TASK MODELS FOR PROACTIVE WEB APPLICATIONS

Melanie Hartmann, Daniel Schreiber
Telecooperation Group, TU Darmstadt, Hochschulstr. 10, Darmstadt, Germany

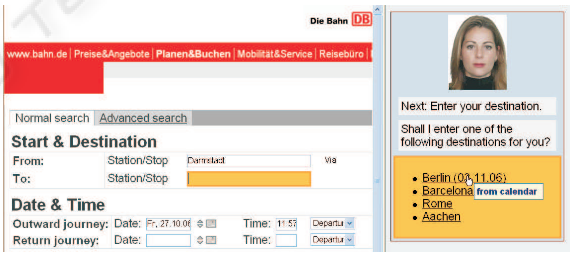
Matthias Kaiser
SAP Research Center Palo Alto, 3475 Deer Creek Road, Palo Alto, CA 94304, U.S.A.

Keywords: Intelligent User Interface, Task Modeling Language, Interface Agents, Human-Computer Interaction.

Abstract: In this paper, we propose a task modeling language for augmenting web applications with a proactive user interface (PUI). PUIs cooperate with the user to accomplish his tasks and facilitate the usage of web applications. They provide user-sensitive support mechanisms and adapt the interface to the user's needs and preferences. Further, they can suggest which data to enter by inferring it from the context or previous interactions. For these purposes, the required knowledge about the application is stored in a task model. We propose a task modeling language that allows to easily enriching the automatically created initial task model with additional semantics. We define requirements for such a language and show that none of the existing languages fulfils all of them. As UML statecharts meet the most requirements, we use them as basis for our task modeling language. We show the applicability of this language and the capabilities of PUIs by enhancing an existing web application with a PUI.

1 INTRODUCTION

Web applications gain more and more functionality, bit by bit reaching the complexity of traditional desktop applications. In addition, many desktop applications are complemented or replaced by their web versions. The increasing complexity of options mostly leads to a decreasing usability of the interface. However, an intuitive and easy-to-use interface is of great importance especially for web applications, because the multitude of existing applications and their simple accessibility enable the user to switch easily to another application. This can be achieved by adapting the interface to the user's needs, by explaining him how to use the application or by assisting the user with his tasks by performing actions on his behalf or by suggesting content for input fields. Proactive user interfaces (PUI) aim at combining all these features in an augmentation of traditional user interfaces. PUIs are a special type of intelligent user interfaces or interface agents that cooperate with the user to accomplish his task, and function as the user's personal assistant. We state that the main features of a PUI are Support Mechanisms (provide online help that adapts


Die Bahn DB
www.bahn.de | Preise&Angebote | Planen&Buchen | Mobilität&Service | Reisebüro
Normal search
Advanced search

Start & Destination

From: Station/Stop Darmstadt Via
To: Station/Stop

Date & Time

Outward journey: Date: Fr, 27.10.06 Time: 11:57 Departur
Return journey: Date: Time: Departur

Next: Enter your destination.

Shall I enter one of the following destinations for you?

- Berlin (03.11.06)
- Barcelona from calendar
- Rome
- Aachen

Figure 1: Screenshot of a PUI that assists the user in booking a train ticket.

to the user and his current working context), Interface Adaptation (adapt the provided options and content to the user's needs and preferences), Content Prediction (suggest data to be entered that is inferred from previous interactions and context information) and Task Automation (recognize usage patterns to automate repetitive tasks).

Support Mechanisms: An inherent problem in user interface (UI) design is that it is hardly ever possible to build a UI that fits the needs of all users and provides everybody with the appropriate amount of functionality. Thus, the application has to i) adapt to each possible user (see *Interface Adaptation*) or ii) offer easy access to the whole functionality. The latter strategy mostly forces the user to find needed functionality using trial and error. This can decrease user-satisfaction, especially if the number of provided options gets too large. The traditional method to cope with this problem is to fall back on classical support mechanisms like manuals or training courses. The most significant drawback of all these mechanisms is that they only provide offline help, i.e., the user is forced to interrupt his work to use them. Although rational behavior would be to read the manual up front as this saves time in total, there are psychological factors preventing users from doing this (Carroll and Rosson, 1987). Microsoft's Office Assistant presents a more advanced support mechanism (Horvitz, E. et al., 1998), as it provides online help according to the user's current task. However, the Office Assistant does not adapt to the user's needs and preferences. Our research prototype proactively displays online support information adapted to the user's needs and preferences in an unobtrusive manner as shown in figure 1.

Interface Adaptation: As stated before, the other way of overcoming the problem of providing the user with easy access to the required functionality is to adapt the provided content and options to the user. A novice user should only be confronted with a basic set of operations to reduce his cognitive load, whereas an expert also needs quick access to more advanced operations depending on his needs. These conflicting demands make it difficult to design a generic interface that fits the needs of all users and thus stress the need for a user-adaptive interface.

Content Prediction and **Task Automation**: A PUI also helps the user performing his task by suggesting how to fill fields and by automating repetitive tasks. Therefore, we need a way to predict the content for input fields based on knowledge inferred from context information or from previous user interactions. This content prediction mechanism recognizes repetitive workflows and enables its automation by providing a generalization of these workflows.

An important issue for PUIs is that the user should never feel losing control of the system nor should the PUI hamper the normal usage of the application. For these reasons, we use different levels of proactive behavior that can be determined by the user. For example, the user can specify to what degree the PUI acts autonomously (just provide suggestions or perform actions autonomously on behalf of the user) or how much support he wants.

The PUI consists of three key components: the **Process Guidance** that assists the user in performing a task (e.g., by providing him with step-by-step instructions, by adapting the interface to his needs, by using context and previous behavior to make suggestions, or by automating repetitive tasks), the **Generation of Justifications** that helps the user understand the system's suggestions and thus induces trust in it and the **Generation of Explanations** that explains terms used within an application or why certain actions are necessary.

All these components need additional knowledge about the web application to be able to provide assistance that is not only based on observed behavior. This knowledge is stored in task models. Ideally, this task model is provided with the web application, otherwise the PUI builds up the task model for the application itself, and allows the user to augment it with additional information.

PUIs can be built into web applications from scratch or can be used as an overlay to existing web applications. The advantage of an overlay architecture is that it relieves programmers of web applications from integrating these features into every single application. Additionally, it easily allows enhancing legacy web applications and sharing knowledge between applications.

The remainder of this paper is organized as follows, in section 2 we give an overview of other intelligent user interfaces and how they gain knowledge for providing support. In section 3.1, we list requirements for a task modeling language for PUIs. We discuss advantages and drawbacks of existing task modeling languages. Next, we present our task modeling language resulting from this discussion that is most suitable for developing proactive web applications. In section 5, we show the applicability of our task modeling language for building PUIs by applying it to a web application. We conclude with a summary and directions for future work.

2 RELATED WORK

There exist two main approaches to intelligent user interfaces: Learning interface agents (LIAs) that have no prior knowledge of the task, and agents that are based on a prebuilt model of the task. Maes, one of the most prominent advocates for LIAs, investigates agents that learn by observing a user working with an application in (Maes, 1994). She states that agents which result from such a learning process are better accepted than pre-programmed expert systems, because they are personalized to the user. The agents' decisions and advices can be explained to the user in an understandable manner, e.g., name the previous observed actions of the user that now lead to the suggestion.

In contrast, the importance of a prebuilt task model is stressed by the Collagen project (Rich and Sidner, 1998). Its goal is to build an architecture for creating collaborating agents, that work together with the user like human experts would. A task model is used as the primary knowledge source for creating the UI of the application as well as the corresponding agent (Eisenstein and Rich, 2002). In Collagen, a hierarchical task decomposition model similar to CTT (Paternò et al., 1997) is used. As the agent has to be created together with the application, the approach lacks flexibility. Moreover, learning patterns that involve multiple applications is difficult with such an approach.

Closely related to our work on task models for web applications is the AGUSINA (Amandi and Armentano, 2004) project that also bases on a prebuilt task model. Amandi and Armentano develop an architecture for separating the agent from the application it works on. This is done by using a modified CTT model as application description. Their agents can offer advice or take actions on behalf of the user. The programmer has to specify when to perform these actions by providing a mapping from encountered situations to the action to take. How and if the created agents learn to enhance themselves is not described in the paper.

In our opinion, the best support can be achieved by combining LIAs with task model driven approaches. LIAs need no modeling of the application in advance, and thus can be easily applied to various applications. However, even Maes concedes in (Maes, 1994) that some minimal background knowledge has to be given to the interface agent. We conclude, that we should allow the developer of the application and also the end-user with minimal technical knowledge to enrich our PUI with as much knowledge as he is able and willing to provide. To encode this knowledge, we need a task model, which has to meet several requirements that are discussed in the next section.

3 ANALYSIS OF EXISTING TASK MODELING LANGUAGES

The task model of an application is composed of the interaction elements of the application. In the case of web applications these are the different form elements and links. It has to be easy to understand and update and has to support constructs for additional semantic information (like dependencies between elements) in a machine readable way. Thus, we define the following requirements for a task modeling language suitable for PUIs:

- **R1: Understandability**. The resulting task models have to be comprehensible for the application designer and for the end-user with minimal IT experience. The easier it is to add additional semantics, the easier it is to improve usability beyond the state reachable with pure machine learning.
- **R2: Incremental modeling**. PUIs may not depend on up front semantic modeling, because not every goal or usage is known during design-time and therefore can not be modeled. Thus, the task modeling language has to support adding semantics later.
- **R3: Conditions**. Conditions refer to any requirements the application imposes on the entered data. This information is used to help the user achieve his goal by telling him which data he still has to provide and which actions he has to perform. For example, a customer relationship management application may require that a principal contact for a customer is defined before an email can be sent to this customer. The PUI uses this knowledge to automatically guide the user to the contact creation mask accompanied by an explanation, in case the user tries to send an email to a customer without principal contact.
- **R4: Dependency relations**. The interaction or the data entered into an application can depend on previously entered data, external sources (context information) or on other user interface elements, e.g., a certain choice may depend on the current location. This restricts the amount of data that has to be considered for predicting data and recognizing usage patterns.
- **R5: Mapping to UI**. The elements of the task model have to be connected to the elements of the UI. This enables the PUI to map user's actions in the normal UI to the task model. This mapping must be easy to generate to gain wide adoption.
- **R6: Mapping to ontology**. Finally, the elements of the task model should be mapped to an ontology to allow the creation of user understandable explanations like in (Wilcock and Jokinen, 2003). This mapping can also be used to gain additional information, like inferring conditions (R3) (e.g., required fields for a valid postal address) or dependencies between elements (R4).

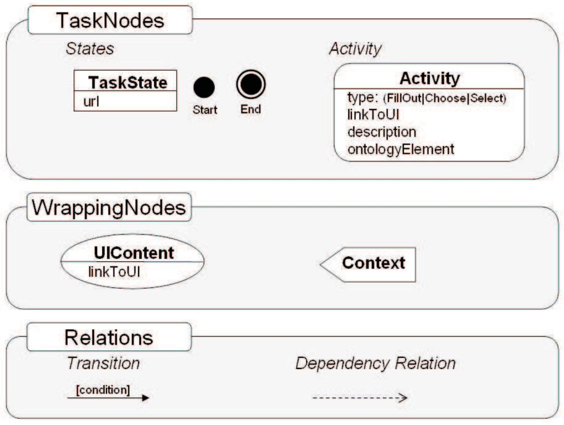


Figure 2: Components of our task modeling language.

3.1 Existing Task Modeling Languages

In this section, we analyze how existing task modeling languages perform with regards to these requirements. We briefly discuss GOMS, STRIPS and CTT. Further, we inspect UML as one of the most widely used modeling languages. Although UML does not have an explicit task model diagram, several diagram types of UML can be used for task modeling. Here we consider activity diagrams and statecharts.

GOMS GOMS (John and Kieras, 1996) represents procedural knowledge required to operate a system by stating its atomic operations like mouse movements or time the user needs to come to a decision. The model is used for studying and comparing performance of existing user interfaces. They model much more than just the interaction with the interface, therefore, the elements are too fine grained for applying it to PUIs.

STRIPS Logical formalisms like STRIPS (Fikes and Nilsson, 1971) have the advantage of having well understood semantics. They are often applied for expressing planning problems. The major drawback is, that they are hard to visualize and understand for a human. Another problem is to consider external sources, because their values do not change deterministically. Thus, it is no appropriate representation for PUIs.

CTT The Concurrent Task Tree (CTT) formalism (Paternò et al., 1997) allows to structure work hierarchically using task / subtask relations. Sibling tasks of the same level can be connected using various temporal relationship operators. To avoid ambiguities in the temporal constraints, it may be necessary to introduce artificial super tasks which are of no value for the user. These make the model harder to understand for the user violating requirement (R1). Further, it is difficult to model conditions (R3) and non-temporal relations between tasks, especially dependence on external sources (R4). Incremental modeling (R2) is also hard with such an approach, as often semantics cannot be added by an edit at one place, and new tasks have to be introduced.

UML 2.0 Activity Diagrams The semantics of UML activity diagrams are based on Petri nets and thus are well defined. Activity diagrams support conditions and dependencies (R3 and R4) in form of Object Constraint Language expressions. However, they do not offer a mapping to UI elements (R5) or to an ontology (R6). Incremental modeling is possible with activity diagrams (R2), as activities are non-atomic and can be refined. In form based web applications, usually many activities can be carried out in an arbitrary order (e.g., filling out an address form starting with first or last name or shipping address). In order to model this, every activity would have to be linked to every other activity resulting in a model that lacks easy understandability (R1).

UML 2.0 Statecharts Statecharts do not suffer from this problem, as web pages can be represented as states and the available actions on this page as transitions. Statecharts allow the modeling of conditions by guard expressions on transitions (R3, R4). Incremental modeling is possible (R2), e.g., the guard expressions of transitions can be transformed into states and transitions themselves. One drawback of statecharts is that it is not possible to refer to user input from previous interactions. Further, as activities are just modeled by transitions without attributes, it is particularly difficult to map UI or ontology elements to activities (R5, R6).

Although workaround solutions to meet the requirements exist for most modeling languages, e.g., by modeling every additional piece of information as extra attribute, these should be avoided, as they introduce unnecessary complexity into the resulting models which decreases understandability and thereby violates (R1). Hierarchical task decomposition is not essential for the user's cognitive model of a web application, because web pages are seldom organized hierarchically and we just need one level of granularity. To build task models suitable for PUIs for web applications we propose a new task modeling language, without focus on hierarchical task decomposition. As

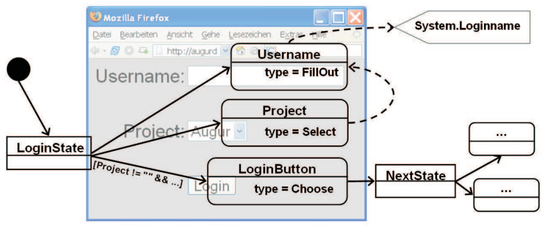


Figure 3: Graphical representation of the task model with underlying web page.

UML statecharts miss only few requirements and are the most intuitive, we chose to build upon them and augment them with activities with richer semantics (e.g., mapping to UI elements or an ontology) as described in the next section.

4 TASK MODELING LANGUAGE FOR PROACTIVE UIS

We chose a task modeling language with a graphical representation, because it is easier to understand (R1). Task models can be represented as directed graphs where the nodes represent **states** and **activities**, using a mixture of statecharts and activity diagrams (see figure 3). We distinguish between states and activities, because this maps naturally to the user's view of a web application where states represent web pages, and activities the different interactors on the page.

An overview of the node types and attributes of our task model can be found in figure 2. We illustrate the elements of our task modeling language with a simple example: A login page containing an input field for the username, a drop down menu for selecting the project to work on and a "Login"-button for submitting the data (see figure 3).

We distinguish three different types of states, the start and end state, representing the start and end of a task without referring to a specific web page, and the task states that refer to a web page. Further, we model three types of interactors: **FillOut** (input field for arbitrary text), **Choose** (clickable UI element) and **Select** (select from a set of predefined items). In our example, we would model the input field for the username as FillOut interactor, the drop down menu as Select interactor and the "Login"-button as Choose interactor.

The state nodes are linked via **transitions** to activities corresponding to the input elements on the site represented by the state. The activity nodes again are linked to the state the user reaches when performing the activity. In the majority of cases, this is the same state as before and the transition from the activity to the state can be omitted in the graphical representation.

Each activity node is coupled to a UI element via its **linkToUI** attribute stating the corresponding XPath expression (R5). **Dependency relations** are also modeled as attributes of the activity nodes (R4). In general, the input of activity nodes can depend on other activity nodes, other UI elements and external information gathered from context sensors. Thus, we also need a way to represent additional UI elements and context data with our task modeling language. For that purpose we introduce wrapping nodes **UIContent** and **Context** for information that is not directly related to the workflow process. The UIContent nodes are also linked to the interface by XPath expressions. The Context nodes hold a reference to the corresponding sensor. In our example, we model two dependencies: the selection of the project is related to the entered username, and the username corresponds to the windows login name that can be retrieved from a context sensor modeled as Context node.

Further, all activity nodes can be linked to an element in a common **ontology** which can be used to generate explanations for these elements (R6). If no ontology is used, the **description** attribute should contain a short explanation of the element. Additional factors that help guide the user through the workflow of a task are **conditions** that state what has to be done to be able to perform an activity (in our example the login-button may only be pressed when a username was entered and a project was chosen) (R3). Conditions are modeled as attributes of the corresponding transitions.

All of the above mentioned attributes are optional, but the more additional information is present the better assistance can be provided (R2). The whole task model for a web application can be partly automatically created by the PUI by analyzing the interactors on a webpage (Paganelli and Paternò, 2003) or by observing the user's interaction with it.

5 APPLICATION OF OUR TASK MODELING LANGUAGE

In order to test our task modeling language, we built a task model and a PUI for an existing web application. We modeled one of the most frequently used functionalities of the Deutsche Bahn website (German Railways, http://www.bahn.de): the booking process for train tickets.

An example of the resulting website augmented with a PUI is depicted in figure 1. The left frame contains the normal website, and the right frame the PUI displaying suggestions and explanations. In the example, the PUI has identified the FillOut activity for the destination as next step. The PUI guides the user by highlighting the corresponding input field of the web application and by suggesting the user what to do next in the PUI frame.

Further, the PUI suggests values to fill in the form element (for FillOut and Select activities). These values are derived from previous interactions or from context sensors specified in the task model. In our example, the PUI suggests four values to be entered as destinations. The first two of them are derived from the user's calendar and the remaining two from previous entered data. The source of the data is shown as tooltip of the suggestions. Moreover, the user can ask for more detailed justifications, including where the data was derived from and why a suggestion was made. Our example shows that PUIs go beyond simple form filling, e.g., by taking into account context data. Our modeling language allows to define a sensor which delivers the value and link it to the corresponding FillOut Activity via the "dependencies" attribute like in our case the calendar sensor. The user still has the possibility to enter any other destination on the website or take a different action. The new data would then be observed by the PUI, and incorporated into future advices. Thus, the PUI does not hamper the normal interaction with the application.

In generating task models for the booking application, our initial assumption that three activity types (FillOut, Choose and Select) are sufficient for modeling interactions could be confirmed. We could also show that it is possible to generate the initial task model automatically with a tool we built for that purpose. Further, it proved to be easy to enhance existing task models with additional semantics, like links to context data, as the corresponding parts of the task model could be easily found with help of the graphical representation.

6 CONCLUSION

In this paper, we presented a step towards improving the usability of web applications with PUIs by creating one important building block: the task modeling language. We defined requirements for such a task modeling language, and showed that none of the existing approaches meets all our requirements. Thus, we defined a new language building upon statecharts fulfilling all requirements. We demonstrated the applicability of our task modeling language with a prototype that augments an existing web application with a PUI.

Building on this task modeling language, we strive to further enhance the support given by the PUI by incorporating knowledge about high level goals of the user. These can be used to deductively generate step-by-step instructions. The support for context data that can be shared by several applications will be improved, as this will be a major step in connecting applications with the needs of the user.

ACKNOWLEDGEMENTS

We would like to thank SAP Research Darmstadt for supporting our research in the AUGUR project.

REFERENCES

Amandi, A. and Armentano, M. (2004). Connecting web applications with interface agents. *International Journal of Web Engineering and Technology*, 1(4).

Carroll, J. M. and Rosson, M. B. (1987). Paradox of the active user. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, chapter 5. MIT Press, Cambridge, MA, USA.

Eisenstein, J. and Rich, C. (2002). Agents and GUIs from task models. In *IUI 2002*. ACM Press.

Fikes, R. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *IJCAI 1971*.

Horvitz, E. et al. (1998). The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. In *UAI 1998*. Morgan Kaufmann.

John, B. E. and Kieras, D. E. (1996). The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Trans. Comput.-Hum. Interact.*, 3(4).

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM*, 37(7).

Paganelli, L. and Paternò, F. (2003). A Tool for Creating Design Models from Web Site Code. *International Journal of Software Engineering and Knowledge Engineering*, 13(2).

Paternò, F., Mancini, C., and Meniconi, S. (1997). Engineering Task Models. In *ICECCS 1997*. IEEE Computer Society.

Rich, C. and Sidner, C. L. (1998). COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3-4).

Wilcock, G. and Jokinen, K. (2003). Generating Responses and Explanations from RDF/XML and DAML+OIL. In *IJCAI Workshop on Practical Dialogue Systems*.