ilmedia

TECHNISCHE UNIVERSITÄT ILMENAU

Müller, Christine; Kaiser, Matthias; Knauf, Rainer:

Utilizing AI-Technologies for intelligent user interface behavior

Publikation entstand im Rahmen der Veranstaltung:
29th Annual German Conference on Artificial Intelligence, 2nd Workshop Knowledge and Software Engineering (KESE 2006), 14. - 19. Juni 2006, Bremen

Utilizing AI-Technologies for Intelligent User Interface Behavior

Christine Müller[1], Matthias Kaiser[2], and Rainer Kauf[3]

[1] Dept. of Information Management
Technical University of Ilmenau
PO Box 100565, 98684 Ilmenau, Germany
[2] SAP Research Center Palo Alto, LCC
Palo Alto, CA, USA
[3] Faculty of Computer Science and Automation
Technical University Ilmenau
PO Box 100565, 98684 Ilmenau, Germany

Abstract. While today's business applications become more complex, support solutions have not improved respectively. This leads to a decrease in the usability of software interfaces. Additionally, the current interaction modalities do not enable users to interact with the system in an adaptive way. This research proposes a framework for automated and adaptive assistance in business applications. Specific focus is placed on cognitive support and multi-channel interaction. An ontology-based approach is applied to represent domains, tasks, and user models. Furthermore, rules are implemented to trigger the generation of context-sensitive support content which can be mapped onto individualized user interfaces according to the device and the modality that the user prefers.

1 Introduction

While today's business applications are characterized by an increasing degree of complexity, user support has not improved respectively. As a consequence the usability of software interfaces is declining. Although efforts in user interface design are being made to reduce the complexity and to improve the usability of software, they cannot satisfy the individual needs of every user [Nie93], [Opp94], [ST01].

Training, e.g. by e-Learning, is one attempt to improve the usability of interfaces. Although, it can help the users to cope with poorly designed interfaces, it is time-consuming, expensive, and not available when using the application.

Support features, such as manuals, hyperlink-based documents, and glossaries also address the problem of user support. However, major drawbacks arise from the presentation of static and anonymous support including irrelevant and redundant information. Most help features are also not embedded in business applications. Consequently, users are forced to frequently interrupt their work and reorient themselves afterwards.This decreases their productivity and satisfaction. Due to a lack of explanation why certain information is given, users are less likely to accept solutions and to place trust in a system [Opp94], [SS02].

Several methods have been investigated that address the improvement of software usability by creating intelligent content-driven or adaptive systems [Opp94]. The approach in proactive UIs relates to adaptive help systems, such as UC [Chi86] and UMFE [Sle85] following a passive, context-independent, and user-sensitive approach, as well as Activist [Sch91] and Passivist [Lem84] offering context-sensitive, user-independent, and active / passive support, respectively [Opp94]. Proactive UIs provide two support modes: Depending on the individual preferences, support is given on initiative of the system (active) or has to be explicitly requested by the user (passive). In contrast to other adaptive help systems, users of proactive UIs initialize the support mode themselves: While novice users might wish to be interrupted by support messages on initiative of the system and therefore choose the active support mode, experts are more likely to reject any kind of interference and prefer passive support on demand.

In addition, this research relates to the distinction between content generation and presentation as proposed in the adaptive help system HyPlan [Opp94]. Respectively, in proactive UIs context-sensitive content is generated and formally represented in the system. The content is then mapped according to the device, such as PDA, mobile phone, or desktop, and modality, such as textual, graphical, or audible, users prefer. Thus, users are supported by an optimal input and output facilities based on their individual needs and preferences. Proactive help has also been investigated in the Lumiere Project by Microsoft [HH98]. Based on the interactions of users, the MS assistant proactively triggers static help messages that are presented by a graphical avatar to guide users towards an assumed goal. In contrast, a proactive UI dynamically generates content through reasoning with ontology-based representation of the domains, tasks, and user models. Instead of the system to assume the user's goal, it explicitly asks for it. Thus, allowing the user full control over the support mode (active or passive). While the MS assistant only offers a single output facility, proactive UIs support a multi-channel interaction.

This paper proposes a framework for automated and adaptive assistance in business applications, with a specific focus on cognitive support and multi-channel interaction. The approach introduced here utilizes a knowledge engineering technology to solve the software engineering problem of proactive UIs.

The structure of the paper is as follows. In section 2, we discuss our model of knowledge representation. Section 3 introduces the architecture of proactive UIs. Section 4 discusses the knowledge modeling by means of ontologies. The conclusion and future work are outlined in section 5.

2 Knowledge Representation

We distinguish between three knowledge types: (1) domain knowledge, (2) task knowledge, and (3) user knowledge.

Domain knowledge, stored in the domain model, is knowledge about all concepts in domain and application, as well as their properties and relations.

Task knowledge, stored in the task model, is knowledge describing the processes in the domain and application. In addition, we assume the existence of three main concepts: (1) actions, (2) tasks, and (3) goals [CJB98], [vWvdV98].

- Actions are atomic preconditioned tasks that trigger a consequence [FN71].
- Tasks are activities performed by user in order to achieve a particular goal. Complex tasks can be composed of smaller sub-tasks, which consist of a number of actions. In addition, it shall be distinguished between three different states of tasks and sub-tasks: *default*, *pending*, and *achieved*.
  - *default*: No action of a task has been executed so far.
  - *pending*: Some but not all actions of a task have been executed.
  - *achieved*: All actions have been executed. The task has been completed.
- Goals are intended outcomes of tasks and can be broken down into smaller sub-goals. Sub-goals are based respectively to their sub-tasks.

User knowledge, stored in a dynamic user model, is knowledge about the goals of the users as well as a history of all previous tasks and actions during his current interaction with an application. While the domain and task model are static and generated during design time of the application, the user model represents dynamic information on the user and has to be inferred during runtime.

Figure 1 presents two state charts visualizing the interdependencies between actions, tasks, and goals. The first state chart represents a goal which is decomposed into two sub-goals. Each sub-goal is achieved by a respective sub-task. The second state chart represents a sub-goal which is achieved by executing two actions in arbitrary order.

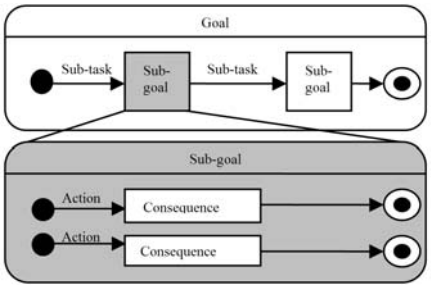

Fig. 1. Declaration of task knowledge. A goal is decomposed into sub-goals. Sub-goals are achieved by executing respective actions.

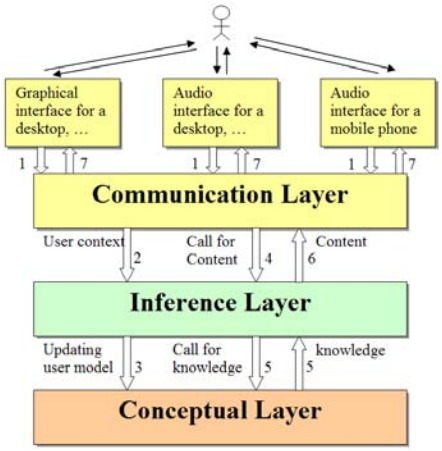

Fig. 2. Architecture of the proactive UI.

3 Introducing Proactive UIs

3.1 Definition

Proactive UIs are intelligent content-driven interfaces that define human computer interaction as a problem solving process in which computers provide cognitive support in order to facilitate users to use a familiar or new application. This requires the employment of cognitive strategies to understand, anticipate and support cognitive user needs in the problem solving process of users during his interaction with the application.

Our approach of cognitive support also includes a flexible environment in which users can choose the device and modality they are most comfortable with. We claim that this feature reduces cognitive overload of users since they can fully concentrate on solving a specific problem. Consistent applicability of user interfaces on diverse platforms as well as an easy adaptability across different interaction modalities is therefore a major requirement for cognitive support [KM05].

3.2 Architecture of Proactive UIs

The architecture of proactive UIs consists of a conceptual layer, an inference layer, and a communication layer.

- The conceptual layer represents the knowledge base including domain, task, and user model.
- The inference layer draws on the knowledge in the conceptual layer to dynamically generate context-sensitive support content. This content is then represented in a formal way and passed onto the communication layer.
- The communication layer interacts with the user. Furthermore, it maps the support content to conform to the device-specific presentation formats set by the user. Thus, depending on the individual preferences of users, the formal support content is mapped into an audible, graphical, or textual format.

Figure 2 presents the architecture of proactive UIs including the interactions between layers. Users intact with the proactive UI via their preferred device, e.g. a mobile phone, and the required modality, such as audible in- and output. In the background, the communication layer monitors the interactions of the user, including data entries and commands (1), and retrieves the user context. The user context includes the goals of the user and his / her interactions with the application. It is transferred to the support component (2) which updates the user model in the conceptual layer (3). Furthermore, the communication layer parses user requests and calls the support component to generate context-sensitive content (4). Based on the knowledge represented in the conceptual layer (5), the support component generates the support content which is then transferred to the communication layer (6). Finally, the support content is mapped according to the device and modality constraints set by the user (7).

3.3 Cognitive Support in Proactive UIs

One feature of proactive UIs is the generation of cognitive support that facilitates users to solve a problem. Further discussions are based on the assumption that the problem solving process of human beings is divided into the following consecutive phases [Pol57], [Ama68], [ea86]:

- Phase One: The user analyzes her current situation to identify relevant information and to get a better understanding of the environment.
- Phase Two: The user identifies all options available to achieve her goal.
- Phase Three: Based on her current situation and available options in the situation, the user chooses one option.
- Phase Four: The user executes the option to achieve the individual goal.
- Phase Five: The user reviews the problem solving process, evaluates its satisfaction, and stores relevant information used in previous similar situation.

Proactive UIs provide cognitive support in each phase. Different cognitive support types have been derived from the cognitive problem solving process of human beings. They facilitate users to accomplish the respective phases:

- Type one: Situational descriptions facilitate users to identify relevant information in their current situation,
- Type two: Overviews of relevant goals facilitate users to identify all options they have in their current situations,
- Type three: Recommendations of goals support users to make their decisions,
- Type four: Goal-directed previews, and step-by-step instruction teach users how to achieve a goal,
- Type five: Reviews present information of how a goal was achieved and point out conclusions that can be drawn, and
- Type six and seven: Justifications of recommendations and terminological definitions facilitate the understanding of users and encourage them to place trust in the support messages. Hence, users are more likely to trust a system, if they can reconstruct how it did retrieve certain solutions or what caused the system to make certain recommendations.

4 Knowledge Modelling

4.1 Reusing Domain and Task Model

In order to model a specific business process, the knowledge about a particular domain and its processes is collected and structured during the design phase. A key feature of our concept is the reusability of the resulting domain and task model for the development of the support feature of the application. The idea is to represent the two knowledge models by ontologies, since they guarantee a high degree of reusability and can be easily maintained. This significantly simplifies the development process since information can be shared and communicated among development teams. In addition, the fact that diverse applications and

their support features draw knowledge from a common ontology facilitates data respectively knoweldge exchange and allows for a consistent user support behavior.

Several ontology development environments are available to support the representation of knowledge in several formats [MFRW00]. We use Protégé [GMF+03] to structure the domain and task model during design time since a Protégé project, including both models, can be directly imported into a Jess reasoning engine [FH03], [FH05]. The reasoning engine transforms the ontological data into a formal representation which is stored in the conceptual layer of the proactive UI. Furthermore, the engine is used to explicate knowledge and infer new knowledge from the conceptual layer, thus facilitating dynamic support generation.

The following sections discuss the ontological engineering for a domain and task model as well as strategies for support generation in more detail by means of a sample scenario.

4.2 The Scenario

To facilitate a better understanding of the principles proposed, we are drawing several examples from an eRecruiting application, a business application used in Human Ressource Departments. The eRecruiting application models the task workflow of two types of users: candidates and recruiters. Recruiters working in the Human Ressource Department can, for example, manage applications, post job offers, and enter paper applications, while internal and external candidates enter their resumes and submit applications. Examples referring to the eRecruiting scenario can be found pervasively throughout the following sections.

4.3 Ontologies in Proactive UIs

We distinguish between three types of ontologies: (1) top-level ontologies, (2) domain ontologies, and (3) task ontologies [Gua97]. The top-level ontology is a domain-independent meta-ontology that defines the structure of the domain and task ontology. The domain ontology is used to represent the domain model. It structures the domain concepts by giving their hierarchical position, relation, and properties. Figure 3 shows a simple domain model of the eRecruiting scenario [MFRW00],[RN03]. The task ontology is used to represent the task model. It describes the tasks and actions in the application as well as their interdependencies [vWvdV98], [Tim02], [MSI96]. Figure 5 shows the hypothetical super goal *Resume-created* of the eRecruiting scenario which requires the achievements of three sub-tasks which can be completed in an arbitrary order. However, sequential orders of task can be implemented as well by setting precondition of specific tasks. The state chart below represents the sub-goal *Skill-entered* which is achieved by executing three parallel actions. In addition, ontologies represent the user model composed of user goal, tasks, and actions. In fact, the user model basically is considered a subset of the task model: Instead of representing all goals, task, and actions in the domain, only individual actions and tasks already completed by the user are included.

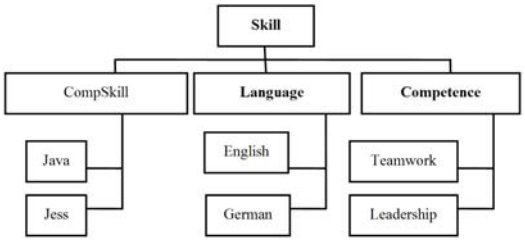

Fig. 3. A simple domain model for an eRecruiting application.

```
(term (is-a term)(has-name Skill)
      (has-super-term nil)
      (has-sub-terms Language Competence CompSkill))

(term (is-a term)(has-name Language)
      (has-super-term Skill)
      (has-sub-terms English German))

(term (is-a term)(has-name Competence)
      (has-super-term Skill)
      (has-sub-terms Teamwork Leadership))
```

Fig. 4. Three facts representing a simplified domain model.

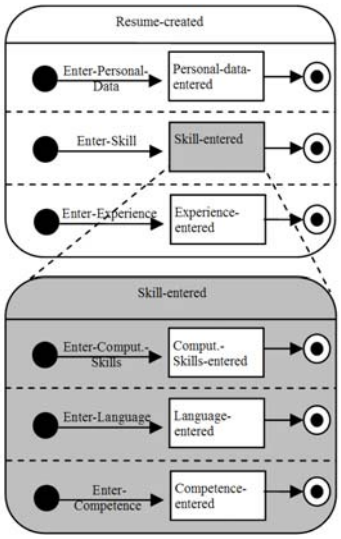

Fig. 5. Detailed view on the goal *Resume-created*.

4.4 Implementation of Ontologies

We suggest the use of a rule-based language in order to implement the knowledge representation of proactive UIs. In rule-based languages, facts are used to specify user, task, and domain models. These facts may be regarded as axiomatically true statements about concepts. The properties of concepts and the relations between concepts are stored in slots which are attached to the facts [FH03].

Figure 4 shows three Jess facts representing the three domain concepts *Skill*, *Language*, and *Competence* introduced in Figure 3. The slots of the facts represent their hierarchal relation, such as super-term and sub-term, as well as properties, such as name and type.

```
(goal (is-a goal) (has-name Resume-created)
      (has-option Create-Resume)))

(task (is-a task) (has-name Create-Resume)
      (has-Precondition none) (has-parts Enter-Personal-Data
       Enter-Skill Enter-Experience)))

(action (is-a action) (has-name Enter-Name)
        (part-of Enter-Personal-Data)
        (has-filler none)))
```

Fig. 6. Three facts representing the domain model.

The task model introduced in Figure 5 is implemented by Jess facts as shown in Figure 6. The first fact represents the user goal *Resume-created* that can

be achieved by the task *Create-Resume*. The second fact implements the task *Create-Resume* which has three sub-tasks and no precondition in order to be completed. Preconditions implement sequential order of tasks. The third fact embodies the action *Enter-Language* which is part of the sub-task *Enter-Personal-Data* and can have any value.

```
(user-goal Resume-created)

(user-task (is-a task)(has-name Create-Resume)
           (part-of nil)
           (has-status default)
           (has-part Enter-Personal-Data
                     Enter-Skill Enter-Experience))
```

Fig. 7. Implementation of the default user model.

The user model can be implemented as shown in Figure 7. The user has already chosen a goal, but has not entered any further data yet. The first fact embodies the goal of the user to create a resume. The second fact represents the current task of the user, *Create Resume*, whose state is flagged *default*.

Whenever the user executes an action, a fact representing the new action is added to the user model. Furthermore, the status of its sub and super task are updated if they are required for the overall goal of the user. The status of a sub or super task in the user model can be default, pending, or achieved. A (sub-)task is default if none, pending if at least one, and achieved if all of its required (actions) sub-tasks have been completed.

4.5 Rules for Support Generation

Domain-independent rules have been implemented to dynamically update the user model and generate definition, situational description, recommendation, and justification. Since rules do not need to be tailored to a specific domain but only depend on the structure of domain, task, and user model, they can be reused in other domains.

Figure 8 shows the Jess implementation of a rule which fires whenever a new action that is part of a *default* sub-task is added to the conceptual layer. On fulfillment of this precondition the old fact representingthe sub-task will be deleted, and a new fact representing the same – but updated – sub-task is added.

The rule for the generation of a definition is given in Figure 9. If a term needs to be explained, the respective fact is retrieved from the knowledge base as is the fact of a parallel concept in the ontology. The information contained in both facts is then used to generate the definition. Next, the definition is structured into a suitable format, such as xml, and is then passed on to the communication layer, which maps it according to the users device and modality preferences.

Figure 10 introduces a simplified rule for the generation of a recommendation. The rule fires whenever the user requests a valid entry for a specific field in the

```
(defrule set-user-model
   (user-action (is-a action)
                (has-name ?action)
                (part-of ?task)
                (has-value $?filler))
   ?fact<-(user-task (is-a task)
                     (has-name ?task)
                     (part-of ?part-of)
                     (has-status default))
=>(retract ?fact)
  (assert(user-task (is-a task)
                    (has-name ?task)
                   (part-of ?part-of)
                   (has-status pending)
                   (has-part ?action))))
```

Fig. 8. Updating the user model.

```
   ?fact<-(user-entry explain ?object)
   (term (is-a ?type) (has-name ?object)
         (has-super-term ?super)
         (has-sub-terms ?sub))
   (term (is-a ?type) (has-name ?analog)
         (has-super-term ?super))
=>
   (retract ?fact)
   (set-message
      <definition>
         <term> ?object </term>
         <type> ?type </type>
         <super-term> ?super </super-term>
         <sub-term> ?sub </sub-term>
         <parallel-term> ?analog </parallel-term>
      </definition>
    ))
```

Fig. 9. Generating a definition.

application. The output of the rule is a list of possible values. This list can be further filtered based on the user model and preferences.

5 Conclusion and Future Work

A framework for automated and adaptive support generation has been proposed with a focus on cognitive support and multi-channel interaction. In particular, we suggest to structure and represent domains, tasks, and user models by means of ontologies and rule-based principles. Furthermore, reasoning strategies have been provided to explicate knowledge and to form context-sensitive support messages. A prototype has been implemented that delivers proof and verification of our concept of a proactive UI.

Future work will be based on the extension and improvement of both knowledge representation as well as support generation strategies of the current version of the proactive UI. We aim to modify the prototype to further implement the

```
  ?fact<-(user-entry recommend-entry ?action)
  (action (has-name ?action)(has-filler $?fillers))
=>
  (retract ?fact)
  (set-message ?fillers))
```

Fig. 10. Recommendation of a valid entry.

capabilities of the conceptual study presented which will be subject to technical and user evaluation. User tests and interviews, as well as task analysis have to take place in order to refine the support concept. Although the prototype and an illustrative demo have been subject to user interviews, only the integration of the prototype and the communication layer into a business application can provide a basis for accurate empirical studies.

Furthermore, the current rather simple user model needs to be expanded. For example information about the user, such as the preferred way of solving a task, the learning preferences, and already acquired knowledge, could be included.

References

[Ama68] S. Amarel. On representations of problems of reasoning about actions. *Machine Intelligence*, 3, 1968.

[Chi86] D.N. Chin. User modeling in uc: The unix consultant. In *Proc. of the CHI86 Conference on Human Factors in Computing Systems*, pages 24–28, 1986.

[CJB98] B. Chandrasekaran, J.R. Josephson, and V.R. Benjamins. Ontology of tasks and methods. In *11th Knowledge Acquisition Modeling and Management Workshop (KAW'98)*, Banff, Canada, 1998.

[ea86] H.A. Simon et al. Decision making and problem solving, 1986.

[FH03] E. Friedman-Hill. Jess in action. rulebased system in java, 2003.

[FH05] E. Friedman-Hill. Jess information, 2005.

[FN71] R. Fikes and N. Nilson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[GMF+03] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of protege: An environment for knowledge-based systems development. *International Journal of HumanComputer Studies*, 58(1):89–123, 2003.

[Gua97] N. Guarino. Formal ontological distinctions for information organization, extraction, and integration. *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, pages 139–170, 1997.

[HH98] E. Horovitz and D. Heckerman. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proc. of 14th Conference on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI, USA, 1998.

[KM05] M. Kaiser and C. Mueller. The shopping scout: A framework for an intelligent shopping assistant. In *Proc. of Workshop of the 2nd Internat. Conference on eBusiness and Telecommunication Networks*, Reading, UK, 2005.

[Lem84] A.C. Lemke. Passivist: Ein passives, natuerlichsprachiges hilfesystem fuer den bildschirmorientierten editor bisy(in german). Master's thesis, University of Stuttgart, Computer Science Department, Stuttgart, Germany, 1984.

[MFRW00] D.L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proc. of the Seventh Internat. Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, Breckenridge, Colorado, USA, 2000.

[MSI96] R. Mizoguchi, K. Sinitsa, and M. Ikeda. Task ontology design for intelligent educational/training systems. In *Position Paper for ITS96 Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs*, Montreal, 1996.

[Nie93] J. Nielson. *Usability Engineering. San Francisco.* California, USA, 1993.

[Opp94] R. Oppermann. *Adaptive User Support: Ergonomic Design of Manually and Automatically Adaptable Software.* Hillsdale, New Jersey, USA, 1994.

[Pol57] G. Polya. *How to Solve It.* Princeton University Press, New Jersey, USA, 1957.

[RN03] S.J. Russel and P. Novig. *Artificial Intelligence: A Modern Approach*, volume Second Edition. Upper Saddle River, New Jersey, USA, 2003.

[Sch91] T. Schwab. A framework for modelling dialogues ininteractive systems. *Human aspects in computing: Design and use of interactive system and information management*, 18 B:940–945, 1991.

[Sle85] D. Sleemann. A user modelling frontend subsystem. *International Journal of Man-Machine Studies*, 23:71–88, 1985.

[SS02] R. Sinha and K. Swearing. The role of transparency in recommender systems. In *Extended Abstracts Proc. of Conference on Human Factors in Computer Systems (CHI2002)*, pages 830–831, Minneapolis, Minnesota, NewYork, USA, 2002. ACM Press.

[ST01] P. Sisler and C. Titta. Help is dead. long live help. In *STC Proceedings, 2001 Annual Conference of the Society forTechnical Communication*, pages 543–548, 2001.

[Tim02] S. Timpf. The need for task ontologies in interoperable gis, 2002.

[vWvdV98] M. van Welie and G.C. van der Veer. Ontology for task world models. In *Proceedings DSV-IS98*, pages 57–70, 1998.