Tailoring the Interface to Individual Users

Melanie Hartmann, Daniel Schreiber, Max Mühlhäuser
Telecooperation Group
TU Darmstadt
Hochschulstrasse 10
64289 Darmstadt, Germany
{melanie,schreiber,max}@tk.informatik.tu-darmstadt.de

ABSTRACT
As the input and output capabilities in Ubiquitous Computing are often very limited, the interaction costs are much higher than for traditional desktop applications. Adapting the interface to the user's needs and preferences is the key to reducing interaction costs and increasing usability of applications. In this paper, we present the AUGUR system that can automatically generate such user-adapted interfaces. We present the architecture of AUGUR along with the task and user model applied by it. Further, this paper introduces the FxL* algorithm that determines which user interface elements to render for an individual user in a given situation. We show that it clearly outperforms algorithms that do not consider the user's situation.

Author Keywords
Proactive User Interfaces, Intelligent User Interfaces.

ACM Classification Keywords
H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General Terms
Design, Economics, Human Factors

INTRODUCTION
Nowadays computer applications are used in more and more diverse settings pushing us into the era of Ubiquitous Computing. Modern mobile devices contain enough computing power, battery lifetime, and network connectivity to run standard desktop applications. They can be easily carried around, but are hence restricted in size and thus have limited input and output capabilities. This causes many difficulties for the user interface (UI). The UI has to cope with more constraints and restrictions than a UI for a desktop application. Further, mobility often also leads to limited attention of the user [13]. Users cannot always direct their full attention towards the UI, as this might be socially inacceptable, e.g., in a restaurant, or even dangerous, e.g. while crossing a busy street. These factors lead to very high interaction costs in Ubiquitous Computing compared to desktop settings, making usability an even more important issue.

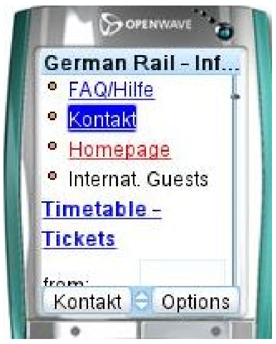
The amount of different mobile devices with varying capabilities makes it nearly impossible to deploy specially tailored and highly usable interfaces for every device. Various approaches for abstracting from these low level details have been proposed. These techniques adapt an abstract UI at runtime or compile-time to a given device, which saves a lot of manual work for the UI designer. A common limitation is that these techniques adapt a UI only to device constraints and to some degree to environmental constraints, like noisy surroundings. However, they do not employ a user model and do not adapt the UI to the specific needs of an individual user.

Due to the high interaction costs for ubiquitous computing applications, we state that the key for increasing the usability of UIs for Ubiquitous Computing is to build user-adaptive interfaces. A special form of user-adaptive interfaces is called *Proactive User Interface* (PUI). The four main features of a PUI are:

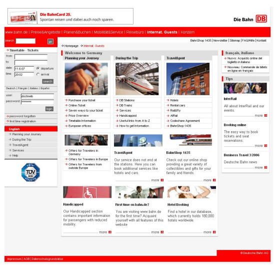
- **Support Mechanisms**: provide online help that adapts to the user and his current context
- **Interface Adaptation**: adapt the provided options and content to the user's needs and preferences
- **Content Prediction**: suggest data to be entered that is inferred from previous interactions and context information
- **Task Automation**: recognize usage patterns to allow automation of repetitive tasks.

A PUI can reduce the interface to the needed functionality on devices where screen space is very limited or even render it as voice interfaces. Further, it can use additional available modalities like speech for providing the user with information without taking up screen estate or requiring visual attention. The concrete realization of the different PUI features thereby always depends on the user model and on the current context.

In this paper, we present our realization of a PUI called AUGUR. We focus in this paper only on its Interface Adaptation feature, i.e. how knowledge about the user's behaviour can be used for rendering user-adapted applications on devices with limited display. In the following we illustrate this with the example of looking up a train connection on the Deutsche Bahn (German railways) website. The original web interface of this application is shown in Figure 1, as it is displayed in a normal browser and in an emulator for mobile devices. Figure 2 shows how the AUGUR system tailors this application to the needs of an example user.

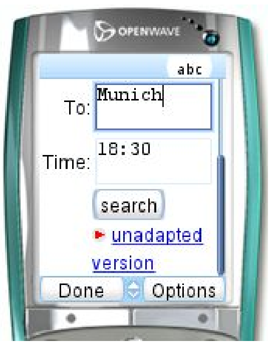


(a) Openwave Browser

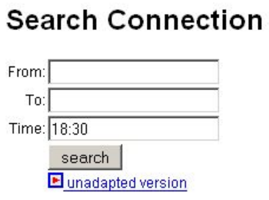


(b) Desktop Browser

Figure 1: Unadapted user interface



(a) Openwave Browser



(b) Desktop Browser

Figure 2: Adapted user interface

In the remainder of the paper, we first relate AUGUR to other approaches for Ubiquitous Computing UIs and UI adaptation, showing the theoretical background that led AUGUR's development. In section "Architecture Overview", we give an overview of AUGUR's main components. For the adaptation, AUGUR needs knowledge about the applications that should be proactively augmented. This knowledge is stored in task models and user models, both models are described in the succeeding section. Next, the FxL* algorithm underlying the UI adaptation process is presented. In the evaluation, we show that an interface with user-adaptation applying this algorithm clearly outperforms UIs that do not consider the users current working context. Finally, we conclude the paper and give some perspectives for future research.

RELATED WORK

Most approaches for UI adaptation described in the literature aim at simplifying the task of adapting to a multitude of platforms and devices instead of adapting to an individual user. They provide a high level, device independent modelling language for describing the user interface. This abstract representation is then automatically transformed into an appropriate presentation for each device. A step further, the CAMELEON project [4] does not only aim at device independence, but on modality independence and thereby introduces further layers of abstraction. [4] describes four layers are described in total, reaching from the domain and concept layer over the modality and device independent abstract user interface, the device independent concrete user interface to the running final user interface. This reference framework is supported by a number of languages from the UsiXML family [12]. A drawback of approaches relying on heavy modelling is the requirement to train UI designers in abstract modeling languages. Even if this was done, building an abstract user interface without seeing the final product is hard. To compensate for this [5] and [2] propose ways to create the needed abstract models from concrete UIs. This allows the designer to build the UI using visual tools, and to infer the abstract model from this model. The same approach can be used with AUGUR, where the role of the concrete model is played by the HTML frontend of a web application and the corresponding abstract model, in a special purpose language called ATML, is automatically created by the system. The ATML models have a scrutable graphical representation which can be overlayed on the web page used to generate them making them easy to understand and edit by the end-user with minimal technical knowledge (see Figure 5), an important point for ubiquitous computing as pointed out e.g. by [1].

The reference framework of CAMELEON does also provide means to adapt to changes of context, like lighting conditions. It is also possible to incorporate static user preferences, but it does not support modelling of dynamic user features, like goals and intends. The presentation of the generated interfaces relies on the static knowledge in the compilers and transformation rules and of course on the abstract models. In contrast to these systems, AUGUR does not primarily aim at providing device independency for user interfaces but to allow user interfaces to be adapted automatically to the user at runtime. We believe that adapting to the user is at least as important as to adapt to device constraints. However, it would be sensible to assume that future systems will incorporate both features for device- as well as user-adaptation.

In contrast to these rule based model transformations, SUPPLE [6] produces a final user interface from an abstract description through numerical optimization of a usability function based on interaction costs. As this is done online, the function can take a dynamic user model into account. Indeed SUPPLE supports limited personalization, e.g. by promoting frequently used UI elements. However, the interface is mainly determined by the static features of the usability function and the device description. The user is modeled in terms of static preferences, e.g. whether she does prefer large buttons.

A mixed initiative approach on personalization is presented by [3]. It relies on an algorithm to suggest customizations of the user interface based on an automatic analysis of interaction costs. The idea of reducing interaction costs is also one of the main goals of the AUGUR approach.

For desktop applications many personalization techniques have been explored, leading to mixed results. An overview of studies on the subject together with a successful adaptation technique can be found in [7]. It shows that automatic personalization can be beneficial in a desktop setting. This indicates that it is even more valuable in ubiquitous computing settings where we have to deal with much higher interaction costs. Thereby approaches are preferable that do not rely on explicit user feedback. [11] provides an overview of applications that apply machine learning techniques for personalization. The user model learned in these applications is mostly concerned with highly application specific concepts This approach is infeasible for ubiquitous computing as it has to be easily applicable to a wide range of different applications and devices. AUGUR faces this challenge by supporting all kinds of HTML application, thus being independent of a specific application domain. Similar is the approach of [14], it does however restrict itself to hierarchical applications, where every point in the application can be reached in only one way and thus avoids to deal with interaction history explicitly.

ARCHITECTURE OVERVIEW

The AUGUR system is designed as an overlay to web applications. Thus, it can be easily applied to existing applications and allows rapid prototyping. The interaction with an applications that is enhanced with a PUI is routed through the AUGUR system as well as the response returned by the web application. This response is augmented and modified depending on the current context and the user's preferences. AUGUR can be installed on the user's device, so that no sensitive data of its user model becomes publicly available. The architecture of AUGUR consists of three major tiers: Knowledge Base, Support Generation and Interaction.

The **Knowledge Base** provides the information needed by the two other tiers. It holds a repository of task models, containing structural knowledge about applications the user interacted with. These task models can be learned by the PUI while the user interacts with unknown applications, they can be provided by application developers and can also be modified by the user with minimal technological knowledge. The Knowledge Base does also contain a user model with information about how the user interacted with the applications and the user's preferences. Details about the knowledge representations used can be found in the next section.

The **Support Generation** tier takes information about the current state of the application and returns support information. An important part of this information are the interaction elements of the UI which will be used next, together with confidence values. For example, that the user will most probably fill out the destination and time after entering the departure information in the train booking example. The FxL* algorithm providing this data is described in detail in Section "Prediction Algorithm". Further, this tier predicts content for input fields depending on the user's interaction history and the current context (also with corresponding confidence values).

Finally, the **Interaction** tier is responsible for interpreting the user input and for adapting the application's output to the user's needs. As all user actions are routed through the AUGUR system, this tier is also responsible for updating the task and user model accordingly. The Interaction tier takes the support information from the Support Generation tier and decides which information and which elements should be presented to the user. Further, it augments the output with proactive features. The output is then rendered to a concrete UI and send to the user's device. If the available display is too small to properly render the output returned by the application, the interface will be reduced to the most relevant parts.

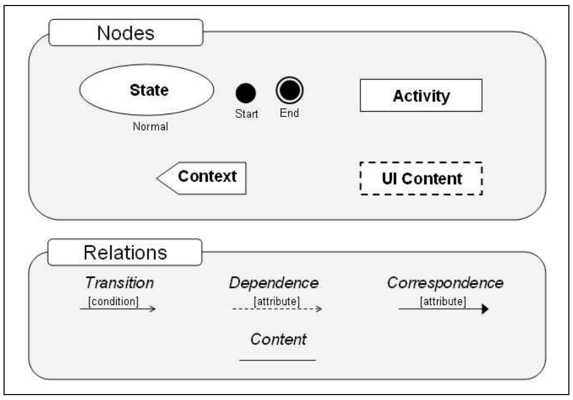


Figure 3: Components of ATML

KNOWLEDGE REPRESENTATION

In this section, we describe the structure of the task and user model needed for a meaningful adaptation. Using the example of the Deutsche Bahn application, we show how these models are build.

Task Model

The structural knowledge about an application is stored in the application's task model. For that purpose AUGUR uses a modeling language called ATML (AUGUR Task Modeling Language) [10] based on statecharts. ATML task models are understood intuitively, enabling the end-user to extend and modify them. In the following, we present the main components of ATML that are relevant for automatically generating a user-adapted interface. An overview of these concepts can be found in Figure 3. For more detailed description of ATML we refer to [10].

In ATML, tasks are modeled as directed graphs where the nodes represent states (visualized as ellipses) and activities (visualized as rectangles). ATML distinguishes 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 a page. Figure 4 illustrates the task model for the train booking example.

We distinguish three different types of states: start and end state, representing the start and end of a task without referring to a web page, and "normal" task states that refer to a web page. We also found that the following three different activity types suffice to describe the interaction with a form based web application: FillOut, Select and Click Activities. FillOut activities refer to input fields, Select activities to elements for selecting an alternative from a list and Click activities to clickable UI element, e.g. a button or a checkbox.

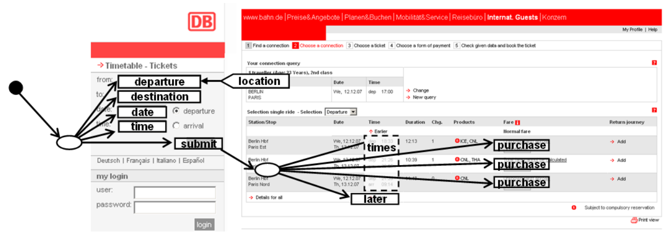


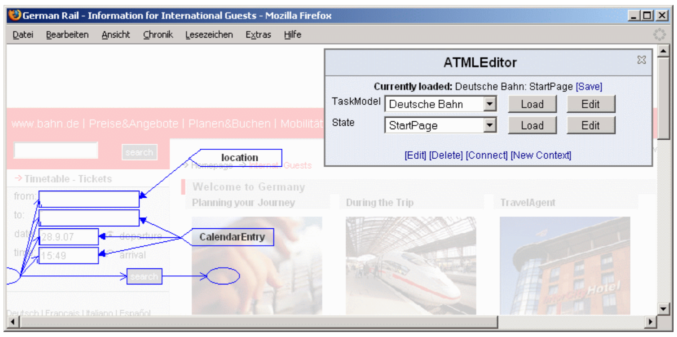
Figure 4: Example task model for the train booking application

The state nodes are linked via transitions to activities corresponding to the input elements on the corresponding site. The activity nodes again are linked to the state the user reaches when performing the activity (e.g. an "OK" button mostly leads to a new website and thus to a new state in the task model). However, 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. The transitions can also contain conditions when an activity may be used, e.g. the "Search" button may only be pressed if a destination was entered. Each activity node is coupled to a UI element via an XPath expression that unambiguously identifies the corresponding interaction element.

If AUGUR needs to generate a scaled down UI of an application, it needs to know which information provided by the application if of interest to the user. Only these parts should be displayed, e.g. the departure times if the user is searching for a train connection. These parts of the interface can be modeled in ATML with UIContent nodes. Such nodes are coupled to a specific UI element via an XPath expression and linked to the corresponding state with a "content" relation. Unfortunately, this introduces some modeling effort as it is very difficult to automatically identify the relevant non-interactive elements. However, this may be less important in a ubiquitous computing environment as the application's feedback may be conveyed through the environment. For example interacting with a home control system for changing the lighting condition gives implicit feedback by e.g. dimming the light, thus making explicitly displayed feedback redundant.

What data is entered, which button is pressed or what information is relevant often depends on the user's current context. For that purpose, we introduce context nodes that refer to a specific context source, e.g. the user's point of departure is often his current location. These can be linked to activities with one of two relations: Correspondence (the value reported by the context source corresponds to the value that needs to be entered) and Dependence relation (the entered data somehow depends on the context information, but is not necessarily the same). The details of integrating context knowledge is out of the scope of this paper.

Most of the above mentioned information is optional, but the more additional information is specified, the better assistance can be provided. For easy editing of task models, AUGUR provides a visual editor as shown in Figure 5. Ideally, an initial taskmodel is provided by the application developer


German Rail - Information for International Guests - Mozilla Firefox

ATMLEditor

Currently loaded: Deutsche Bahn: StartPage [Save]

TaskModel: Deutsche Bahn — Load — Edit

State: StartPage — Load — Edit

[Edit] [Delete] [Connect] [New Context]

Figure 5: ATML Editor

as not all constraints and dependencies can be learned reliably only by observation. However, the initial task model can also be automatically created by monitoring the user's interaction with an application. This way, the task model only contains the interaction elements the user really needs. In order to adapt to the user's needs and to reduce the effort for the user, AUGUR constantly observes the user's interactions to learn more relations between the activities and the context. These relations are tagged with an additional attribute to mark them as inferred relation and with a confidence value. This confidence is determined by the fraction of observed interactions that support the relation.

User Model

AUGUR assumes that the user's behaviour can be inferred, at least to some degree, from her interaction history, which is also supported by the analysis of our datasets. The actions contained in the interaction history refer to elements in a task model, thus allowing to add more semantics to them. In the example taskmodel in Figure 5, a typical sequence may be to type in the place of departure, the destination and then submit the form. We do not store the complete user history, but only the frequency of observed sequences up to a certain length in a trie structure. Each sequence is represented by a node in the trie. The elements of the sequence correspond to the symbols on the path from root to the node. The value stored in the node besides the actual action represents the absolute frequency in the interaction history. Figure 6 shows an interaction history of the example after 24 usages, e.g., the highlighted sequence "departure, submit" occurred 4 times in the interaction history.

For determining the next actions the user will most likely perform, we use the FxL* algorithm as described in the next section. Considering the train booking example, we see that the user frequently accessed the input fields for the point of departure, the destination and the submit button in that order. Thus, the PUI automatically reduces the interface to these elements as can be seen in Figure 2. Besides the most important interaction elements, the adapted UI contains a link to the unadapted user interface, thus still allowing access to the whole functionality. In cases where the user wants to access new functionality not present in her application task model, she can always fall back and use the unadapted interface.

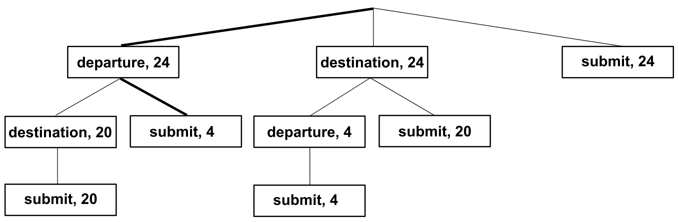


Figure 6: Example usage sequences for the train booking application

PREDICTION ALGORITHM

In [9] we evaluated different algorithms for predicting the next user action, so-called sequence prediction algorithms (SPAs). A SPA returns for a given sequence $a_1...a_i$ a probability distribution P over all possible next actions. We found that the FxL algorithm [9] is best suited for applying it in PUIs as it has the best prediction accuracy and very little demands regarding computational and memory ressources. Its predictions are based on a usermodel as described in the previous section. However, for reducing the interface to the most relevant functionality it is not sufficient to predict only the next action, we need to know the next n actions the user will most probably use. Thereby, n depends on the size of the available display and on how much additional information is presented, i.e. how many interaction elements can still be displayed. For that purpose, we use the FxL algorithm and extend it as follows (resulting in FxL*): For the most probable actions $x_1, ..., x_n$ returned by FxL, we apply it again on every sequence $a_1...a_i x_i$ with $a_1..a_i$ being the most recent actions. The resulting probabilities are multiplied by the formerly computed probabilities for x_i succeeding $a_1...a_i$ as the probability of an action cannot exceed the probability of its preceding action. Further, the resulting probabilities are merged with the probabilities calculated so far. If a probability was calculated for an action that is already stored, the maximum probability of both actions is taken. This process is repeated n times, so that the maximum depth that needs to be considered (the next n steps) is reached. The following pseudo-code illustrates the simplified behavior of the algorithm:

Algorithm 1 FxL*

```
Require: $a_1...a_i$ Sequence of most recent actions
    $p$ parent probability (initialized with 1)
    $n$ amount of elements to be displayed
    $R$ global hash
Ensure: $R$ contains the probabilities for all actions that they
    will be needed in the next $n$ user actions

1: $P(x|a_1 \dots a_j) = \text{FxL}(a_1...a_j)$
2: for all $x$ do
3:     $q(x) = P(x|a_1...a_j) \cdot p$
4:     $R(x) = \text{MAX}(R(x), q(x))$
5:     if $n > 0$ then
6:         FxL*($a_1...a_j x$, $q(x)$, $n-1$)
7:     end if
8: end for
```

The n actions with the highest probabilities are then used for presentation. For providing easy access to the interface elements, they are ordered by the sequences in which they will most probably occur and not just simply by probability.

EVALUATION

For evaluating the benefits of a user-adaptive interface, we compared the interaction costs of three interfaces with different adaptation strategies: the first two UIs use user-adaptive strategies and operate incrementally, this means they update their user model after each observed action. One strategy applies FxL* for computing the most probable next actions, whereas the other presents the actions the user has most often used so far, neglecting the current interaction history (user-adapted maximum). The third unadapted strategy bases on a static generalized user-model, as is today mostly applied for interface design. This strategy presents the actions to the user that are most frequently used averaged over all users (static maximum). We chose this strategy because it represents the best usability reachable by a non user adaptive interface, as this has to optimize for the average user. For the evaluation we applied the three strategies on three real usage datasets: The Greenberg dataset [8] containing UNIX commands, the CrossDesktop log data from a web application for managing files and emails[1] and the Word dataset with logs of MS Word usage[2].

We varied the amount of elements n that can be displayed at the same time (corresponding to different display sizes) and measured how often the action that is performed next can be found among the currently presented elements. We recalculated the elements that are displayed whenever an action was requested that was not present among the current elements. Due to space limitations we only present the results for the word dataset in Figure 7. The “static maximum” version performs only slightly worse than the “user-adapted maximum” version for the word datasets, in the other two datasets even no difference could be detected. This indicates that the frequently used actions for the given datasets are the same for most users. However, the user-adaptive FxL* strategy clearly outperforms the two maximization strategies (the difference in the hit ration ranges for $n \in [2, 10]$ from 6.0% to 27.1% for the Word dataset, 25.3% to 30.2% for the Greenberg dataset and 3.2% to 15.2% for the CrossDesktop data). The hit ratio ranges from about 47% to 86% for all dastasets (Word: 48% to 83%, Greenberg: 47% to 83% and CrossDesktop: 50% to 86%). This shows that local presentation strategies depending on the recent interaction history model the user's behaviour much better than global strategies.

A better estimation for the benefit of user-adapted interfaces using FxL* can be determined as follows: If the next action is part of the currently displayed elements in p percent of all cases, we have to change to the unadapted version in $1-p$ percent of all cases. We now assume that the costs for using one of the n elements in the adapted version or for changing to the unadapted version is c_a and that the average costs for selecting an action in the unadapted version is c_u. Thereby the interaction costs can be seen as the amount of clicks,

[1]http://www.crossdesktop.com

[2]http://www.cs.rutgers.edu/ml4um/datasets/

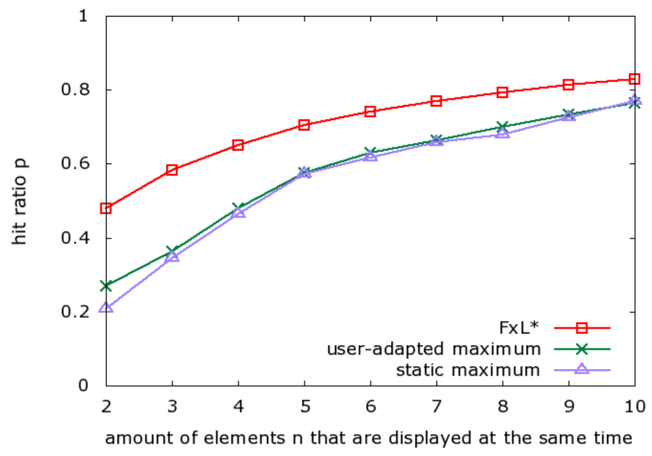


Figure 7: Evaluation results for word dataset. The y-axis represents how often the next user action was currently displayed on the UI.

navigational movements and keys that need to be pressed to interact with this element. The value of c_u and c_a depend on the individual user. We use the time taken per interaction as estimation. The benefit b of using the adapted version can then be calculated as difference between the average costs for selecting an item in the unadapted version (c_u) and the average costs in the adapted version (c_a in p percent of all cases, and otherwise $c_a + c_u$, i.e. c_a for selecting the link to the unadapted version and c_u for performing the operation in the unadapted interface):

$$b = c_u - [p \cdot c_a + (1-p)(c_a + c_u)] = p \cdot c_u - c_a$$

Thus, using an adapted version is beneficial if $c_u/c_a \geq 1/p$. For example, if four elements can be displayed, c_u/c_a has to exceed 1.5 for the word dataset. This value can easily be reached as more than 100 actions (or even 1000 for the Greenberg dataset) need to be made available by the unadapted interface compared to four actions by the adapted one.

CONCLUSION

In this paper, we presented the AUGUR system that allows to automatically tailor existing web applications to the needs of an individual user. This is especially important in Ubiquitous Computing, where the interaction costs are high. We presented here the FxL* algorithm for predicting the next actions the user will most probably perform depending on his past behavior that is stored in a user model. Further, we showed that this adaptation strategy is superior to strategies that do not take the user's current situation into account.

As stated before, the automatic identification of the (non-interactive) information that is relevant for the user is a challenging task. Although we often have to deal with implicit feedback in Ubiquitous Computing, there are still a multitude of applications that require to explicitly display feedback (e.g. the train booking application). For that purpose we want to target this issue in our future work. Towards the adaptation to devices we aim at providing the user with the ability to use different devices concurrently at his discretion, fusioning the input data at the PUI level. We suppose that this further reduces the user's interaction cost by liberating her from switching devices.

ACKNOWLEDGEMENTS

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

REFERENCES

1. M. Assad, D. J. Carmichael, J. Kay, and B. Kummerfeld. PersonisAD: Distributed, active, scrutable model framework for context-aware services. In *Pervasive*, pages 55–72, 2007.
2. L. Bouillon and J. Vanderdonckt. Retargeting of Web Pages to Other Computing Platforms with VAQUITA. In *WCRE '02*, page 339, Washington, DC, USA, 2002.
3. A. Bunt, C. Conati, and J. McGrenere. Supporting interface customization using a mixed-initiative approach. In *Proceedings of IUI '07*, pages 92–101, New York, NY, USA, 2007. ACM.
4. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15:289–308(20), June 2003.
5. M. Florins and J. Vanderdonckt. Graceful degradation of user interfaces as a design method for multiplatform systems. In *Proceedings of IUI '04*, pages 140–147, New York, NY, USA, 2004. ACM Press.
6. K. Gajos and D. S. Weld. Supple: automatically generating user interfaces. In *Proceedings of IUI '04*, pages 93–100, New York, NY, USA, 2004. ACM Press.
7. K. Z. Gajos, M. Czerwinski, D. S. Tan, and D. S. Weld. Exploring the design space for adaptive graphical user interfaces. In *Proceedings of AVI '06*, pages 201–208, New York, NY, USA, 2006. ACM Press.
8. S. Greenberg. Using unix: collected traces of 168 users. Research report 88/333/45, 1988.
9. M. Hartmann and D. Schreiber. Prediction algorithms for user actions. In *Proceedings of ABIS 2007*, pages 349–354, Sept. 2007.
10. M. Hartmann, D. Schreiber, and M. Kaiser. Task Models for Proactive Web Applications. In *Proceedings of WEBIST 2007*, pages 150–155. INSTICC Press, Mar. 2007.
11. P. Langley. Machine learning for intelligent systems. In *AAAI/IAAI*, pages 763–769, 1997.
12. Q. Limbourg and J. Vanderdonckt. USIXML: A user interface description language supporting multiple levels of independence. In *ICWE Workshops*, pages 325–338, 2004.
13. M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 2001.
14. B. Smyth and P. Cotter. The plight of the navigator: Solving the navigation problem for wireless portals. In *Proceedings of AH '02*, pages 328–337, London, UK, 2002. Springer-Verlag.