicycle_price
- 1personbicyclecar_price_Kohl
- 1personbicyclecar_price_TheBest

Relevant results

- vehicle_price
- car_pricecolor
- cheapcar_price
- _RedFerrariprice
- fastcar_yearprice
- 4wheeledcar_price
- autocycle_maxprice
- cheapcar_pricecolor
- carbicycle_taxedprice
- autobicycle_maxprice
- 4wheeledcar_yearprice
- _3WheeledOpelCarPrice
- 4wheeledcarbicycle_price
- bicycle4wheeledcar_price
- carbicycle_recommendedprice
- cheapcar1personbicycle_price
- autobicycle_recommendedprice
- 1personbicycle4wheeledcar_price
- 4wheeledcar1personbicycle_price
- cyclecar_recommendedpriceineuro

Potentially relevant results

- car_price
- Toyotaprice
- auto_price
- lenthu_rentcar
- car_priceauto
- car_yearprice
- auto_yearprice
- machine_price
- autocycle_price
- car_pricereport
- auto_pricecolor
- car_pricequality
- fastcar_pricecolor
- 3wheeledcar_price
- expensivecar_price
- car_taxedpricereport
- cheapcar_yearprice
- car_taxedpriceprice
- autocycle_taxedprice
- cheapcar_pricereport
- autobicycle_taxedprice
- expensivecar_pricecolor
- car_recommendedprice
- 4wheeledcaryear_price
- 3wheeledcaryear_price
- car2personbicycle_price
- expensivecar_yearprice
- _3WheeledAudiCarprice
- auto2personbicycle_price
- amount-of-moneycar_price
- cheapcar_taxedpricereport
- fastcar_recommendedprice
- 4wheeledcaryear_pricereport
- auto2personbicycle_maxprice
- auto_recommendedpricecolor
- car_recommendedpriceindollar
- cheapcar2personbicycle_price
- autocycle_recommendedprice
- car_recommendedpriceineuro
- auto2personbicycle_taxedprice
- fastcar_recommendedpricecolor
- objectpersoncreditaccount_price
- amount-of-moneycheapcar_price
- 4wheeledcar2personbicycle_price
- 2personbicycle4wheeledcar_price
- amount-of-moneycar_pricecompany
- cheapcar_recommendedpricecolor
- cheapcar2personbicycle_maxprice
- caryear_recommendedpriceineuro
- cheapcar1personbicycle_maxprice
- amount-of-money4wheeledcar_price
- amount-of-money3wheeledcar_price
- amount-of-moneyexpensivecar_price
- 4wheeledcar2personbicycle_maxprice
- 4wheeledcar1personbicycle_maxprice
- expensivecar_recommendedpricecolor
- auto2personbicycle_recommendedprice
- amount-of-moneycheapcar_recommended-price
- amount-of-money3wheeledcar_recommended-price
- amount-of-money4wheeledcar_recommended-price
- amount-of-moneyexpensivecar_recommended-price

B.2.1.24. Solutions for novel_author

This request is expected to have the following matching results:

Highly relevant results

- novel_author
- book_author
- book_authortext
- novel_authortime
- novel_authorprice
- publication_author
- novel_authorgenre
- book_authorprice
- monograph_author
- book_authorbook-type
- novel_authorbook-type
- book_author_EncSS
- novel_authormaxprice
- novel_person_Writer
- novel_author_MyOnto
- novel_userreviewauthor
- novel_authortaxedprice
- novel_authorcommitting
- book_authorprice_Novel
- novel_author_BookOntoservice
- novel_authorrecommendedprice

Relevant results

- isbn_bookauthor
- romanticnovel_authorprice
- romanticnovel_authorbook-type
- romanticnovel_authormaxprice
- romanticnovel_authortaxedprice
- publication-number_bookauthor
- science-fiction-novel_authorprice
- science-fiction-novel_priceauthor
- sfnovel_authorauthor_BookOnto
- academic-item-number_bookauthor
- sciencefictionbook_authormaxprice
- sciencefictionbook_authortaxedprice
- science-fiction-novel_authortaxedprice
- sciencefictionnovel_author_MyOnto
- science-fiction-novel_authorbook-type
- romanticnovel_authorrecommended-price
- science-fiction-novel_author-recommended-price

Potentially relevant results

- SFNovelReview
- book_publisher
- novel_publisher
- short-story_author

- author_novelprice
- encyclopedia_author
- publication_publisher
- monograph_publisher
- short-story_authorprice
- isbn_publicationauthor
- ScienceFNovelReview
- author_noveltaxedprice
- book_person_Publisher
- romanticnovel_publisher
- sciencefictionbook_author
- short-story_authormaxprice
- short-story_authorbook-type
- book_readerreviewperson
- sciencefictionbook_publisher
- short-story_authortaxedprice
- science-fiction-novel_publisher
- encyclopedia_authorbook-type
- sciencefictionbook_authorprice
- author_novelrecommendedprice
- sciencefictionbook_authorbook-type
- publication-number_publicationauthor
- short-story_authorrecommendedprice
- science-fiction-novel_authormaxprice
- publication-number_bookauthor-publisher
- academic-item-number_publication-author
- sciencefictionbook_author-recommended-price

B.2.1.25. Solutions for preparedfood_price

This request is expected to have the following matching results:

Highly relevant results

- MerkelD
- MarkoPS
- preparedfood_price
- preparedfood_priceday
- food_pricequantity_Aldi
- food_price_AnimalFood
- food_pricephysical-quantity_Aldi
- preparedfood_taxedpriceindollarprice

Relevant results

- Ben
- ZAD
- MAK
- SPD-Grune
- food_price
- food_maxpricequantity
- preparedfood_GSprice
- preparedfood_maxprice
- preparedfood_taxedprice
- food_recommendedprice

- preparedfood_taxfreeprice
- preparedfood_SpainishTax
- food_taxedpricequantity_Aldi
- food_taxfreepricequantity_Aldi
- preparedfood_recommendedprice
- food_maxpricephysical-quantity_Aldi
- food_taxedpricephysical-quantity_Aldi
- food_taxfreepricephysical-quantity_Aldi

Potentially relevant results

- coffee_maxprice
- price_Fish
- butter_maxprice
- butter_taxedprice
- coffee_taxedprice
- sandwich_maxprice
- coffeesandwich_price
- preparedfood_USTax
- grocerystore_teaprice
- sandwich_taxedprice
- tea_recommendedprice
- meat_pricequantity_Aldi
- butter_recommendedprice
- coffee_recommendedprice
- user_price_ShoppingStatus
- food_maxpricequantity_Aldi
- sandwich_recommendedprice
- grocerystore_preparedfoodprice
- meat_pricephysical-quantity_Aldi
- breadorbiscuit_recPricetaxedpriceineuro

B.2.1.26. Solutions for degreegovernment_scholarship

This request is expected to have the following matching results:

Highly relevant results

- awardgovernment_funding
- degreegovernment_funding
- governmentaward_scholarship
- governmentdegree_scholarship
- degreegovernment_scholarship
- governmentdegree_scholarshipquantity
- governmentdegree_scholarship_TheBest
- degreegovernmentorganization_scholarship

Relevant results

- government_scholarship
- government_funding_Missile
- award_funding_GermanGov
- government_funding_ForPhD
- degree_funding_GermanGov
- nationalgovernment_scholarship

- award_scholarship_GermanGov
- degree_scholarship_GermanGov
- government_scholarshiporganization
- governmentorganization_scholarship
- award_fundingduration_GermanGov
- degree_fundingduration_GermanGov
- academic-degreegovernment_funding
- award_scholarshipduration_SwissGov
- academic-degree_funding_GermanGov
- nationalgovernment_scholarshipquantity
- degreenationalgovernment_scholarship
- award_scholarshipduration_GermanGov
- degree_scholarshipduration_GermanGov
- academic-degreegovernment_scholarship
- academic-degree_scholarship_GermanGov
- nationalgovernment_scholarship quantity duration
- academic-degree_fundingduration_GermanGov
- academic-degreegovernment-organization_funding
- academic-degree_scholarship-duration-GermanGov
- nationalgovernment_physical-quantity-scholarship-landarea

Potentially relevant results

- government_lending
- government_welfare
- citygovernment_lending
- governmentdegree_welfare
- nationalgovernment_lending
- degreegovernment_lending
- award_lending_GermanGov
- degree_lending_GermanGov
- governmentdegree_givingback
- degreegovernment_unilateralgiving
- award_lending-duration_GermanGov
- degree_lendingduration_GermanGov
- academic-degree-government_lending
- academic-degree_lending_GermanGov
- academic-degree-government_unilateralgiving
- academic-degree_lendingduration_GermanGov
- academic-degree-government-organization-lending
- academic-degree-government-organization-unilateral-giving

B.2.1.27. Solutions for getLocationOfCityState

This request is expected to have the following matching results:

Highly relevant results

- getLocationOfCityState

Relevant results

- city_state_ZipCodes
- checkAndLookupAddress
- getCoordinatesOfAddress
- getLocationOfCityWorldwide

Potentially relevant results

- addressGeocoder
- getPlaceOfAddress
- queryParserLocation
- googleGeocodingAPI
- real-time_geocoding
- getATMLocationsInCity
- getLocationOfAddress
- gazetteerLookupLocation
- findPlaceNamePostalCode
- getZipCodesWithinCityState
- getLocationOfAddressWorldwide
- getLocationOfAddressYahooMaps
- real-time_geocodingStreetAddress

B.2.1.28. Solutions for geopolitical-entity_weatherprocess

This request is expected to have the following matching results:

Highly relevant results

- geopolitical-entity_weatherprocess
- geographical-region_weatherprocess
- geographical-region_weatherprocess_GRW

Relevant results

- icing_German
- warmfront_Italy
- country_lightning
- country_warmfront
- country_weatherfront
- municipal-unit_drought
- country_weatherseason
- country_weathersystem
- country_weatherprocess
- municipal-unit_warmfront
- weatherprocess_German
- geopolitical-entity_drought
- geopolitical-entity_lightning
- municipal-unit_weatherfront
- geopolitical-entity_warmfront
- geographical-region_drought
- geographical-region_lightning
- municipal-unit_weathersystem
- municipal-unit_weatherseason
- municipal-unit_weatherprocess
- geopolitical-entity_weatherfront
- geographical-region_warmfront
- geographical-region_weatherfront

- geopolitical-entity_weathersystem
- geopolitical-entity_weatherseason
- geographical-region_weatherfront
- geopolitical-entity_weatherprocess
- geographical-region_weatherseason
- geopolitical-entity_weatherseasonproposition
- geopolitical-entity_weatherseasontimeposition

Potentially relevant results

- country_drought
- city_weathersystem
- municipal-unit_lightning
- geopolitical-entity_internalchange

B.2.1.29. Solutions for government_funding_Missileservice

This request is expected to have the following matching results:

Highly relevant results

- missile-government_giving
- government-missile_funding
- government_funding_Missile
- projectile-government_funding
- missile-government_giving-range
- missile-government_funding-range
- missile-government_giving_Borrow
- government-missile-weapon_funding
- government-missile-weapon_funding
- government-missile_funding_Reliable
- missile-government-organization_funding
- missile-government-organization_giving-range
- missile-government-organization_funding-range
- government-organization-selfpowered-device_funding

Relevant results

- missile_funding_Pak
- missile_funding_India
- missile_financing_US
- missile_funding_Asian
- missile_funding_NKorea
- missile_financing_China
- missile_financing_Russian
- government-weapon_funding
- government-missile_financing
- weapon-missile_funding_Iraq
- government_funding_ABomb
- government_funding_BallMissile
- missile-government_financing-range
- national-government-weapon_funding
- ballisticmissile-government_giving-range
- ballisticmissile-government_funding-range

- ballisticmissile-government_lending-range
- ballisticmissile-government_financing-range
- missile-government organization_financing-range
- ballisticmissile-government-organization_giving-range
- ballisticmissile-government-organization_funding-range
- ballisticmissile-government-organization_financing-range

Potentially relevant results

- missile-government_lending-range
- missile-government organization_lending-range
- government-organization-missile_unilateral-giving
- ballisticmissile-government-organization_lending-range

B.2.1.30. Solutions for lock_door

This request is expected to have the following matching results:

Highly relevant results

- lock_door

Relevant results

- close_door

Appendix C

Listing C.1: Establishing connection service via Twitter API.

```
1  import pprint
2  import logging
3  logging.basicConfig(level=logging.DEBUG)
4  logging.getLogger('spyne.protocol.xml').setLevel(logging.DEBUG)
5  logging.getLogger('sqlalchemy.engine.base.Engine').setLevel(logging.DEBUG)
6
7  from spyne.application import Application
8  from spyne.decorator import rpc
9  from spyne.error import InternalError
10 from spyne.model.complex import ComplexModel
11 from spyne.model.complex import Iterable
12 from spyne.model.fault import Fault
13 from spyne.model.primitive import Float
14 from spyne.model.primitive import Mandatory
15 from spyne.model.primitive import Unicode
16 from spyne.model.primitive import DateTime
17 from spyne.protocol.soap import Soap11
18 from spyne.server.wsgi import WsgiApplication
19 from spyne.service import ServiceBase
20 import base64
21
22 #from urllib import urlopen
23 import urllib2
24 import json
25 from datetime import datetime
26
27 from sqlalchemy import create_engine
28 from sqlalchemy.orm import sessionmaker
29 db = create_engine('sqlite:///:memory:')
30 Session = sessionmaker(bind=db)
31 auth_token = ""
32
33 def twRequest(ctx,path,parameters=""):
34   global auth_token
35   headers = {'Authorization':'Bearer '+auth_token}
36   data = None
37   request = urllib2.Request("https://api.twitter.com/1.1/"+path+"?"+
         parameters,headers=headers,data=data)
38   response = urllib2.urlopen(request).read()
39   return_value = json.loads(response)
40   return return_value
41
42 def twCreateDate(dateTimeString):
43 #Wed Aug 29 17:12:58 +0000 2012
44   return datetime.strptime(dateTimeString,'%a %b %d %H:%M:%S +0000 %Y')
```

```
45
46  class Person(ComplexModel):
47    name = Unicode
48    fb_id = Unicode
49    tw_id = Unicode
50    def __hash__(self):
51      if self.fb_id == None:
52        has = int(self.tw_id)
53      if self.tw_id == None:
54        has = int(self.fb_id)
55        return has
56    def __eq__(self, other):
57      if isinstance(other, Person):
58        if self.fb_id == None:
59          if self.tw_id == other.tw_id:
60            return True
61        if self.tw_id == None:
62          if self.fb_id == other.fb_id:
63            return True
64         else:
65            return False
66
67  class Range(Float):
68    unit = Unicode
69
70  from math import sin,pow,cos,atan,sqrt,pi
71  class Coordinate(ComplexModel):
72    longitude = Mandatory.Float
73    latitude = Mandatory.Float
74    def distance(ctx, self, other):
75      if isinstance(other, Coordinate):
76        logging.debug(self)
77        logging.debug(other)
78        l = (self.longitude-other.longitude)/2*pi/180
79        t = 1/298.257223563
80        a = 6378.137
81        F = (self.latitude+other.latitude)/2*pi/180
82        G = (self.latitude-other.latitude)/2*pi/180
83        logging.debug(F)
84        logging.debug(G)
85        S = pow(sin(G),2)*pow(cos(l),2)+pow(cos(F),2)*pow(sin(l),2)
86        C = pow(cos(G),2)*pow(cos(l),2)+pow(sin(F),2)*pow(sin(l),2)
87        logging.debug(S)
88        logging.debug(C)
89        w = atan(sqrt(S/C))
90        D = 2*w*a
91        if w==0:
92          R=0
93        else:
94          R = sqrt(S*C)/w
95        H1 = (3*R-1)/(2*C)
96        if S==0:
97          H2 = 0
98        else:
99          H2 = (3*R+1)/(2*S)
100       s = D * (1+t*H1*pow(sin(F),2)*pow(cos(G),2)-t*H2*pow(cos(F),2)*pow(
              sin(G),2))
101       logging.debug("Distance is %s km"%str(s))
102       return s
103     else:
104       raise "Wrong Type"
105
```

```
106 class Place(ComplexModel):
107   name = Mandatory.Unicode
108   coordinate = Coordinate
109
110 pp = pprint.PrettyPrinter()
111 class TwitterService(ServiceBase):
112   @rpc(Mandatory.Unicode,_returns=bool)
113   def initSession(ctx,token):
114     global auth_token
115
116     consumer = '4o6gv9DFGMDmPbtnEh5PsSv3M'
117     consumer_secret = 'PVjvFONKvtQfDsgkDeSHbYStENllCH4cJXdChgMvRXtvdguW6k'
118     bearer_token = consumer + ':' + consumer_secret
119     base64_bearer_token = base64.b64encode(bearer_token)
120
121     headers = {'Authorization':'Basic '+base64_bearer_token,
122       'Content-Type' : 'application/x-www-form-urlencoded;charset=UTF-8'}
123     data = 'grant_type=client_credentials'
124     request = urllib2.Request('https://api.twitter.com/oauth2/token',
          headers=headers,data=data)
125     response = urllib2.urlopen(request).read()
126     data = json.loads(response)
127     logging.debug(data)
128         if 'access_token' in data:
129       if data['token_type'] == 'bearer':
130         auth_token = data['access_token']
131         return True
132       return False
133   @rpc(_returns=Unicode)
134   def getAuthToken(ctx):
135     global auth_token
136     return auth_token
137   @rpc(str,DateTime,_returns=Iterable(Person))
138   def recentContacts(ctx,username,d):
139     path = 'search/tweets.json'
140     r = twRequest(ctx,path,"q="+username+"&count=200")
141     persons = set()
142     for item in r['statuses'] :
143       p= Person()
144       p.name = item['user']['name']
145       p.tw_id = str(item['user']['id'])
146       persons.add(p)
147         return persons
148   @rpc(str,DateTime,_returns=Iterable(Place))
149   def recentPlaces(ctx,username,d):
150     path = 'statuses/user_timeline.json'
151     r = twRequest(ctx,path,"screen_name="+username+"&count=200")
152     places = set()
153     for item in r:
154       if (item.has_key('created_at')):
155         stamp = twCreateDate(item['created_at'])
156         if (d != None):
157           if stamp<d:
158             break
159       if item['place']!=None:
160         place = Place()
161         place.name = item['place']['name']
162         place.coordinate = Coordinate()
163         latitude = 0
164         counter = 0
165         longitude = 0
166         for i in item['place']['bounding_box']['coordinates'][0]:
```

```
167             longitude = longitude + i[0]
168             latitude = latitude + i[1]
169             counter = counter + 1
170           place.coordinate.longitude = longitude/counter
171           place.coordinate.latitude = latitude/counter
172           places.add(place)
173       return places
174   @rpc(str,str,DateTime,_returns=Iterable(Place))
175   def recentPlacesWith(ctx,firstPerson,otherPerson,d):
176       path = 'statuses/user_timeline.json'
177       r = twRequest(ctx,path,"screen_name="+firstPerson+"&count=200")
178       places = set()
179       for item in r:
180         if (item.has_key('created_at')):
181           stamp = twCreateDate(item['created_at'])
182           if (d != None):
183             if stamp<d:
184               break
185         found = False
186         if item['entities']!=None:
187           if item['entities']['user_mentions']!=None:
188             for i in item['entities']['user_mentions']:
189               p = Person()
190               p.name = i['name']
191               #p.tw_id = i['id']
192               if (p.name!=otherPerson):
193                 found = True
194         if item['place']!=None and found == True:
195           place = Place()
196           place.name = item['place']['name']
197           place.coordinate = Coordinate()
198           latitude = 0
199           counter = 0
200           longitude = 0
201           for i in item['place']['bounding_box']['coordinates'][0]:
202             longitude = longitude + i[0]
203             latitude = latitude + i[1]
204             counter = counter + 1
205           place.coordinate.longitude = longitude/counter
206           place.coordinate.latitude = latitude/counter
207           places.add(place)
208       return places
209   @rpc(str,Coordinate,Range,DateTime,_returns=bool)
210   def isNear(ctx,username,coordinate,range,d):
211       path = 'statuses/user_timeline.json'
212       r = twRequest(ctx,path,"screen_name="+username+"&count=2")
213       places = set()
214       for item in r:
215         if (item.has_key('created_at')):
216           stamp = twCreateDate(item['created_at'])
217           if (d != None):
218             if stamp<d:
219               break
220         if item['place']!=None:
221           place = Place()
222           place.name = item['place']['name']
223           place.coordinate = Coordinate()
224           latitude = 0
225           counter = 0
226           longitude = 0
227           for i in item['place']['bounding_box']['coordinates'][0]:
228             longitude = longitude + i[0]
```

```
            latitude = latitude + i[1]
            counter = counter + 1
          place.coordinate.longitude = longitude/counter
          place.coordinate.latitude = latitude/counter
          distance = place.coordinate.distance(place.coordinate,coordinate)
          print "distance:",distance
          if (distance<=range):
            return True
      return False

class UserDefinedContext(object):
  def __init__(self):
    self.session = Session()
def _on_method_call(ctx):
  ctx.udc = UserDefinedContext()
def _on_method_context_closed(ctx):
  if ctx.udc is not None:
  ctx.udc.session.commit()
      ctx.udc.session.close()

class GatewayApplication(Application):
  def __init__(self,services,tns,name=None,in_protocol=None,out_protocol=
      None):
    super(GatewayApplication,self).__init__(services,tns,name,in_protocol,
        out_protocol)
    self.event_manager.add_listener('method_call',_on_method_call)
    self.event_manager.add_listener('method_context_closed',
        _on_method_context_closed)
  def call_wrapper(self, ctx):
    try:
      return ctx.service_class.call_wrapper(ctx)
    except Fault, e:
      logging.error(e)
      raise
    except Exception, e:
      logging.exception(e)
      raise InternalError(e)

from wsgiref.simple_server import make_server
from spyne.protocol.http import HttpRpc
if __name__=='__main__':
  app = GatewayApplication([TwitterService], 'social.sensor.mercury.http',
    in_protocol=Soap11(),
    out_protocol=Soap11())

  wsgi_app = WsgiApplication(app)

  server = make_server('127.0.0.1',7788,wsgi_app)
  # Create a server listening on port 7788
  print "listening to http://127.0.0.1:7788"
  print "wsdl is at: http://localhost:7788/?wsdl"

  server.serve_forever()
```

Listing C.1: Establishing connection service via Twitter API.

Listing C.2: Twitter client for retrieving user context.

```
import logging
logging.basicConfig(level=logging.INFO)
logging.getLogger('suds.client').setLevel(logging.DEBUG)
from suds.client import Client
```

```
5  import sys
6  import random
7
8  client = Client('http://localhost:7788/?wsdl')
9  #The client port is corresponding to Server port
10 logging.debug(client)
11
12 #Init Session w/ FB Auth
13 result = client.service.initSession()
14 assert result == True
15
16 username = sys.argv[1]
17 username2 = sys.argv[2]
18
19 #Choose a random person from recent contacts
20 result = client.service.recentContacts(username)
21 person = result[0][random.randint(0,len(result[0])-1)]
22 logging.debug(person)
23
24 #Get your own recently visited places
25 result = client.service.recentPlaces(username)
26
27 #Where have you been with yourself? Same as above...
28 result2 = client.service.recentPlacesWith(username,username2)
29 place = result[0][0]
30
31 #Are you within 1km to one of these places? Rounding calibration <100m
32 result = client.service.isNear(username,place.coordinate,0.1)
33
34 #Have you been within 200km to the center of Germany?
35 coordinate = client.factory.create('Coordinate')
36 coordinate.longitude = 9
37 coordinate.latitude = 51
38 result = client.service.isNear(username,coordinate,200)
```

Listing C.2: Twitter client for retrieving user context.

Listing C.3: Establishing connection service via Facebook API.

```
1  import pprint
2  import logging
3  import urllib
4  import urlparse
5  logging.basicConfig(level=logging.DEBUG)
6  logging.getLogger('spyne.protocol.xml').setLevel(logging.DEBUG)
7  logging.getLogger('sqlalchemy.engine.base.Engine').setLevel(logging.DEBUG)
8
9  from spyne.application import Application
10 from spyne.decorator import rpc
11 from spyne.error import InternalError
12 from spyne.model.complex import ComplexModel
13 from spyne.model.complex import Iterable
14 from spyne.model.fault import Fault
15 from spyne.model.primitive import Float
16 from spyne.model.primitive import Mandatory
17 from spyne.model.primitive import Unicode
18 from spyne.model.primitive import DateTime
19 from spyne.protocol.soap import Soap11
20 from spyne.server.wsgi import WsgiApplication
21 from spyne.service import ServiceBase
22
23 from urllib import urlopen
```

```
import json
from datetime import datetime, timedelta

from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
db = create_engine('sqlite:///:memory:')
Session = sessionmaker(bind=db)

auth_token = ""

def fbRequest(ctx,path,parameters=""):
   global auth_token
   response = urlopen("https://graph.facebook.com/v2.6/"+path+"?
       access_token="+auth_token+parameters)
   data = json.loads(response.read())
   return data

def fbCreateDate(dateTimeString):
   #2012-09-27T08:38:16+0000
   return datetime.strptime(dateTimeString,'%Y-%m-%dT%H:%M:%S+0000')

class Person(ComplexModel):
   name = Unicode
   fb_id = Unicode
   tw_id = Unicode
   def __hash__(self):
      if self.fb_id == None:
         has = int(self.tw_id)
      if self.tw_id == None:
         has = int(self.fb_id)
         return has
   def __eq__(self, other):
      if isinstance(other, Person):
         if self.fb_id == None:
            if self.tw_id == other.tw_id:
               return True
         if self.tw_id == None:
            if self.fb_id == other.fb_id:
               return True
      else:
         return False

class Range(Float):
   unit = Unicode

from math import sin,pow,cos,atan,sqrt,pi
class Coordinate(ComplexModel):
   longitude = Mandatory.Float
   latitude = Mandatory.Float
   def distance(ctx, self, other):
      if isinstance(other, Coordinate):
         logging.debug(self)
         logging.debug(other)
         if ((self.longitude==other.longitude) & (self.latitude==other.
             latitude)):
         return 0
         l = (self.longitude-other.longitude)/2*pi/180
         t = 1/298.257223563
         a = 6378.137
         F = (self.latitude+other.latitude)/2*pi/180
         G = (self.latitude-other.latitude)/2*pi/180
         S = pow(sin(G),2)*pow(cos(l),2)+pow(cos(F),2)*pow(sin(l),2)
```

```
84          C = pow(cos(G),2)*pow(cos(l),2)+pow(sin(F),2)*pow(sin(l),2)
85          w = atan(sqrt(S/C))
86          D = 2*w*a
87          R = sqrt(S*C)/w
88          H1 = (3*R-1)/(2*C)
89          H2 = (3*R+1)/(2*S)
90          s = D * (1+t*H1*pow(sin(F),2)*pow(cos(G),2)-t*H2*pow(cos(F),2)*pow
                (sin(G),2))
91          logging.debug("Distance is %s km"%str(s))
92          return s
93       else:
94          raise "Wrong Type"
95
96 class Place(ComplexModel):
97    name = Mandatory.Unicode
98    coordinate = Coordinate
99
100 pp = pprint.PrettyPrinter()
101 class FacebookService(ServiceBase):
102    @rpc(Unicode,_returns=bool)
103    def initSession(ctx,token):
104       global auth_token
105       auth_token = token
106       path = "me"
107       r = fbRequest(ctx,path,"")
108       logging.debug(r)
109       if r.has_key('id'):
110          return True
111       return False
112    @rpc(_returns=Unicode)
113    def getAuthToken(ctx):
114       global auth_token
115       return auth_token
116    @rpc(_returns=Person)
117    def whoami(ctx):
118       path = "me"
119       r = fbRequest(ctx,path,"")
120       logging.debug(r)
121       person = Person()
122       person.name = r['name']
123       person.fb_id = r['id']
124       logging.debug(person)
125       return person
126    @rpc(Person,DateTime,_returns=Iterable(Person))
127    def recentContacts(ctx,person,d):
128       path = str(person.fb_id)+"/feed"
129       r = fbRequest(ctx,path,"&fields=story_tags,updated_time,to,from&limit
              =10")
130       persons = set()
131       for item in r['data']:
132          # Get recent contacts from all time
133          if (item.has_key('story_tags')):
134             tags = item['story_tags']
135             for tag in tags:
136                if (item.has_key('type')):
137                   if (tag['type']=='user'):
138                      person = Person()
139                      person.name = tag['name']
140                      person.fb_id = tag['id']
141                      persons.add(person)
142          if (item.has_key('to')):
143             i = item['to']['data'][0]
```

```
144             person = Person()
145             person.name = i['name']
146             person.fb_id = i['id']
147             persons.add(person)
148          if (item.has_key('from')):
149             i = item['from']
150             person = Person()
151             person.name = i['name']
152             person.fb_id = i['id']
153             persons.add(person)
154       return persons
155    @rpc(Person,DateTime,_returns=Iterable(Place))
156    def recentPlaces(ctx,person,d):
157       path = str(person.fb_id)+"/feed"
158       r = fbRequest(ctx,path,"&fields=updated_time,place&limit=10")
159       logging.debug(pp.pformat(r))
160       places = set()
161       for item in r['data']:
162          if (item.has_key('updated_time')):
163             stamp = fbCreateDate(item['updated_time'])
164             if (d != None):
165                if stamp<d-timedelta(days=4):
166                   break
167          if (item.has_key('place')):
168             i = item['place']
169             place = Place()
170             place.name = i['name']
171             place.coordinate = Coordinate()
172             place.coordinate.latitude = i['location']['latitude']
173             place.coordinate.longitude =i['location']['longitude']
174             places.add(place)
175       return places
176    @rpc(Person,Person,DateTime,_returns=Iterable(Place))
177    def recentPlacesWith(ctx,firstPerson,otherPerson,d):
178       path = str(firstPerson.fb_id)+"/feed"
179       r = fbRequest(ctx,path,"&fields=story_tags,updated_time,to,from,place
              &limit=10")
180       logging.debug(pp.pformat(r))
181       places = set()
182       for item in r['data']:
183          # Get recent contacts from all time
184          found = False
185          if (item.has_key('story_tags')):
186             tags = item['story_tags']
187             for tag in tags:
188                if (tag.has_key('type')):
189                   if (tag['type']=='user'):
190                      person = Person()
191                      person.name = tag['name']
192                      person.fb_id = tag['id']
193                      if (person==otherPerson):
194                         found = True
195          if (found == False and item.has_key('to')):
196             i = item['to']['data']
197             person = Person()
198             person.name = i[0]['name']
199             person.fb_id = i[0]['id']
200             if (person==otherPerson):
201                found = True
202          if (found == False and item.has_key('from')):
203             i = item['from']
204             person = Person()
```

```
205             person.name = i['name']
206             person.fb_id = i['id']
207             if (person==otherPerson):
208                found = True
209          if (item.has_key('place') and found == True):
210             i = item['place']
211             place = Place()
212             place.name = i['name']
213             place.coordinate = Coordinate()
214             place.coordinate.latitude = i['location']['latitude']
215             place.coordinate.longitude =i['location']['longitude']
216             places.add(place)
217       return places
218    @rpc(Person,Coordinate,Range,DateTime,_returns=bool)
219    def isNear(ctx,person,coordinate,range,d):
220       path = person.fb_id+"/feed"
221       r = fbRequest(ctx,path,"&fields=updated_time,place&limit=10")
222       logging.debug(pp.pformat(r))
223       for item in r['data']:
224          if (item.has_key('updated_time')):
225             stamp = fbCreateDate(item['updated_time'])
226             if (d != None):
227                if stamp<d-timedelta(days=4):
228                   break
229          if (item.has_key('place')):
230             i = item['place']
231             place = Place()
232             place.name = i['name']
233             place.coordinate = Coordinate()
234             place.coordinate.latitude = i['location']['latitude']
235             place.coordinate.longitude =i['location']['longitude']
236             distance = place.coordinate.distance(place.coordinate,
                    coordinate)
237             if (distance<=range):
238                return True
239       return False
240    @rpc(Person,DateTime,_returns=Place)
241    def getUserAddress(ctx,person,d):
242       place = Place()
243       path = person.fb_id
244       r = fbRequest(ctx,path,"&fields=location")
245       if (r.has_key('location')):
246          locid = r['location']['id']
247          path = locid
248          l = fbRequest(ctx,path,"&fields=location")
249          if (l.has_key('location')):
250             loc = l['location']
251             place.name = loc['city']+','+loc['country']
252             place.coordinate = Coordinate()
253             place.coordinate.latitude = loc['latitude']
254             place.coordinate.longitude = loc['longitude']
255       return place
256
257 class UserDefinedContext(object):
258    def __init__(self):
259       self.session = Session()
260 def _on_method_call(ctx):
261    ctx.udc = UserDefinedContext()
262 def _on_method_context_closed(ctx):
263    if ctx.udc is not None:
264       ctx.udc.session.commit()
265       ctx.udc.session.close()
```

```
266
267 class GatewayApplication(Application):
268     def __init__(self,services,tns,name=None,in_protocol=None,out_protocol=
            None):
269         super(GatewayApplication,self).__init__(services,tns,name,in_protocol
                ,out_protocol)
270         self.event_manager.add_listener('method_call',_on_method_call)
271         self.event_manager.add_listener('method_context_closed',
                _on_method_context_closed)
272     def call_wrapper(self, ctx):
273         try:
274             return ctx.service_class.call_wrapper(ctx)
275         except Fault as e:
276             logging.error(e)
277             raise
278         except Exception as e:
279             logging.exception(e)
280             raise InternalError(e)
281
282
283 from wsgiref.simple_server import make_server
284 from spyne.protocol.http import HttpRpc
285 if __name__=='__main__':
286     app = GatewayApplication([FacebookService], 'social.sensor.mercury.fb',
287         in_protocol=Soap11(),
288         out_protocol=Soap11()
289         )
290
291     wsgi_app = WsgiApplication(app)
292
293     server = make_server('127.0.0.1',7789,wsgi_app)
294
295     print("listening to http://127.0.0.1:7789")
296     print("wsdl is at: http://localhost:7789/?wsdl")
297
298     server.serve_forever()
```

Listing C.3: Establishing connection service via Facebook API.

Listing C.4: Facebook client for retrieving user context.

```
1  import logging
2  logging.basicConfig(level=logging.INFO)
3  logging.getLogger('suds.client').setLevel(logging.DEBUG)
4  from suds.client import Client
5  import sys
6  import random
7  import urllib
8  import urlparse
9  import webbrowser
10 import warnings
11 import os
12 import time
13
14 client = Client('http://localhost:7789/?wsdl')
15 logging.debug(client)
16
17 class FBOAuth(object):
18     # Parameters of your app and the id of the profile you want to mess with
          .
19     APP_ID = '174216792758017'
20     APP_SECRET = 'a533d83ffe9cb79cddc5e9bb8ccfa2aa'
```

```
21      SECRET_CODE = None
22      ACCESS_TOKEN = None
23      REDIRECT_URI = 'http://localhost:7790/'
24      def authorize(self):
25          warnings.filterwarnings('ignore', category=DeprecationWarning)
26          savout = os.dup(1)
27          os.close(1)
28          os.open(os.devnull, os.O_RDWR)
29          try:
30              webbrowser.open('https://graph.facebook.com/oauth/authorize?'+
                    urllib.urlencode(
31                              {'client_id':self.APP_ID,
32                               'redirect_uri':self.REDIRECT_URI,
33                               'scope':'public_profile,user_friends,user_location
                                     ,user_posts'}))
34
35          finally:
36              os.dup2(savout, 1)
37
38      def access_token(self):
39          if not self.SECRET_CODE:
40              self.authorize()
41          while True:
42              time.sleep(0.5)
43              self.SECRET_CODE = urllib.urlopen(self.REDIRECT_URI+"?request=
                    secret").read()
44              if self.SECRET_CODE == "None": #the return type is string....
45               pass
46              else:
47               break
48          print self.SECRET_CODE
49
50          args = {'redirect_uri': self.REDIRECT_URI,
51                'client_id' : self.APP_ID,
52                'client_secret':self.APP_SECRET,
53                'code':self.SECRET_CODE,}
54
55          response = urllib.urlopen("https://graph.facebook.com/oauth/
                access_token?" + urllib.urlencode(args)).read()
56          response = urlparse.parse_qs(response)
57          token = response['access_token'][0]
58          return token
59
60  if __name__ == '__main__':
61      #Init Session w/ FB Auth
62      fb = FBOAuth()
63      fb.SECRET_CODE = None
64      token = fb.access_token()
65      result = client.service.initSession(token)
66      assert result == True
67      me = client.service.whoami()
68
69      #Choose a random person from recent contacts
70      result = client.service.recentContacts(me)
71      person = result[0][random.randint(0,len(result[0])-1)]
72      logging.debug(person)
73
74      #Get your own recently visited places
75      result = client.service.recentPlaces(me)
76
77      #Is the location in the status close to your current living city?
            Rounding calibration <100m
```

```
78     place = client.service.getUserAddress(me)
79     result = client.service.isNear(me,place.coordinate,0.01)
80
81     #Are you within 200km to the center of Germany?
82     coordinate = client.factory.create('Coordinate')
83     coordinate.longitude = 9
84     coordinate.latitude = 51
85     result = client.service.isNear(me,coordinate,200)
```

Listing C.4: Facebook client for retrieving user context.

Listing C.5: Redirecting service for Facebook API.

```
1  import BaseHTTPServer
2  import time
3  import sys
4
5
6  HOST_NAME = 'localhost'
7  PORT_NUMBER = 7790
8  REDIRECTIONS = {"/facebook/": "http://facebook.com/"}
9  LAST_RESORT = "http://facebook.com/"
10 SECRET_CODE = None
11
12 class RedirectHandler(BaseHTTPServer.BaseHTTPRequestHandler):
13    def do_HEAD(s):
14       s.send_response(301)
15       s.send_header("Location", REDIRECTIONS.get(s.path, LAST_RESORT))
16       s.end_headers()
17    def do_GET(s):
18       global SECRET_CODE
19       params = s.path.split("/?")[1].split("&")
20       for param in params:
21        key = param.split("=")[0]
22        val = param.split("=")[1]
23        if key=="code":
24           s.do_HEAD()
25           SECRET_CODE = val
26        elif key=="request":
27           if val=="secret":
28              s.send_response(200)
29              s.send_header('Content-type','text/html')
30              s.end_headers()
31              # Send the html message
32              s.wfile.write(SECRET_CODE)
33              SECRET_CODE=None
34              return
35
36 if __name__ == '__main__':
37    server_class = BaseHTTPServer.HTTPServer
38    httpd = server_class((HOST_NAME, PORT_NUMBER), RedirectHandler)
39    print time.asctime(), "Server Starts - %s:%s" % (HOST_NAME, PORT_NUMBER)
40    try:
41       httpd.serve_forever()
42    except KeyboardInterrupt:
43       pass
44    httpd.server_close()
45    print time.asctime(), "Server Stops - %s:%s" % (HOST_NAME, PORT_NUMBER)
```

Listing C.5: Redirecting service for Facebook API.

D

Appendix D - Licenses and Permissions

- In Copyright - Non-Commercial Use Permitted
  http://rightsstatements.org/page/InC-NC/1.0/

- Open Data Commons Attribution License (ODC-By) v1.0
  https://opendatacommons.org/licenses/by/1.0/

- License Number: 4157670887545
  Licensed Content Publisher: Springer
  Licensed Content Title: Toward a cooperative programming framework for context-aware applications
  Licensed Content Author: Bin Guo
  Type of Use: Thesis/Dissertation
  Portion: Figures/tables/illustrations
  Original figure numbers: Figure 4

- License Number: 4157690951493
  Licensed Content Publisher: Elsevier
  Licensed Content Title: The design and realisation of the Experimentmy Virtual Research Environment for social sharing of workflows
  Licensed Content Author: David De Roure,Carole Goble,Robert Stevens
  Type of Use: reuse in a thesis/dissertation
  Portion: figures/tables/illustrations
  Original figure numbers: Figure 1

- License Number: 4160211006307
  Licensed Content Publisher: Elsevier
  Licensed Content Title: A survey of context modelling and reasoning techniques
  Licensed Content Author: Claudio Bettini,Oliver Brdiczka,Karen Henricksen,Jadwiga Indulska,Daniela Nicklas,Anand Ranganathan,Daniele Riboni
  Type of Use: reuse in a thesis/dissertation
  Portion: figures/tables/illustrations
  Original figure numbers: Figure 1

- License Number: 4160211287322
  Licensed Content Publisher: Elsevier
  Licensed Content Title: A survey of context modelling and reasoning techniques
  Licensed Content Author: Claudio Bettini,Oliver Brdiczka,Karen Henricksen,Jadwiga Indulska,Daniela Nicklas,Anand Ranganathan,Daniele Riboni
  Type of Use: reuse in a thesis/dissertation
  Portion: figures/tables/illustrations
  Original figure numbers: Figure 2

- License Number: 4160220133428
  Licensed Content Publisher: Springer
  Licensed Content Title: Context-driven personalized service discovery in pervasive environments
  Licensed Content Author: Katharina Rasch
  Type of Use: Thesis/Dissertation
  Portion: Figures/tables/illustrations
  Original figure numbers: Figures 1, 6

- License Number: 4160221149121
  Licensed Content Publisher: Springer
  Licensed Content Title: Adaptive and Context-Aware Service Discovery for the Internet of Things
  Licensed Content Author: Talal Ashraf Butt
  Type of Use: Thesis/Dissertation
  Portion: Figures/tables/illustrations
  Original figure numbers: Figure 1

- License Number: 4163330385414
  Licensed Content Publisher: Springer
  Licensed Content Title: OpenIoT: Open Source Internet-of-Things in the Cloud
  Licensed Content Author: John Soldatos
  Type of Use: Thesis/Dissertation
  Portion: Figures/tables/illustrations
  Original figure numbers: Figure 4

- License Number: 4163560138337
  Licensed Content Publisher: Elsevier
  Licensed Content Title: A knowledge-based resource discovery for Internet of Things
  Licensed Content Author: Charith Perera,Athanasios V. Vasilakos
  Type of Use: reuse in a thesis/dissertation
  Portion: figures/tables/illustrations
  Original figure numbers: Figure 1

Curriculum Vitae

Personal Data

Name, Address	Kobkaew Opasjumruskit Ernst-Abbe-Platz 5 07743 Jena
	kobkaew.opasjumruskit@uni-jena.de http://fusion.cs.uni-jena.de/fusion/members/kobkaew-opasjumruskit/
Nationality	thai
Marital status	single
Birth date	7 April 1983
Birth place	Bangkok, Thailand

Education

Apr 2007	Master's Degree in Electrical Engineering, Chulalongkorn University, Bangkok, Thailand
Apr 2005	Bachelor's Degree in Electrical Engineering, Chulalongkorn University, Bangkok, Thailand

Work Experience

Since Sep 2011	Ph.D.candidate and researcher at Heinz-Nixdorf Endowed Chair for Distributed Information Systems, Department of Mathematics and Computer Science - Friedrich Schiller University of Jena
Oct 2009 - Aug 2011	Software Engineer, Thomson Reuters Software, Thailand
May 2008 - Oct 2009	Embedded Software Engineer, COD Co., Ltd., Thailand
May 2007 - Apr 2008	Junior Software Engineer, ASK Media Co., Ltd., Thailand