Context-Aware Intelligent User Interfaces for Supporting System Use

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades Dr.-Ing.

von

Dipl.-Inform. Melanie Hartmann

geboren in Seeheim-Jugenheim

Referenten: Prof. Dr. Max Mühlhäuser (TU Darmstadt)
Prof. Dr. Rainer Malaka (Universität Bremen)

Tag der Einreichung: 4. Dezember 2009
Tag der mündlichen Prüfung: 18. Januar 2010

Darmstadt 2010
Hochschulkennziffer D17

Ehrenwörtliche Erklärung[1]

Hiermit erkläre ich, die vorgelegte Arbeit zur Erlangung des akademischen Grades "Dr.-Ing." mit dem Titel "Context-Aware Intelligent User Interfaces for Supporting System Use" selbständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben. Ich habe bisher noch keinen Promotionsversuch unternommen.

Darmstadt, den 4. Dezember 2009

Melanie Hartmann

[1] Gemäß §9 Abs. 1 der Promotionsordnung der TU Darmstadt

Abstract

Two trends can be observed in current applications: the amount of functionality offered is constantly increasing and applications are more often used in mobile settings. Both trends often lead to a decrease in the usability of applications. This effect can be countered by Intelligent User Interfaces that facilitate the interaction between user and application. In order to support the user in an optimal way, the Intelligent User Interface has to be aware of the user's needs and adapt the provided support to her current situation, i.e. her current context. The relevant context information can thereby be gathered from the environment (e.g. her current location) or from sophisticated user models that reflect the user's behavior.

In this thesis, we present a novel approach of combining user and environmental context information for supporting the system use. As context information is often very error-prone and the user‘s workflow should not be disrupted by erroneous interaction support, we adapt the presentation of the support to the reliability of the context information. Therefore, we use different levels of proactivity from unobtrusive highlighting to automatically performing tasks. The presented approach –called AUGUR– is application independent and is able to support the interaction for arbitrary existing applications, even across application boundaries. For that purpose, we developed a novel application modeling language that is able to model applications and their relationships to context. The application models can thereby be defined by the application developer, learned by AUGUR, and augmented by the end-user.

The interaction can be facilitated by an Intelligent User Interface in two different ways: on the one hand by supporting the entering of data and on the other hand by simplifying the navigation within and between applications. For supporting the user in **entering data**, we contribute three approaches based on (i) the user's previous interactions, (ii) the information represented in the application models, and (iii) the semantics of the data required by the user interface and the context information currently relevant for the user. For the latter, we developed a novel algorithm that combines string-based and semantic similarity measures.

AUGUR supports the user's **navigation** in three different ways: It can (i) guide the user through an application, (ii) provide navigation shortcuts to other applications, and (iii) reduce the user interface to the most relevant functionality for mobile use.

For guiding the user, we developed a novel algorithm called FxL that is able to predict the next relevant interaction element. For the interface adaptation, we introduce a novel approach based on FxL to determine the elements that should be presented to the user for mobile use according to her current situation.

We realized the developed concepts of AUGUR in a working prototype. Further, we evaluated the usability of the context-aware support provided by AUGUR in a user study, and showed that it can significantly increase the usability of an application.

Zusammenfassung

Heutige Anwendungen werden zum einen immer komplexer und zum anderen zunehmend auf mobilen Endgeräten verwendet. Beide Faktoren beeinträchtigen häufig die Gebrauchstauglichkeit der Anwendungen. Intelligente Benutzungsschnittstellen wirken dem entgegen indem sie den Benutzer bei der Interaktion mit einer Anwendung unterstützen. Um die optimale Unterstützung bieten zu können, muss sich die intelligente Benutzungsschnittstelle an die Bedürfnisse des Benutzers und an seine aktuelle Situation, d.h. seinen Kontext, anpassen. Kontextinformationen können dabei aus der Umgebung des Benutzers gewonnen werden (z.B. sein aktueller Aufenthaltsort) oder aus Benutzermodellen, die das Verhalten des Benutzers widerspiegeln.

In dieser Arbeit präsentieren wir einen neuen Ansatz zur kontextsensitiven Interaktionsunterstützung, der sowohl den Umgebungs- als auch den Benutzerkontext berücksichtigt. Da Kontextinformationen häufig fehlerbehaftet sind und der Arbeitsfluss des Benutzers nicht mit fehlerhafter Unterstützung gestört werden soll, müssen wir die Zuverlässigkeit der genutzten Kontextdaten bei der Interaktionsunterstützung berücksichtigen. Daher passen wir die Darstellung der Interaktionsunterstützung an die Zuverlässigkeit der zugrunde liegenden Daten an. Die Darstellung reicht von unaufdringlichem Hervorheben relevanter Elemente bis zur automatischen Ausführung von Anwendungsschritten. Der vorgestellte Ansatz zur kontextabhängigem Interaktionsunterstützung, genannt AUGUR, ist anwendungsunabhängig und kann den Benutzer auch über Anwendungsgrenzen hinweg unterstützen. Dies wird durch eine neue Anwendungsmodelierungssprache ermöglicht, die uns erlaubt, Anwendungen und ihren Zusammenhang zu Kontextinformationen zu modellieren. Die Anwendungsmodelle können dabei vom Anwendungsentwickler erstellt werden oder von AUGUR erlernt werden. Zusätzlich können sie jederzeit vom Endbenutzer überprüft und erweitert werden.

Wir konzentrieren uns in der vorliegenden Arbeit auf die Unterstützung des Benutzers bei der Dateneingabe und bei der Navigation in und zwischen Anwendungen. Zur Unterstützung bei der **Dateneingabe** verwenden wir folgende Informationsquellen: (i) die bisherigen Interaktionen des Benutzers, (ii) das zugehörige Anwendungsmodell und (iii) die Semantik der von der Anwendung benötigten Daten und des aktuellen Kontexts des Benutzers. Für Letzteres haben wir einen neuen Al-

gorithmus entwickelt, der zeichenfolgenbasierte und semantische Ähnlichkeitsmasse kombiniert und neue Zusammenhänge erlernt.

Die **Navigation** des Benutzers wird von AUGUR auf drei verschiedene Arten unterstützt: AUGUR kann (i) den Benutzer durch eine Anwendung führen, (ii) Navigationsshortcuts zu anderen Anwendungen vorschlagen und (iii) automatisch eine reduzierte Version der Anwendungsoberfläche für mobile Benutzung generieren. Um den Benutzer durch eine Anwendung zu führen, haben wir einen neuen Algorithmus namens FxL entwickelt, der in der Lage ist, das nächste relevante Interaktionselement auf Basis der bisherigen Interaktionen vorherzusagen. Für die Generierung der reduzierten Benutzungsoberfläche stellen wir einen neuen Algorithmus basierend auf FxL vor, der die Interaktionselemente bestimmen kann, die idealerweise dem Benutzer angezeigt werden sollten.

Die entwickelten Konzepte wurden in einem Prototyp umgesetzt und getestet. Darüberhinaus haben wir die Gebrauchstauglichkeit der kontextabhängigen Interaktionsunterstützung in AUGUR in einer Benutzerstudie überprüft. Wir konnten zeigen, dass eine solche Unterstützung die Gebrauchstauglichkeit einer Anwendung signifikant erhöhen kann.

Acknowledgments

This work would not have been possible without the continuous support and encouragement of my colleagues, family and friends over the last years, which I would like to acknowledge here.

First and foremost, I would like to thank my advisor, Max Mühlhäuser, for his unlimited support, excellent advice and faith in my work. I am also grateful to Rainer Malaka (Universität Bremen) for acting as second referee.

I am grateful to all at Telecooperation and RBG for providing me with a friendly and supportive place to work. Especially, I would like to thank Daniel Schreiber for all the fruitful discussions and support (It is really a pleasure working with you). I am also grateful to all the other members in the Telecooperation Group who proofread papers and provided feedback (Andreas Behring, Dirk Schnelle-Walka, Felix Flentge, Tobias Klug, Sebastian Ries, Jürgen Steimle to name only few of them).

I would also like to thank Manuel Görtz and Andreas Faatz from SAP research for their support in the AUGUR project and all the others from SAP we cooperated with. Furthermore, I would also like to acknowledge Holger Ziekow (HU Berlin), Dominikus Heckmann (DFKI), Frederik Janssen (TUD), Anthony Jameson (DFKI), and Christine Müller (Jacobs University Bremen) for their advice. Many thanks are also due to Marcus Ständer, Matthias Beckerle and Markus Miche for their support and for continuously supplying me with chocolate.

I am highly grateful to my brother and parents for their support and patience during the course of this work. Finally, many thanks are due to Torsten Zesch for his unlimited support and for standing all the stressful time with me when finishing the thesis.

Contents

1 Introduction

Users of current applications face the problem of increasing interaction effort caused by two main reasons. Firstly, applications are becoming more and more feature-laden (as evidenced by Figure 1.1). Contrary to the intention, this increase in provided functionality often leads to a decrease in the usability of the application (Sikora and Swan, 1998). Secondly, applications are increasingly used in mobile or ubiquitous computing settings. However, the devices used in these settings usually have limited input and output capabilities. For example, a mobile phone only provides a small screen and few or only very small buttons. This leads to increased effort which hampers the entering of data and reduces the amount of information that can be

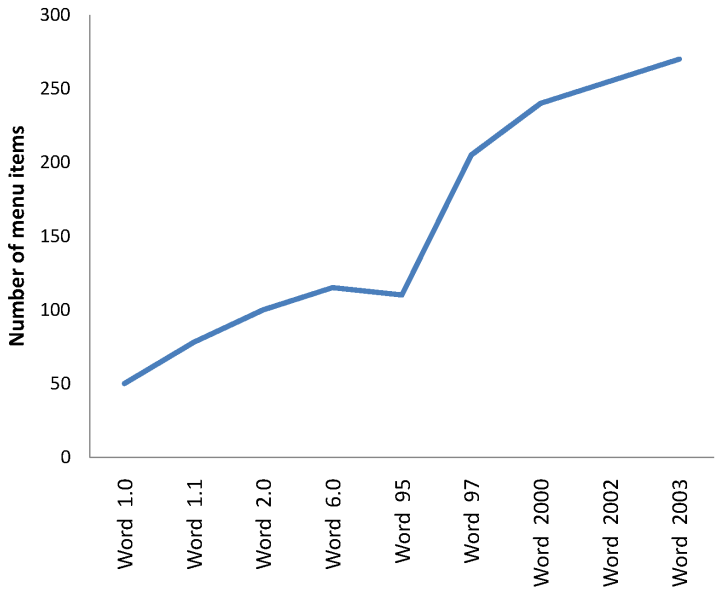


Figure 1.1. Number of top-level menu items in Microsoft Word, by release (adapted from http://blogs.msdn.com/jensenh/archive/2005/10/24/484131.aspx)

conveyed to the user. Furthermore, mobility often leads to limited attention of the user (Satyanarayanan, 2001). Users cannot always direct their full attention towards the application, as this might be socially unacceptable (e.g. in a restaurant), or even dangerous (e.g. while crossing a busy street). These problems can be countered with *Intelligent User Interfaces* which support the user in performing her tasks.

Intelligent User Interfaces Intelligent user interfaces try to improve the efficiency, effectiveness, and naturalness of human computer interaction by applying artificial intelligence techniques. In this thesis, we focus on Intelligent User Interfaces that support the user (i) in entering data (*Content Support*) and (ii) in navigating within and between applications (*Navigation Support*).

Content support assists the user by suggesting data to be entered in input elements. For example, if an address needs to be filled in an input element, the input element could be linked to the user's address database. Whenever the user clicks in the input element, a list of addresses is presented to her from which she can choose the correct one instead of having to type the address herself.

Navigation support guides the user through an application or provides explanations about its functionality. For example, an Intelligent User Interface could highlight the interaction element (e.g. a button or a link) that needs to be activated to finish the current task. Another type of navigation support is to adapt the user interface to facilitate access to the most relevant functionality and information. This helps the user to cope with the increasing amount of features, to focus on the relevant information, and to reduce the required effort, i.e. the required *interaction costs*. For example, on a mobile device, the number of interaction elements presented to the user should be reduced to the most relevant ones.

The amount of content or navigation support that should be provided by an Intelligent User Interface strongly depends on the specific needs of a user. For example, a novice user needs more assistance and a simpler interface than an expert user who needs access to more advanced functionality.

Some tasks involve interaction with several applications, e.g. booking a trip often involves looking up train connections, booking a hotel, etc. In order to provide content and navigation support for those tasks, an Intelligent User Interface should be able to operate with a variety of different applications, i.e. it should be application-independent, and also offer support across application boundaries.

Context-Aware Support The user's current needs are influenced by the context in which the user performs a task. For example, a user usually performs different tasks at home than in the office. Thus, Intelligent User Interfaces should be able to take the current context of the user (e.g. her location) into account, i.e. they should be *context-aware*. This ensures that the interaction support can be better adapted to the user's needs.

We refer to all information as *context* that can be used as auxiliary information for facilitating the interaction, i.e. information which is not required for the normal

functionality. A formal definition of the term "context" is given in Chapter 2. We distinguish between two types of context: *user context* and *environmental context*. The user context contains all information describing the user, e.g. the individual working habits of a user, which are represented in the form of *user models*.

The environmental context comprises all information that cannot be described in a user model, but is related to the user's current situation, e.g. the temperature of a room or the user's current location.

However, context information is often error-prone and hence also the support that an Intelligent User Interface generates from this context information. For that reason, an Intelligent User Interface which uses context information needs to cope with the unreliable nature of context information. For example, it should not perform tasks automatically that rely on uncertain context information.

Application Knowledge In order to support the interaction with an application, an Intelligent User Interface needs knowledge about the application, i.e. which information is required and which information is provided by an application. Depending on how this knowledge is acquired, we distinguish between three types of Intelligent User Interfaces (Maes, 1994): (i) knowledge-based, (ii) end-user programmed, and (iii) learning Intelligent User Interfaces.

Knowledge-based Intelligent User Interfaces rely on elaborate models for each application that provide reliable knowledge, but induce a huge modeling effort. This prevents the provision of support for yet unknown applications. Moreover, interaction support provided by knowledge-based Intelligent User Interfaces cannot be adapted to the individual user's needs, as it relies on predefined knowledge.

End-user programmed Intelligent User Interfaces rely on the end-user to model the required knowledge about the application. This allows the provision of user-adapted support, but involves a large effort for the end-user. Additionally, it requires the end-user to possess a good understanding of the Intelligent User Interface.

Learning Intelligent User Interfaces gather an application model from observing the user's interaction with the application. Thus, they are able to provide user-adapted support and do not require any modeling effort. However, the user cannot be supported right from the first interaction, as it takes some time to learn a reliable application model. Furthermore, learning Intelligent User Interfaces are usually limited to providing support for one specific application.

As each approach has specific advantages and disadvantages, the best support can be provided by combining the three approaches: An Intelligent User Interface should (i) use application models if they are provided along with the application, (ii) enable the end-user to easily modify the application models, and (iii) learn additional information from observing the user and thus enhance and adapt the application models over time.

We have seen that current applications face the problem of increasing interaction costs due to (i) an increasing number of features, and (ii) being used on mobile and

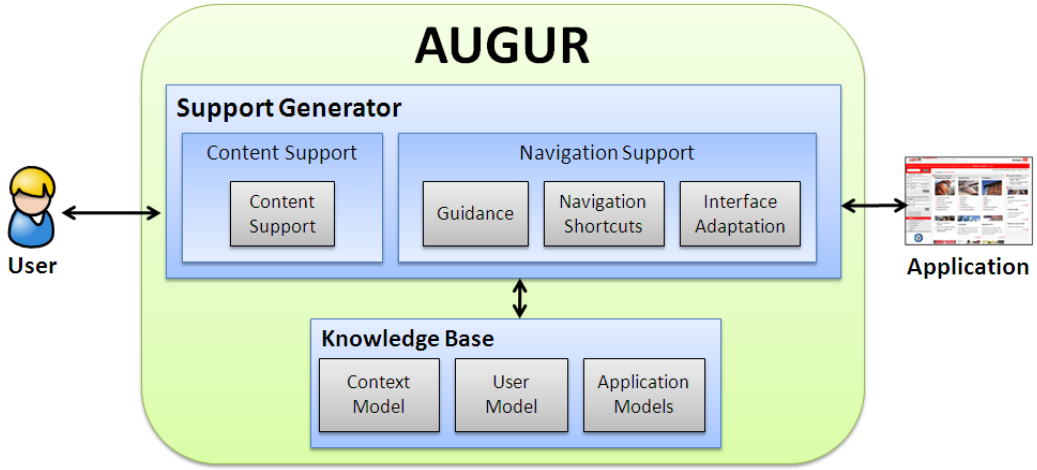


Figure 1.2. Conceptual building blocks of AUGUR

ubiquitous devices with limited interaction capabilities. The problem can be tackled with Intelligent User Interfaces providing content and navigation support. In order to support the user in an optimal way, Intelligent User Interfaces should take user context and environmental context into account, and should combine the three main approaches to acquire knowledge about the applications as described above. In the next section, we introduce a novel approach to Intelligent User Interfaces that considers all these conclusions.

1.1. AUGUR

In this thesis, we present a novel approach (called AUGUR) for using context information in Intelligent User Interfaces. In contrast to existing approaches, we consider the user context as well as the environmental context to provide content and navigation support. Our approach is able to adapt its support to the reliability of the context information. Furthermore, the provided support is not limited to a specific application, but is able to facilitate the interaction with different applications by using a proxy-based architecture. This means that all events evoked by the application and by the user are routed through AUGUR. This enables AUGUR to provide interaction support for existing applications without the need to modify them and to provide support even across application boundaries.

Figure 1.2 gives an overview of the main conceptual building blocks of AUGUR. It contains two main components: The *Support Generator,* which contains all algorithms for generating context-aware content and navigation support, and the *Knowledge Base,* which maintains all knowledge models required for that purpose. In this section, we briefly introduce the knowledge models used by AUGUR and describe how it provides content and navigation support.

Knowledge Models

In order to provide context-aware content and navigation support which is adapted to the user's needs, we at first require a model for the context and the user. The context model is able to keep track of all context information which is currently relevant for the user's interactions. The user model contains information on how the user interacts with applications. Moreover, we need an application model which represents the relations between context elements and application elements. For that purpose, we developed a novel application modeling language called ATML. The ATML application models can be provided by the application developer (*knowledge-based* approach), modified by the end-user (*end-user programmed* approach), and learned by AUGUR from observing the user's interactions (*learning* approach). Thus, AUGUR combines the three approaches for acquiring knowledge about applications discussed before.

Content Support

In order to facilitate the entering of information for the user, we can rely on a-priori modeled information about the applications and their relations to context information. However, these models are not able to reflect the usage patterns of an individual user. Thus, AUGUR observes the user's interactions to learn how the user interacts with an application and adapts the application models accordingly. Besides providing content support based on these *learned and modeled relations* between application elements and context information, AUGUR is able to generate content support based on the user's *previous interactions* and based on the *semantics* of context and UI information. For the latter, AUGUR computes the semantic relatedness of the elements in the UI with the context information currently relevant for the user. AUGUR determines the best mapping between context elements and application elements using a combination of string-based and semantic similarity measures.

Navigation Support

AUGUR supports the user's navigation in three different ways: (i) By guiding her through an application, (ii) by providing navigation shortcuts, and (iii) by reducing the user interface to the most relevant interaction elements. For *guiding* the user through an application, AUGUR highlights the next relevant interaction element in order to draw the user's attention to it. This is especially useful for novice users or when navigating in large menu structures. For predicting the next relevant interaction element, we developed a sequence prediction algorithm called FxL. The predictions are based on models of the user's interactions with an application. For novice users this model can be initiated with the data of an expert user to guide her right from the beginning.

Additionally, AUGUR is able to generate an *adapted version of a user interface* for mobile and ubiquitous use. For that purpose, it reduces the user interface to those

elements that are currently relevant for the user's interactions. For predicting the most relevant interaction elements, we developed an algorithm called FxL*, which is based on the FxL algorithm. It is able to take the user's current interactions as well as the device used into account.

Finally, AUGUR suggests *shortcuts* based on events triggered by an external context event (e.g. an incoming phone call) or by the user interface (e.g. when address information is presented). For example, if address information is contained in a user interface, the Intelligent User Interface can suggest a shortcut to a map application to look up the respective address.

1.2. Main Contributions

In this section, we give a brief summary of the main contributions of this thesis:

- *Overall Concept:* We developed an overall concept for integrating context information into Intelligent User Interfaces. In this concept, we take user context and environmental context into account, and combine the three main approaches for building Intelligent User Interfaces (*knowledge-based systems*, *end-user programming,* and *learning systems*). Our approach is able to provide application-independent support across application boundaries, and to determine which context information is currently relevant for the user's interactions. In addition, we performed a user study to evaluate the effects of using context information in Intelligent User Interfaces on the usability of an application.

- *ApplicaTion Modeling Language* (ATML): We developed a new language to store relevant information about an application, e.g. about its interaction elements and about its relations to context. We also developed an editor for ATML to enable end-users with minimal technical experience to augment the application models.

- *FxL Algorithm:* We developed this sequence prediction algorithm in order to predict the next relevant interaction element in performing a task. FxL learns from observing the user's interactions, and is thus able to provide user-adapted support. The algorithm is applied to guide the user through a user interface. Thus, it facilitates the use of an application and reduces the user's cognitive load by drawing the user's attention to the relevant interaction elements.

- *Interface Adaptation using FxL* Algorithm:* We developed an augmented version of the FxL algorithm that is able to predict which parts of the user interface are most relevant for the user in her current situation. Thus, it enables us to generate a reduced version of a user interface. Depending on the available screen size, we are able to generate a user-adapted version of the user interface of the application which is better suited for mobile usage and thus facilitates navigation.

- *Semantic Mapping Algorithm:* We tackled the problem of deciding which context elements should be suggested as input to the user by applying semantic similarity measures. For that purpose, we compare the textual descriptions available for the UI and context elements. This requires a reliable detection of the label of input elements. This is problematic for Web applications as the tag for marking labels in HTML is scarcely used in practice. For that reason, we developed an algorithm called *LabelFinder* which is able to identify the label from the actual visual representation of the user interface of a Web application.

- *AUGUR Prototype:* We implemented all the developed concepts and algorithms for a context-aware Intelligent User Interface in a working prototype applicable to different (form-based) Web applications.

1.3. Publication Record

This thesis builds on a number of publications in peer-reviewed books, conferences, journals, and workshop proceedings. In the book chapter Hartmann and Austaller (2008), we give a general overview of context and its usage in context-aware applications. In Hartmann (2009), we summarize the challenges which have to be faced when developing user-adaptive Intelligent User Interfaces.

The basic concepts of the context-aware Intelligent User Interface described in this thesis were first presented in Hartmann et al. (2009). In this paper, we describe how context information can be used for content and navigation support. We also introduce the concept of a component for keeping track of all the context information currently relevant for the user's interactions. In Mühlhäuser and Hartmann (2009), we contributed to the identification of problems arising when context information is used for interaction support. In this paper, we stress that the user context as well as the environmental context should be considered for that purpose. In Schreiber et al. (2007, 2008), we report the results of a user study for evaluating the usability of applications which are enhanced with context-aware interaction support.

In Hartmann et al. (2007), we present the application model that we developed for specifying the relations between context and application. Improvements to the model are presented in Hartmann and Schreiber (2009) and Hartmann et al. (2009).

The FxL algorithm for guiding the user is presented in Hartmann and Schreiber (2007). Its extension for generating adapted user interfaces (called FxL*) is published in Hartmann et al. (2008a); Hartmann and Schreiber (2008) and Hartmann and Schreiber (2009). The semantic mapping algorithm for providing content support is described in Hartmann et al. (2008b) and Hartmann and Mühlhäuser (2009).

1.4. Thesis Outline

Figure 1.3 illustrates the overall structure of this thesis. The light bulbs thereby refer to the main contributions as listed in Section 1.2. In Chapter 2, we define the most

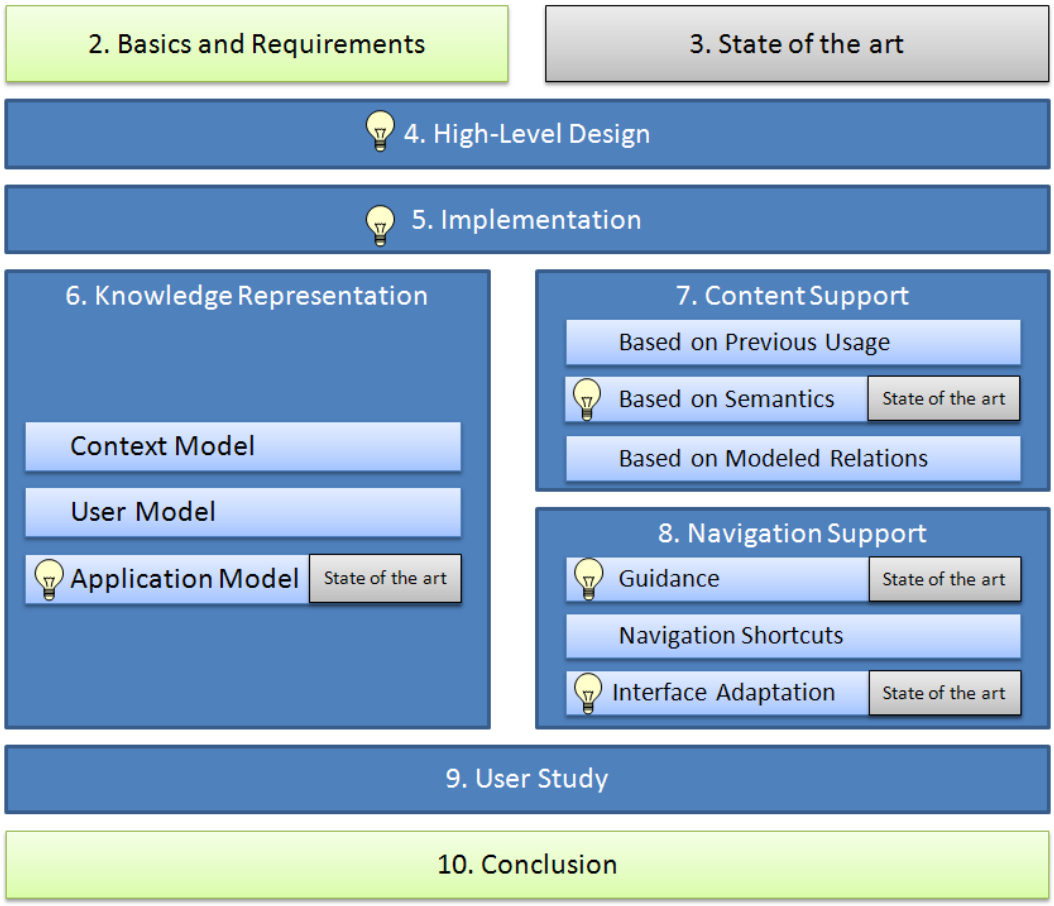


Figure 1.3. Structure of this thesis

important concepts in this thesis –i.e. "context" and "Intelligent User Interfaces"–, and relate our approach to the different research areas subsumed under the term Intelligent User Interface. We also review challenges identified in literature for developing Intelligent User Interfaces, and identify requirements that arise when using context information for supporting the user's interaction. In Chapter 3, we give an overview of state-of-the-art Intelligent User Interfaces which are related to the approach presented in this thesis, and analyze them with respect to the requirements identified in Chapter 2. However, we focus here only on related work considering complete Intelligent User Interface systems. The state of the art for the novel models and algorithms introduced in this thesis is explained in the respective sections. We decided in favor of this structure as we assume that the requirements we pose on the different models and algorithms and thus the analysis of the respective state of the art, can be better understood in this context.

Chapter 4 presents the overall design of the context-aware Intelligent User Interface called AUGUR presented in this thesis. We introduce its different conceptual

building blocks, and show how they meet the identified requirements. In Chapter 5, we then describe our reference implementation of AUGUR for Web applications.

Chapter 6 introduces the different models required for using context information for providing interaction support. We put a special focus on the application model, as it describes the relations between context and application elements. For the application model, we give an overview of the state of the art. We show how our proposed modeling language overcomes the shortcomings of existing languages regarding the usage in context-aware Intelligent User Interfaces.

In Chapter 7, we give an overview of the algorithms for providing content support. We describe how we generate content support based on previous usage, on the semantics of the context and UI elements, and on modeled relations. We also show how these relations can be learned from observing the user's interactions to reduce the modeling effort and adapt the support to the user's needs.

Chapter 8 presents the algorithms for providing navigation support, i.e. for guiding the user through an application, for providing navigation shortcuts, and for adapting the interface.

In Chapter 9, we describe the user study that we performed for evaluating the impact of our context-aware Intelligent User Interface on the usability of an application.

Finally, Section 10 concludes this thesis and discusses further research directions.

2 Basics and Requirements

In this chapter, we describe the basics for using context information in Intelligent User Interfaces (IUIs). To get a common understanding, we first define the meaning of the term "context" in the scope of this thesis (Section 2.1). We also introduce the research area of IUIs and point out the aspects on which we focus in this thesis (Section 2.2). In Section 2.3, we identify the challenges and requirements which we have to consider when using context in IUIs.

2.1. Context

Humans use all kinds of information for characterizing their current situation, like the current time, location, or the identity of persons nearby. They use the information to adapt their behavior to the situation. For example, when we speak to a person, we adapt what we say and how we say it to the social rank of the person (e.g. most people would not say "that's nonsense" to their boss, but they would say it to a friend). None of this information is easily captured, represented, or processed by a computer. However, this information is an extremely important knowledge source for building IUIs, as it enables IUIs to provide support that is better adapted to the current user needs. For example, knowing the current context enables an IUI to filter information which is irrelevant to the current situation and thus to reduce the user's cognitive load. This is especially necessary in the area of Ubiquitous Computing, where the user has to deal with a multitude of different computers and thus with a multitude of possible distractions. Furthermore, the interaction costs in Ubiquitous Computing settings are very high – much higher than in a desktop setting. Using context information to better understand the user's needs enables IUIs to better support her interactions and thus to increase the usability of an application.

To support common understanding of the term context, we provide a definition in Section 2.1.1. We review different categories of context from a computational perspective and from a human computer interaction (HCI) perspective, and advocate the combination of both to gather a better understanding of the user's needs (Section 2.1.2). Finally, we point out which properties differentiate context from traditional information sources (Section 2.1.3).

2.1.1. Definition

The word "context" has its origins in the Latin word "contextus" meaning "to weave together", originally denoting the construction of a text. Nowadays, the term is loaded with a variety of different meanings. According to Merriam-Webster's Collegiate Dictionary[2], context is defined as "the interrelated conditions in which something exists or occurs". Even in computer science itself, context is used with a number of different meanings. For example, context in context-free or context-sensitive grammars refers to the symbols that surround a placeholder and determine which strings can replace it. In contrast, "context" in the area of context-aware computing refers to any auxiliary information that can be used to enhance an application, especially focusing on the interaction with the user. The term "context-aware computing" became popular in the mid-1990s, when researchers started to develop applications that incorporated the current location of users. By then, location-awareness was regarded as the most important subset of context-awareness and often used synonymously.

However, there were also a growing number of applications, such as navigation systems, that did not regard location as auxiliary but rather as mandatory information. These applications required location information in order to provide their normal functionality and could not operate without it. As location-awareness was still regarded as a subset of context-awareness, the latter lost its connotation of using auxiliary information for enhancing the interaction. Thus, the term context became even more difficult to define.

We illustrate the problem of defining context with an example application of a booking process for train tickets. In Figure 2.1, we show some information sources that are available when using the booking application. Some of the information is mandatory for the normal functionality of the system (i.e. the customer number and how many persons want to travel), some information cannot be processed by the application (i.e. the temperature and which people are near the user), and some is optional as it is just used as additional information (e.g. the user's current location and her calendar information). The latter information can be used to infer the station of departure as well as the arrival and travel times, and saves the user from having to enter this information manually.

We define context as follows:

> ***Context*** *characterizes the actual situation in which the application is used. This situation is determined by information which distinguishes the actual usage from others, in particular characteristics of the user (her location, task at hand, etc) and interfering physical or virtual objects (noise level, nearby resources, etc). We thereby only refer to information as context that can actually be processed by an application (relevant information), but that is not mandatory for its normal functionality (auxiliary information).*

[2] http://www.merriam-webster.com/

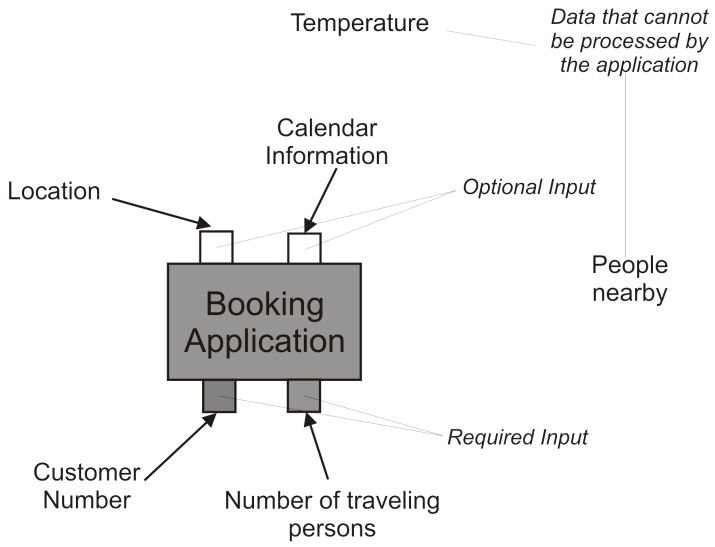


Figure 2.1. Available information sources when using a booking application

Thus, we refer to context as the intersection of "Relevant Information" and "Auxiliary Information" as visualized in Figure 2.2. In this way, our definition overcomes the limitations of existing definitions which we discuss below.

The simplest definitions of context are given by enumerating all constituents of context. For example, Schilit et al. (1994) attempted to define context by specifying three categories of context: computing, user, and physical context. Chen and Kotz (2000) later extended this definition by the time context (e.g. time of day, season of the year).

A common problem of these definitions is that they do not specify a bound, for which information can be referred to as context. According to these definitions everything that fits into one of the categories can be named context, no matter whether or not it has any relevance for the application. In our example application, the temperature and the people nearby would also be referred to as context for the application, even though the application cannot make use of it.

The most prominent context definition by Dey (2001) solves this problem by limiting the context to all information that is relevant to the interaction between user and application: *"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"*. In our example, all information that can be processed by the application would be referred to as context (see "Relevant Information" in Figure 2.2). However, the definition has the shortcoming of defining context by means of other ill-defined terms such as "situation" or "relevance". Another problem that arises from Dey's definition is that even information that the application needs to fulfill its

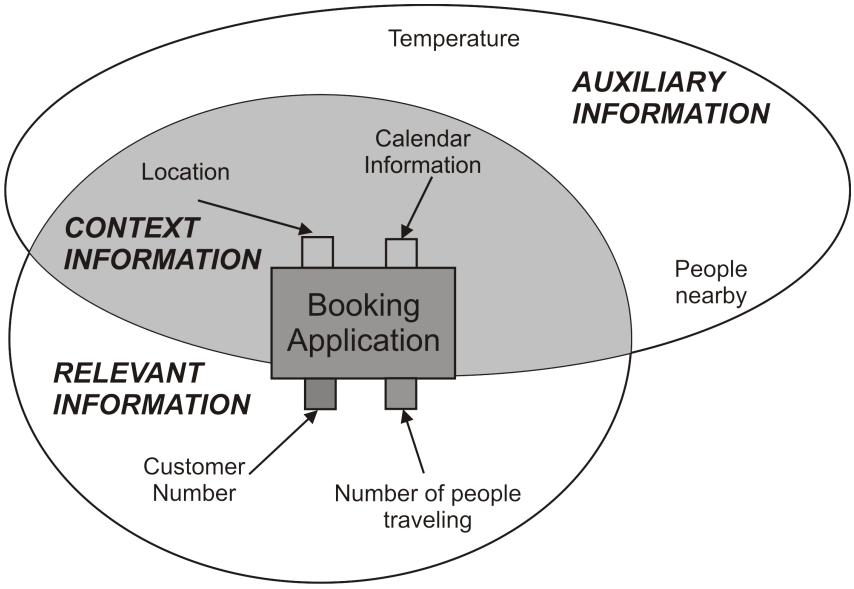


Figure 2.2. Classification of the different information sources from Figure 2.1

tasks, in our example the customer number and the number of people traveling, can be referred to as context. Thus, every application could be called context-aware.

To sum up, we need a context definition that defines an upper and a lower bound of what we consider as context for an application. The upper bound is needed to exclude information that is irrelevant for the application, and the lower bound is needed to exclude information that is mandatory for its normal functionality. These bounds are defined in our definition, so it overcomes the limitations of the existing definitions.

2.1.2. Categories of Context

The research in the context-aware computing community usually takes a computational view on context, because it often focuses on the derivation of context information. For that reason, the widely used categorization (e.g. by Baldauf et al. (2007)) distinguishes between physically sensed, virtually sensed, and derived context:

- **Physically sensed:** Context information that is sensed from the real world, e.g. temperature, acceleration, or location.
- **Virtually sensed:** Context information derived from software sources, e.g. the location included in a calendar entry.
- **Derived:** Higher-level context information which is gathered by filtering, aggregating, interpreting (e.g. mapping GPS coordinates to city names), or combining context information (e.g. infer the user's activity from her location and her movements).

The view of the HCI community is orthogonal to this computational perspective; the considered context information can be roughly classified as user context and environmental context:

- The **User Context** contains all information describing the user. User models which are often used for that purpose can range from simple user profiles to complex models that describe the emotional or cognitive state of the user (e.g. Pantic et al. (2005); Susan and McEvoy (2003)). This also includes models of how the user interacts with applications and which goals she is currently pursuing. These more sophisticated models are usually learned from observing the user and interpreting the interaction with the help of some predefined background knowledge, e.g. about available goals.

- The **Environmental Context** comprises all information that cannot be described in a user model, but is related to the user's current situation, e.g. the temperature of a room or her current location.

Until now, the HCI community focuses on the user context and does not take much environmental context into account. In contrast, context-aware computing puts more emphasis on the environmental context. The user context they consider is usually limited to simple user profiles (e.g. the user's mother tongue). Like Jameson (2001), we argue that we have to consider both context sources – sophisticated user models and environmental context – to bring us closer to understanding the user's needs, and thus to enable us to provide a level of support which is of maximum benefit for the user.

2.1.3. Difficulties in Using Context

Context is difficult to use because it differs from most other data sources used in traditional applications in the following properties:

- Context is gathered from *heterogeneous sources* which use different representations.

- Context is *error-prone*: Context is often acquired by external sensors that can get out of reach, report unreliable data, or completely fail. Furthermore, context is often inferred using machine learning approaches (e.g. activity recognition or sophisticated user models as discussed in the previous section), which induce additional uncertainty.

For those reasons, most applications available today do not take any context information into account and if they do, they only consider a very limited number of context sources, like location.

The problem of heterogeneous sources can be tackled by using a middleware which provides a uniform context representation to the applications. In this thesis, we present a uniform representation which is suitable for using context information in

IUIs. However, we will not go into detail about how this uniform representation is gathered from sensor information. We leave it up to the middleware used (e.g. the Context Server described in Aitenbichler et al. (2007b)) to provide such a uniform representation.

The problem of error-prone context information has to be addressed by every application that uses context information. In Section 2.3, we describe the implications of this property of context for the development of context-aware IUIs.

2.2. Intelligent User Interfaces

The area of Intelligent User Interfaces (IUIs) is a very heterogeneous research subject, covering all kinds of different disciplines, which makes it difficult to give a general definition. IUIs try to facilitate the interaction between user and computer by means of artificial intelligence (AI). In contrast to traditional human computer interaction (HCI), IUIs do not only focus on enabling the user to perform intelligent actions but also on ways to incorporate knowledge to be able to *assist* the user in performing actions. In contrast to traditional research in the area of AI, IUIs do not focus on making the computer smart by itself (e.g. by developing sophisticated problem solvers) but on making the *interaction* between computer and human smarter.

In this section, we give an overview of the research field of IUIs and relate the approach in this thesis to the different research topics subsumed under the term IUI. In Section 2.2.1, we give a definition of IUIs. As one of the most important building blocks of IUIs are models, we further describe the models they rely on. In Section 2.2.2, we present a classification of IUIs based on a literature survey. We describe the two research areas relevant to the approach presented in this thesis, i.e. *Personal Assistants* and *Interface Adaptation*, in more detail (Section 2.2.3 and 2.2.4).

2.2.1. Definition and Models

Giving a clear definition of the term "Intelligent User Interfaces" is almost as difficult as defining context, because already the term "intelligence" is hard to grasp. Malaka (2008) refers to a technical solution as "intelligent" if it (i) has some built-in intelligent computation that solves a problem for the user or (ii) enables the user to solve a problem[3]. The goal of IUIs is to better support the user's current needs by making the interaction itself as well as the presentation of information more effective and efficient. Based on the definition by Maybury and Wahlster (1998), we define IUIs as follows:

> ***Intelligent User Interfaces*** *are human-machine interfaces that aim to improve the efficiency, effectiveness, and naturalness of human machine*

[3] Malaka (2008) only refers to "otherwise unsolvable problems". However, this definition is too strict, because Intelligent User Interfaces aim at facilitating the interaction and not enabling the interaction in the first place.

> *interaction by representing, reasoning, and acting on models of the user, domain, application, discourse, context, and media and device.*

As stated by the definition, IUIs need models to be able to provide meaningful support or adapt the interaction in a meaningful way. We provide below an overview of the different types of models that can be used. Which models are available in an IUI differs from system to system:

- **User Model** contains all information that was gathered about a user, e.g. her skills, preferences, or how she usually interacts with an application. This information can be either modeled or learned from observing the user. Besides modeling individual users, IUIs sometimes also use models of a group of users or of the average user.
- **Context Model** holds information about the current context of use, e.g. lighting conditions. As stated in the previous section, we also consider the user model as context information. However, as the user model is of particular importance for IUIs, we address these two models separately. We subsume all information concerning the user context in the user model and all information concerning the environmental context in the context model.
- **Application Model** contains available information about an application, e.g. its structure, the goals that can be achieved, or help instructions.
- **Domain Model** contains knowledge of a given domain, e.g. the domain 'travel' for travel booking applications. This model contains semantic information about the different terms that are used in this domain to enable the IUI to interpret the user's input.
- **Discourse Model** contains the description of the syntax, semantics, and pragmatics of a dialog as it proceeds (Wahlster, 1988). This is especially important for natural language systems.
- **Media and device model** states the capabilities of available in- and output devices and properties of media. This comprises for example which modalities can be provided by an interaction device, which modalities are needed by a media, or the characteristics of the media e.g. its resolution. These models are needed for example in multimodal environments to determine which modalities should be used in a given setting.

2.2.2. Classification of IUIs

According to Jameson (2007), we distinguish between two main categories of IUIs (see also Figure 2.3):

- IUIs that **support the system use** can perform parts of routine tasks on behalf of the user, adapt the UI, and provide help.

- IUIs for **supporting information acquisition** assist the user to find relevant information, filter data for the user, or provide her with additional information. Recommender systems or eLearning systems are examples of this category.

In this thesis, we focus on IUIs for supporting the system use. For that reason, we take a closer look at the major research areas for this category of IUIs. We identified the following five main research areas and briefly introduce each one of them:

- **Personal Assistants** support the user's interaction by augmenting an application with support information. They are also able to interact with an application by themselves and thus perform tasks on behalf of the user. Their support relies on models of the application that are either provided by the application developer, the end-user, or that are learned from observing the user. Personal assistants are often also called user interface agents, adaptive agents, or learning apprentice systems.

- **Help Systems** can assist the user by explaining how to use an application. In contrast to Personal Assistants, they do not require access to the functionality of the application, as they do not perform any tasks by themselves.

- **Programming by Demonstration Systems** learn to perform a task according to a few examples (usually only one) demonstrated by the user.

- **Interface Adaptation Systems** are able to generate UIs for an application which are adapted to the needs of an individual user, the current context, or the available in- and output devices.

- **Post Desktop Interaction Systems** try to provide a very natural interaction for the user by supporting modalities other than the standard keyboard,

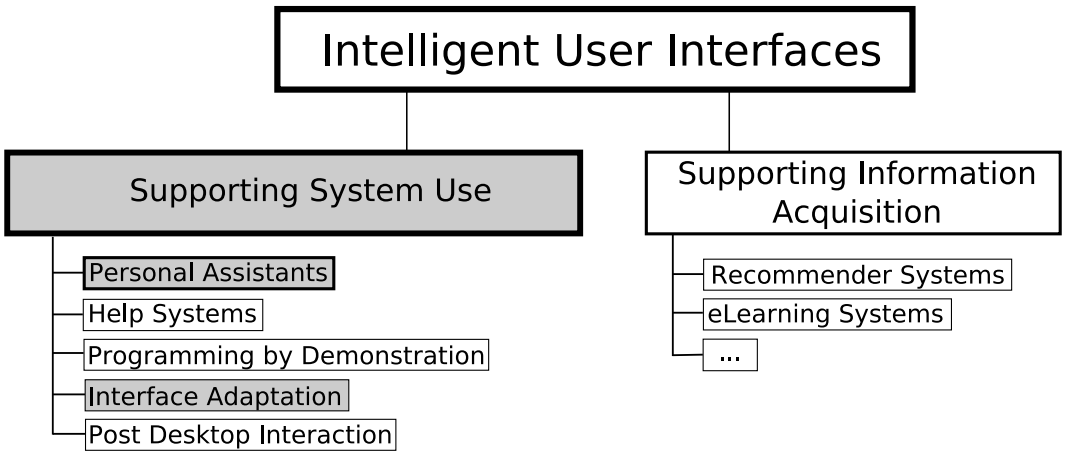


Figure 2.3. Classification of research areas of IUIs (gray research areas: Focus of this thesis)

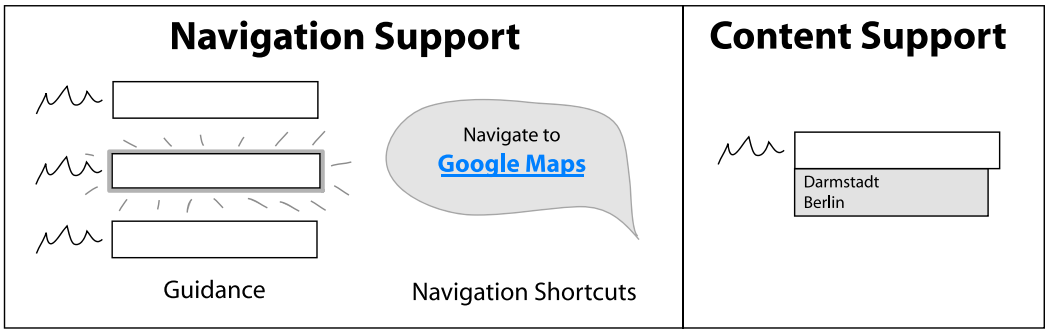


Figure 2.4. Support classes provided by personal assistants

mouse, and screen interaction. For example, they use natural language for the communication or allow the user to interact via various input modalities. Examples of post desktop interaction systems are multimodal user interfaces, gesture- and sketch-based interfaces, or natural language interfaces.

Existing systems often fall into more than one of these research areas, e.g. many personal assistants contain help systems or programming by demonstration systems. In this thesis, we focus on *Personal Assistants* and how they can benefit from context information as they provide the most profound support for facilitating system use. In ubiquitous computing settings, the standard UIs are usually difficult to use due to the limited screen space. To support also the interaction in those settings, we reduce UIs to the given context. For that reason, we also contribute to the area of *Interface Adaptation*. In the following sections, we point out how the user's interaction is supported in these two areas.

2.2.3. Classification of Personal Assistants

We analyzed the state of the art of personal assistants to identify in which ways they support the user's interactions. We found that we can distinguish between two categories (see Figure 2.4):

- **Navigation support** helps the user in navigating within and between applications. This comprises guiding her through an application and providing navigation shortcuts.

  Guidance highlights interaction elements to draw the user's attention to it. It thus reduces the user's cognitive load and is especially useful for navigation in large menu structures or for novice users. Guidance is applied for example in COLLAGEN (Rich et al., 2001) and AGUSINA (Amandi and Armentano, 2004).

  Navigation shortcuts provide the user with shortcuts to applications she might want to switch to depending on the information which is presented to her. For example, a shortcut to a map application can be suggested if address

information is presented. Navigation shortcuts are for example provided by CyberDesk (Dey et al., 1997), onCue (Dix et al., 2006), and FolderPredictor (Bao et al., 2006).

- **Content support** predicts which data should be entered, and suggests it to the user or even fills it into the corresponding elements. This reduces the required interaction costs for entering the data, which is especially important for mobile applications (see Rukzio et al. (2008)). Content Support is applied for managing meetings for example by LookOut (Horvitz, 1999) or CMRadar (Modi et al., 2005) and used more generally for example by VIO (Zimmerman et al., 2007) or Citrine (Stylos et al., 2004).

In some applications **Task Automation** is also named as an additional method of supporting the interaction. The techniques needed to automate a task are the same as for providing navigation or content support: The system has to know which data to enter or where to navigate to. For that reason, we consider task automation as content and navigation support that is executed in a very proactive way.

2.2.4. Classification of Interface Adaptations

Interface adaptation systems aim to generate interfaces that best fit the current usage situation. Thereby, they can adapt *what* is shown to the user and *how*. Thus, we can distinguish between the following two classes:

- **Adapting the content** (i.e. *what* to present) refers to adapting the displayed information as well as the structure of the UI. For example, systems focusing on this aspect can provide a reduced version of a UI for mobile usage or for reducing the user's cognitive load.
- **Adapting the presentation** (i.e. *how* to present) refers to adapting the modality used and how the different interaction elements, e.g. buttons, are presented. For example, systems focusing on this aspect increase the button size for touch screen displays or for persons with disabilities. Another way of supporting the interaction is to automatically generate UIs for another modality, e.g. using voice.

In this thesis, we focus on the content adaptation and on reducing the UI to relevant interaction elements, as this is very important for the ubiquitous usage of applications. We also support to render the reduced UI using another modality, but we do not support further adaptation features for this aspect.

2.2.5. Summary

In this section, we introduced the research area of IUIs. The approach presented in this thesis falls into the research area of Personal Assistants. Personal assistants focus on collaborating with the user to facilitate the entering of data (Content Support)

and navigation (Navigation Support). However, they do not provide any assistance for the use of applications in mobile and ubiquitous settings. As the navigation in these settings can be dramatically enhanced when using a reduced version of a UI with the most relevant functionality, we also contribute to the research area of Interface Adaptation. As stated before, our approach also supports the rendering of the reduced UI using another modality, but we will not go into detail of this aspect. To sum up, we focus on the following methods of supporting the user's interaction:

- **Content Support**: Facilitate entering data
- **Navigation Support**: Facilitate the navigation by
  - guiding the user (*Guidance*),
  - providing *Navigation Shortcuts*, and by
  - reducing the UI to the most important elements (*Interface Adaptation*).

2.3. Requirements for Context-Aware IUIs

In this section, we give an overview of the challenges and requirements that arise when using context in IUIs, i.e. when building context-aware IUIs. This analysis builds the basis for the development of the concepts and algorithms presented in the remainder of this thesis.

In Section 2.3.1, we first present an overview of general challenges identified in literature for developing IUIs. In Section 2.3.2, we list the additional requirements which have to be faced when using context information for supporting the user's interaction.

2.3.1. Challenges in Developing IUIs

The main goal in developing IUIs is that they should be usable, useful, and trustable (Myers, 2007). This aligns with the main problems identified by Maes (1994): *Presentation*, *Competence,* and *Trust*. **Presentation** is concerned with the human computer interaction part of IUIs, whereas **Competence** focuses on the artificial intelligence techniques that can be applied. The development of IUIs has to take special care of **Trust**, as the user is not willing to delegate tasks to an IUI she does not trust, thus rendering the IUI useless. In this section, we point out which challenges have to be faced when developing IUIs, and describe possible ways to cope with them. We thereby focus on IUIs that are able to adapt their behavior to the user's needs. An overview of the identified challenges is given in Table 2.1. The challenges are not disjunctive and heavily interrelated; they should just give some of the focus points for developing IUIs. This list is not meant to be complete and not all challenges have to be faced in each IUI, e.g. collaborative filtering systems usually do not have to cope with the problem of little usage data.

Presentation	Interaction design
	Unobtrusiveness
	Adaptivity
Competence	Little usage data
	Changing user behavior
Trust	Controllable behavior
	Intelligibility
	Privacy

Table 2.1. Challenges in developing IUIs

Presentation

For the presentation of IUIs, we at first need to consider how to **design the interaction** between the user and the IUI. Especially personal assistants are often only augmentations of existing UIs. Thus, they have to be integrated into the existing layout, offering the user a way to communicate with the assistant. In doing so, it should *not hamper the normal usage* of the application. The interaction should also support some kind of *forgiveness* that allows the user to easily correct previously performed actions using an undo capability (Apple, 2008). Many researchers argue that IUIs should *fail softly* – this means if the provided support is not (entirely) correct, it should still move the user closer to the goal (Lieberman et al., 2004) and should not cause problems when it is wrong (Dix et al., 2000). Furthermore, the design of the interaction tackles whether and how the user can *instruct* the IUI (Norman, 1994) or whether an anthropomorphic agent is used to allow the user to communicate with the IUI (Wexelblat and Maes, 1997). These issues are closely related to trust issues that will be discussed later, i.e. how the user can control an IUI and which expectations are raised by the IUI.

Another important factor regarding the presentation is **unobtrusiveness** (Jameson, 2007; Langley and Fehling, 1996). The intelligent support should not distract the user from normal usage of the application. A counterexample is the Microsoft Office Assistant, which is constantly moving and thus drawing the user's attention to it without providing any relevant help for the user's current task. Wexelblat and Maes (1997) propose reducing the distraction of the user by minimizing the amount of interruptions and deferring interruptions until they are less disruptive. Another way to cope with this issue is to support different levels of obtrusiveness (or proactivity) depending on the information importance or the certainty in the action, e.g. as applied by Horvitz (1999) and Maes (1994).

Furthermore, an IUI should be able to **adapt** its presentation to different *users*, *devices,* and *situations*. For example, a novice user needs more explanations than an expert user, or voice output is suitable for mobile usage, but not if the user is sitting in a library. As the costs of interacting with applications via a mobile phone are

much higher than in a traditional desktop setting, more support may be desirable in mobile settings. However, adaptivity not only influences the presentation of an IUI, but also how much the user trusts the IUI or which demands an IUI puts on the underlying algorithms (i.e. affecting the competence of the IUI).

Competence

The competence of an IUI is determined by the underlying algorithms. If an IUI adapts its behavior to the individual user's needs, its level of competence depends on the amount of training data available. However, this is usually limited at the beginning. Thus, the algorithms used for IUIs mostly have to be able to deal with **little usage data**. For that reason, IUIs that just learn from observation are usually not of great aid at the beginning ("slow-start problem"). This problem can be overcome for example by relying on predefined models or by using a default model that is inferred from the models of other users. However, the former requires a large modeling effort by a developer and the latter can cause privacy problems (as discussed in the *Trust* section).

A second problem is that the **user's behavior changes** over time (Höök, 2000). Especially the usage patterns of a novice user often dramatically differ from the usage patterns of an expert. For that purpose, ageing can be used that decreases the weight of older interactions.

Trust

It is much more challenging for IUIs to induce user's trust in it than for traditional user interfaces, because IUIs apply AI techniques whose results can often not be directly foreseen by the user and thus reduce the user's feeling of being in control of the system.

The trust the user puts in an IUI is influenced by many factors, especially by presentation issues as discussed before. We state below the main challenges identified in literature that have to be considered when building trustable IUIs. At first, it is essential that the user feels in **control** of the IUI. The user should be able to correct and adjust the actions of an IUI, and to control its autonomy (Höök, 2000; Bellotti and Edwards, 2001; Glass et al., 2008; Dey and Newberger, 2009). One possibility to control the actions of an IUI is to require the user to approve or disapprove every action (Cypher, 1991), or by letting the user specify confidence thresholds for actions (Maes, 1994). However, requiring the user to always control the application is usually not desirable as the users differ in their desire for control (Jameson and Schwarzkopf, 2002) and too much control may lead to distraction and time-wasting (Kay, 2001). The amount of control should also depend on the criticalness of the task, e.g. for non-critical tasks, like filling data in input fields, a lower level of control is needed than for automatically buying goods. Another factor that influences how much control the user wants to exert is her trust in the IUI that (hopefully) evolves over time. For all those reasons, an IUI should support variable levels of control that

can also be adjusted by the user.

Another important issue for establishing the user's trust in the IUI is **intelligibility**, i.e. that the user is able to understand the actions of the IUI (Bellotti and Edwards, 2001; Malaka, 2008; Dey and Newberger, 2009). As stated by Maes (1994), a user is more likely to trust an IUI if she sees in advance what the agent would do. A study of context-aware systems by Barkhuus and Dey (2003) also showed that users become very frustrated when they do not understand why a system has performed an action, or when they do not have the ability to fix it. One way to achieve intelligibility is *transparency*, i.e. that the IUI helps the user understand its actions. Transparency can be realized by an IUI by giving feedback of its actions (Maes, 1994), by being able to justify its actions, or by making the user aware of automatic adaptations. Another way of increasing the intelligibility of an IUI is to give the user *access to the knowledge source* that was used for providing support (Glass et al., 2008). For example, Cook and Kay (1994) argue that the IUI should let the user inspect and modify the user models used by the IUI. The intelligibility of its actions should thus support the user in developing an appropriate model of the behavior of the IUI (Malaka, 2008). In this way, it is not necessary to mediate a complete model of the IUI, as "understanding comes from a careful blend of hiding and revealing agent state and functioning" (Wexelblat and Maes, 1997). They argue for example that for driving a car, it is also not necessary to have a complete model of how the engine or the brakes work. This blend can be achieved by applying a black box in a glass box system (Höök et al., 1996), i.e. complex inferences are hidden from view in a black box system, whereas a simpler model is conveyed to the user, like cartoons as used by Kozierok and Maes (1993). A good mental model of the behavior of an IUI also influences how *predictable* its actions are perceived by the user, and finally which *expectations* she has of the IUI (Glass et al., 2008). Erroneous higher expectations can easily lead to disappointment among users and discourage them from using the IUI. This is also one of the main reasons why many researchers argue against using anthropomorphic agents, as they are perceived by the user to be similar to a human being, and thus could also take responsibility for their actions.

Finally, for IUIs that share information between users, **privacy** has to be regarded. The requirements that are posed on privacy differ between applications. For example, the users of FireFly[4], an application for sharing preferences for music or movies, did not perceive this sharing as critical, whereas users of the Doppelgänger system (Orwant, 1994), which provides personalized news by considering the type of news that a colleague usually reads, had strong privacy concerns against the system. This might be the case as the data differed in their level of importance to the user and as the data in the Doppelgänger system, in contrast to the Firefly system, was not anonymized. Besides anonymizing the data, another solution proposed for this problem is to split the user model into a private and a public part (Cook and Kay, 1994).

[4] A company founded by a group of engineers from MIT media lab, including Pattie Maes, which was sold to Microsoft in April 1998.

2.3.2. Requirements for Using Context in IUIs

Besides the general challenges for IUIs, we have to face additional requirements when using context information in IUIs, that is for building *context-aware IUIs*. In our opinion, considering the user's context is the key to providing useful support, as the required assistance often heavily depends on it. The relevant context can thereby be gathered from the user's environment as well as from models describing the user's behavior. For that reason, it is important to consider the *environmental context* as well as the *user context* (see Section 2.1.2):

R1 (Awareness of user context and environmental context) An ideal context-aware IUI should be able to consider user context as well as all kinds of environmental context to support the user's interaction in a user-adapted way.

This also requires that the specific nature of context has to be taken into account, i.e. that context information is often unreliable. For example, if physical context sensors provide inaccurate values, or if the user's behavior has changed. The provided support thus has to adapt to this uncertainty. This is especially important as erroneous support can easily lead to losing the user's trust (Tiernan et al., 2001).

R2 (Cope with error-prone context) An ideal context-aware IUI should be able to cope with the error-prone nature of context information and adapt the support according to the reliability of the data.

To get a better understanding of the user's context, we also have to consider her interactions. Many tasks a user performs involve the interaction with several applications. For example, booking travel often involves looking up train connections, booking a flight, etc. using different applications. Information entered into one application can be relevant context for another one. For that reason, we state that it is important to support the interaction with all kinds of existing applications and not just to focus on a single one.

R3 (Application-independence) An ideal context-aware IUI should be able to facilitate the interaction for all kinds of applications.

R4 (Support across application boundaries) An ideal context-aware IUI should be able to provide support across application boundaries.

We cannot expect that all applications the user interacts with are developed using our approach. However, the interaction with applications is an important information source for understanding the user's context. For that reason, an ideal context-aware IUI should be able to gather context information from existing applications. It should also be able to provide interaction support for these existing applications to gain wide applicability.

R5 (Support for legacy applications) An ideal context-aware IUI should be able to provide support for existing applications and gather information from the user's interaction with them.

Thus, a context-aware IUI should be able to cope with a variety of different applications and context information. Moreover, which context information is relevant for a given application also depends on the individual user. It is impossible to foresee all relevant relations between context and application at design time. However, these relations can often be stated by the end-user herself. Moreover, it increases the user's trust in the system if she is able to control the provided support.

R6 (Involving end-user) An ideal context-aware IUI should enable the end-user to specify additional information about an application and its relation to context.

Manually specifying this information is a tedious task and not all end-users are willing and able to provide this information. For that reason, it is important that an ideal context-aware IUI is also able to learn the information from observing the user's interaction.

R7 (Learning capabilities) An ideal context-aware IUI should learn additional information about an application and its relation to context from observing the user's interactions.

2.4. Chapter Summary

In this chapter, we presented our understanding of context and why it is an important information source for IUIs. We refined the definition of context to overcome the limitations of existing definitions. We advocated that we have to consider user context as well as environmental context to support the user's interactions in an optimal way. We introduced the area of IUIs and gave a categorization of the different research areas subsumed under that term. We pointed out to which of these areas this thesis contributes and elaborated on the features of the two relevant research areas, *Personal Assistants* and *Interface Adaptation*. Finally, we identified challenges and requirements that have to be considered when using context information in IUIs. In the next chapter, we analyze the state of the art and its shortcomings with respect to those requirements.

3 State of the art

In this chapter, we give an overview of the state of the art of Intelligent User Interfaces (IUIs). We thereby concentrate on the research area of *Personal Assistants,* as this is the main focus of this thesis. The state of the art for the algorithms and models needed for using context information in IUIs will be reported in Chapters 6 to 8, because the requirements we pose on the algorithms and models can be better understood in the context of these chapters.

In Section 3.1, we introduce the three classes of personal assistants: knowledge-based systems, end-user programming, and learning systems. In Sections 3.2 to 3.4, we review the state-of-the-art systems for each of these three classes that are closest related to our approach. Table 3.1 summarizes which requirements for context-aware IUIs, which were identified in the previous chapter, are met by the different systems.

3.1. Classification of Personal Assistants

A personal assistant is an Intelligent User Interface that collaborates with the user to facilitate the system use, and is able to perform tasks autonomously. Following Maes (1994), we distinguish between three main classes of personal assistants:

- **Knowledge-based systems** that rely on extensive domain-specific knowledge about the application and the user. This knowledge is used at runtime to recognize the user's plans, and to find a way to assist her in executing them.
- **End-user programming** enables the end-user to define how she wants to be assisted in her tasks, for example by specifying rules on how emails should be sorted.
- **Learning systems** use machine learning techniques to gather this knowledge from observing the user's interactions.

Each of these approaches has its own advantages and disadvantages. *Knowledge-based approaches* rely on extensive domain knowledge, so they require great effort by the knowledge engineer in advance. It is often also not possible to model all needed knowledge at design time as not all ways of using it can be foreseen and the usage

can vary from user to user. Knowledge-based approaches cannot be customized to individual user habits and preferences as their knowledge is fixed. Moreover, it is difficult for the user to gain a mental model of how the system works, thus reducing the trust in the personal assistant.

End-user programming requires much understanding from the end-user and introduces additional modeling effort. Approaches of this type can usually support the interaction with a variety of applications.

Learning systems are able to provide assistance that is adapted to the specific user's needs. They have the drawback that they cannot provide adapted support from the beginning, because they require initial training data ("slow start problem"). They are furthermore mostly limited to one specific application.

3.2. Knowledge-based Personal Assistants

Existing knowledge-based approaches are used in two main application domains: (i) For supporting the execution of a task by making suggestions or by explaining steps (COLLAGEN and AGUSINA), and (ii) for providing navigation shortcuts (CyberDesk, onCue, and Miro).

The **COLLAGEN** (COLLaborative AGENt) project by MERL (Mitsubishi Electric Research Laboratories) aims at building an agent that cooperates with a human via dialogues (Rich and Sidner, 1997; Rich et al., 2001). It thereby tries to incorporate discourse strategies governed by the same principles that underlie human collaboration. The knowledge for assisting the user is stored in hierarchical task models whose elements are linked to GUI elements. In order to reduce the necessary modeling effort, they also developed a task model learning algorithm that converts a sequence of demonstrated actions into a task model (Garland et al., 2001). In this learning process, the user can also annotate which actions are only optional in the sequence. COLLAGEN can tell the user which steps she needs to perform. It can also interpret the user's input and thus perform tasks for the user. COLLAGEN is integrated into several prototypes, e.g. DiamondHelp for networked home products (Rich et al., 2006).

AGUSINA (AGent USer INteraction Architecture) is another knowledge-based assistant developed by Amandi et al. (Amandi and Armentano, 2004). They focus on building interface agents for any standard Web application without having access to the Web application code itself. Thus, the agent and application code are strictly separated in their architecture. They are connected by a task model. As building these task models is a tedious task, they developed an algorithm for automatic task model generation from given usage sequences (Eyharabide and Amandi, 2005). The agent observes the user's actions and computes, with the help of the predefined task models, the currently active tasks. In addition, the observed actions are forwarded to an intention detector that contains a Bayesian network to infer the current intentions of the user (Armentano and Amandi, 2003). An intention is thereby defined as tasks that can be performed within a specific task model, and can thus also be observed by

		Content support	Navigation shortcuts	Guidance	Awareness of user context (R1)	Awareness of environmental context (R1)	Cope with error-prone context (R2)	Application-independence (R3)	Support across application boundaries (R4)	Support for legacy applications (R5)	Involving end-user (R6)	Learning capabilities (R7)
Knowledge-based	COLLAGEN	•		•				•				○
	AGUSINA	•		•	•			•		•		•
	CyberDesk		•			•		•	•	•		
	onCue		•					•	•	•		
	Miro		•					•	•	•	•	
	ActiveBadge					•	•					
End-user programming	Citrine	•						•		•	•	
	Context Rules					•				•		
Learning	CAP	•			•						•	•
	LookOut	•			•		•					•
	CMRadar	•			•							•
	PTIME	•			•						•	•
	Maxims	•			•		•				•	•
	CAIA	•			•							•
	Chusho	•						•		•	•	•
	VIO	•						•				•
	Folder Predictor		•		•							•

Table 3.1. Overview of the state-of-the-art approaches with respect to the provided support types and the requirements for context-aware IUIs (see Section 2.3.2, ○: is able to learn task models from usage sequences, *Context Rules* subsumes several systems with the same characteristics)

the agent. The Bayesian network is also updated, thus involving learning of user's intentions (R7: +). The user can ask the agent for help or the agent can proactively offer advice (this can be a suggestion, a warning, or a reminder), offer to take action, act autonomously, or ask the user for missing information. For that purpose, the application developer has to predefine triggers and actions for the various support types. Thus, AGUSINA has only limited possibilities to adapt to the user's needs and preferences.

To sum up, these two systems are able to support the user in entering data and in navigating for all kinds of applications (R3: +). However, their support cannot be provided across application boundaries (R4:-) and they do not consider the environmental context for that purpose (R1: -).

CyberDesk (Dey et al., 1997) is a context-aware data detector. It considers data that the user has currently selected in the user interface, but time and location information is also used. For example, if the user has selected an email address, CyberDesk can suggest a shortcut for writing an email to this person. The incoming context data is processed by converters that for example transform coordinates to rooms or transform a string into a name object. If CyberDesk contains services that are registered for the resulting type of data, CyberDesk provides shortcuts to them. CyberDesk also allows the chaining of several services, or the combining of several data types to make more powerful services. For example, if the user selects a name, a service can be used to gain the corresponding email address so that the user can be provided with a shortcut for writing an email.

Very similar to this work is **onCue** (Dix et al., 2006), an intelligent toolbar that also suggests shortcuts for facilitating the processing of data. In contrast to CyberDesk, it only considers the data in the user's clipboard, and focuses on the provision of more complex services for visualizing data, e.g. for displaying a selected table as a pie tree.

One drawback of these two data detectors is that they are hard to extend for an end-user. For that reason, Faaborg and Lieberman developed a data detector, **Miro** (Faaborg and Lieberman, 2006), which cooperates with a programming by demonstration system called Creo[5]. The user can record macros for executing a task with Creo. The data entered in the demonstration session is generalized using semantic networks like ConceptNet (Liu and Davenport, 2004), e.g. it generalizes "Coke" to "Soda" and "Food Brand". Creo allows the end-user to state which of the computed generalizations are really relevant for the task. Miro is then able to turn all terms on a Web page that also map to one of those generalizations into hyperlinks to the created macro.

The described data detectors are able to provide shortcuts for different applications, and can thus be considered to be application-independent (R3: +) and able to provide support across application boundaries (R4: +). Most of the described data

[5]This system is not described in more detail because it falls in the "Programming by demonstration" research area, which is not in the focus of this thesis.

detectors also consider no or only some specific types of context (R1: -) and none of them is able to cope with the error-prone nature of context (R2: -). Moreover, they are not able to enhance themselves via observation (R7: -).

In the area of context-aware computing, few approaches exist for supporting the system use with the help of personal assistants. The most prominent approach is the **Active Badge** system by Want et al. (1992). This system tracks the location of all users and displays this information to a receptionist who is responsible for forwarding phone calls to the respective persons. In doing so it also displays the uncertainty of the location estimation in order to provide more transparency to the receptionist. It is therefore able to cope with error-prone context information (R2: +). However, it is limited to one specific application (R3: -, R4: -) and cannot be enhanced by the end-user (R6: -) nor learn additional support from observation (R7: -).

3.3. End-user Programmed Personal Assistants

There is a fluent passage between knowledge-based systems and end-user programmed systems, as some knowledge-based systems also support the user to add new knowledge and new functionality. However, they usually focus on users with some technical background and who have some experience with the system, thus not on the end-user.

A system that focuses explicitly on supporting the end-user to add new functionality is **Citrine** (Stylos et al., 2004). It facilitates the entering of contact information, citations, and calendar appointments into forms via copy-and-paste. For that purpose, it analyzes the information copied by the user with heuristic text parsers that check for patterns and keywords to assign it to one of the mentioned data types. The structured data can then be pasted into the corresponding form fields. Citrine supports the entering of information into some existing Web applications, like Outlook or the Palm Desktop. The user can further define by example which parts of the data should be mapped to which field in a Web form. Thus, Citrine is able to provide support for different Web applications (R3: +), but it considers neither user context nor environmental context for that purpose (R1: -).

In the area of context-aware computing, several approaches exist that allow the end-user to invoke actions depending on context changes using a graphical editor (R6: +) (Humble et al., 2003; Korpipää et al., 2005; Dey et al., 2006) (here subsumed as **Context Rules**). The user can specify rules like "if John is outside and the phone rings in the kitchen, then turn up the phone volume". Their applicability is rather limited to predefined triggers and actions and cannot be extended for the use with other applications (R3: -).

3.4. Learning Personal Assistants

Learning personal assistants are most often applied in two application domains: (i) managing meetings and emails, and (ii) filling forms. One of the first representatives

for systems for scheduling meetings is **CAP** by Dent et al. (1992). It assists the user in organizing her appointments by suggesting values for the different attributes of an appointment. For gaining the suggestions, they apply offline learning that can be combined with hand coded rules (R6: +).

Another implementation for scheduling meetings is **LookOut** (Horvitz, 1999), a tool for managing meetings integrated into Microsoft Outlook. It analyzes the user's incoming emails. If it contains a meeting request, LookOut tries to predict whether the user wants to schedule a meeting and for which date. This data can then be used to automatically fill all important information in the appointment form and thus to reduce the user's interaction costs, even if the predictions are not entirely correct. LookOut supports different interaction modalities. The user can decide whether the support should be provided automatically or just on demand. In the latter case, LookOut only takes action if the user explicitly invokes it by clicking on an icon in the system tray. In this mode, LookOut can further display an alerting symbol in the system tray to indicate that it would have taken action if it were in automated mode. In the automatic mode, the system takes the uncertainty, costs, and benefits of taking an action into account to decide whether it should automatically perform an action or ask the user. LookOut is therefore able to handle the uncertainty of context (R2: +). However, it only considers the user context for that purpose (R1: +).

DARPA's "Personalized Assistant that Learns" program funds two artificial intelligence projects called CALO and RADAR that aim to build assistants that can support the user in her daily activities. In both projects a meeting scheduling assistant was developed: CMRadar and PTIME. **CMRadar** (Modi et al., 2005) is a multi-agent scheduling system. It is able to extract the relevant information for scheduling a meeting from emails; it negotiates possible schedules with other CMRadar meeting agents, visualizes these suggestions using RhaiCAL (Faulring and Myers, 2005), and finally learns the user's preferences from the user's decisions. **PTIME** (Berry et al., 2006) learns the user's preferences from observation and from explicit instructions (e.g. "I like meetings in the morning"), and suggests several possible meeting suggestions to the user.

Pattie Maes, one of the main advocates of learning personal assistants, also developed a tool for scheduling meetings (Kozierok and Maes, 1993). She and her team further developed the electronic mail agent **Maxims** (Lashkari et al., 1994) which assists the user in managing her emails. It learns to prioritize, delete, forward, sort, and archive messages by observing the user's interactions. Maxims memorizes all situation-action pairs, and later uses them to make predictions using case-based reasoning. Maxims also computes a confidence in the suggestions (depending on how similar the examples are, how many examples the agent has memorized, etc.). This confidence value is used to decide whether the predicted action should be suggested to the user or even automatically executed (R2: +). For that purpose, the user can specify two thresholds ("tell-me" and "do-it" thresholds). Another feature that should increase the user's trust in the system is that the displayed Maxims icon communicates its internal state, e.g. how confident it is in a suggestion, via its facial expression. To cope with the slow start problem, the user can explicitly teach

Maxims by showing it an example situation and the corresponding action (R6: +).

In summary, all these meetings and email assistants consider only the user context for their predictions (R1: -), and some are able to cope with the error-prone nature of context (R2). However, they are all limited to the scope of a meeting application (R3: -) and are thus not able to provide support across application boundaries (R4: -).

A more general application domain for learning personal assistants than scheduling meetings is form filling. One of the first systems that focused on filling forms is **CAIA** by Hermens and Schlimmer (1994). They built an adaptive system that is able to predict defaults or string completion for filling out leave reports. They use the values of surrounding fields as features for the learning process. However, the algorithms only perform well when the forms are not filled in a random order. Moreover, this approach is –like all previously presented learning approaches– limited to one specific application (R3: -).

A more generic approach is taken by Chusho et al. (2002). They build cognitive rules to enter information in Web forms. A cognitive rule has the form `IF #case THEN #action` whereby the `#case` consists of the description and value of the input field located around the target input field, e.g. `IF (UPPER: (@address TEXTFIELD), LEFT: @phone, RIGHT: NONE, LOWER: (@email Textfield)) THEN @phone`. To cope with the problem of synonyms, they introduce concept names like `@phone` that refer to all possible stored synonyms for that concept. For determining the correct concept for an input field, they compute for each of the four conditions (upper, left, lower, right) which actions would result from it with the corresponding probability. Out of these four probability distributions, they choose the action with the highest confidence value. The list of keywords can be extended during usage, however the user has to explicitly state which of the surrounding words is considered as the keyword for the input field.

Another approach for form filling is taken by **VIO** (Zimmerman et al., 2007), a mixed-initiative agent to support a webmaster in updating information in a database, which was developed in the RADAR project. For that purpose, VIO analyzes incoming email requests to suggest a form to the webmaster for changing the information (e.g. the contact form) and also which values to change. It learns from observing the user's corrections and thus improves the support over time. The focus of VIO is to understand the natural language input to map it to the Web forms; for that reason they only support interactions that were triggered by an email request.

The latter two approaches are application-independent (R3: +), but are also not able to provide support across application boundaries (R4: -). Furthermore, they consider neither user nor environmental context information (R1: -).

Other learning personal assistants aim at facilitating the user's navigation. **Folder Predictor** (Bao et al., 2006), developed in the CALO project, aims at minimizing the clicks needed to locate a file. It builds upon TaskTracer (Dragunov et al., 2005) and TaskPredictor (Stumpf et al., 2005). TaskTracer traces various user activities in Windows, including Microsoft Office and Internet Explorer. TaskPredictor is able to

predict the task the user is currently working on from these traces. FolderPredictor takes the information about the current task and about all folders that are associated with this task, and assigns a weight to each folder that depends on the recency of use. The three folders that minimize the expected click costs the most are then suggested to the user in the left bar of the standard file selection menu of Windows. This approach is thus again limited to a single application (R3: -), and it does not consider any environmental context (R1: -).

3.5. Chapter Summary

In this chapter, we gave an overview of the state of the art of personal assistants and showed that no existing approach meets all the requirements identified in Section 2.3. In particular, there are no approaches that consider the user context as well as the environmental context. In the area of personal assistants, only very few approaches exist that consider environmental context at all.

Which context information is useful to support the user depends on her needs and her current task. As discussed before, a context-aware IUI should for that reason be able to learn which context information can be used to support the user which is covered by *learning systems* (R7). Furthermore, the user should be able to inspect and modify these relations between context and application which is targeted by *end-user programming* approaches (R6). Finally, learning systems cannot provide support right from the start, and end-user programming introduces high modeling costs for the end-user. For that reason, the application developer should also be able to provide some initial application models stating which context information is relevant for the application (*knowledge-based systems*). The quality of these prebuilt models is usually higher than the quality that can be achieved by pure machine learning approaches or by end-user modeled data. For that reason, the best support can be provided by combining all three existing approaches for building personal assistants. However, current systems focus on only one of these approaches and provide none or only very limited support for the others.

Furthermore, there are very few existing systems that consider the unreliability of context, which we assume to be very important when using context information in IUIs (R2). Many approaches also focus only on a single application and are thus not application-independent (R3). Finally, only some approaches that are limited to providing navigation shortcuts are able to provide support across application boundaries at all (R4).

In the following chapters, we will present our own application-independent approach, which overcomes those limitations and considers the user as well as the environmental context to provide content and navigation support.

4 High-Level Design

In this chapter, we present the conceptual building blocks of our approach for context-aware IUIs which meets the requirements identified in Chapter 2. We will refer to the combination of presented concepts as AUGUR. Chapter 5 illustrates the implementation of our approach in the AUGUR prototype. The underlying models and algorithms are presented in detail in Chapters 6 to 8. In Chapter 9, we report on the results of a user study for evaluating the usability of interaction support as presented in this thesis.

In this chapter, we at first introduce the main conceptual building blocks of AUGUR (Section 4.1). In Section 4.2, we introduce the models required for the interaction support. Section 4.3 presents the different interaction support types provided by AUGUR and which algorithms are required for their realization. Then, we point out how AUGUR copes with the error-prone nature of context information by using different presentations of the support types to reduce the adverse effect of erroneous support (Section 4.4). Finally, we illustrate the capabilities of AUGUR with two application scenarios in Section 4.5.

4.1. Conceptual Building Blocks

In this section, we give an overview of the major conceptual building blocks of AUGUR (see Figure 4.1), and how they reflect the requirements identified in Section 2.3. AUGUR consists of three major blocks: (i) The *Support Tier,* which generates the required support (*Support Generator*) and adapts it accordingly (*Representation Manage*r), (ii) the *Knowledge Base*, and (iii) *Editors* to enable the end-user to inspect and modify the models in the *Knowledge Base*. The relations between requirements and the conceptual blocks are summarized in Table 4.1, and will be described in more detail below.

Models are the most important components of IUIs. As stated by requirement R1 (*Awareness of user context and environmental context*), user context and environmental context are important information sources for understanding the user's needs. For that reason, the *Knowledge Base* of AUGUR requires a context and a user model. The context model stores all information which we referred to as en-

Requirement	Architectural Component
R1 (*Awareness of user and environmental context*)	*Knowledge Base*, containing *User Model* and *Context Model*
R2 (*Cope with error-prone context*)	*Representation Manager* adapts support to confidence in context information
R3 (*Application-independence*)	Proxy architecture
R4 (*Support across application boundaries*)	
R5 (*Support for legacy applications*)	
R6 (*Involving end-user*)	*Editors*
R7 (*Learning capabilities*)	*Support Generator* updates the *Knowledge Base*

Table 4.1. Relations between requirements and architectural components

vironmental context and the user model provides user context. Moreover, AUGUR requires an application model to store the relations between context and applications, and thus to be able to actually use context information for providing support. A high level description of all three models is given in Section 4.2. In Chapter 6, we elaborate on them and show how they overcome the shortcomings of state-of-the-art models.

The second important building block of IUIs is the support generation (i.e. *Support Tier* in Figure 4.1). As discussed in Section 2, we focus on supporting the user in the following ways:

- **Content Support**: to assist the user in entering data
- **Navigation Support**: to guide the user through an application (*Guidance*), to provide *Navigation Shortcuts*, and to adapt the UI to the user's needs (*Interface Adaptation*). For the interface adaptation, we thereby focus on reducing the UI to most relevant UI elements.

In this thesis, we present novel algorithms for realizing these different support types by incorporating user context as well as environmental context. The algorithms also learn from observing users' behavior to enable AUGUR to automatically enhance the provided support (R7 - *Learning capabilities*). In Section 4.3, we describe the different support types provided by AUGUR. In Chapters 7 and 8, we elaborate on the underlying algorithms and compare them to state-of-the-art approaches.

According to requirement R2 (*Cope with error-prone context*), AUGUR has to be able to deal with unreliable context information. AUGUR copes with this issue in two different ways. Firstly, it adapts the provided support to the confidence of the support information (*Representation Manager* in Figure 4.1). For example, AUGUR

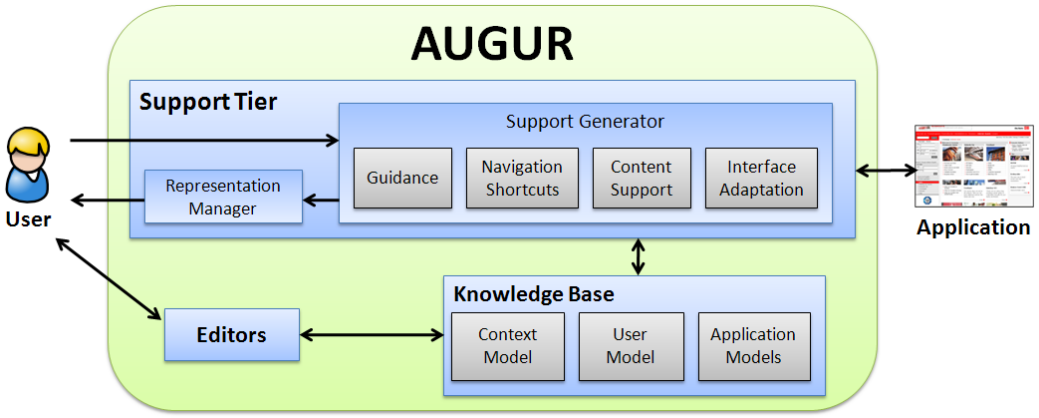


Figure 4.1. Major conceptual building blocks of AUGUR

does not suggest context information as input to the user in which it has only low confidence. This is described in more detail in Section 4.4. Secondly, AUGUR visualizes the confidence in the support information to the user. This increases the transparency of the provided support, and thus increases the user's trust in AUGUR (as discussed in Section 2.3.1).

In order to fulfill requirement R6 (*Involving end-user*), AUGUR enables the end-user to provide additional information about applications and their relation to context via integrated editors (*Editors* in Figure 4.1). The end-user can inspect the models used by AUGUR, and modify them to adapt the support provided by AUGUR.

Finally, AUGUR needs to support the interaction with all kinds of existing application (R3 - *Application-independence*, R5 - *Support for legacy applications*) and provide support across application boundaries (R4 - *Support across application boundaries*). In this thesis, we exemplify this by focusing on form-based applications. AUGUR uses a proxy-based approach to fulfill all these requirements. It is thus able to provide support for different existing applications without the need to modify them. All user events are routed through AUGUR in order to trigger appropriate support and to learn from observing the user's interaction. AUGUR can be run in two different modes (see Figure 4.2):

- **Augmenting UI**: AUGUR displays the original UI of the application, and augments it with content and navigation support information.
- **Generating adapted UI**: AUGUR reduces the UI to the relevant UI elements, and presents it to the user in the desired representation language, e.g. VoiceXML. The resulting UI is then also augmented with content and navigation support information.

In this way, the user has full control over whether interface adaptation is used or not. This is important as the interface adaptation used in AUGUR generates a novel UI,

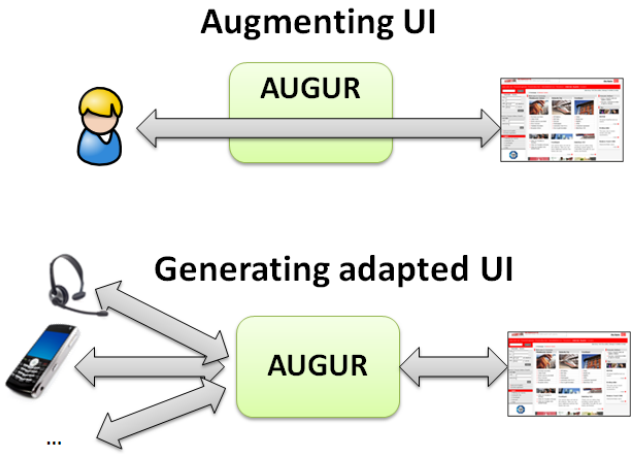


Figure 4.2. Interface adaptation modes supported by AUGUR

and thus does not maintain the spatial stability of the UI, which is regarded as a very important factor for usable adaptation (Gajos et al., 2006). However, rendering the original UI on a small screen device or via voice is usually also not appropriate, as it is difficult to access the relevant interaction elements in those cases. Thus, we decided to leave it up to the user whether or not she wants to interact with a reduced version of the UI.

In the following section, we provide details about the models used, and how the support types are realized in AUGUR.

4.2. Models in the Knowledge Base

In this section, we give an overview of the data which is stored in the different models and how it is gathered. In Chapter 6, we describe the models in more detail and relate them to existing approaches.

4.2.1. Context Model

The context model manages all environmental context information. The context model consists of two components: (i) *Context Server* that gathers and manages all environmental context information and (ii) a component called *Current Context* that keeps track of all context information currently relevant for the user and her interactions.

- **Context Server**: The context server gathers context information from virtual and physical sensors and derives higher-level context information from it. All the context information is represented in a uniform way. This enables AUGUR to treat it in the same way and thus facilitates the integration of novel context sources.

In order to enable AUGUR to cope with error-prone context information, AUGUR needs to be aware of the quality of the context information. Thus, the context representation has to include a quality metric. Buchholz et al. (2003) suggest using the following quality dimensions: (i) probability of correctness, (ii) trustworthiness, (iii) precision, (iv) resolution, and (v) up-to-dateness. On the one hand, such a multi-dimensional representation allows for more accurate quality estimation than a single quality dimension. On the other hand, it puts greater demands on the application, and it is difficult to perceive by the end-user. The most important constituents of quality information which are applicable to all types of context information are its *confidence* (or probability of correctness) and whether the information is *up-to-date*. All other quality metrics are either not available for all context types (e.g. precision, resolution) or hard to obtain (e.g. trustworthiness). AUGUR computes a single quality measure for the context information from the confidence and up-to-date values, i.e. the *current confidence* of the context information. This single-dimensional representation enables AUGUR to use and represent the quality information in a way which can be easily understood by the end-user.

- **Current Context**: The novel concept of the *Current Context* stores all context information which is currently relevant for the user's interaction. *Current Context* contains information gathered from (i) the UI, (ii) the user's interaction, as well as (iii) relevant environmental context information which is provided by the integrated context server. The *Current Context* enables AUGUR to provide content support without requiring any knowledge about the application in advance. For that purpose, it considers only the semantics of the context information and tries to relate it to required input. Furthermore, the *Current Context* enables AUGUR to focus on the relevant context information to learn new relations between context and application for enhancing the provided support.

4.2.2. User Model

Context information relating to the user is stored in the user model. AUGUR needs to be aware of her preferences and of her behavior. Thus, the user model in AUGUR contains the following two components:

- The **preferences** of the user regarding the behavior of AUGUR, which will be explained in Section 4.4. This information is provided by the user herself.
- **Usage Models** for every application: A usage model describes how the user interacted with an application in the past. It enables AUGUR to infer future behavior of the user. It is the basis for the *Guidance* and the *Interface Adaptation* feature of AUGUR, and is also used for providing *Content Support*. The usage models are learned by AUGUR from observation. They can additionally be initialized with usage models from expert users and adapted by AUGUR during usage. This is especially useful for assisting novice users.

4.2.3. Application Model

An application model stores information about an application. This comprises for example information about the available interaction elements, and how they relate to context information. The content of interaction elements can thereby depend on context information (and on other UI elements) in two ways:

- **Direct relations**: The content of an interaction element directly corresponds to the information provided by context information. For example, the point of departure which needs to be entered corresponds to the user's current location.
- **Rules**: The relation between context information and content of an interaction element can be described by simple IF-THEN rules. For example, `if location="at home" then from="Heidenreichstr."` with `location` being context information and `from` an input element.

We could think of supporting more complex relations, e.g. using variables or mathematical operations, but they are more difficult to model and to learn. Thus, we leave this issue to future work.

In the ideal case, the application model is provided by the application developer along with the application, otherwise it is learned by AUGUR from observation. The end-user can augment it with additional information. Moreover, AUGUR constantly tries to learn new information and relations from observing the user's interactions.

4.3. Interaction Support

In this section, we describe the different support types provided by AUGUR, and how they are triggered. As discussed before, interface adaptation is only applied if AUGUR is used in the corresponding mode (*Generating adapted UI*). For the other types, support is invoked as follows:

- At the **start of an interaction** with an interaction element, i.e. focusing an interaction element, the generation of *Content Support* for the interaction element is triggered. For example, if the user clicks on an input field, AUGUR computes content support for this element, and thus tries to facilitate entering data.
- At the **end of an interaction** (e.g. if the user clicks a button, or has entered data in an input field) the generation of *Guidance* support for the current UI is triggered. For example, if the user has entered data in an input element, AUGUR highlights the next relevant interaction element.
- At the **raise of an event** a *Navigation Shortcut* is triggered if it is associated with another application. The event can in this way be raised by external context sources or by information present on the UI. For example, if address information is contained in the current UI and "address" events are associated

with a map application, AUGUR suggests a shortcut to the map application to facilitate the navigation.

The start and end of an interaction can be invoked by the user or by AUGUR. For example, if AUGUR highlights an interaction element (*Guidance* support), the interaction with this element is started automatically.

In the following section, we describe the different support types in AUGUR, and which algorithms are required for their realization.

4.3.1. Content Support

AUGUR is able to suggest content for interaction elements or even to fill it in automatically. AUGUR is thereby able to provide content support for several interaction elements at once. This is especially useful for mobile devices, where the interaction costs for entering information are much higher than in traditional desktop settings. For providing content support, AUGUR considers the user context as well as the environmental context. Existing approaches build their content support either on learned user models or on predefined relations for a single application. However, we advocate considering also the semantics of the information which is currently relevant to the user (i.e. the information stored in *Current Context*). This also enables AUGUR to provide support for yet unknown applications. Thus, AUGUR uses the following three ways to compute content support (see Figure 4.3):

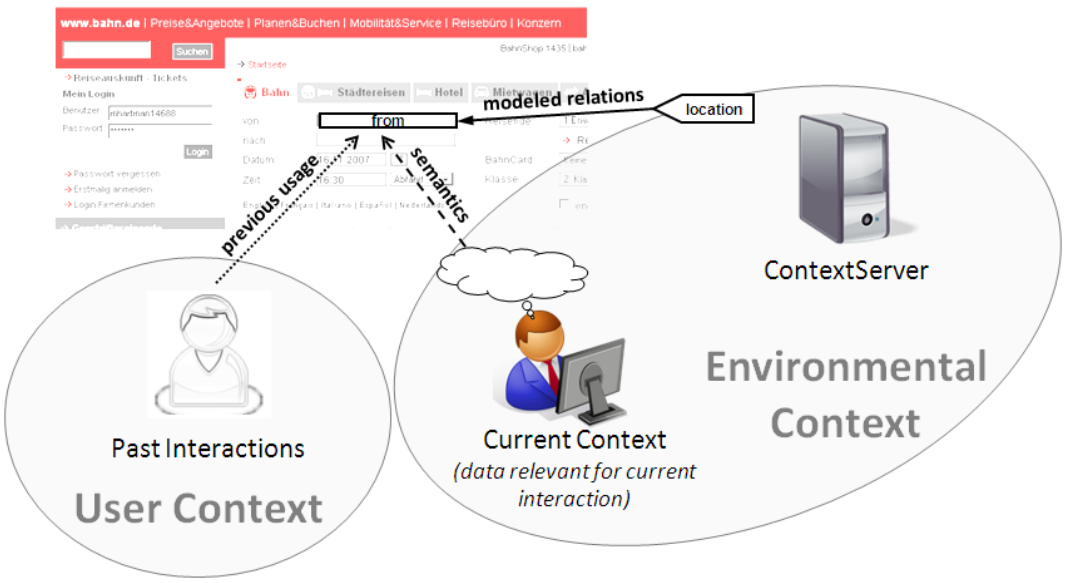


Figure 4.3. Sources for providing content support

- **Previous Usage**: AUGUR constantly tracks the user input, and uses it to predict future user input. This is similar to the standard technique used in modern Web browsers. The support of AUGUR goes beyond this by providing content support which comprises several interaction elements. This can dramatically reduce the required interaction costs. Incorporating more advanced user models is left for future work.

- **Semantics:** The context information which is currently relevant for the user is an important information source for the data required by a UI. For example, when booking a trip, the corresponding information (e.g. destination, dates) is required as input by several applications. For that reason, we developed a novel approach which analyzes the semantics of the data available in the *Current Context* and the data required by the UI. If context information matches the required input, it is offered as input to the user.

- **Modeled Relations:** AUGUR makes use of the relations specified in the corresponding application model to suggest data gathered from context. As stated before, AUGUR supports two types of relations: Direct relations, i.e. which context information can be used as input for an interaction element, and rules, which state interdependencies between elements. These relations are also learned from observation.

For all content support elements returned by the three different sources, AUGUR checks whether it conflicts with the information the user has already entered in the UI. For example, if the user already entered "Frankfurt" in the "from" field, no content support should be offered which suggests another value for the "from" field (unless the user directly clicks on the "from" field and thus indicates that she wants to change the data). Each content support element is associated with a confidence $c_{support}$ to enable AUGUR to decide whether this information should be suggested to the user or not. All compliant content support elements are sorted by confidence and all content support elements which are covered by another content support element with higher confidence are removed. For example, content support element A suggests "Darmstadt" for the "from" interaction element and "Frankfurt" for the "to" element. A content support element B contains the same information as well as "12:00" for the "time" interaction element. If $c_{support}(A) = 80\%$ and $c_{support}(B) = 90\%$, only B will be presented to the user as it covers all the data in A.

4.3.2. Guidance

AUGUR assists the user in navigating through an application by (i) highlighting the interaction element she most probably interacts with next, or (ii) by clicking on navigational elements on her behalf. For that purpose, we need an algorithm which is able to predict this interaction element based on the user's past behavior. For that purpose, we developed a sequence prediction algorithm which also takes changing user behavior into account and outperforms state-of-the-art approaches.

4.3.3. Navigation Shortcuts

As mentioned before, the presentation of navigation shortcuts is triggered by specific events. These events are raised either

- by external context sources (e.g. an incoming phone call), or
- by a UI itself (e.g. if address information is presented on the UI).

For example, if a user wants to browse the details of every person that calls in her address book, AUGUR can provide a shortcut to the corresponding UI as soon as someone calls. AUGUR enables the application developer and the end-user to model the relations between events and applications via its application model. However, AUGUR is also able to learn new relations from observing the user.

4.3.4. Interface Adaptation

Interface adaptation is especially important for mobile use due to the limited screen size and the limited attention of the user (Satyanarayanan, 2001). One possibility to increase usability in these settings is to decrease the required amount of interactions, e.g. key presses or scroll movements to fulfill a task with the application (Buchanan et al., 2001). This can be achieved by reducing the UI to the most important parts.

For that reason, AUGUR is able to generate a reduced UI which is adapted to the user's needs and her current context. This generated UI also contains a link to the unadapted UI, thus still providing access to the whole functionality of the application. If the user wants to access functionality which is not yet present in the application model of the application, she can always fall back to the unadapted UI. AUGUR thus does not reduce the functionality of the application, but provides more efficient access to the elements that are really relevant to the user. In this thesis, we present a novel algorithm for determining which elements in a UI are relevant based on the user's past behavior and her current context.

4.4. Representing Support

The support types described in the previous section can be represented in different ways, e.g. by suggesting input for interaction elements or by automatically filling them. The different representations differ in their level of proactivity. On the one hand, the required interaction costs decrease with increasing proactivity. For example, the interaction costs for entering data are higher if data is suggested than if it is automatically entered. On the other hand, the obtrusiveness increases with increasing proactivity.

For *Interface Adaptation*, AUGUR does not consider different levels of proactivity. The user can directly control whether interface adaptation is used by the mode in which she invokes AUGUR (*Generating adapted UI* and *Augmenting UI*). For all other support types AUGUR supports different levels of proactivity. Which level of

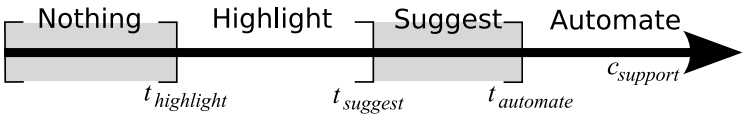


Figure 4.4. Levels of proactivity depending on the confidence $c_{support}$ in the support

proactivity should be chosen depends on the user's preferences and on the confidence $c_{support}$ in the provided support which is provided along with the support information. If a system automatically fills in erroneous data, the user quickly loses trust in the system. This is especially important to consider for support which bases on unreliable information like learned knowledge or context information. Two other factors that can be taken into account are the interruptibility of the user and the potential benefit of the support information for the user. However, we did not incorporate these two factors, as they are difficult to estimate reliably, because they heavily depend on the user's perception. Furthermore, we wanted to keep our model as simple as possible to enable the end-user to state her preferences with respect to the proactivity of AUGUR.

The user can state thresholds for the confidence in the support to determine when which level of proactivity is used (see Figure 4.4). This approach is similar to the one taken by Horvitz (1999). Horvitz (1999) uses two thresholds for choosing the right level of proactivity: One for suggesting actions to the user and one for automatically executing them. We suggest taking a third threshold into account which is especially relevant for navigation support (which was not considered by Horvitz (1999)): The highlighting of interaction elements. We therefore distinguish between three levels of proactivity:

- **Highlight**: Highlight elements to draw the user's attention to it.
- **Suggest**: Display suggestions of which data to enter or where to navigate to.
- **Automate**: Automatically perform actions on behalf of the user, e.g to fill data into input fields, select data from a drop-down menu, or click on navigational elements.

Figure 4.5 shows an overview of all possible combinations of support types and levels of proactivity. However, not all representations are applicable to all support types: Guidance cannot be represented as *Suggest* and content support not as *Highlight*. Moreover, navigation shortcuts are never automatically followed to avoid the user feeling that she has lost control of the system. For guidance the same problem arises: AUGUR should not automatically click on any navigational element. However, the user can explicitly allow AUGUR to perform this action if the confidence of AUGUR in this action is high enough, i.e. above $t_{automate}$.

The suitable level of proactivity for *Navigation Shortcuts* is thus chosen as follows: If $c_{support} > t_{suggest}$, the shortcuts are directly displayed as suggestions to the user. If $t_{highlight} \leq c_{support} < t_{suggest}$, an unobtrusive highlighting is used to inform the

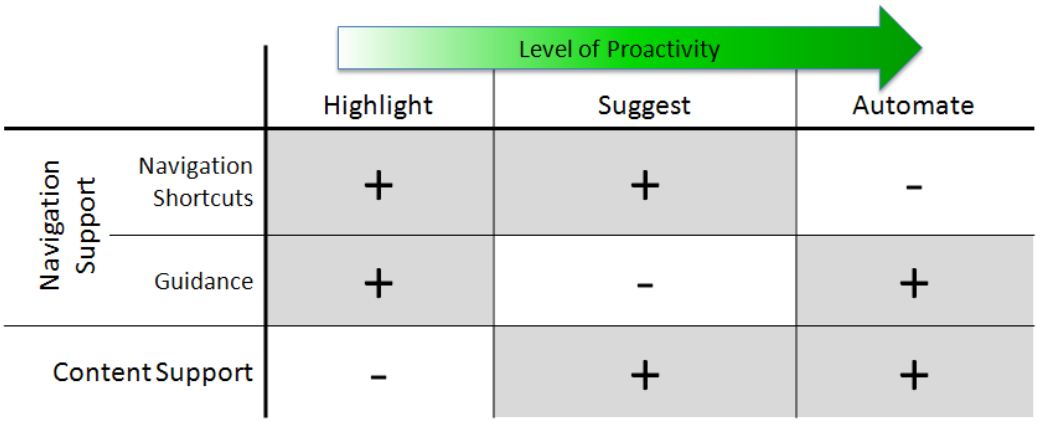


Figure 4.5. Possible combinations of support types and levels of proactivity

user that navigation shortcuts are available. Otherwise, no navigation shortcuts are presented.

For *Guidance*, the next interaction element is highlighted if $c_{support} > t_{highlight}$. If this element is a navigational element and $c_{support} > t_{automate}$, AUGUR automatically clicks on it only if the user has stated that this action should be performed automatically.

If the confidence in the most probable *Content Support* exceeds the threshold $t_{automate}$ and if there is only one content support with this confidence, its data is automatically entered into the corresponding interaction elements, and the user is made aware of this automatism by the elements being highlighted. Otherwise, all data with a confidence above $t_{suggest}$ is suggested to the user.

4.5. Scenarios

In order to illustrate the capabilities of AUGUR described in this chapter, we describe two brief examples of how AUGUR can support the user's interaction for Web applications.

Restaurant Reviewer

Jane is a restaurant reviewer. She always looks up the address of the restaurant she has to review on her favorite restaurant website. Then she navigates to her favorite map application and enters the address information of the restaurant. As AUGUR has no prior information of any of these applications, it has to learn how it can support Jane's interactions from observation. After the first usage, AUGUR has already learned that Jane might switch to the map application when address information is available. Furthermore, AUGUR has learned that the street and city information of the address information can be used as input for the map application.

Thus, AUGUR would be able to support Jane's interaction already at the second usage. However, Jane has stated in her preferences that she does not want to be disrupted by frequent proactive support. For that reason, AUGUR does not perform any action, because it is not yet confident enough in the learned relations. During further usage, AUGUR observes the relation more often and the confidence of AUGUR in it increases. When Jane navigates to the restaurant website containing an address, AUGUR then suggests a navigation shortcut to the map application to her. Jane clicks on the provided link. AUGUR suggests entering the address information for her and Jane accepts. Jane decides that AUGUR should perform more actions autonomously and changes her preferences accordingly. The next time she performs the task and navigates to the map application, AUGUR automatically fills in the address information for Jane in the map application.

Looking up a Train Connection

John wants to look up a train connection. For that purpose, he navigates to the Deutsche Bahn website. AUGUR has already learned an application model and usage model for this Web page from John's past interactions. AUGUR guides him to the first input field in which he has to enter data (the "from" field) by highlighting it. As this field is associated with the user's current location via the application model, AUGUR suggests entering John's current location. John accepts and AUGUR guides him to the "to" input field. As AUGUR has learned from previous interactions that the information for the "to", "date',' and "time" field can be derived from John's calendar, AUGUR queries John's calendar for all relevant data and suggests it to John. John chooses one suggestion and all corresponding data is entered into the corresponding fields. However, as John wants to arrive a bit earlier, he modifies the time and submits the data.

4.6. Chapter Summary

In this chapter, we introduced the conceptual building blocks of AUGUR. We described the general functionality of AUGUR and which models are required for that purpose. Furthermore, we showed how the different support types are realized in AUGUR, and illustrated the capabilities of AUGUR with two brief usage scenarios. In the next chapter, we describe how all the presented concepts are realized in the AUGUR prototype. In Chapters 6 to 8, we will then go into detail of the models and algorithms used in AUGUR.

5 Implementation

In this chapter, we introduce the AUGUR prototype which implements all concepts introduced in the previous chapter. The AUGUR prototype is able to augment any existing form-based Web application with content and navigation support. Figure 5.1 shows an example screenshot of the AUGUR prototype, proactively augmenting the Web page of Deutsche Bahn (German railways). It shows how the AUGUR prototype augments the UI of a Web application by integrating suggestions that are derived from context.

Although we focused on Web applications for the implementation, the developed concepts are applicable to all kinds of interactive form-based systems. We decided in favor of Web applications, as they are reaching the complexity of traditional desktop applications thus increasing the need for interaction support. In addition, many desktop applications are complemented or replaced by a Web version. Thus,

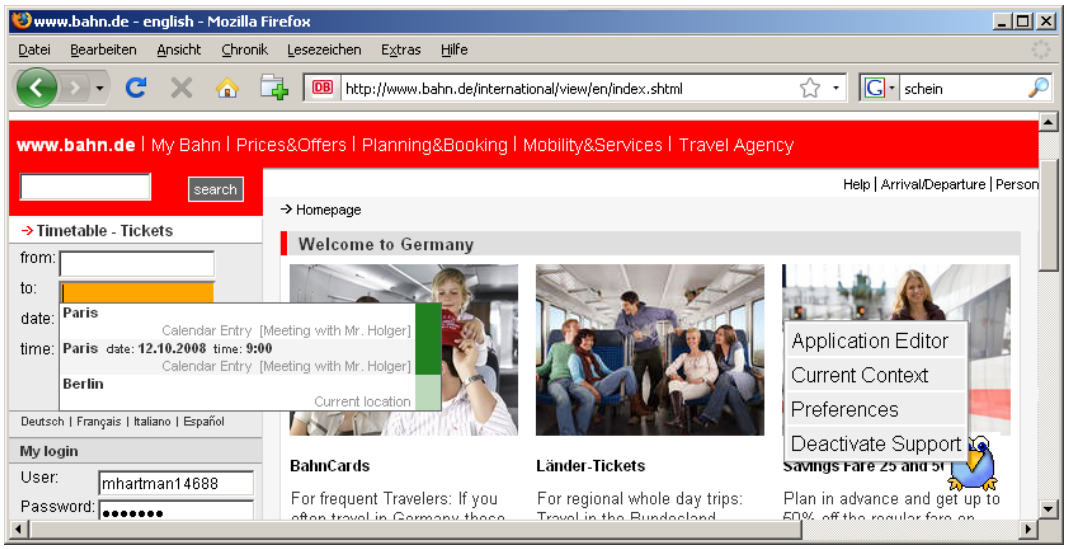


Figure 5.1. Screenshot of the Deutsche Bahn webpage (`http://www.bahn.de`) that is augmented by the AUGUR prototype

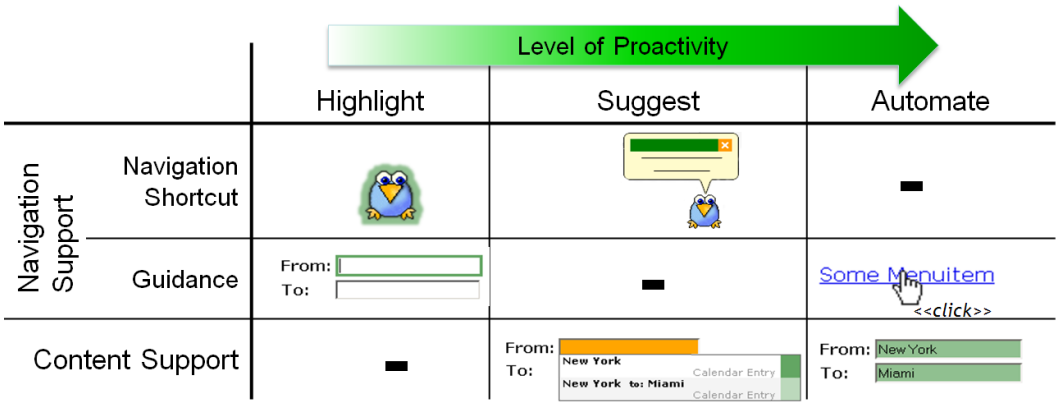


Figure 5.2. Presentations of support types for the different levels of proactivity

our approach can be applied for a wide range of existing applications. Moreover, for supporting the user, we need to be able to observe her interactions with the application and to influence the appearance of the UI (e.g. by highlighting elements or inserting drop-down menus) which can be easily achieved for Web applications.

After introducing the prototype in Section 5, we present in Section 5.1 how the different support types are realized in the prototype. In Section 5.2, we point out how the end-user can adjust the support provided by AUGUR. In Section 5.3, we describe the general architecture of AUGUR and its components.

5.1. Interaction Support

In the following, we present how we realized the support types in the AUGUR prototype. We describe the concrete presentation of the various support types in the AUGUR prototype using the different levels of proactivity. An overview of the different visualizations can be found in Figure 5.2.

In order to make the user aware of the uncertainty of the support, AUGUR visualizes the confidence to the user for all support types. The confidence is visualized with a shade of green, ranging from white for 0% confidence to dark green for 100% confidence.

5.1.1. Content Support

The AUGUR prototype is able to suggest content for a single or multiple interaction elements or to automatically fill it in corresponding interaction elements. If the user accepts a suggestion or if the data is automatically filled in, the AUGUR prototype highlights all affected interaction elements to make the user aware of the action. The two supported levels of proactivity are

- *Suggest*: The suggestions are visualized as drop-down menu for the correspond-

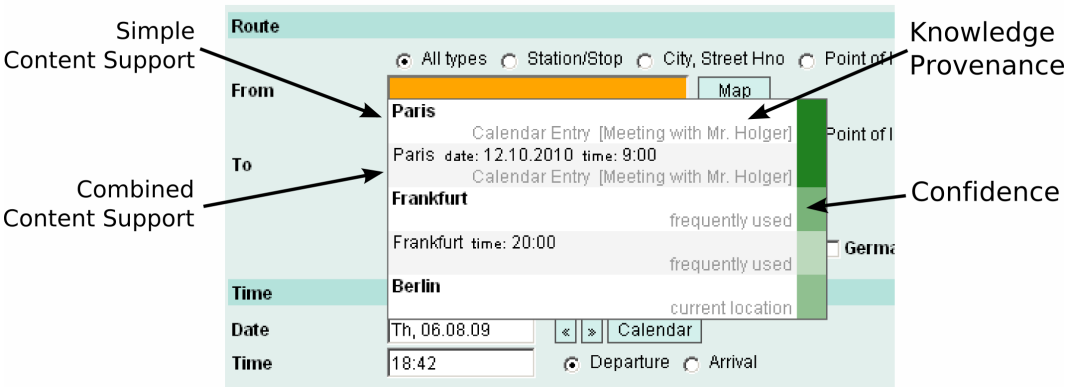


Figure 5.3. Example of suggested content support

ing interaction element (see Figure 5.3). If a suggestion refers to several interaction elements (*combined content support*), it displays the suggested content as a list of key value pairs, whereby the key is the label of the interaction element and the value the suggested data. Only the label for the currently focused interaction element is omitted. For example, if the user focuses on the "to" input element, the AUGUR prototype computes content support "Frankfurt" for the "to" input element and "20:00" for the "time" field and displays "Frankfurt time: 20:00". As these combined suggestions are sometimes only partially correct, the AUGUR prototype also suggests entering only the value for the currently focused element (*simple content support*), in our example only "Frankfurt" is suggested for the "to" input element. Thus, the user is able to select step by step all correct parts provided by a combined content support. The suggestions are grouped by the data they suggest for the interaction element in question (i.e. "Paris", "Frankfurt" and "Berlin" for the example in Figure 5.3). They are ordered by the confidence in the simple content support for each group.

The AUGUR prototype visualizes the knowledge provenance for each content support element to indicate how the data was derived: (i) "frequently used" for suggestions gathered from previous usage, (ii) the context source for modeled relations (e.g. "current location"), and (iii) "current context" for data from the *Current Context* which semantically matches the input required by the UI. The confidence in each suggestion is indicated by a green box for every content support element. If the user hovers over this box the exact confidence value is displayed.

If the user accepts a suggestion, the AUGUR prototype automatically fills in

Figure 5.4. Example of automated content support. The filled data is highlighted to make the user aware of the automatism.

the data in all affected interaction elements and highlights them. Thus, it can dramatically reduce the required interaction costs especially with combined content suggestions as one click is sufficient to enter data in several elements.

- *Automate*: AUGUR automatically fills in data in one or several interaction elements if the confidence in the support is high enough. To make the user aware of this automatism, the AUGUR prototype highlights the affected interaction elements using a red background color (see Figure 5.4). The confidence in the action is displayed by a green border around the interaction element. We do not use this green color for highlighting the interaction element itself as for all the other support types, because uncertain actions would be highlighted in a light green which would not sufficiently grab the user's attention.

5.1.2. Navigation Support - Guidance

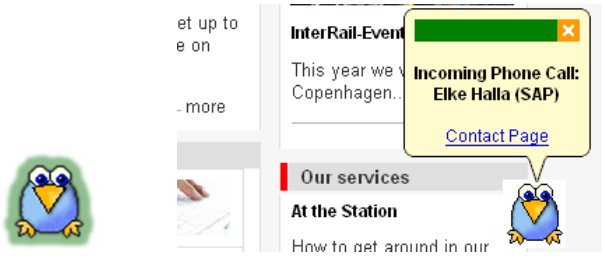
The AUGUR prototype assists the user in navigating through an application (i) by highlighting the interaction element she most probably interacts with next or (ii) by clicking on navigational elements on her behalf. This reflects two different levels of proactivity:

- *Highlight*: highlight the interaction element by displaying a border around the corresponding element. The color of the border reflects the confidence in this support.
- *Automate*: the AUGUR prototype is able to automatically click on navigational elements if the confidence in this action is high enough. However, this leads to the problem that the user might feel to loose control of the system. To avoid this, the user has to explicitly allow the AUGUR prototype to perform this action for every relevant navigational element which can be stated in the corresponding application model.

5.1.3. Navigation Support - Navigation Shortcuts

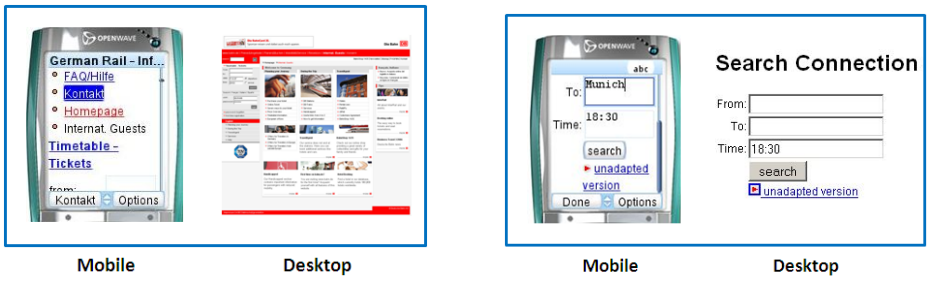
Navigation shortcuts are visualized as balloons (see Figure 5.5) containing a brief description of the event (e.g. "Incoming Phone Call: Elke Halla (SAP)" in Figure 5.5) and the suggested shortcut (e.g. "Contact Page" in Figure 5.5). Further, a bar at the top of the balloon indicates the confidence in the shortcut.

The navigation shortcuts can be presented in two levels of proactivity:



(a) (b)

Figure 5.5. Representations for a navigation shortcut to the contact page of a caller as (a) highlighting and as (b) suggestion



(a) Unadapted **(b)** Adapted

Figure 5.6. (a) Unadapted and (b) adapted user interface

- *Highlight*: Non-intrusive by making the AUGUR icon glow without displaying the suggestion. Thereby, the color used as halo around the icon reflects the confidence in this action. The glowing AUGUR icon makes the user aware that navigation shortcuts are available, but without interrupting her workflow. For displaying the suggestion the user just has to hover over the AUGUR icon.

- *Suggest*: Display the suggested navigation shortcut. The shortcut fades out after a few seconds similar to the notifications used, for example by Microsoft Outlook.

5.1.4. Interface Adaptation

Figure 5.6 shows an example of how the AUGUR prototype automatically adapts the Web site of the Deutsche Bahn to the needs of an exemplary user. The user can at any time switch to the unadapted version of the UI by following the link "unadapted version". This can be necessary if the desired functionality is not present in the adapted version, e.g. because it is novel functionality, or the user does not often use it.

5.2. Controlling AUGUR

As we discussed in Section 2.3.1, one of the key usability issues when designing IUIs is that the user can control its behavior. For that reason, the AUGUR prototype offers various ways to control and adjust its behavior. To enable the user to communicate with the AUGUR prototype, it embeds its icon – a little bird[6] – in the UI as can be seen in Figure 5.7. Right-clicking on this icon opens a menu which allows the user to (i) provide and inspect additional information about the current application ("Application Editor"), (ii) inspect and modify the information in her current user context ("Current Context"), (iii) state her preferences regarding when and how the AUGUR prototype should proactively approach her ("Preferences"), and (iv) activate and deactivate the support. In the following, we briefly describe these four options.

Application Editor

With the application editor, the user can specify further information about the current application, e.g. stating relations between context and interaction elements, or annotating elements. Figure 5.8 shows an example screenshot of the application editor. We provide more details about the underlying application model and its editor in Section 6.3.

Current Context

The *Current Context* stores all context information which AUGUR assumes to be relevant for the user's current interactions (an example screenshot is shown in Figure 5.9). This context is derived from the user's interaction, from external context sources (e.g. location tracking sensors), or from the UI of the application. The user can inspect and modify the context information that is currently stored in her current context. More details about the *Current Context* can be found in Section 6.1.1.

[6] Referring to the augurs in ancient times who interpreted the will of the gods by studying the flight of the birds.

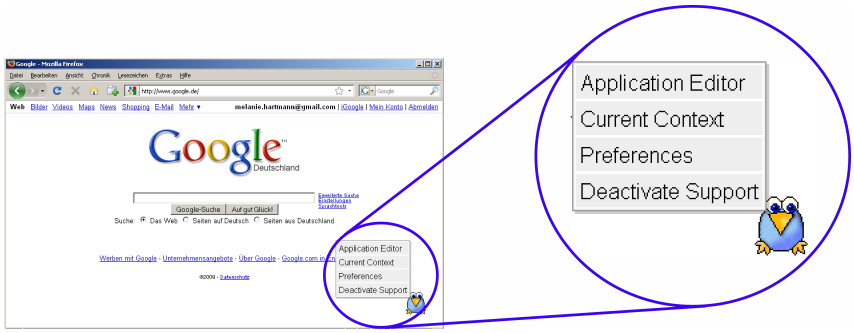


Figure 5.7. Menu of the AUGUR prototype augmenting `http://www.google.de`

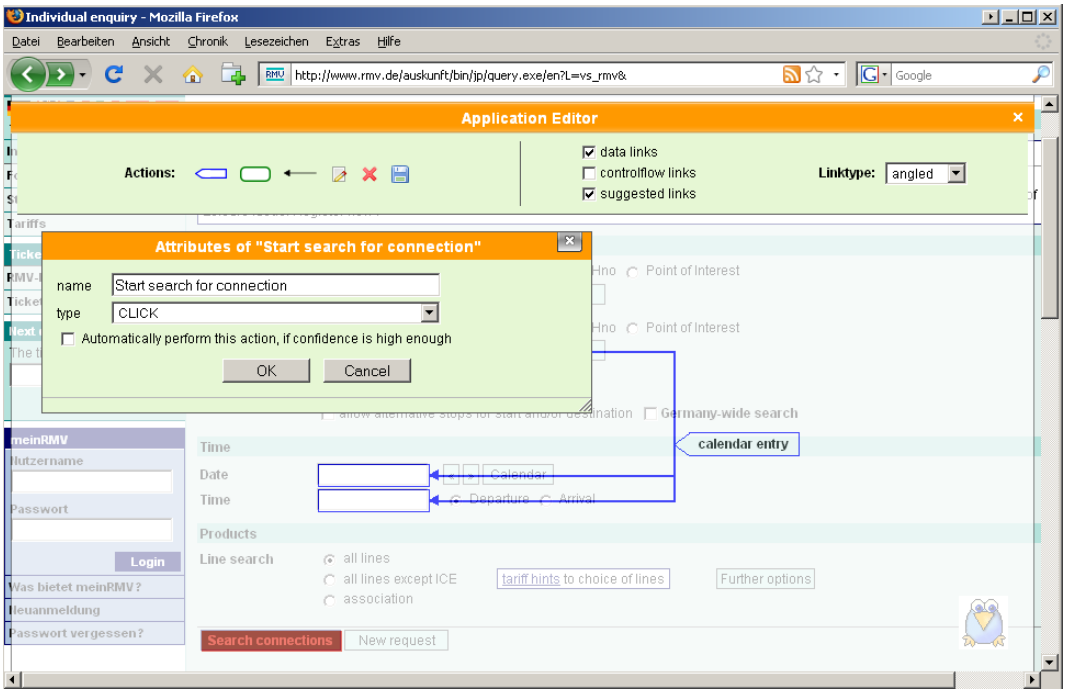


Figure 5.8. Screenshot of the application model editor integrated in AUGUR

Preferences

As stated before, the presentation of the different support types depends on the confidence in the support information and on the preferences of the user. Figure 5.10 shows the dialog for adjusting the proactive behavior, i.e. for setting the thresholds $t_{highlight}$, $t_{suggest}$ and $t_{automate}$. The user can adjust sliders for the confidence that is necessary for the different levels of proactive support. With the exemplary settings

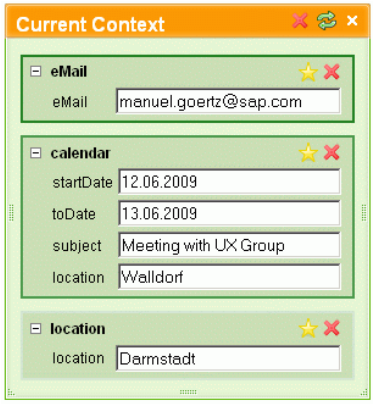


Figure 5.9. Screenshot of *Current Context*

in Figure 5.10, computed navigation shortcuts are ignored if their confidence is below 10%, presented as *Highlight* (i.e. with a glowing AUGUR icon) if the confidence is between 10% and 40% and presented as *Suggest* if the confidence is above 40%.

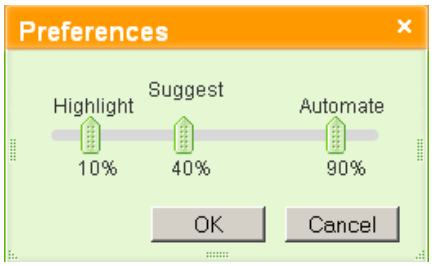


Figure 5.10. Dialog for specifying the thresholds for the proactive presentations

De/Activate Support

The user can easily activate or deactivate the proactive support of the AUGUR prototype using this option. The AUGUR icon indicates whether support is currently activated or not as can be seen in Figure 5.11.

(a) (b)

Figure 5.11. AUGUR icon indicating whether the proactive support is (a) enabled or (b) disabled

5.3. Architecture

As described in Section 4.1, AUGUR uses a proxy-based architecture. The AUGUR prototype receives user requests and forwards them to the corresponding Web application. The returned HTML page is then received by the AUGUR prototype and a modified version of the displayed UI is forwarded to the user. As discussed in Section 4.1, AUGUR can be run in two interface adaptation modes (see Figure 5.12):

(i) **Augmenting UI**: The AUGUR prototype augments the UI with additional proactive features and shares a browser with the user.

(ii) **Generating Adapted UI**: The AUGUR prototype generates a reduced version of the UI. For that purpose, it hosts the UI in an internal Web browser and generates a reduced version of the UI in the desired representation language,

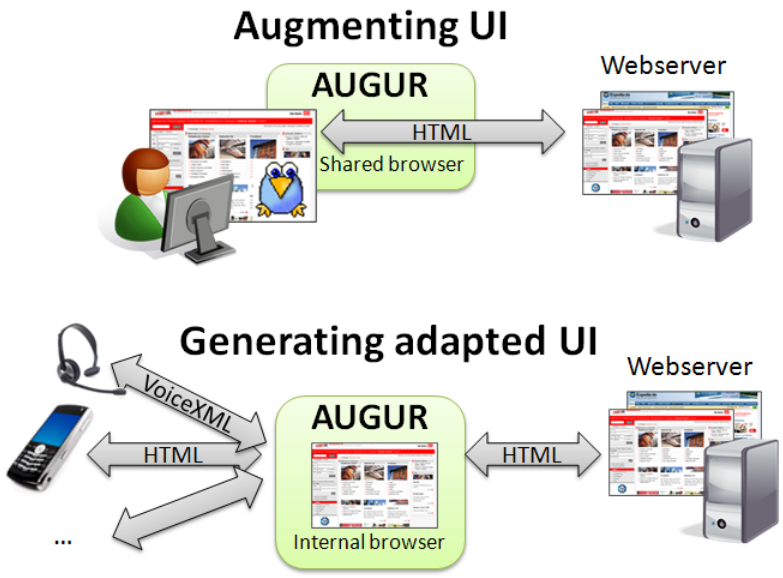


Figure 5.12. Interface adaptation modes supported by the AUGUR prototype

e.g. VoiceXML. The events invoked by the user are forwarded by the AUGUR prototype to the original UI hosted in the internal browser. Changes in the original UI are in turn reflected back to the generated UI. The application on the Web server cannot distinguish between input coming from the AUGUR prototype and input coming from the user.

The AUGUR prototype is thus always coupled to the UI representation that is hosted in a Web browser, either in an internal one or in the browser that is shared with the user. This enables the AUGUR prototype to operate on the actual visual representation of the UI and not on its static representation that is given by its HTML code. Thus, the AUGUR prototype is able to cope with highly dynamic Web pages using scripting languages, AJAX, etc.

In order to provide proactive support, the AUGUR prototype requires push communication, i.e. to send data to the browser without requiring the browser to explicitly demand for it. This is not supported by HTTP. For that reason, we use the JavaScript library Bayeux[7] that simulates push communication via HTTP. As these JavaScript files need to be embedded in the UI of the Web application, we use a proxy architecture that enables us to add these files in the HTTP responses returned by the Web applications. These JavaScript files (i) inform the AUGUR prototype which interaction elements are available and which user actions are invoked (e.g. `onfocus`, `onchange` events), (ii) they are responsible for augmenting the UI with proactive support, and (iii) integrate the AUGUR icon in the UI that allows the user

[7] Bayeux is a protocol for transporting asynchronous messages (primarily over HTTP) with low latency, see http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html

to interact with the AUGUR prototype itself, e.g. for setting preferences.

As described in Section 4.1, the architecture of AUGUR consists of a *Support Tier*, a *Knowledge Base,* and *Editors* as shown in Figure 5.13. The *Support Tier* is responsible for handling the communication between user and application. The *Knowledge Base* provides all the knowledge that is required for that purpose. Finally, *Editors* enable the end-user to access and modify the information stored in the *Knowledge Base*.

5.3.1. Support Tier

The Support Tier intercepts the user's input, forwards it to the application, interprets the result returned by the application, and adapts the output accordingly. The components that are needed for that purpose are (i) an interpreter for the events invoked by the user and the application (*Interpreter*), (ii) a component that generates the required support information (*Support Generator*), and (iii) a component to decide which proactive support should be included in the UI and which parts of the application should be presented at all in the *Generating adapted UI* mode (*Representation Manager*).

At first, the *Interpreter* handles incoming user events with the help of the *Knowledge Base*. It possibly updates the user model and the application model of the current application by adding new elements and new relations. Further, it transforms the events into appropriate actions on the DOM tree of the Web browser. Most events are not directly reported back to the Web application, e.g. entering text in input fields is usually transmitted not until the user presses the submit button. If the event is forwarded to the Web application, the response returned by the application is then again processed by the *Interpreter*, e.g. to update the application model, and sent to the *Support Generator*.

The *Support Generator* takes information about the user's actions and the current UI from the *Interpreter* (e.g. which UI elements are displayed) and generates interaction support. For that purpose, the *Support Generator* contains components for computing navigation and content support, and for determining the most relevant UI elements for the interface adaptation. In order to provide navigation shortcuts, the *Support Generator* is subscribed to context events. The generated support is associated with a confidence value $c_{support}$ which enables the *Representation Manager* to adapt the support according to this confidence.

Finally, the *Representation Manager* takes the support information with the associated confidence information, and decides whether and how this support information should be presented to the user. When interface adaptation is used, the *Representation Manger* is also responsible to generate the adapted UI in the required representation language (e.g. VoiceXML).

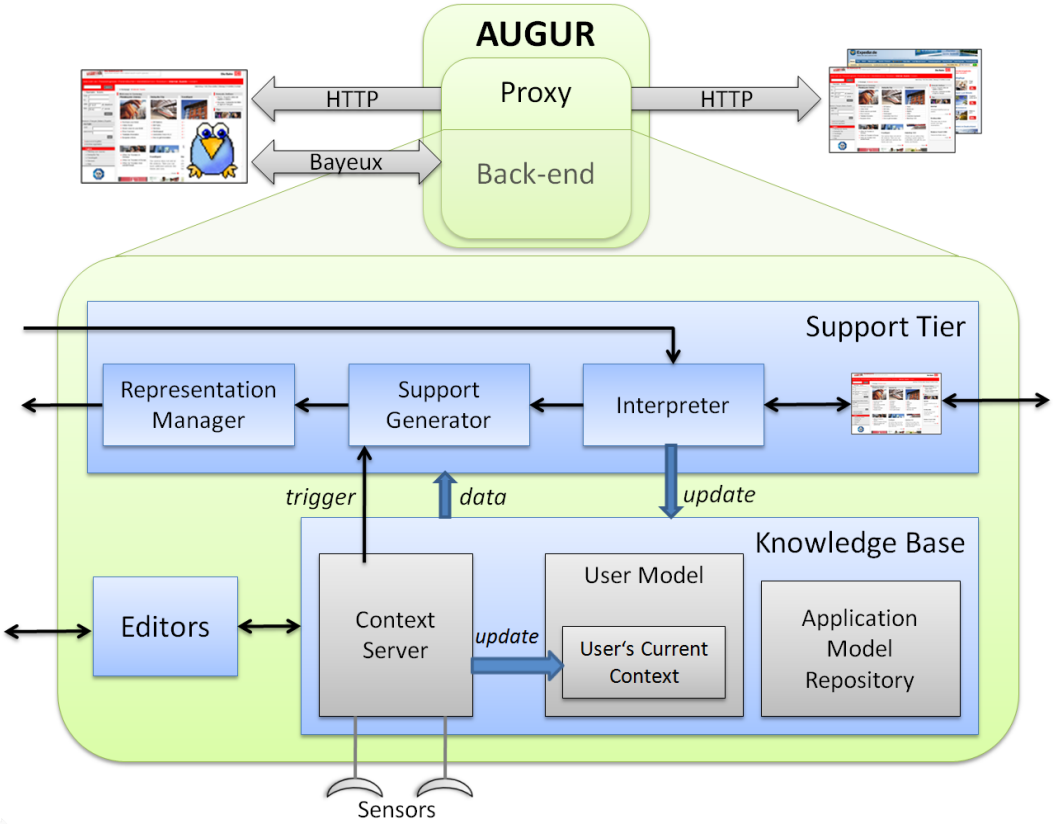


Figure 5.13. Architecture of the AUGUR prototype

5.3.2. Knowledge Base

The **Knowledge Base** provides the information needed by the *Support Tier*. It holds a repository of application models, the user model, and information about current context.

Context Model As mentioned before, the context model consists of two components: (i) a *Context Server* which provides all environmental context information and (ii) the *Current Context* component that stores all context information currently relevant for the user's interactions. To keep the context information, like her location, in the *Current Context* up-to-date, the *Current Context* component is subscribed to the respective context information at the context server.

As implementation of the *Context Server,* we use the Mundo Context Server provided by Aitenbichler et al. (2007b). It supports query- and subscription-based access. It is based on the publish subscribe middleware called Mundo (Aitenbichler et al., 2007a) that facilitates the communication with context sensors by supporting various programming languages and communication protocols. This allows for easily integrating new context sources. The *Context Server* provides the context informa-

tion in a uniform representation and manages the corresponding metadata like the confidence in the context information as will be described in Section 6.1.

As the UI is a valuable source for context information, the *Context Server* contains JavaScript UI sensors for Web applications. The JavaScript files which are required for that purpose need to be embedded in the UI code, or can run as Greasemonkey[8] Scripts. They sense context data, and send it to the *Context Server*. The current UI sensors are able to recognize microformats[9] (a semantic markup for some standard information like address or calendar information) and email addresses.

User Model The *User Model* contains the user's preferences regarding the proactive behavior of the AUGUR prototype and usage models reflecting how the user interacts with applications.

Application Model Repository The *Application Model Repository* stores knowledge about the applications the user interacted with via the AUGUR prototype. This comprises knowledge about the structure of the application as well as semantic information like labels and their relation to context information.

5.3.3. Editors

The user can inspect and modify parts of the models used by the AUGUR prototype via the integrated editors. She can access the *Current Context* as part of the context model, the user's preferences as part of the user model and the application models. This increases the user's trust in the AUGUR prototype. Further, it enables the end-user to enhance the provided support by providing additional information, e.g. about the relations between context and application. Screenshots of the editors are shown in Figures 5.8 to 5.10.

5.4. Chapter Summary

In this chapter, we illustrated how we realized all the concepts introduced in the previous chapter in the AUGUR prototype. The AUGUR prototype is able to provide navigation and content support for arbitrary form-based Web applications. In the next chapters, we will go into detail for the models and algorithms which are needed to support the user in a meaningful way by incorporating user as well as environmental context information.

[8] http://www.greasespot.net/
[9] http://microformats.org/

6 Knowledge Models

In this chapter, we give an overview of the different models that are required for providing content and navigation support based on context information, i.e. the context, application and user model. These models are used by the algorithms presented in the next chapters to generate the interaction support for the user.

In Section 6.1, we describe how the context information in the context model is represented, and introduce a new approach for keeping track of the context information currently relevant for the user's interactions. Section 6.2 lists the components contained in the user model. In Section 6.3, we present a new application modeling language for representing the relations between context and application elements.

6.1. Context Model

The context model comprises all environmental context information provided by the integrated *Context Server*. Further, the context model stores all context information relevant for the user's current interactions in the *Current Context*. In the following, we describe how context information is represented in AUGUR. In Section 6.1.1, we describe which information is stored in the *Current Context* and how its relevance for the user is computed.

Context Representation

The representation of the context information suitable for AUGUR has to meet two requirements:

- **R1$_{CM}$ (Understandability):** The context representation needs to be comprehensible for the end-user. This results from the requirement R6 (*Involving end-user*) stating that the end-user should be able to add additional information about the application including which context information is related to which interaction element.
- **R2$_{CM}$ (Quality Indication):** The context representation should indicate the quality of the context information. This directly results from the general requirement R2 (*Cope with error-prone context*), because AUGUR has to be

aware of the reliability of the context information to adapt the support accordingly.

There exist five approaches to represent context information (Hartmann and Austaller, 2008): key-value, markup-scheme, ontology-based, object-based, and logic-based representations. According to Strang and Linnhoff-Popien (2004) only object-oriented and ontology-based representations are able to inherently support quality indications ($R2_{CM}$). However, ontology-based representations are often criticized of being too complex and difficult to understand (e.g. Robinson et al. (2007)), thus contradicting $R1_{CM}$ (*Understandability*). For those reasons, we decided in favor of object-oriented representations. Each context object O_i consists of an object which represents the actual context information and additional metadata. The context information can range from a simple value (e.g. representing the current temperature) to a more complex object (e.g. representing contact information). The context information is represented as a set of context attributes ca_i with their respective values v_i, i.e. $\{(ca_i, v_i)\}$. For referring to the value of the attribute ca_j of a context object O_i, we will use the abbreviated form $O_i.ca_j$ in the remainder of the thesis.

To fulfill requirement $R2_{CM}$, the metadata stored for a context object has to provide an estimation of its quality. To keep the representation simple and understandable for the end-user ($R1_{CM}$), we focus on a single quality measure as discussed in Section 4.2.1, i.e. the **current confidence** c_{ctx} in the context object. The current confidence depends on (i) the confidence in the context information as it was initially sensed, i.e. c_{init} at time t_0 and on (ii) the time past since the sensing. As the confidence in a context object decays over time, the current confidence does not correspond to its initial confidence c_{init}. We use linear approximation to estimate the current confidence c_{ctx} at time t as follows:

$$c_{ctx}(t) = \begin{cases} c_{init} \cdot (1 - \frac{t-t_0}{ttl}) & \text{if } t < (t_0 + ttl) \\ 0 & \text{else} \end{cases} \tag{6.1}$$

thereby, *ttl* refers to the time to live of the context object. The time to live is defined for every context type. To reduce the amount of information which needs to be stored in a context object, we only store a reference to its type. The *ttl* value is derived from a context type database in the context server which contains all meta-information about a context type. Besides the *ttl* the context type database stores for each context type:

- a human readable *label:* The label is used for providing additional information to the user to make her aware of the knowledge provenance and thus to induce the user's trust in the context information (see Section 2.3.1).

- its potential *attributes:* This information is required when modeling relations between context and interaction elements (see Section 6.3)

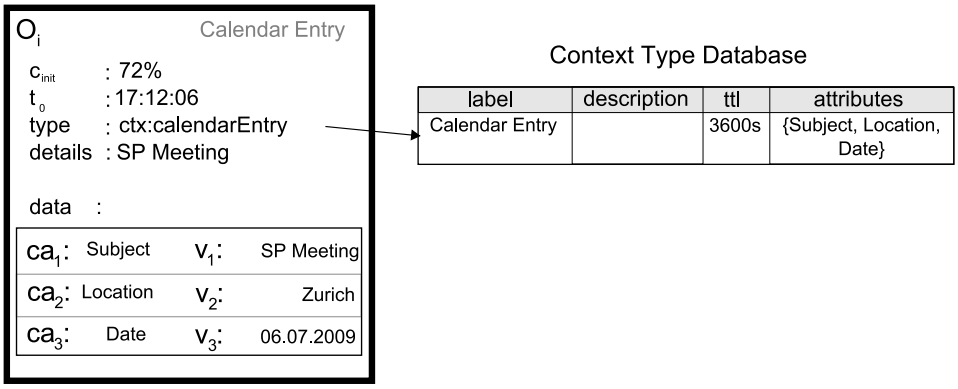


Figure 6.1. Example context object of a calendar

- optionally some further *descriptive texts:* These texts are used for enabling AUGUR to better understand the meaning of the context information and its attributes, and thus to better map context information to interaction elements based on its semantics (see Section 7.3).

A context object can furthermore contain *details* about its knowledge provenance which is not reflected in the label of its context type. For example, a calendar entry can provide details about its subject (e.g. "SP Meeting"). Figure 6.1 shows an exemplary context object representing a calendar entry.

To sum up, each context object used in AUGUR has the following metadata:

- **initial confidence** in the represented context data (c_0)
- **timestamp** of the context item (t_0)
- **type** of the context information referring to an element in the context type database, e.g. `ctx:location`.
- **details** (optional) about its knowledge provenance (e.g. the subject of the calendar entry)

6.1.1. Current Context

The purpose of the *Current Context* component is to store all context objects O_i that might be relevant for the user's current interactions, formally *Current Context*= $\{O_i\}$. The relevant context objects are used for content support and for learning dependencies between context and application elements. In this section, we describe which information is stored in the *Current Context*, how its relevance is modeled, and how it is visualized in the AUGUR prototype.

All context objects O_i that might be relevant for the user can be retrieved from the following sources:

- *user input*, i.e. data the user has entered or selected. This also includes data that was automatically entered on behalf of the user. Entered data is sent to the *Current Context* as soon as the data is submitted to the application, i.e. as soon as the user clicks on a navigational element.

- *the UI*, i.e. data that is delivered by the UI sensors of the *Context Server* (see Section 5.3), e.g. address and appointment information that are marked with microformats tags.

- *associated context data*, i.e. related context objects like the calendar entry in the train example. If the current application is associated with context data of a given type via its application model, AUGUR subscribes to this information and stores it in the *Current Context*. The corresponding context information is then constantly updated by the *Context Server*.

Computing Relevance of Context Objects

For each context object, we need to be aware of its relevance r_{user_ctx} for the user to provide only meaningful support. The computation of the relevance r_{user_ctx} has to meet the following two requirements:

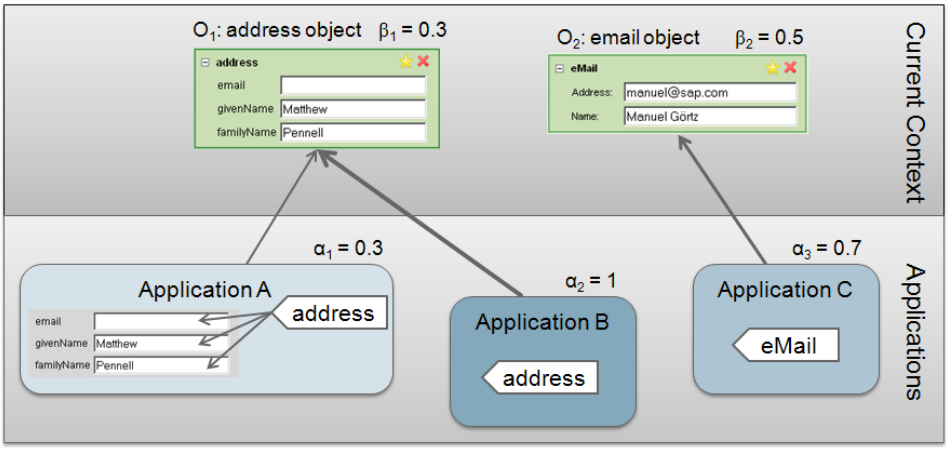
- **R1**$_{UM}$**:** The relevance r_{user_ctx} in a context object in the *Current Context* should decay over time. The decay prevents that the *Current Context* gets overloaded with –in the meantime– irrelevant data.

- **R2**$_{UM}$**:** If the user switches between applications, the context for the respective application should get (re)activated, i.e. its relevance r_{user_ctx} should increase. This is especially important when resuming interrupted tasks.

To our knowledge there exists no state-of-the-art system for supporting the user's interaction that considers the relevance of context information for the user. In this section, we present a novel approach for mimicking the user's short term memory, and thus keeping track of the relevant information. We thereby build on the widespread cognitive architecture ACT-R (Anderson and Lebiere, 1998) to meet requirement R1$_{UM}$. ACT-R models the activation of information in the user's short term memory. In our application, the relevance of a context object is also determined by its confidence c_{init} when being stored in the *Current Context*. Thus, we define

$$r_{user\_ctx} = c_{init} \cdot a \tag{6.2}$$

c_{init} is 1 for data that is gathered from the user's input as we assume it to be correct. For UI data and associated context data, c_{init} is c_{ctx} provided by the *Context Server* (see Section 6.1). As soon as the user uses information from the *Current Context*, i.e. fills it in input elements, c_{init} is set to 1 as the context information was then confirmed by the user, and can thus be considered correct.



$$a_1 = \beta_1 + \alpha_1 + \alpha_2 = 0.3 + 0.3 + 1 = 1.6 \Rightarrow a_1 = 1$$
$$a_2 = \beta_2 + \alpha_3 = 0.5 + 0.7 = 1.2 \Rightarrow a_2 = 0.8$$

Figure 6.2. Example calculation for the activation of context objects O_1 and O_2

In ACT-R, the activation a of an object is determined by its own base activation β that decays over time and by the activation a_i of all associated objects, the so-called associated activation sources. For our application domain, we have to consider the relations between applications and context information to address requirement $R2_{UM}$. This means that the activation sources for a context object are all applications which are related to the context object. This comprises applications from which the data was gathered, or that are associated with the given context type via their application model. For future work, we could also consider the relations between the context objects themselves. However, this would require a common ontology that can state these relations, e.g. location is related to travel. The activation a of a context object is defined as[10]

$$a = \beta + \sum_j \alpha_j$$

with α_j being the activation of an associated application. As stated before, the base activation β and the activations α_j decay over time. ACT-R models this fact with the power law of forgetting

$$\beta = -ln(\Delta t/T)$$

and

$$\alpha_j = -ln(\Delta t/T)$$

where Δt is the time since last usage and T a time scaling factor (i.e. after time T the activation level equals 0). To obtain only activation values between 0 and 1,

[10] ACT-R further considers the strengths of these associations. For simplification, we assume that they are all equally strong.

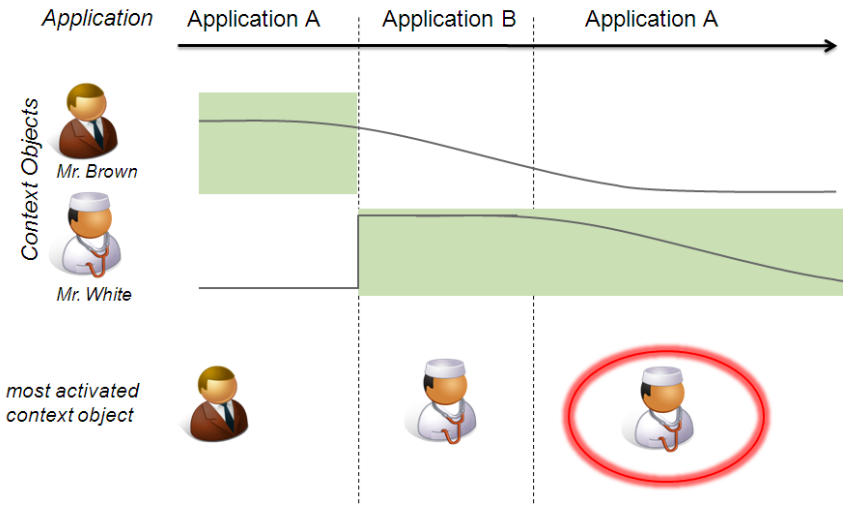


Figure 6.3. Example activation flow using normal decay

we map all activation values > 1 to 1 and all values < 0 to 0. T and Δt are not necessarily measured in real time, also alternative clocked measures can be used. For our purpose, we use the user's actions as clock. As the resulting c_{user_ctx} has to be in $[0, 1]$ again and $a \in [0, \infty[$, we normalize this value with $a = MIN\{a/a_{max}, a_{max}\}$. Thereby, a_{max} is an upper bound for the activation value. We empirically determined 1.5 to be a good estimate for this value. If the confidence c_{user_ctx} of a context object drops under the user-defined threshold $t_{suggest}$ for suggestions, it is removed from the *Current Context*, because it is then no more relevant for any content support.

Figure 6.2 shows an example calculation to illustrate the computation: a context object "address" is associated with applications A and B and a context object "email" with application C according to the respective application models. The user interacted with the applications in the following order: A,C,B. This results in the different activation values α_1, α_2, and α_3. As the context objects are already in the *Current Context* without being used, their base activation β_1 and β_2 has decayed to 0.3 and 0.5, respectively. This results in an activation for the context object "address" of 1.6 which is normalized to 1 and an activation value for the context object "email" of 1.2, normalized to 0.8.

To show that our proposed model meets $R2_{UM}$ and thus ensures that the context for an application gets reactivated if the user switches to it, we illustrate it with another example shown in Figure 6.3: At first the user is editing the contact of Mr. Brown with application A, then Mr. White calls and the user has to switch to application B. If the phone call is over, she switches back to application A where the context object representing Mr. Brown should again be more relevant than the one of Mr. White. If we use normal decay of the context objects without considering the related applications, the context object Mr. White will get the higher activation values, thus violating $R2_{UM}$. If we however also consider the activation of the related

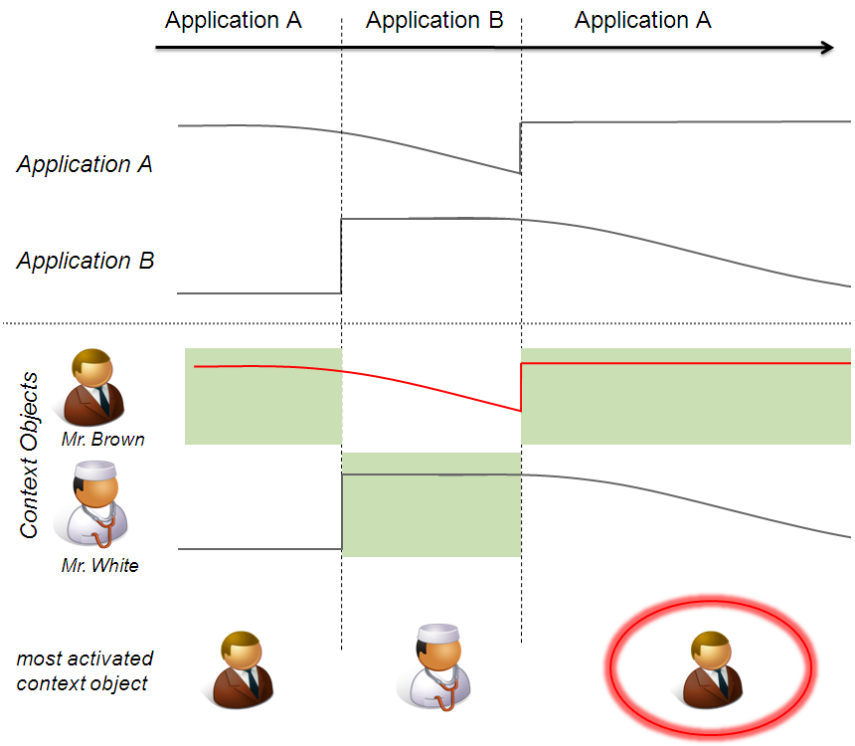


Figure 6.4. Example activation flow considering the activation of the applications

applications as proposed before, this results in a flow of activation as illustrated in Figure 6.4: If the user switches back to application A, the context object for Mr. Brown is more activated, thus $R2_{UM}$ is met by our proposed approach.

Visualization in the AUGUR prototype In the AUGUR prototype, the user can always inspect and modify which information is currently stored in the *Current Context* (see Figure 6.5). Each context object is represented as a box. The relevance r_{user_ctx} of a context object is visualized in its border color ranging from white (relevance 0) to dark green (relevance 1). The user can edit and remove (✖) context objects, or increase the confidence in them (★), i.e. setting r_{user_ctx} to 1.

All interaction elements that are associated with a context type via the corresponding application model are grouped as one context object of this specific type. In contrast to data from the UI and associated context, data gathered from the user input is often not associated with a context type because this would require that the context type is specified in the corresponding application model. In this case, it only consists of a set of labels with associated values, each representing an interaction element. They are grouped as an "Unknown" context object with an *edit* button (✎) and we leave it up to the user to associate it with a context type (see Figure 6.5).

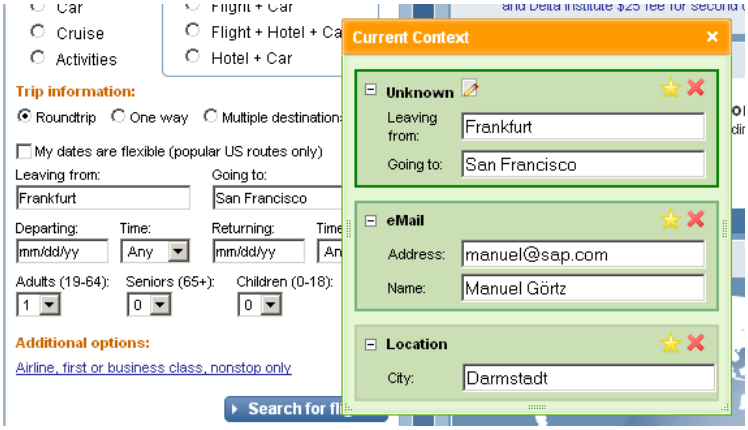


Figure 6.5. Example for *Current Context* with three context objects.

6.2. User Model

In this section, we present the user model applied in AUGUR. The user model contains:

- the *preferences* of the user regarding the proactivity of the support (i.e. the thresholds $t_{highlight}$, $t_{suggest}$, and $t_{automate}$ as described in Section 5.2)
- *Usage Model U* for every application: The usage model describes how the user interacted with an application in the past to infer future behavior of the user. It is the basis for the *Guidance* and the *Interface Adaptation* feature of AUGUR. The corresponding algorithms will be described in Section 8.1 and 8.3.

In the following, we describe the *Usage Model* in more detail.

6.2.1. Usage Model

A usage model $U(x)$ stores a model of the user's interaction history with an application x. Each action a_i in the interaction history refers to an activity in the corresponding application model. In the example application of looking for a train connection, a typical usage sequence is to type in the place of departure, the destination and then to submit the form. We do not store the entire user history, but only the frequency fr of observed sequences $a_1, ...a_n$ up to a predefined length n. This reduces the amount of data to be stored, and suffices for obtaining a good performance as will be shown in Section 8.1.6. The data is stored in a *trie*, because this represents an efficient storage structure for sequence data. An example can be found in Figure 6.6. After every observed action a_n, the usage trie is updated as described in Algorithm 1. Thereby, Σ is the set of available actions in the application model and $\alpha \in [0, 1]$ is an aging factor to reduce the influence of older sequences. The update mechanisms thus increase the influence of the currently observed sequence in comparison to all related sequences, i.e. all sequences with same prefixes. We do not

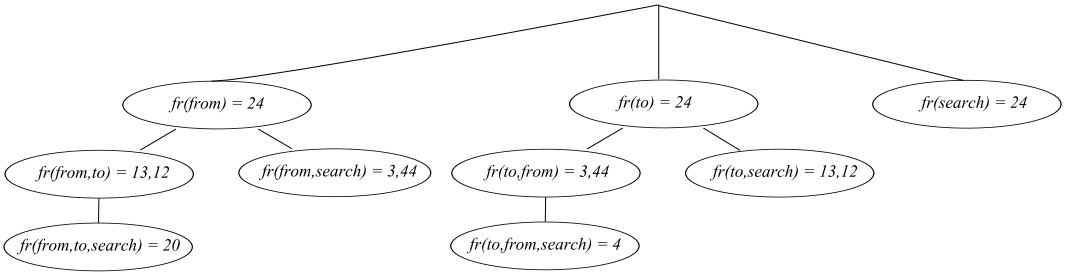


Figure 6.6. Example *Usage Model* trie for 24 traces for the train booking application with "from","to", and "search" interaction element (for $\alpha = 0.9$).

apply the aging to all stored sequences, because sequences that are not related to the current task of the user should not be affected by the aging. For example, when looking up a train connection this should not influence the parts of the usage model that deal with setting the personal preferences on the Bahn website.

Algorithm 1 Update Usage Model

for $(i: 1 \text{ to } n-1)$ **do**
  for all $(x \in \Sigma)$ **do**
    $fr(a_{n-i}...x) \leftarrow \alpha \cdot fr(a_{n-i}...x)$
  end for
  $fr(a_{n-i}...a_n) \leftarrow fr(a_{n-i}...a_n) + 1$
end for

6.3. Application Model

An *Application Model* represents all interaction elements which are available in an application. In the case of form-based applications, these are the different form elements and links. In the ideal case, an initial application model is provided by the application developer as not all dependencies can be learned reliably solely from observation. However, the initial application model can also be automatically created by monitoring the user's interaction with an application. In order to adapt the support to the user's needs and to reduce the effort for the user, the user's interactions are constantly observed to learn more relations between activities, states, and context information.

In the following, we at first list requirements for an application modeling language for context-aware IUIs (Section 6.3.1). In Section 6.3.2, we analyze existing modeling languages in this respect. From this analysis, we come up with our own application modeling language called ATML which overcomes the limitations of the existing languages (Section 6.3.3). As introduced in Section 5, we focus on Web applications for the AUGUR prototype. However, for Web applications the most important information for describing UI elements, i.e. their labels, cannot be directly determined from

the HTML representation of the UI. For that reason, we developed a novel approach for extracting the label of UI elements from the visual layout (Section 6.3.4). This algorithm called LabelFinder outperforms existing algorithms and is able to cope with highly dynamic Web pages. In Section 6.3.5, we finally show how the ATML models are visualized in the AUGUR prototype and how the user can modify them.

6.3.1. Requirements for Application Modeling Language

As we consider the end-user as an important information source for adding additional knowledge (R6 *Involving end-user*), the *Application Model* has to be easy to understand and update for the end-user. Further, it has to support constructs for additional semantic information (like relations to context) in a machine readable way. Thus, we define the following requirements for an application modeling language suitable for AUGUR:

- **R1$_{AM}$ (Intuitive Visual Representation):** The resulting application models have to be comprehensible for the application designer and for the end-user with minimal IT experience (R6 *Involving end-user*). The easier it is to add additional semantics, the easier it is to improve interaction support beyond the state reachable with pure machine learning.
- **R2$_{AM}$ (Model relations to context):** As discussed before, context is a primary information source for supporting the interaction with an application (R1 *Awareness of user context and environmental context*). For that purpose, the application modeling language has to be able to model relations between context and interaction elements. These relations can be *direct relations* (i.e. context information X can be used as input for interaction element Y) or it can be specified by *rules* (e.g. if the location is at home, then enter "Heidenreichstr." in the "to" input element).
- **R3$_{AM}$ (Mapping to existing UI):** In order to fulfill R5 (*Support for legacy applications*), the elements of the application model have to be connected to the elements of an existing UI. This mapping should also be easy to perceive for the end-user.

6.3.2. Existing Application Modeling Languages

In this section, we analyze how existing application modeling languages meet the requirements we identified. Many application modeling languages are provided by the model-based UI community. For example, the widespread ConcurTaskTree (CTT) formalism by Paterno et al. (1997) structures an application into hierarchical tasks that are connected with temporal relation operators. These tasks are coupled to abstract UI elements like "select 1:n" and transformed into a concrete UI representation at runtime. Other examples of those languages are UIML (Abrams et al., 1999) or XIML (Puerta and Eisenstein, 2002). Many of these languages also bring their

	$\mathbf{R1}_{AM}$	$\mathbf{R2}_{AM}$	$\mathbf{R3}_{AM}$
Model-based UI approaches (CTT, UIML, XIML)	-	-	-
UML (Activity Diagrams, State Charts)	•	-	-
XPDL	○	-	-

Table 6.1. Comparison of existing application modeling languages

own editor for specifying these models (e.g. (Paterno et al., 2008)), however they are usually build for application developers and are not intuitive to understand for end-users ($R1_{AM}$: -). Furthermore, as their goal is to provide an application independent representation, they cannot be linked to existing UI elements ($R3_{AM}$: -). Finally, they do not consider context information or model it only as attributes for their elements, which makes relations between application and context difficult to perceive and model for the end-user ($R1_{AM}$: -, $R2_{AM}$: -).

Application modeling languages that focus on the application itself like UML or XPDL[11] are more wide-spread. For that reason, there exist editors that are usually more intuitive to understand for the end-user with minimal technical knowledge ($R1_{AM}$: o) than those from the model-based UI community. As UML models are widely used, they are usually well understood by the end-user ($R1_{AM}$: +). However, they do not support relations to existing interaction elements ($R2_{AM}$: -) or context information ($R3_{AM}$: -).

Table 6.1 summarizes the requirements that are met by the existing application modeling languages. There are workaround solutions for most modeling languages to meet the requirements $R2_{AM}$ and $R3_{AM}$, e.g. by modeling every additional piece of information as an extra attribute. However, these workarounds should be avoided, as they introduce additional complexity into the resulting models. Thus, the most important information cannot be seen at first glance, which decreases understandability and thereby violates $R1_{AM}$. For that reason, we decided to develop our own application modeling language called ATML which we introduce in the next section.

6.3.3. ATML: ApplicaTion Modeling Language

In this section, we describe the application modeling language that we developed to overcome the shortcomings of the existing modeling languages for usage in AUGUR.

Our application modeling language called ATML (ApplicaTion Modeling Language)[12] builds on UML diagrams. We use a combination of activity diagrams and state charts, because this maps naturally to the user's view of an application, where states represent the different UIs of a form-based application (for Web application these are the different Web pages), and activities represent the different interaction elements available in this UI. The model is visualized as an overlay to the existing

[11] http://www.wfmc.org/xpdl.html
[12] Formerly also known as AUGUR Task Modeling Language

application. Each activity is displayed on top of the corresponding interaction element to facilitate the perception of the mapping from application model elements to UI elements ($R3_{AM}$).

An overview of the node types and attributes of ATML can be found in Figure 6.7. The application model is stored in an XML-format similar to XPDL (the corresponding DTD can be found in Appendix A.1). In the following, we illustrate the elements of our application modeling language with a simple example (see Figure 6.8): The Deutsche Bahn website provides a UI for querying train connections (Figure 6.8 (a)). It contains input elements for the place of departure ("from"), the destination ("to"), the "date" and the "time" of the planned train travel. Further, the user can select whether "date" and "time" state the time of departure or time of arrival using two radio buttons ("departure" and "arrival"). The entered data can be submitted via the "search" button. The search results are then displayed containing buttons for purchasing the respective train tickets (Figure 6.8 (b)). In the following, we present the different components of ATML using this example. We describe their attributes and how the relations between them can be modeled. Most of the above attributes are optional, but the more additional information is specified, the better assistance can be provided.

State nodes Each state node in the application model refers to a UI. For our running example, this are the "bahn.de" and the "Your timetable" state as illustrated in Figure 6.9. The state is associated with a UI via a reference *ref* which is stored as attribute of this node. For Web pages the URL itself is not a good choice for the reference as the URL often contains many additional parameters (e.g. a session key) that change from one usage of the application to the next. For that reason, we cut off all the parameters of the URL. However, this sometimes generalizes too

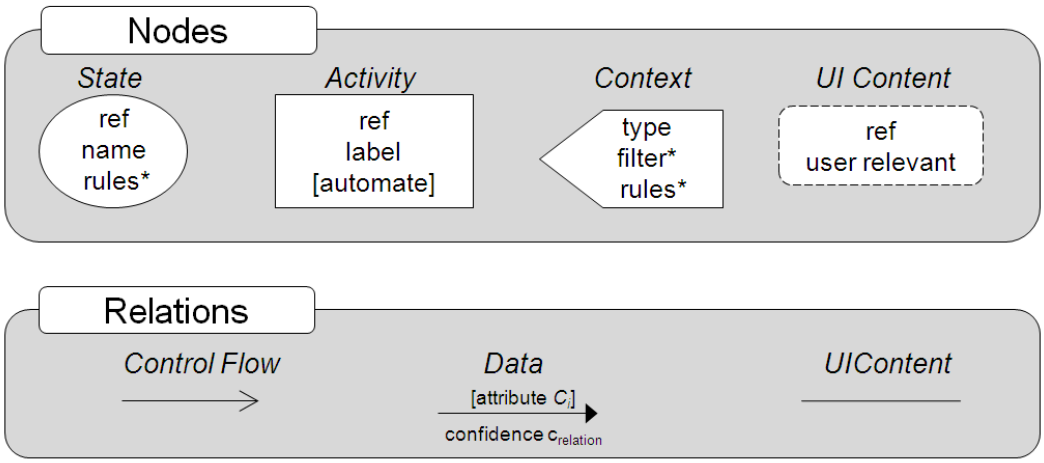
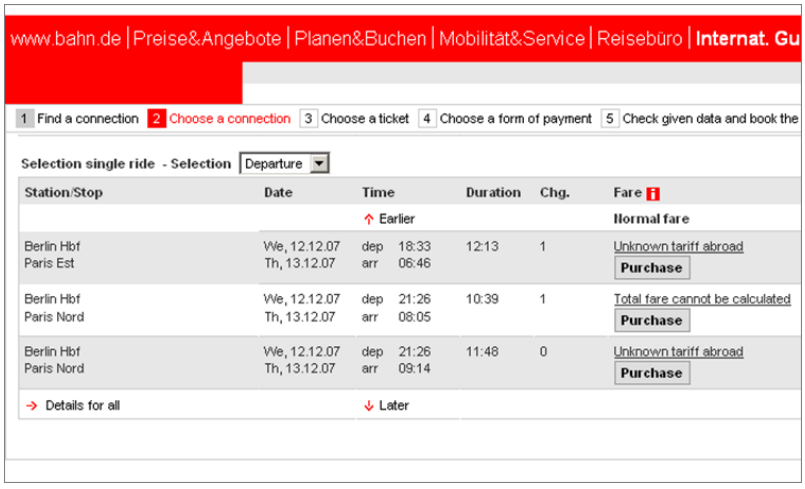


Figure 6.7. Components of ATML and their attributes.

much when applications offer different functionality that is just distinguished by a parameter. Hence, we add the title of the Web page to the URL which usually identifies the content on the current Web page better. For example, the Web page of the Deutsche Bahn with the URL `www.bahn.de/sth?sessionid=13&...` and the title `Your timetable` results in the id `www.bahn.de/sth[Your timetable]`.

We also store the *name* of the UI as an attribute of the state node. This name is used as title for generated UIs and eases the identification of the state for the end-user when editing the application model. For Web pages, we store the title of the page as name.

(a) Search



(b) Search results

Figure 6.8. Example screenshots of the DB website (`http://www.bahn.de/`) for (a) searching train connections and for (b) displaying the results

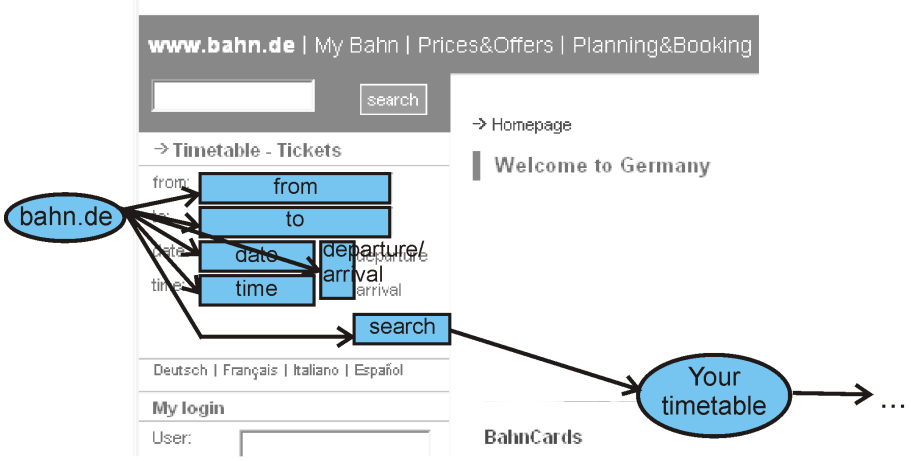


Figure 6.9. Control flow of the search UI of our running example

Activity nodes Each activity node is coupled to a UI element via the *ref* attribute that unambiguously identifies the corresponding interaction element within a UI, thus satisfying requirement $R3_{AM}$. For Web applications, this is their XPath expression. Further, each activity node contains a *label* for the interaction element. For Web applications, the label often cannot be directly gathered from the HTML representation. For that reason, we developed an algorithm called LabelFinder (presented in Section 6.3.4) for inferring the label from the visual representation which outperforms existing approaches.

Moreover, activities describing navigational elements, i.e. buttons and links, have a boolean attribute *automate* which states whether AUGUR should automatically click on these elements if the confidence in the corresponding navigation support is high enough.

Control Flow The state nodes are linked via control flow relations to activity nodes which correspond to the interaction elements on the UI. Activity nodes again are linked to the state which the user reaches when executing the activity. For example, navigational elements in Web applications usually lead to a new Web page, and thus to a new state in the application model. However, for interaction elements for entering text this is usually the same state as before. For ease of readability the transition from the activity to the state node is omitted in the graphical representation in those cases. Figure 6.9 shows the ATML graph of the search UI of our example with all its control flow relations.

Context nodes To represent context data, we introduce context nodes that represent specific context types. In our example, the input required by the application is related to "location" and "calendar entry" context information. The location information can be used to suggest content for the "from" field and the location, date and

time of a calendar entry can be used as input for the "to", "date" and "time" field.

The context data that is referred to by a context node can be limited by specifying *filters*, i.e. conditions like "equals" or "contains" for each of its attributes. For example, only those calendar entries are considered for which a location is stated (i.e. its location attribute does not equal the empty string). The *type* attribute of the context node refers to an element in a common context type database, e.g. `ctx:location` (see Section 6.1).

Data relations Data relations connect these context nodes with activity or state nodes, thus allowing to model direct relations as required by $R2_{AM}$. Figure 6.10 shows the data relations for our running example. Data relations to activity nodes are used for providing content support, data relations to state nodes for navigation shortcuts. If a context node is linked to a state node, this means that a navigation shortcut to the corresponding UI is suggested to the user if an event of the given context type is raised. In the following, we go into detail of modeling direct relations between context and activity nodes and its implications for the provided content support.

An activity node can be linked to several context nodes, as often information can be obtained from various context sources, e.g. the user's location can be gathered from location sensors or from entries in her calendar.

For context types that have more than one attribute, we have to specify which of these *attributes* should be actually used to compute content support for an associated interaction element. In our running example, we need to specify that the "location" attribute of the "calendar entry" is related to the "to" interaction element. This information is stated as additional attribute of the data relation. If the relevant attribute is not specified, the user's interactions are observed in order to infer the missing information.

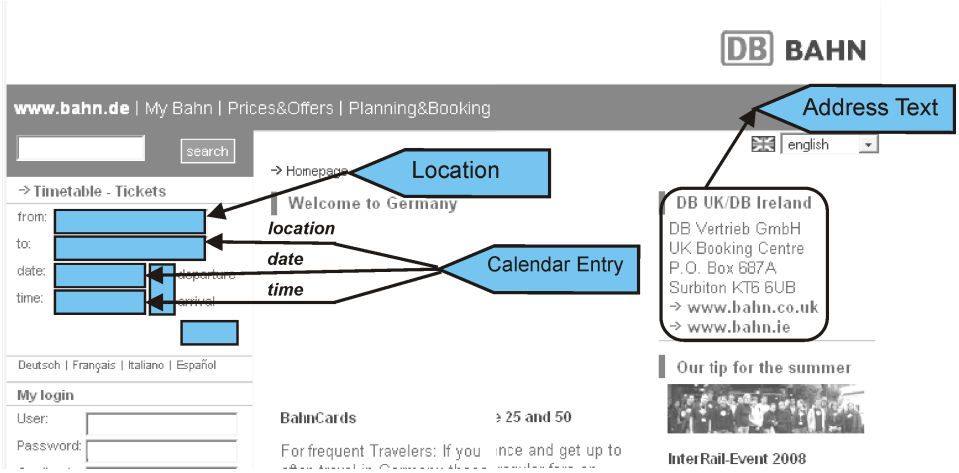


Figure 6.10. Data relations of the search UI of our running example

In many cases, a context node is linked to several activity nodes, like the "calendar entry" in our example. If this context node is used for generating support, only data for the associated interaction elements is suggested that is obtained from the same context object. This means that the content of several context objects is not mixed. For example, "location", "date", and "time" for one suggestion are always derived from the same "calendar entry" object in our running example. If the user has already entered data in one of the relevant interaction elements, only data is suggested that matches this input. For example, if the user already entered the travel date, only those locations from calendar entries are suggested as travel destinations whose date matches the already entered value.

The direct relations between context and interaction elements can also refer to fixed values. For example, if data from a calendar entry is suggested in our running example, the "departure/arrival" option should be set to "arrival", because the start time indicated in a calendar entry usually does not consider the time needed for traveling.

A context node can also be associated with an application without specifying any data relations. This is the case if the user is unsure about the exact relation between context and interaction elements, or if she is unwilling to provide this information. The relevant data relations are then learned from observation. As it is not possible to consider all context information that is potentially available, these "unbound" context nodes state which context information should be considered in the learning process.

If the data relations are learned from observation, they are not always reliable. For that reason, we store for each learned data relation its *confidence* $c_{relation}$ in terms of how often this relation has been observed, and how often this relation could have been observed. The confidence in the relation influences the confidence in the corresponding content and navigation support, and thus whether and how it is presented to the user (see Section 5.1.1).

Rules Besides specifying these direct data relations between context and activity nodes, we also want to be able to model more complex relations in form of rules ($R2_{AM}$). We define a rule as follows:

`if (<contextAttribute>=<value>|<interactionElement>=<value>)`$^+$
`then (<interactionElement>=<value>)`$^+$

with `interactionElement` being the id of an activity node. The rules are stored in the corresponding context nodes. For example, in a context node describing seminar talks, we could specify a rule like `if advisor="Daniel Schreiber" then area="Smart Interaction"` (with `advisor` being an attribute of the context and `area` the id for a UI element) which means that "Smart Interaction" is entered in the `area` field if the advisor is "Daniel Schreiber". Similar rules can also be specified for state nodes to describe interdependencies between activities. The only difference is that no `<contextAttribute>=<value>` conditions can be defined. As the rules can also be learned, they are associated with a confidence value c_{rule}, similar to $c_{relation}$

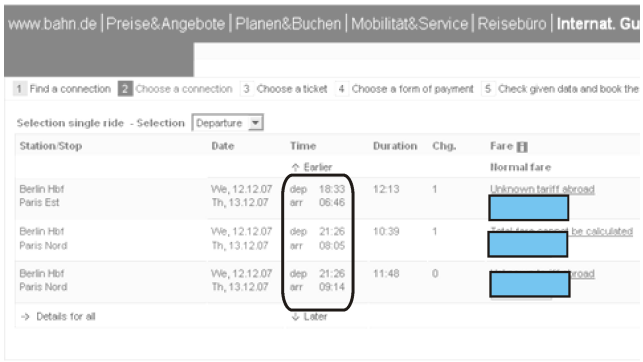


Figure 6.11. Activity and UIContent nodes for the search result UI of our running example

for data relations (for more details see Section 7.4.3).

UI Content node The UI is a valuable context source, however it is difficult to determine which part of the UI can provide which context information. There exist some semantic markup languages like microformats for address and appointment information which can provide a hint of the meaning of the represented information. This information is automatically used by the *Context Server* as described in Section 5.3. However, such semantic markup is not widely used, as it induces additional modeling effort for the application developer. For that reason, ATML supports to annotate relevant information in the UI using UIContent nodes. These are coupled to the UI via the *ref* attribute (the XPath expression for Web applications) and are linked to the corresponding state via a **content relation**. They can then be linked to context nodes via data relations (see e.g. the address information in Figure 6.10). If the user navigates to the corresponding UI, the data contained in the respective part of the UI is interpreted as context information of this specified type and sent to the context server which can then extract further information from it, e.g. address information. An alternative approach is to use natural language processing on the whole Web page to extract the relevant information. However, not all information is as easy to identify as address information. For that reason, we limit the information which needs to be analyzed, and allow the user to annotate it with the context type which should be searched in the data.

The UIContent nodes are also required for generating a scaled down UI of an application. In this case, we need to know which information provided by the application is of interest to the user and should be displayed. However, this is very difficult to determine for non-interactive elements like the timetable in our running example (see Figure 6.11). For that purpose, they can also be annotated as UIContent nodes. As not all UIContent nodes that provide context information are relevant for the user, UIContent nodes have an additional attribute *user relevant* for tagging the elements which should be displayed in a generated UI.

6.3.4. LabelFinder: Recognizing Labels

For the ATML representation of activities and for semantically mapping context to UI elements (see Section 7.3.8), we need to be able to determine the labels of the different interaction elements. The human readable label is the most descriptive representation that is located somewhere near the input element in the visual representation of a Web page (e.g. "from" in Figure 6.8 (a)). For Web applications, the HTML syntax defines a tag LABEL for marking a label that is associated to an input element; however, it is rarely used in practice (only about 20% of the input elements we used for evaluating the performance of label recognition approaches had an associated label attribute).

As most Web forms are similar in their layout, we assume that we can define some common heuristics that are applicable to a wide range of Web applications. Our approach called LabelFinder thereby focuses on the actual visual layout. In contrast to most existing approaches, it also considers the exact visual position of texts to make it independent of the underlying HTML structure.

Related Work Some approaches for extracting the label rely on a high-level description of the UI that is extracted from the HTML representation. Kaljuvee et al. (2001) apply string matching to find the best match for the input element's name attribute from the text elements surrounding the input element. He et al. (2004) define heuristics to determine the best label. However, these approaches have the drawback that they only consider a simplified textual representation of the website and not its actual visual layout. Raghavan and Garcia-Molina (2001) address this problem by rendering a pruned version of the HTML representation with a custom layout engine for gaining the visual layout. However, this does not cope with the growing complexity and dynamic layout of today's Web applications, e.g. using Ajax. Zhang et al. (2004) tackle this by using the HTML DOM API of a browser. They introduce a grammar (called 2P grammar) that describes the visual patterns in terms of directions (left, above, etc.). This grammar is used to gain a parse tree for the UI. Their best effort parser combines multiple possible parse trees to get the best representation of the Web page.

Another system working on the actual visual representation is CoScripter (Little et al., 2007), a popular Firefox plugin for recording intelligent macros, which also uses heuristics to assign a label to each input element. As CoScripter is publicly available, we also use it as a baseline system for our evaluation.

LabelFinder For identifying the best label for each input element, we at first determine all available input elements and all potential label candidates, i.e. all text elements on the website. Every input element is represented by its coordinates, its size, its type, and its HTML LABEL attribute, if available. A label candidate is also described by its coordinates and its size. Further, its representation contains its textual content and its type as described in the following. The text elements are often embedded into larger divisions, thus the exact position of the texts themselves

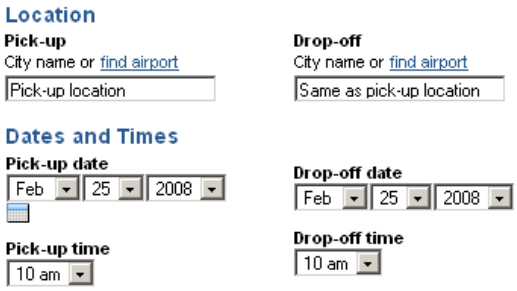


Figure 6.12. Example form (http://www.usairways.com)

cannot be directly determined. For coping with this problem, we temporarily insert a SPAN tag around them, and determine its position. The type of the corresponding label candidate is referred to as **inner label candidate**. However, as labels sometimes refer to various input elements and are thus not placed directly above all of them (see e.g. "Pick-up date" in Figure 6.12), we also keep the position information for the surrounding HTML tag that usually spans a greater section (**outer label candidate**).

From the analysis of various forms, we found that most labels are positioned on top or to the left of the corresponding input element. Checkboxes and radio buttons often do not have an explicit label. Especially ungrouped checkboxes are hardly ever explicitly labeled. For that reason, we also determine the labels of single checkboxes and radio buttons that are usually located on their top or to their right. Thus, grouped checkboxes and radio buttons have two possible labels: the **group label** or the **element label** in the beginning.

In the following, we list the heuristics which we apply to determine the best label candidate for an input element:

1. If the element has a corresponding LABEL element, we take it as label.
2. We ignore
   (i) all label candidates that are not located directly above or on the corresponding side (depending on its type) of the input element, i.e. all candidates that do not have a minimal predefined overlap in the corresponding dimension,
   (ii) all label candidates that have another label candidate between them and the input element, and
   (iii) *outer label candidates*, if an *inner label candidate* is available. For every remaining label candidate we compute the minimal Euclidean distance d between the input element and the label candidate. We take the square root of the distance for the horizontal dimension, as the horizontal distance grows much faster than the vertical distance (e.g. if another input element is positioned between the element and the label candidate). Finally, we take the label candidate with the smallest d.

3. For grouped checkboxes and radio buttons: If the distance d of the *group label* is smaller than n pixels (we empirically determined $n = 15$ to yield good results), we take it as the best label. Otherwise, we assign the label of the currently checked element to the group, or the label of the first element if no element is checked.

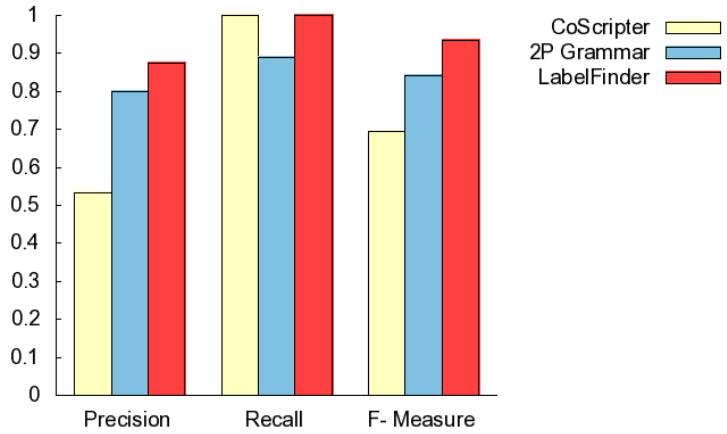


Figure 6.13. Overall performance of label recognition approaches

Evaluation For evaluating the heuristics, we use the *IWRandom* dataset UIUC (2003) provided by Zhang et al. (2004). The dataset contains 33 forms randomly sampled from the Web, mainly gathered from the website `invisible-web.net`. Two forms were dropped as they were difficult to annotate even for humans.[13] In total, we analyzed 216 interaction elements. We compared the performance of our LabelFinder with the results reported for the 2P Grammar by Zhang et al. (2004) and with the labeling component used in CoScripter (Little et al., 2007). Figure 6.13 shows the micro-average results in terms of precision, recall, and the resulting F-measure. The F-measure is defined as:

$$F = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{6.3}$$

LabelFinder reaches a precision of .88 and perfect recall, resulting in an F-measure of .93. It thus clearly outperforms the label recognition of the 2P Grammar (F =.84) and of CoScripter (F =.69).[14] On the dataset that we also used for the evaluation in Section 7.3.7, LabelFinder yields a precision of .95 and perfect recall resulting in an F-measure of .97.

[13] Zhang et al. also used only 30 of the forms for their evaluation.
[14] The macro-average results are consistent with these findings.

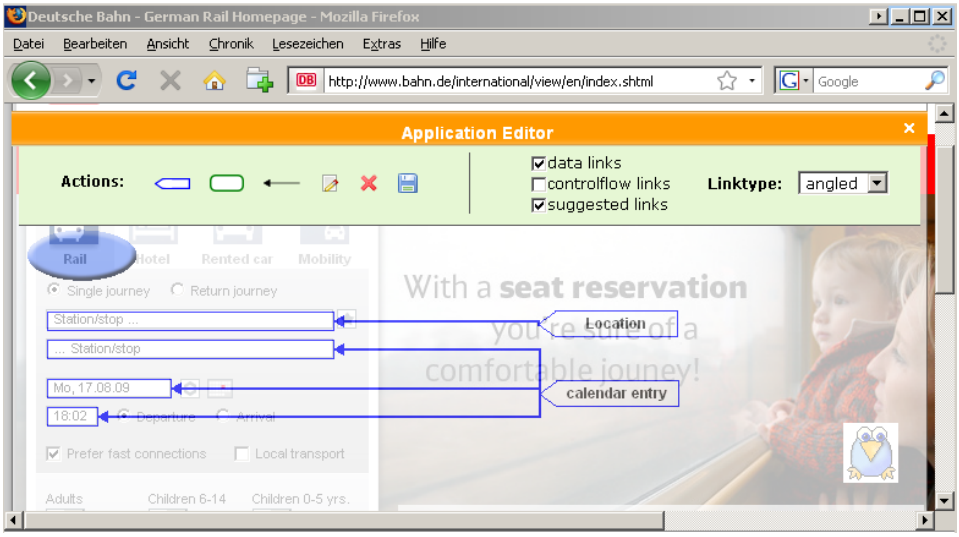


Figure 6.14. ATML Editor

6.3.5. Visualization in the AUGUR Prototype

For enabling even the end-user to easily modify the application models, and augment it with additional knowledge, we developed an application model editor that is integrated into the AUGUR prototype (see Figure 6.14). The editor is implemented as an overlay to the corresponding Web page. The editor highlights all interaction elements that are currently contained in the application model, and shows all associated *State* and *Context nodes*. It allows the user to add new *Context nodes* (by clicking on ▭) and to connect them to *State* and *Activity nodes* (⟵). New *UIContent nodes* can be inserted (▭) and connected to *Context nodes*. Further, the user can delete (✖) application model elements and edit their attributes (✎).

The user can select whether she wants to see data links and/or control flow links to keep the amount of visualized information as low as possible. Usually only data links are required as the other control flow links are mainly relevant for internal use. Further, the user can select whether new links which are suggested by the AUGUR prototype should also be shown. These suggested relations are displayed using a different color (green instead of blue for the fixed links). The hue of the line color reflects the confidence in the suggestion (ranging from white to dark green) as shown in Figure 6.15. The user can include these learned relations in the application model via the attribute editor for the relations (☆, see Figure 6.16).

Finally, the user can select how the links should be visualized: as straight lines (see Figure 6.15) or angled (see Figure 6.14). Straight lines can often be more easily perceived, however these straight lines require a lot of memory, as they are not supported by HTML and are for that reason made up of many small `div` elements.

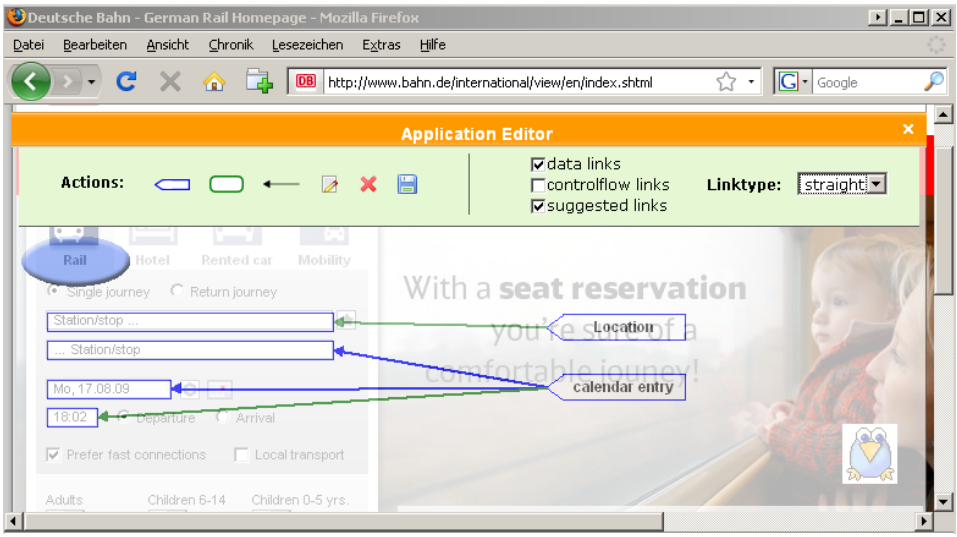


Figure 6.15. ATML Editor with two learned relations (the location link with 75% confidence and the link from calendar entry to time with 95% confidence)

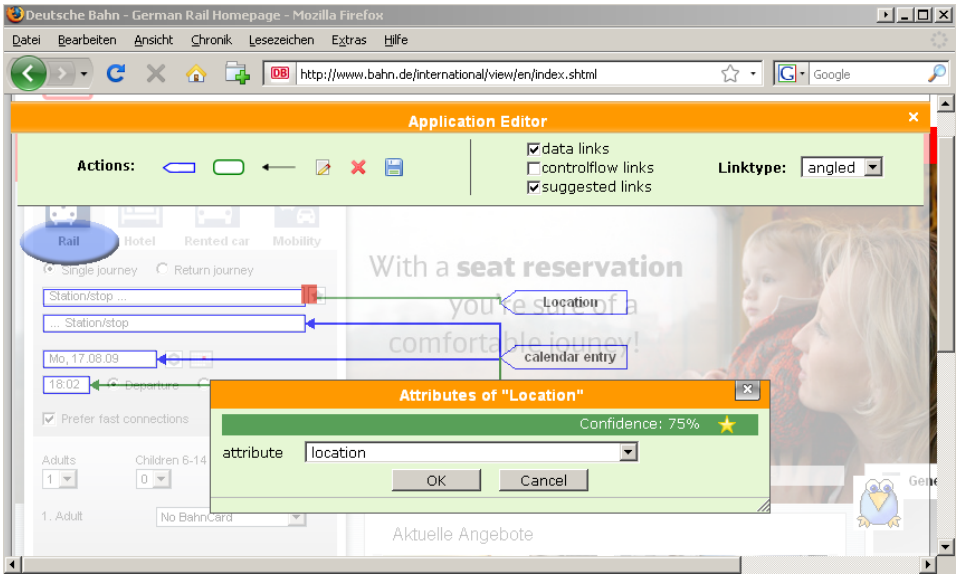


Figure 6.16. Attribute editor for the "Location" link. It shows the confidence in the link and allows the user to add the link to the application model by clicking on the star.

	$\mathbf{R1}_{AM}$	$\mathbf{R2}_{AM}$	$\mathbf{R3}_{AM}$
Model-based UI approaches (CTT, UIML, XIML)	-	-	-
UML (Activity Diagrams, State Charts)	●	-	-
XPDL	○	-	-
ATML	●	●	●

Table 6.2. Comparison of existing application modeling languages with ATML

6.3.6. Summary

In this section, we presented a novel application modeling language ATML which overcomes the shortcomings of existing modeling languages. ATML builds on the intuitive representation of state charts and activity diagrams and visualizes the application model as overlay to the existing application. This makes it more intuitive to understand for the end-user ($R1_{AM}$: +). Further, ATML explicitly visualizes context elements ($R2_{AM}$: +) and maps the application model elements to existing UI elements ($R3_{AM}$: +). Thus, it meets all the requirements which we identified in Section 6.3.1. Table 6.2 summarizes the comparison with state-of-the-art approaches.

When using ATML for Web applications, we have to face the problem that the label of the interaction elements can usually not be directly gathered from the corresponding HTML representation. For that reason, we also introduced a novel approach for extracting the label from the visual representation of a UI which outperforms state-of-the-art approaches.

Finally, we illustrated how the ATML in the AUGUR prototype visualizes the application models and enables the end-user to add further information.

6.4. Chapter Summary

In this section, we described the models applied in AUGUR, i.e. the context, user and application model. We contributed a new approach to model the relevance of context objects for the user in context-aware IUIs. Further, we introduced a novel modeling language for applications which overcomes the shortcomings of existing approaches and a new approach to extract labels from the visual representation of HTML UIs.

7 Content Support

In Section 4.3.1, we identified three ways of providing content support: based on (i) previous usage, (ii) the semantics of context and application elements, and (iii) modeled relations between these elements. In this chapter, we present the algorithms for providing content support in these three ways and how the confidence $c_{support}$ in their support is computed. In the following, we represent content support as a set of interaction elements a_i with the corresponding predicted values v_i, i.e. $C = \{(a_i, v_i), ..., (a_j, v_j)\}$. As stated in Section 4.3.1, AUGUR considers the content support provided by all three sources and drops all content support elements that conflict with the data the user has already entered in the UI. Further, AUGUR removes all content support elements that are covered by another element with higher confidence. The remaining content support elements with a confidence $c_{support} > t_{suggest}$ are finally provided as content support to the user.

In this chapter, we at first introduce how the confidence $c_{support}$ in the content support is computed in general (Section 7.1). The concrete computations for the three ways of providing content support are described in the respective sections: In Section 7.2, we describe how we realize content support based on previous usage. Section 7.3 presents a novel algorithm for using semantic information to map context information to interaction elements. In Section 7.4, we report how modeled relations are used for providing content support and how new relations can be learned.

7.1. Confidence in Content Support

We define the confidence in content support as follows:

$$c_{support} = c_{data} \cdot c_{relation} \tag{7.1}$$

with c_{data} being the confidence in the data, and $c_{relation}$ the confidence in how well this data sorts with the required input for the UI. The latter is especially important for learned relations. We define $c_{relation} = 1$ for all relations specified in the application model that are added by the end-user or the application developer. For all learned or computed relations, we discuss in the following sections how $c_{relation}$ is derived.

7.2. Content Support based on Previous Usage

In order to provide content support based on previous interactions, we need to know which data the user already entered in the UI of an application. For that purpose, we store this data for each state of an application. This log L_{usage} is a set of usages U_i. Each usage U_i is in turn represented as a set of activities a_i with corresponding values v_i which were entered by the user, i.e. $U_i = \{(a_1, v_1), ..., (a_n, v_n)\}$.

The confidence in the relation between the entered data and the UI depends on how often this relation has already been observed. For that purpose, we define the confidence $c_{relation}$ for a content support $C = \{(a_i, v_i), ..., (a_j, v_j)\}$ as its probability

$$c_{relation}(C) = \frac{|\{U_i \in L_{usage} : C \subseteq U_i\}|}{\max\{|L_{usage}|, min_{usage}\}}$$

Thereby, min_{usage} is the minimal amount of usage sequences which need to be stored for an application to provide relevant results. This accounts for the fact that content support inferred from few observed usage sequences, is rather unreliable and should thus be assigned a lower confidence value. We empirically determined $min_{usage} = 5$ to be an appropriate value.

Algorithm 2 illustrates how the content support for an activity a_i is computed based on previous usage. At first, we determine all possible content support elements $C = \{(a_i, v_i)\}$ with $c_{relation}(C) > t_{support}$. Then, we compute all content support elements with an additional activity-value pair (and also with $c_{relation}(C) > t_{support}$). We repeat this procedure until no new content support elements can be found. The union of all identified content support elements is the content support based on previous usage.

Algorithm 2 Content Support based on Previous Usage

Purpose: Calculate content support for activity a_i
Input: a_i: activity for which content support is computed
$L_{usage} = \{U_j\}$: Set of usage sequences
$t_{support}$: Threshold for displaying content support

$CS_1 = \{C = \{(a_i, x)\} : (\exists U_j \in L_{usage} : \{a_i = x\} \subseteq U_j) \wedge c_{relation}(C) > t_{support}\}$
$k = 1$
while $CS_k \neq \emptyset$ **do**
    $CS_{k+1} = \{C = C_k \cup \{(a_j, x)\} : C_k \in CS_k \wedge c_{relation}(C) > t_{support}\}$
    $k++$
end while
return $\bigcup_i CS_i$

Confidence in Content Support based on Previous Usage

The data for the content support is gathered from previous usage of the user. As this data was entered by the user, we assume that it is correct, i.e. $c_{data} = 1$. Thus,

$$c_{support}(C) = c_{relation}(C)$$

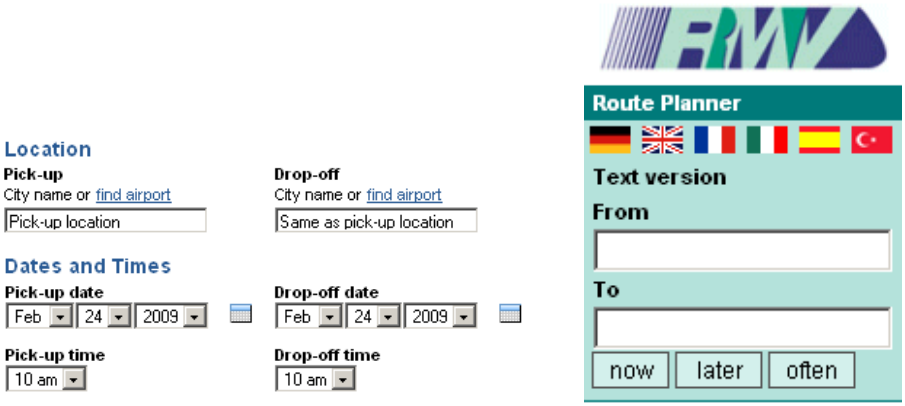
for content support based on previous usage.

7.3. Content Support based on Semantics

The terminology used in the UI often differs from the one used to describe the context. For example, the location information in a calendar entry has the label "location" whereas the location information for the UI in Figure 7.1 (b) is labeled with "To". For that reason, we need a mapping process that is able to bridge the existing vocabulary gap between the two representations. In this section, we contribute a novel approach for mapping context information to UI elements relying on the semantics of the data. Our approach combines the strengths of string-based and semantic similarity measures, which rely on general knowledge bases like Wikipedia or WordNet. Further, it is able to learn new relations from observing the user's interactions. At first, we identify requirements for the mapping process in context-aware IUIs (Section 7.3.1). In Section 7.3.2, we compare our approach to related work. As basis for the mapping process, we need a representation of the UI elements which is described in Section 7.3.3. Most existing approaches only consider the label of the UI elements. However, we show that the label does not provide sufficient information for that purpose and that incorporating further information about UI elements, like their tooltips, can dramatically increase the performance. In Section 7.3.4, we present our novel mapping process. Section 7.3.5 describes how the relevance of a context object for the UI can be determined based on the mapping, and defines how the confidence $c_{support}$ in the resulting content support is computed (Section 7.3.6). Finally, we report on the evaluation of our mapping process in Section 7.3.7.

7.3.1. Requirements

For the mapping process, we cannot rely on UI elements that are already tagged with a semantic concept (e.g. from a common ontology) nor on a large amount of training data to learn these semantic concepts, because we want to be able to provide



(a) http://www.usairways.com/awa/cars/ (b) http://www.rmv.de

Figure 7.1. Example forms

content support for arbitrary applications (R3 - *Application-independence*). Thus, the mapping process has to meet the following requirements:

- **R1**$_{sem}$: The mapping process shall not rely on pre-tagged data.
- **R2**$_{sem}$: The mapping process shall not rely on initial training data.
- **R3**$_{sem}$: The mapping process shall not rely on additional information about the UI despite the information that can be directly gathered from the UI itself.

Further, the result of the mapping should enable the context-aware IUI to decide which context object should be offered as content support to the user. For that reason, the mapping should state a confidence value for the mapping between context and UI.

- **R4**$_{sem}$: The mapping shall report a confidence value for the relevance of a context object for the UI to enable the context-aware IUI to distinguish between relevant and irrelevant context objects.

7.3.2. Related work

The task of mapping context information to input elements is strongly related to ontology mapping (Kalfoglou and Schorlemmer, 2005) and database schema matching (Rahm and Bernstein, 2001), where concepts have to be mapped to ontology entries or column names, respectively. However, in these areas most approaches benefit from additional information like constraints or instances that are more distinctive than the information from the UI (R3$_{sem}$: -) or they rely on a large amount of training data (R2$_{sem}$: -). Both is not available for context-aware IUIs. Also related is the research on the deep Web[15], as it is concerned with mapping textual representations of several Web forms. For example, Wu et al. (2004) use cosine similarity for determining the similarity of label and name attributes. However, their approach also relies on a large corpus of training data (R2$_{sem}$: -).

Other approaches for automatically filling in forms (e.g. Citrine (Stylos et al., 2004)) either require apriori tagging of Web sites (R1$_{sem}$: -), or a manually crafted list containing the labels or names of input elements which refer to a semantic concept (R3$_{sem}$: -). Thus, these approaches can only be applied to a specific domain (e.g. Stylos et al. (2004) focus on address information) or need explicit advice by the user. Chusho et al. (2002) apply a more generic machine learning approach to learn which content should be entered depending on which labels are located around the UI elements. However, they only consider the elements' labels for that purpose which often does not convey enough information as will be shown in the following.

[15] Deep Web refers to all information in the Web that cannot be accessed via conventional search engines following hyperlinks.

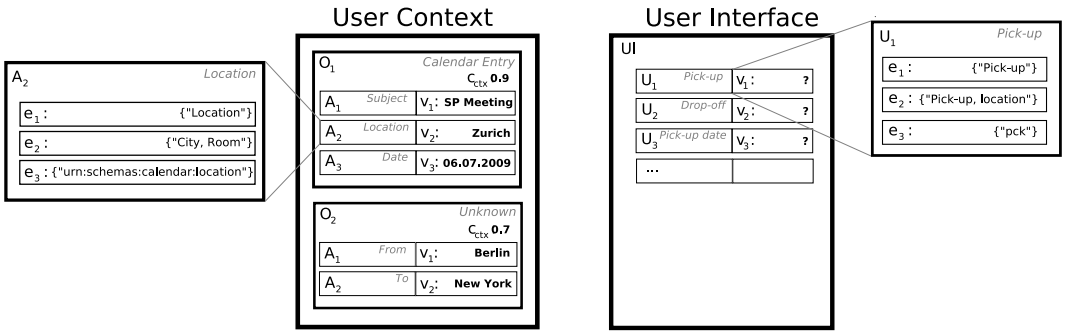


Figure 7.2. Example formalization of two context objects and a UI

7.3.3. Representing UI and Context Objects

For mapping context information to UI elements, we first need a representation for UI elements and context information that is stored in the *Current Context*. The representation for context objects was already described in Section 6.1. The UI representation available from our application models (see Section 6.3) only stores the label of the elements. However, the label of the elements is often not sufficient for unambiguously describing the element as it sometimes refers to several interaction elements (e.g. "Pick-up date" in Figure 7.1 (a)) or does not convey much information (e.g. "To" in Figure 7.1 (b)). For that reason, we take all available texts that describe the element into account, in contrast to Chusho et al. (2002).

In the following, we point out which texts can be used to represent the various elements and how the representation is formalized. A visualization of the formalization is shown in Figure 7.2.

Representing UI elements

For most UI representation languages, each UI element provides the following information that can be used to represent the element:

- its **label** which is human readable (e.g. "Pick-up date" in Figure 7.1 (a))
- the **name** attribute of the input element that gives us its technical label, though this is often not human readable (e.g. "pck" for the "Pick-up" element)[16],
- the corresponding **tooltip** ("alt" attribute),
- the data that is **prefilled** to give the user a hint of what to enter (e.g. "Pick-up location" is filled in the "Pick-up" element in Figure 7.1 (a)), and
- the **values** in drop down menus, radio buttons, or grouped checkboxes.

[16] This attribute is often a camel case word consisting of several concatenated words like "arrivalDate" because the attribute may not contain white spaces.

For Web applications, *name*, *tooltip*, *prefilled,* and *values* can be extracted directly from the HTML representation. The HTML syntax also defines a tag LABEL for marking a *label* of an input element; however, it is scarcely used in practice (only about 20% of the input elements we used for the evaluation had an associated label attribute). For that reason, we developed our own approach called LabelFinder for extracting the labels by analysing the visual layout (see Section 6.3.4).

Each UI element E_i can thus be represented by a list of descriptive texts e_r (see also Figure 7.2). These texts are ordered by their relevance r. This is important as the best (initial) mapping can be determined if we consider only the most relevant information. Further information should only be incorporated if this text does not convey enough information. In initial experiments, we determined the following order of relevance according to the expressiveness of the different attributes: *label*, *values*, *tooltip*, *prefilled*, *name*. The individual texts e_r are separated into a set of single words, so-called **tokens**. For example, the first element in the example form in Figure 7.1 (a) is represented by its *label*, *prefilled* and *name* attributes E_1=(e_1, e_2, e_3)=$(\{Pick\text{-}up\}, \{Pick\text{-}up, location\}, \{pck\})$.

The entire UI is finally represented as a set of elements E_i with corresponding values v_i, i.e. $\{(E_1,v_1), (E_2, v_2), ...\}$. The values v_i represent the data which needs to be entered in the UI and thus the information we want to predict with our mapping process.

Representing context objects

The *Current Context* contains all **context objects** O_j that might be relevant for the user's current interaction. As described in Section 6.1.1, this data is gathered from (i) the user input, (ii) the UI, and (iii) from the *Context Server*. Each O_j consists of a set of **context attributes** ca_i with associated values v_i as introduced in Section 6.1. For example, the calendar entry in Figure 7.2 consists of a subject, a location, and a date element. The goal of the semantic content support is to determine which context object O_j from the *Current Context* can be used as input for the current UI and which of its values v_i should be suggested as input for which UI element E_i.

For the semantic mapping, we represent each attribute ca_i as a set of descriptive texts A_i in analogy to UI elements. If the context object O_j was derived from the user's interaction, it is represented by the same attributes as the corresponding UI element. If it was gathered from the *Context Server*, we have to rely on additional information provided by the *Context Server* (see Section 6.1). This information is at least a label like "start date" which we therefore take as most descriptive text e_1. Sometimes, additional information like a brief description is provided by the *Context Server* which is used as additional descriptive text.

7.3.4. Mapping textual representations

Having determined the representation of the available input elements of a UI, we try to find relevant objects in the *Current Context* that can serve as input for the UI

elements, i.e. which v_i should be suggested for which input element. As stated by $R3_{sem}$, we can only make use of the available textual descriptions as described in the previous section to determine the mapping.

As basis for the mapping process, we first need a similarity measure that computes the similarity between two tokens, e.g. between "Pick-up" and "Location" (see *Computing similarity of tokens*). This similarity measure is then applied to determine the similarity of two elements described by several descriptive texts (see *Computing similarity of elements*), e.g. the elements A_2 and E_1 in Figure 7.2. The resulting similarity value between elements is then used to determine the best mapping between context and UI elements, e.g. which elements of O_1 should be assigned to which elements in the UI (see *Mapping context and UI elements*). Which of the context object present in the *Current Context* (e.g. O_1 and O_2) should be finally suggested to the user is described in Section 7.3.5.

Computing similarity of tokens

As basis for our mapping process, we need a similarity measure $sim_{token}(a, b)$ which computes the similarity between two tokens a and b on a scale from 0 to 1. For sim_{token}, we can use similarity measures that are based on string comparison (**string-based measures**) or that rely on an additional knowledge base, i.e. a domain model, like Wikipedia[17], Wiktionary[18], or WordNet (Fellbaum, 1998) (**semantic measures**). In contrast to string-based measures, the semantic measures recognize similarities between strings from different terminologies (e.g. "destination" and "airport"). However, they are only able to provide similarity measures if the terms are reflected in the underlying knowledge base. This means that they often do not recognize the similarity of strings that differ in their spelling variants (e.g. "e-mail" and "email") and cannot be applied to arbitrary new domains. In contrast, string-based measures are able to cope with spelling variants and can be applied to new domains without requiring any modeling. However, they are not able to bridge vocabulary gaps when different terminologies are used.

In a preprocessing step, we turn all tokens in lower case representation, split camel case words, and skip all tokens that only consist of one character. Further, we lemmatize all tokens, i.e. use their canonical form (e.g. "mouse" instead of "mice" and "run" instead of "ran"). This is necessary as the knowledge bases that are used for the semantic measures only contain lemmas.

The distribution of the actual sim_{token} values over the [0,1] interval vary for the different measures, e.g. for some measures 0.6 is a relatively low value. To ensure that only really similar tokens are considered in the mapping process, we state a threshold θ for sim_{token} for every similarity measure which we apply, i.e. if $sim_{token} < \theta$, then $sim_{token} = 0$.

[17] http://www.wikipedia.org
[18] http://www.wiktionary.org

Computing similarity of elements

We compute the similarity $sim^k(A_i, E_j)$ between two elements A_i and E_j as the average similarity for every possible pair of tokens. Thereby, we take the k most relevant descriptive texts into account. We limit the number of texts we consider, because we first try to determine a mapping with the most relevant descriptive texts (e.g. only considering the labels with $k = 1$), as we assume that it returns the best results. Further descriptive texts are only considered for those elements that could not be assigned in the first step (see *Mapping context and UI elements*). We define

$$sim^k(A_i, E_j) = \frac{\sum_{a \in X^k_{A_i}, b \in X^k_{E_j}} sim_{token}(a, b)}{max(|X^k_{A_i}|, |X^k_{E_j}|)}$$

with $X^k_{A_i} = \bigcup_{r \leq k} e_r \in A_i$ and $X^k_{E_j} = \bigcup_{r \leq k} e_r \in E_j$. For example, for computing the similarity $sim^2(A_2, E_1)$ between A_2 of the context object O_1 and E_1 of the example in Figure 7.2, we determine at first $X^2_{A_2} = \{Location, City, Room\}$ and $X^2_{E_1} = \{Pick\text{-}up, Pick\text{-}up, location, pck\}$. If we use a string-based similarity measure for sim_{token} that returns 1 if the tokens are the same and otherwise 0, only $sim_{token}(location, location)$ results in 1, all other similarity values in 0. Thus, $sim^2(A_2, E_1) = 1/max(3, 4) = 1/4$.

Mapping context and UI elements

For determining the mapping M_m between a context object O_m and the UI, we first identify the best assignments of context to UI elements using only the most relevant descriptive text e_1 (i.e. its label). We compute $sim^1(A_i, E_j)$ for every element $A_i \in O_m$ and every element $E_j \in UI$. This results in a mapping like the one in Figure 7.3 (a). M_m initially contains all pairs (A_i, E_j) with a similarity value greater than 0. As we assume that each concept is only used once in a context object or in a UI, the mapping has to be unambiguous. This means that for example the location is only used for a single UI element. For that reason, we need to solve all generated ties, i.e. to resolve all conflicts in the mapping. The **Solve Ties** process iterates over all pairs $(A_i, E_j) \in M_m$ ordered by their similarity value. Each pair that is in conflict with the currently considered pair, is deleted from M_m. If two pairs have the same similarity measure, both are deleted from the mapping. Figure 7.3 (b) shows the result of Solve Ties for our example mapping.

All elements that are now unambiguously assigned to another element are excluded from the following mapping iteration. We again compute the similarities between the various remaining elements, but this time we take one more descriptive text into account, i.e. we use $sim^2(A_i, E_j)$. Emerging ties are solved and the unambiguously assigned elements also excluded from the next mapping iteration. The process is repeated with further descriptive texts until all descriptive texts are considered, or

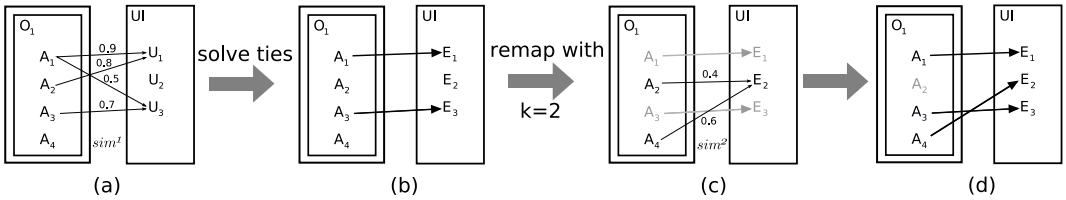


Figure 7.3. Applying the mapping process to map a context object to a UI

all context or UI elements are assigned. The pseudo-code for the whole process can be found in Algorithm 3. To further improve the performance of the mapping process regarding the mapping quality, we suggest the following two extensions to the process.

Second mapping step As stated before, string-based measures have the advantage that they can be applied to arbitrary domains that use the same terminology, whereas semantic measures can bridge the vocabulary gap between domains, but only if the used knowledge base contains the required terms. To benefit from both types of similarity measures, we extend our process with a second mapping step. We at first determine a mapping with one type of measure and then try to map the unassigned elements using the other type of measure. We expect that the string-based mappings provide a better initial mapping than semantic measures, as they usually achieve higher precision values than semantic measures. Thus, they should be applied for the first mapping step and semantic measures for the second.

Learning related terms The existing vocabulary gap between a context and a UI representation cannot always be bridged with the help of semantic similarity measures, as the knowledge base may not contain all relevant terms. For example, the term "pick-up", which is often used for car rental forms, is not contained in any of the knowledge bases that we considered. For that reason, we further apply learning in our mapping process. In those cases where the mapping could not be determined automatically, we learn which terms are related to each other from observing the user's input. If we observe that the user has entered data in a UI element E_i that corresponds to a value v_i contained in the *User Context*, we store the label (e_1) of the corresponding A_i as related expression to the label (e_1) of E_i. For example, if the user entered "Zurich" for v_1 in the UI in Figure 7.2, we store that "Pick-up" (the label of E_1) is related to "Location" (the label of A_2). Note that these expressions can also consist of several tokens (e.g. "Pick-up time"). $sim_{token}(a, b)$ then returns 1 if the token a is contained in an expression that is related to an expression that contains the token b.

Algorithm 3 Mapping Context elements to UI elements

Purpose: Determine a mapping M_m between the elements of O_m and the UI
Input: Set $\{A_i\}$ of attributes of a context object O_m
Set $\{E_j\}$ of UI elements

1: $CO = \{A_i\}$; $UI = \{E_j\}$; $k = 1$; $M_m = \{\}$;
2: k_{max} = max number of descriptive texts in CO and UI
3: **while** $CO \neq \emptyset \ \wedge UI \neq \emptyset \ \wedge k \leq k_{max}$ **do**
4: $M = CO \times UI$
5: **for all** $(A_i, E_j) \in M$ **do**
6: Compute $sim^k(A_i, E_j)$
7: **end for**
8: Order all elements in M by their similarity sim^k
9: SOLVE TIES for M
10: **for all** $(A_i, E_j) \in M$ **do**
11: Remove A_i from CO
12: Remove E_j from UI
13: **end for**
14: $M_m = M_m \cup M$
15: $k = k + 1$
16: **end while**

7.3.5. Measuring the Relevance of a Context Object

Besides computing how a context object could be mapped to the UI, we also need to determine the relevance of a context object for the UI ($R3_{sem}$). The relevance estimation enables a context-aware IUI to decide whether or not a context object is relevant for the current UI and thus which context object should be offered as content support to the user.

We define the relevance $r(O_i)$ of a context object O_i for a UI as

$$r(O_i) = |M_i|/|O_i|$$

i.e. the amount of elements A_j of O_i which could be assigned to UI elements with the mapping M_i. For example in Figure 7.3 (d), $r(O_1)$ is 3/4. The more elements could be assigned, the more likely it is that O_i is relevant for the current UI.

7.3.6. Confidence in Content Support based on Semantics

The confidence in the content support C which results from mapping a context object O_i to the UI, is determined as stated in Equation 7.1 as $c_{support} = c_{data} \cdot c_{relation}$. Thereby, c_{data} is the relevance r_{user_ctx} of the context object for the user (see Section 6.1.1) and $c_{relation}(O_i) = r(O_i)$, i.e. the relevance of the context object for the UI as defined in Section 7.3.5. Thus,

$$c_{support}(C) = r_{user\_ctx}(O_i) \cdot r(O_i)$$

7.3.7. Evaluation

In this section, we evaluate the performance of the proposed mapping process. We examine (i) which similarity measure is best suited for the mapping process, (ii) whether a combination of string-based and semantic measures increases the performance, (iii) whether learning of related terms increases the performance, (iv) whether the proposed mapping process outperforms approaches which only base on the label of the UI elements, and (v) whether the proposed mapping process enables a context-aware IUI to distinguish between relevant and irrelevant context objects ($R4_{sem}$).

We evaluated the mapping process for a variety of Web applications. However, a dataset containing possible context objects for a number of Web applications and their mappings is hard to obtain. Furthermore, we want to be independent of how the context information is actually represented. For that reason, we decided to use the Web forms themselves as possible context representations. This means that we take the representation given by a source Web form as a potential context object and try to map it to a target Web form. The Web forms were taken from several domains, whereby some of them are **related domains**, i.e. the same information can be entered but they often use different terminology. For example, Web forms for booking a hotel room and for booking a car are related, as they both need a start and end date but use different terminology like "check in" and "pick up". For the evaluation within a **single domain**, we used every combination of source and target Web form from this domain. For related domains, we combined one form from the one domain with a form from the related domain.

Evaluation dataset We took 45 Web forms from 4 domains: car rental (consisting of 7 Web forms), flights (12), hotels (9), and address (17). Most Web forms for the cars, flights, and hotels domains were taken from the TEL-8 dataset of the UIUC dataset (UIUC, 2003). Thus, we have four single domains (*cars*, *flights*, *hotels*, and *address*) and three related domains (*cars&flights*, *cars&hotels*, *hotels&flights*).

Evaluation Metric In order to judge the quality of the resulting mapping, we have to consider that often no direct mapping between elements is possible. For example, one representation uses two fields for entering the name, one for the first and one for the last name, while another representation uses only one field for the full name. Assigning the element representing the first name to the one representing the full name is not entirely correct, but it enables us to suggest at least half of the input for the UI element. For that reason, we include these partially correct assignments in our rating but with a lower weight than correct assignments. For each mapping, we compute the precision and the recall as follows:

$$Precision = \frac{correct + 0.5 \cdot partial}{correct + 0.5 \cdot partial + wrong}$$

$$Recall = \frac{correct + 0.5 \cdot partial}{maxScore}$$

In this formalization, *correct* denotes the number of correct assignments, *partial* the number of partially correct assignments, and *wrong* the number of wrong assignments. *maxScore* is the score that is reached by an optimal mapping.

In the following, we first list the similarity measures that we assessed in our comparison and describe the evaluation data set. Then, we report how the different measures perform in a single domain and in related domains (*Comparing similarity measures*). Next, we show that our proposed mapping process –including a second mapping step using another similarity measure and including learning– increases the overall quality of the mapping (*Second mapping step* and *Learning related terms*). Finally, we determine how well our approach can distinguish a relevant from an irrelevant context object and is thus able to provide only relevant content support (*Distinguishing context objects*).

Similarity measures *String-based measures* determine the similarity between two strings by comparing their characters. We use two baseline string measures: The exact string match measure (abbreviated as **exact**) returns 1 if the strings are exactly equal, and 0 otherwise. The bounded substring match measure (**b-substr**) returns 1 if the strings have a shared substring of at least 3 characters that is a prefix or a suffix of the other string (this matches strings like "arrival" and "arrive"). We also consider three more sophisticated measures that return a value in the interval $[0, 1]$: (i) the measure by Jaro (1995) (abbreviated as **jaro**) that takes typical spelling deviations into account, (ii) an adaptation of the *jaro* measure by Winkler and Thibaudeau (1991) (**jaro-w**) which increases similarity scores in the case of shared prefixes, and (iii) the measure by Monge and Elkan (1996) (**monge-elkan**) that uses an affine gap model penalizing many small gaps in the string match more than a large gap. For our experiments, we implemented the *exact* and *b-substr* measures ourselves, and used the SecondString library (Cohen et al., 2003) for the *jaro*, *jaro-w*, and *monge-elkan* measures.

Semantic measures use knowledge bases for determining similarities, such as WordNet (Fellbaum, 1998), Wikipedia[19], or Wiktionary[20]. A typical source of alternative wordings for the same concept is the use of synonyms (e.g. "city" and "town") or other terms closely related by a lexical semantic relation such as hypernymy/hyponymy or holonymy/meronymy (e.g. "city" and "New York"). Thus, we created a semantic similarity measure that returns 1, if the target string is a direct synonym, hypernym, hyponym, holonym, or meronym of the source string, and 0 otherwise. The measure is very similar to the multitude of semantic similarity measures defined on WordNet (see (Budanitsky and Hirst, 2006) for an overview). However, these measures rely on special properties of WordNet, while the proposed measure can also be used with other knowledge bases like Wiktionary. These measures are abbreviated in the following with **wiktionary-rel** and **wordnet-rel**. We do not use the relation measure

[19] http://www.wikipedia.org
[20] http://www.wiktionary.org

with Wikipedia, as it does not contain explicitly labeled lexical semantic relations. We also use a concept vector based measure (Qiu and Frei, 1993) for each knowledge base, where the meaning of a string w is represented as a high dimensional concept vector $\overrightarrow{d}(w) = (d_1, ..., d_N)$. Each vector element d_i represents a document in the knowledge base, and the value of d_i is the string's tf.idf score (Salton and McGill, 1983) in the document. Semantic relatedness of two strings can then be computed as the cosine of their corresponding concept vectors. We apply this to all three knowledge bases, abbreviated in the following with **wordnet**, **wikipedia**, and **wiktionary**.

For our experiments, we used (i) WordNet 3.0 together with the freely available JWNL WordNet API[21], (ii) the English Wikipedia dump from Feb 6th, 2007 together with the JWPL Wikipedia API[8] (Zesch et al., 2008), and (iii) the English Wiktionary dump from Oct 16th, 2007 with the JWKTL Wiktionary API[22] (Zesch et al., 2008).

For normalizing inflected forms of tokens, we used lemmatization as provided by the TreeTagger (Schmid, 1995). We empirically determined the optimal value of the threshold θ for including similarity values in the mapping process on a dataset that is not used in the experiments. We used the following thresholds: 0.75 for the *jaro*, *jaro-w*, and *monge-elkan* measures and 0.05 for *wordnet*, 0.4 for *wikipedia*, and 0.25 for *wiktionary*. All other measures return either 0 or 1, thus no threshold is needed.

Comparing similarity measures

At first, we compared the baseline performance of the different similarity measures without using learning or a second mapping step. Table 7.1 shows the results for the various similarity measures in terms of recall, precision, and F_1-measure for the four single domains. The results differ slightly on a very high level of performance with an F_1-measure between .76 and .87. The string-based measures slightly outperform the semantic measures. The semantic similarity measures based on concept vector outperform the similarity measure based on relations for all knowledge bases and domains. This confirms with recent research on semantic similarity measures (Gabrilovich and Markovitch, 2007; Gurevych et al., 2007) which indicates that concept vector based measures are superior to relation-based measures. For that reason, we focus only on the concept vector based measures in the following.

Table 7.2 summarizes the results across related domains. The performance for the related domains varies heavily, however there are also mostly only minor differences between the various similarity measures. In that setting, the semantic measures perform slightly better than the string-based measures.

As the *b-substr* measure has the best overall performance of all string-based measures, we consider this measure as representative for all string-based measure in the remainder. As the semantic measures rely on different knowledge bases, we expect

[21] http://jwordnet.sourceforge.net/
[22] http://www.ukp.tu-darmstadt.de/software/JWPL

that they vary in their ability to bridge vocabulary gaps. For that reason, we still take all three semantic measures based on concept vectors into account.

Second mapping step

In order to improve the quality of the mapping especially across related domains, we evaluated whether a second mapping step with another similarity measure increases the performance. We compared the results for using a string-based or a semantic measure at first. As expected, we found that it yields better performance to use

	Domain	cars			flights			hotels			address		
		P	R	F	P	R	F	P	R	F	P	R	F
string-based	exact	.93	.82	.87	.95	.72	.82	.88	.87	.87	.89	.71	.79
	b-substr	.94	.83	**.88**	.95	.76	**.84**	.87	.87	.87	.91	.79	**.85**
	jaro	.89	.80	.84	.90	.74	.81	.88	.88	**.88**	.82	.76	.79
	jaro-w	.89	.81	.85	.90	.73	.81	.87	.88	**.88**	.77	.75	.76
	monge-elkan	.94	.83	**.88**	.90	.75	.82	.86	.87	.86	.87	.81	.84
semantic	wordnet	.93	.82	.87	.93	.72	.81	.85	.86	.85	.84	.75	.79
	wiktionary	.93	.82	.87	.95	.72	.82	.88	.87	.87	.89	.75	.81
	wikipedia	.93	.82	.87	.86	.69	.76	.88	.87	.87	.82	.78	.80
	wiktionary-rel	.93	.82	.87	.95	.71	.82	.88	.88	.87	.86	.72	.78
	wordnet-rel	.93	.80	.86	.84	.47	.60	.86	.88	.87	.60	.48	.54

Table 7.1. Micro-average precision (P), recall (R) and F-measure for single domains (best values in bold and blue)

	Domain	cars & hotels			cars & flights			hotels & flights		
		P	R	F	P	R	F	P	R	F
string-based	exact	.84	.40	.54	.52	.10	.17	.62	.25	**.35**
	b-substr	.79	.39	.52	.50	.12	.19	.54	.24	.34
	jaro	.74	.40	.52	.48	.12	.19	.36	.24	.29
	jaro-w	.68	.39	.50	.41	.11	.18	.35	.24	.29
	monge-elkan	.73	.39	.51	.35	.12	.18	.44	.24	.31
semantic	wordnet	.81	.44	**.57**	.46	.10	.17	.60	.25	**.35**
	wiktionary	.84	.40	.54	.52	.10	.17	.62	.25	**.35**
	wikipedia	.78	.42	.54	.48	.15	**.23**	.47	.25	.33

Table 7.2. Micro-average precision, recall and F-measure across related domains (best values in bold and blue)

a string-based measure first. For that reason, we only report here on the results for using *b-substr* in the first mapping step and a semantic measure in the second mapping step. To evaluate whether a second mapping step itself can increase the performance, we also computed the performance when using *b-substr* for both steps. Figure 7.4 shows the absolute increase or decrease in the F-measure for the second mapping step compared to using only a single mapping step. A second mapping step dramatically increases the performance by about 13% points for the *cars&hotels* domain, no matter which similarity measure is used. This indicates that the second mapping step itself can be of advantage, because sometimes some conflicting elements are assigned in the first mapping step so that remaining elements can then be assigned in the second step. Further, *wikipedia* increases the performance for the *cars&flights* domain by 8% points. For the other domains the second mapping step has only marginal influence on the results. This shows that a second mapping step has rarely negative effects on the quality of the mapping, but can increase the performance especially across related domains. As the combination of *b-substr* for the first and *wikipedia* for the second mapping step yields the best results, we only report the results for this combination in the following if not stated otherwise.

Learning related terms

Next, we determined the influence of learning related terms from observation on the mapping quality. For the evaluation, we randomly selected 20 pairs of the *hotels&flights* domain and successively computed the mapping while tracking the F-measure for each mapping. This process was repeated 80 times and the results were averaged. Figure 7.5 shows how the average F-measure increases with the number of observed mappings. It shows that 4 observed mappings are already sufficient to increase the F-measure relatively by 40% and 10 to reach an overall performance of about 60%.

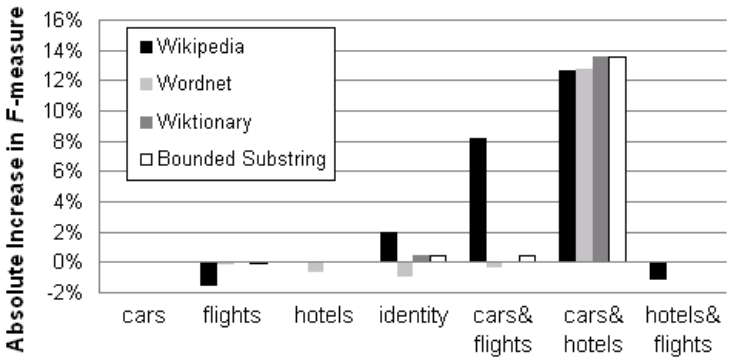


Figure 7.4. Absolute difference between the F-measure of a single mapping step using *b-substr* and a two-step mapping process additionally using one of the listed similarity measures for the second mapping.

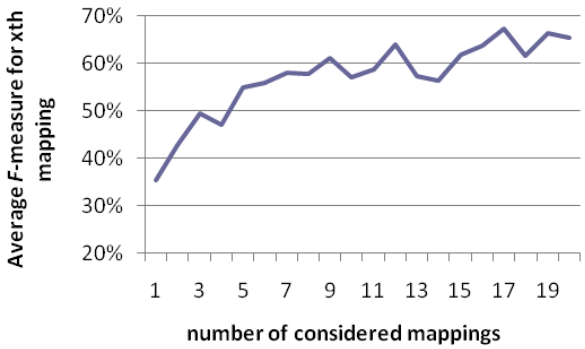
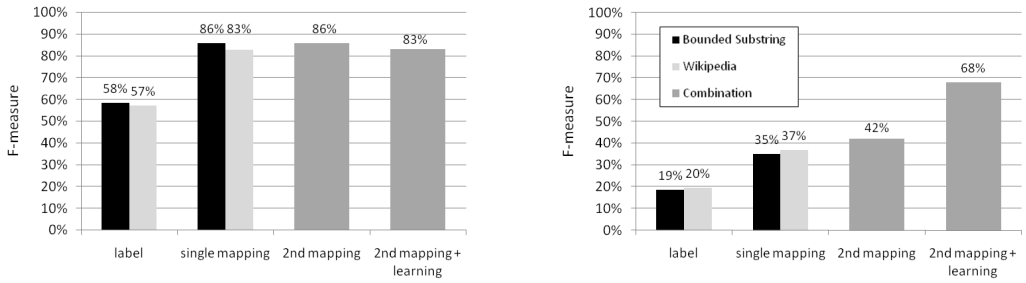


Figure 7.5. Average F-measure for the *hotels&flights* domain given the number of observed mappings

Overall performance

To be able to estimate the influence of the different factors and to judge the overall performance of the presented mapping process compared to approaches which only consider the labels of the UI elements, we compare the F-measures for different configurations for a single domain and across related domains. The result can be seen in Figure 7.6. As a baseline, we use the standard approach that only considers the label of the input elements (*label*). Using our approach with a single mapping step (*single mapping*) already outperforms *label* by 44% for a single domain and by 89% across related domains. Using a two-step mapping process with the similarity measures *b-substr* and *wikipedia* (*2nd mapping*) further increases the results across related domains by more than 13% without having an influence on the results in a single domain. Finally, if related terms are additionally learned (*2nd mapping+learning*), the average F-measure for single domains slightly decreases by 2.5%, however the average F-measure across related domains dramatically increases by 60%.



(a) In a single domain

(b) Across related domains

Figure 7.6. Overall performance of our mapping process

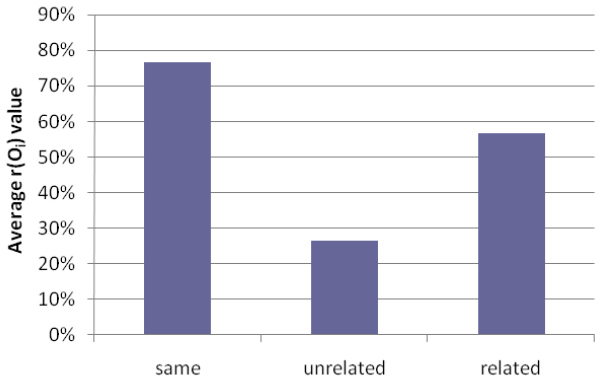


Figure 7.7. Average $r(O_i)$ for mapping a context object to a UI from the same, from an unrelated or from a related domain.

Distinguishing context objects

Besides determining a good mapping for relevant context objects, it is also important that the relevance values $r(O_i)$ for irrelevant context objects are rather low, so that a context-aware UI is able to distinguish relevant from irrelevant context objects ($R4_{sem}$). This means that in the ideal case the mapping process does not assign any context element of an irrelevant context object to a UI element. To evaluate this behavior for our proposed mapping process, we compare the average $r(O_i)$ value for a context object and a UI from the *same* domain, from *related* domains, and from *unrelated* domains (i.e. *hotels&identity* etc.). As Figure 7.7 shows, the average $r(O_i)$ value for the *same* domain (76%) is three times as high as for *unrelated* domains (27%). The average value for *unrelated* domains is comparably high, as there is quite few information in the two domains that requires similar information, e.g. a city name needs to be entered for stating the desired location of a hotel and the address information. This accounts for about 10-20% of the erroneous assignments. The results for the *related* domain (57%) is in between the two extremes. This shows that it is unlikely that a context-aware UI using the proposed mapping process will suggest an irrelevant context object to the user.

7.3.8. Summary

In this section, we presented a novel approach for mapping context objects to UI elements. It does not require any pre-tagged data ($R1_{sem}$) or large amount of training data ($R2_{sem}$) and can determine the mapping based only on the information available in the UI and from the context objects ($R3_{sem}$). Our approach outperforms existing approaches that only rely on the labels of the UI elements. We showed that the performance of the algorithm is increased if it learns related terms from observing the user's interaction. Further, it enables us to distinguish well between relevant and

irrelevant context objects ($R4_{sem}$).

We showed that the quality of mapping context elements to UI elements can be increased by using semantic measures. Due to practical reasons it can however be of advantage to use only string-based measures, as they require far less resources than semantic measures: Semantic similarity measures need an additional knowledge base which requires about 20MB up to several GB of disk space (WordNet: 23MB, Wiktionary: 42MB, Wikipedia: 6.6GB) and are much slower than string-based measures. When applying the presented process for example on mobile devices, it is for that reason of advantage to rely only on string-based measures or to use semantic measures only for mapping the labels in a first step.

7.4. Content Support based on Modeled Relations

In this section, we present how relations between context and interaction elements are modeled (Section 7.4.1), learned (Section 7.4.2 and Section 7.4.3), and how they are used for providing content support (Section 7.4.4). As stated before, AUGUR supports two types of relations: (i) *direct relations* (i.e. context information X can be used as input for interaction element Y) and (ii) *rules* (e.g. if the location is at home, then enter "Heidenreichstr." in the "to" input element). For learning new rules to describe the relation between context and application elements, we build on the Apriori algorithm for mining association rules and adapt it to our needs. To our knowledge, the presented approach for providing content support is the first approach which is able to use arbitrary context information (R1 - *Awareness of user context and environmental context*) (see also Section 2.3).

7.4.1. Relations in AUGUR

As stated in Section 4.2.3, AUGUR supports two types of relations: direct relations and rules.

Direct Relation A direct relation in AUGUR is defined as

```
<interactionElement>=(<contextAttribute>|<value>)
```

Direct relations specify which context attributes or fixed values can be used as input for an interaction element. For example, the ATML model for the train booking application shown in Figure 7.8 contains two direct relations: The "date" and the "time" attribute of a calendar entry can be directly used as input for the date and time input elements.

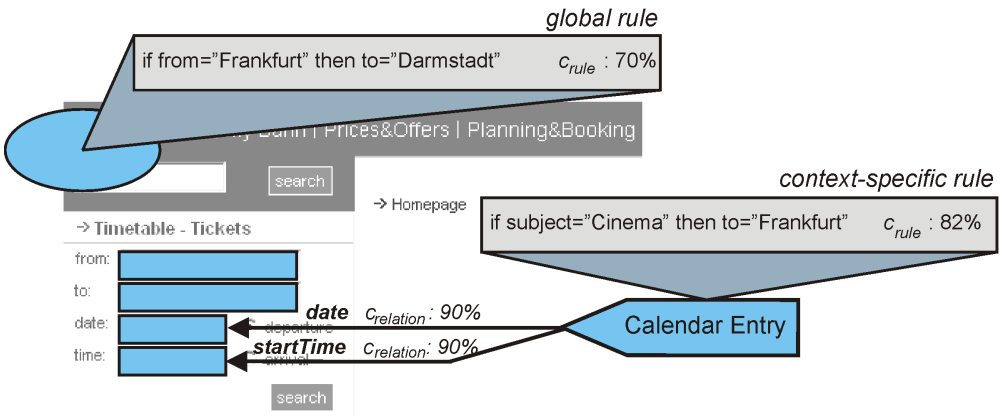


Figure 7.8. Example ATML model for the train booking example with direct relations and rules

Rules Rules are defined in AUGUR as

```
if (<contextAttribute>=<value>|<interactionElement>=<value>)+
then (<interactionElement>=<value>)+
```

Rules allow to state more complex dependencies between elements than direct relations, e.g. if the current location is "at home', suggest "Heidenreichstr." for the "point of departure" field for looking up the bus connections. The rules can state conditions on context attributes as well as on other interaction elements.

Rules that consider context attributes are called **context-specific rules** and those that only rely on interaction elements are called **global rules** for a UI. The example in Figure 7.8 contains two rules: a context-specific rule which depends on data of a calendar entry (`if subject="Cinema" then to="Frankfurt"`) and a global rule which only considers the content of interaction elements (`if from="Frankfurt" then to="Darmstadt"`).

7.4.2. Learning Direct Relations

In order to automatically enhance the provided support, AUGUR learns relations between context and interaction elements from observation. Thereby, AUGUR is able to cope with little training data. Relations are learned by comparing data the user has entered with the data which was at that time available in the *Current Context*. As soon as the user submits data to an application, i.e. as soon as she clicks on a navigational element, the learning process is initiated. The pseudo-code of the learning algorithm is presented in Algorithm 4. If a matching between an attribute in the *Current Context* and entered data is found, AUGUR checks whether the corresponding direct relation is already contained in the application model. If this is not the case, AUGUR adds a new direct relation to the application model. Further, AUGUR sets or updates the confidence $c_{relation}$ in the relation. $c_{relation}$ is thereby defined as

$$c_{relation} = \frac{x}{\max\{n, min_{relation}\}}$$

where n is the number of user interactions with the specific interaction element, x captures how often the user entered context information of the same type during these interactions and $min_{relation}$ is a predefined minimal value which is required to trust in the viability of the relation. The minimal support $min_{relation}$ is required to avoid that the confidence in a relation that has only been observed once, is 100%. We empirically found that 5 is a good estimation for $min_{relation}$.

For example, the user enters five times a value in the "from" interaction element. In three out of these five times, she entered the data that was also stored as the "location" attribute of the "Location" information in the *Current Context*. Thus, a new relation from context type "Location" to the "from" interaction element is learned with $c_{relation} = 60\%$.

For learning relations by comparing input with the *Current Context* the corresponding context information has to be present in the *Current Context*. For context

Algorithm 4 Updating and learning direct relations

Purpose: Updating and learning new relations between activities and context nodes
Input: Set $\{(a_i, v_i)\}$ of input entered in the current UI (with a_i being the different activities and v_i the corresponding values)
Set $\{O_i\}$ of elements in the *Current Context* (with $O_i.ca_j$ denoting the values of the different attributes ca_j of O_i)
Set $R = \{(a_i, cn_j, ca_k)\}$ of direct relations between activities a_i and attributes ca_k of context nodes cn_j

1: **for all** $v_i = O_k.ca_j$ **do**
2: **if** $\exists cn_l.(a_i, cn_l, ca_j) \in R \wedge cn_l$ of same type as O_k **then**
3: update the confidence $c_{relation}$ in the direct relation (a_i, cn_l, ca_j)
4: **else**
5: introduce new context node cn of the same type as O_k
6: add (a_i, cn, ca_j) to R
7: tag it with confidence $c_{relation}$
8: **end if**
9: **end for**

sources that can only be accessed via queries, like the user's address book, we have to compare the entered data with all context objects available of this type. However, this does not scale for large amount of data at runtime, thus it is stored in a log $L_{relations}$ (which will be described in more detail in the next section) and left for offline analysis.

7.4.3. Learning Rules

In this section, we describe how rules can be learned from observation. We use the Apriori algorithm (Agrawal and Srikant, 1994) for identifying rules and adapt it to our application domain. We evaluate which interestingness measure is best suited for our needs and show how the algorithm is integrated in AUGUR.

For identifying rules, we rely on an extended version $L_{relations} = \{U_i\}$ of the log L_{usage} introduced in Section 7.2. Each usage U_i does not only contain the values entered in the different interaction elements as in L_{usage}, but also the values that were present in the *Current Context* at that time, i.e.

$$U_i = \{(a_1, v_1), ..., (a_n, v_n)\} \cup \{(O_i.ac_1, v_1), ..., (O_i.ac_m, v_m) : O_i \in \textit{Current Context}\}$$

The format for storing the context values thereby is `<type>.<no>.<attribute>=<value>`, with `<no>` being a consecutive number to distinguish the different context objects in the *Current Context*. For example,

U_i=
```
{from="Frankfurt",
 to="Darmstadt",
 CalendarEntry.1.location="Darmstadt",
```

```
CalendarEntry.1.time="12:00",
CalendarEntry.2.location="Berlin",
CalendarEntry.2.time="22:00",
Location.1.location="at home"}
```

with `from` and `to` being identifiers for activities (referring to interaction elements of the application).

We want to find association rules in the database $L_{relations}$ which express dependencies of the form "if X then Y", abbreviated $X \rightarrow Y$, with the antecedent X and consequent Y being sets of items. For example a context-specific rule for the "Location" context type could state `if location="at home" then from="Frankfurt", to="Darmstadt"`. The consequent Y has to consist of at least one item (i.e. $|Y| > 1$). In our case, an item is a key-value-pair like `from="Frankfurt"` or `CalendarEntry.location="Berlin"` (ignoring the number to distinguish different context objects). For our application domain, we pose the following requirements on the rule-learning algorithm:

- **R1**$_{rule}$: The items in the consequent of a rule may only contain items describing interaction elements, not context information, as we only aim at predicting values for interaction elements.
- **R2**$_{rule}$: Further, the context objects used in the antecedent of a rule may only be gathered from a single context object in the *Current Context*.
- **R3**$_{rule}$: The algorithm has to support asymmetric rules, i.e. if the rule $X \rightarrow Y$ was learned, $Y \rightarrow X$ is not necessarily a valid rule. For example, if the rule *if I go to the cinema, then I go to Frankfurt* is true, this does not mean that the rule *if I go to Frankfurt, then I go to the cinema* is also true.
- **R4**$_{rule}$: The algorithm has to be able to cope with few usage data (see Section 2.3.1).

Apriori Algorithm

For identifying association rules, we apply the well known Apriori algorithm (Agrawal and Srikant, 1994) for association rule mining and adapt it to our needs. Apriori returns all rules that exceed a predefined minimal support $minsupp$ and confidence value $minconf$. *Support* for an itemset X is defined as the probability of its occurrence in the given log $L_{relations}$, i.e.

$$support(X) = P(X) = \frac{|\{Y \in L_{relations} | X \subseteq Y\}|}{|L_{relations}|}$$

The *confidence* in a rule is defined as its conditional probability:

$$confidence(X \rightarrow Y) = P(Y|X) = \frac{|\{T \in L | X \cup Y \subseteq T\}|}{|\{T \in L | X \subseteq T\}|}$$

The Apriori algorithm consists of two steps: (i) determining all itemsets with sufficient support and (ii) extracting rules from these itemsets that exceed a predefined minimal confidence $minconf$.

For **identifying frequent itemsets,** the Apriori algorithm at first determines the set I_1 containing all itemsets with one element (1-itemsets) with a support value of at least $minsupp$. Next, Apriori calculates all candidates for I_2 (set of 2-itemsets) by combining the elements in I_1.[23] As already mentioned, we only consider rules which contain items in the antecedent that are derived from the same context object ($R2_{rule}$). For that reason, we remove all itemsets from I_2 with items from two different context objects. For each remaining candidate set, its support is calculated and only those elements are included in I_2 whose support exceeds $minsupp$. This process is repeated for I_3, I_4... until no additional frequent itemsets can be found, i.e. $I_k = \emptyset$. The final set of frequent itemsets for our application is $\bigcup_{k>1} I_k$. Algorithm 5 illustrates the Apriori algorithm for gathering the frequent itemsets with the adaptations made for our application domain.

Algorithm 5 Apriori - Creating Itemsets

1: calculate $I_1 = \{1\text{-itemset } x : support(x) > minsupp\}$
2: k=1
3: **while** $I_k \neq \emptyset$ **do**
4: k++
5: calculate set of candidates C_k with k elements by combining itemsets of I_{k-1}
6: remove all elements in C_k which contain items derived from different context objects *(Adaptation to fulfill $R2_{rule}$)*
7: $I_k = \{x \in C_k : support(x) > minsupp\}$
8: **end while**
9: **return** $\bigcup_{i>1} I_i$

In the second step, the Apriori algorithm **generates association rules** from these frequent itemsets. At first, the confidence values for all rules $X \rightarrow Y$ with a single item as consequent (i.e. $|Y| = 1$) are calculated. For our application, Y must not contain context elements ($R1_{rule}$). All of these rules with a confidence above $minconf$ form the set H_1. Similar to the identification step, one more item is then added to the antecedent in every iteration (forming H_k) until no more new rules can be found with sufficient confidence. Algorithm 6 shows the pseudo code for this algorithm.

The confidence of a rule is not always a good measure for the interestingness of a rule (Brijs et al., 2003). For that reason, several other interestingness measures were proposed (Tan et al., 2002). For identifying the measure which is most suited for our application domain, we compared the performance for the most popular interestingness measures which support asymmetric rules ($R3_{rule}$). Besides the *confidence*

[23] Using an efficient subroutine called Apriori-Gen which will not be described in this thesis.

Algorithm 6 Apriori - Generating Rules

1: calculate $H_1 = \{x \in \{X \rightarrow Y\} : \ confidence(x) > minconf \ \wedge \ |Y| = 1 \ \wedge \ Y \notin \ Current\ Context\}$
2: k=1
3: **while** $H_k \neq \emptyset$ **do**
4: k++
5: calculate set of candidate rules C_k from H_{k-1} with k elements in the conclusion (by moving one item from the condition of a rule to its conclusion)
6: remove all elements in C_k which contain an item derived from a context object in the conclusion *(Adaptation to fulfill $R1_{rule}$)*
7: $H_k = \{x \in C_k : confidence(x) > minconf\}$
8: **end while**
9: **return** $\bigcup_i H_i$

measure, we consider the following metrics (using the definitions given in Tan et al. (2002)):

- Laplace: $laplace(X \rightarrow Y) = \frac{n \cdot P(X,Y)+1}{n \cdot P(X)+2}$ with n being the amount of entries in the underlying log, i.e. $|L_{relations}|$.
- Conviction: $conviction(X \rightarrow Y) = \frac{P(X) \cdot P(\neg Y)}{P(X,\neg Y)}$
- Certainty: $certainty(X \rightarrow Y) = \frac{P(Y|X)-P(Y)}{1-P(Y)}$
- Added value: $addedValue(X \rightarrow Y) = P(Y|X) - P(Y)$
- Klosgen: $klosgen(X \rightarrow Y) = \sqrt{P(X,Y)} \cdot (P(Y|X) - P(Y))$
- J-Measure: $jMeasure(X \rightarrow Y) = P(X,Y) \cdot \log \frac{P(Y|X)}{P(Y)} + P(X,\neg Y) \log \frac{P(\neg Y|X)}{P(\neg Y)}$
- Gini Index: $gini(X \rightarrow Y) = P(X) \cdot [P(Y|X)^2 + P(\neg Y|X)^2] + P(\neg X) \cdot [P(Y|\neg X)^2 + P(\neg Y|\neg X)^2] - P(Y)^2 - P(\neg Y)^2$
- Gain: $gain(X \rightarrow Y) = P(X,Y) - \theta \cdot P(X)$ with $\theta \in [0,1]$

Evaluation Datasets For evaluating the performance of the different interestingness measures for our application domain, we used two different datasets: (i) We created our own *Seminar Talks* dataset and (ii) the *Balloon* dataset from the UCI repository (Asuncion and Newman, 2007). Both datasets contain only few usage data to account for requirement $R4_{rule}$.

The *Seminar Talks* dataset contains 52 instances with 7 attributes (presenter, topic, abstract, data, room, type, and advisor) which represent talks given in our group. We identified six rules that describe the relations of the attributes based on the knowledge we have about the structure of the group and the seminar talks, (e.g. `if advisor:"Daniel Schreiber" then group:"MSE"`).

The *Balloon* dataset consists of four files with 16 instances each. Each instance consists of four different attributes describing the conditions of an experiment and its outcome. The instances were generated using some predefined rules, which differed in the different files: The *adult-stretch* dataset relies on two simple rules of the form `if A then B`, the *adult+stretch* dataset and the *yellow+small* dataset on a single compound rule of the form `if A and B then C`, and the *yellow+small-adult+stretch* dataset (abbreviated *complex*) on two compound rules of the form `if A and B then C`. As the *adult+stretch* and the *yellow+small* dataset base on the same type of rule, they provide about the same results for the interestingness measures. For that reason, we report here only on the findings for the *adult+stretch* dataset. Although the expressiveness and validity of the results obtained on this dataset is limited due to the small amount of data and its artificial character, we expect some further insights into the performance of the different interestingness measures.

Experiment We ran Apriori on the two datasets with the different interestingness measures to determine which one is best suited for our application domain. Due to the little amount of available data, we used an absolute support value of $minsupp = 3$. We ordered the rules according to their interestingness using one of the measures introduced above. For that reason, we did not exclude any rules in the generating rules part of Apriori, i.e. we used the minimal confidence $minconf$ of 0.0. We used $\theta = 0.5$ for the *gain* measure.

For comparing the results, we used the *precision at n* ($P@n$) measure, which computes the precision when only the first n rules are considered. However, we have to cope with the problem that the same confidence level is assigned to many rules. This makes it often difficult to identify the n rules which should be considered. For example, if the elements 4 to 8 have the same interestingness measure (see example in Figure 7.9), $P@5$ cannot be directly computed. For that reason, we take the rules with the same interestingness measure as the nth elements ($i@n$) proportionally into account. This means for our example that we weight the amount of correct rules with interestingness value $i@5$ with the factor 2/5. Thus, we calculate $P@n$ as follows:

$$P@n = \frac{|\{x : x \text{ correct} \wedge i(x) > i@n\}| + w \cdot |\{x : x \text{ correct} \wedge i(x) = i@n\}|}{\min\{n, |correctRules|\}}$$

with x being a rule, $i(x)$ its interestingness, $|CorrectRules|$ the amount of correct rules in the dataset and the weight being defined as

$$w = \frac{n - |\{x : \ i(x) > i@n\}|}{|\{x : \ i(x) = i@n\}|}$$

Results Table 7.3 and Table 7.4 provide an overview of the results obtained for the different measures for our two datasets. Overall, the *laplace* and the *jmeasure* metrics achieve the best results. As the results from the *Balloon* dataset are limited in their expressiveness and validity, we focus on the results obtained by the *Seminar Talks*

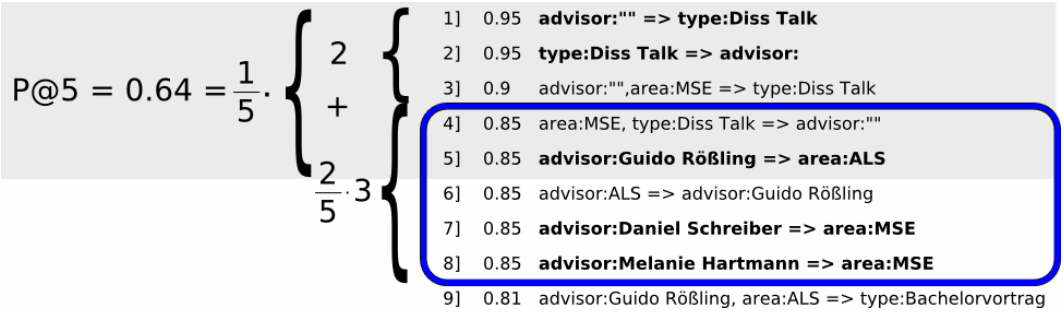


Figure 7.9. Example calculation for precision at n measure for evaluating the association rules (with $|correctRules| = 6$). The rules in bold are correct.

dataset, and decided in favor of the *laplace* measure for learning rules. However, we are aware that this decision is only based on a small dataset, and that it remains unclear whether our assumptions hold for all scenarios. For that reason, the applied interestingness measure can be easily exchanged in AUGUR.

Confidence in learned rules

The confidence c_{rule} in each rule x is calculated as follows:

$$c_{rule}(x) = laplace(x) \cdot \frac{support(x)}{\max\{min_{rule}, support(x)\}}$$

with min_{rule} being a predefined parameter which allows to adjust the influence of rules with low support. For example, if a high min_{rule} value is used, the confidence c_{rule} remains low until the rule was observed sufficiently often (i.e. at least min_{rule} times).

n	2	5	7	10
Confidence	.40	.40	.46	.66
Laplace	**1.00**	**.60**	**.66**	**.88**
Conviction	.40	.40	.46	.66
Certainty	.40	.40	.46	.66
AddedValue	.66	.40	.50	.73
Klosgen	**1.00**	**.60**	.50	.66
JMeasure	**1.00**	**.60**	.50	.50
GiniIndex	.00	.00	.00	.00
Gain	**1.00**	**.60**	.50	.50

Table 7.3. $P@n$ measures for the different interestingness measures on the *Seminar Talks* dataset (best values in bold and blue)

Dataset	adult-stretch			adult+stretch			complex		
n	2	5	7	2	5	7	2	5	7
Confidence	.06	.11	.15	.06	.11	.15	.60	.60	.70
Laplace	.37	.62	.87	.37	.62	.87	**1.00**	**1.00**	**1.00**
Conviction	.06	.11	.15	.06	.11	.15	.60	.60	.70
Certainty	.06	.11	.15	.06	.11	.15	.60	.60	.70
AddedValue	.08	.13	.19	.08	.13	.19	**1.00**	.40	.50
Klosgen	**.50**	.33	.33	**.50**	.33	.33	**1.00**	**1.00**	**1.00**
JMeasure	**.50**	**.66**	**.88**	**.50**	**.66**	**.88**	**1.00**	**1.00**	**1.00**
GiniIndex	.00	.00	.00	.00	.00	.00	.00	.00	.00
Gain	.37	.62	.87	.37	.62	.87	.60	.60	.70

Table 7.4. $P@n$ measures for the different interestingness measures on the *Balloon* dataset (best values in bold and blue).

To ensure that AUGUR does not learn too many irrelevant rules, we only consider rules with an interestingness measure above a given threshold. We empirically determined a threshold of 0.8 on the two datasets used for the experiment.

7.4.4. Computing Content Support based on Relations

In this section, we describe the algorithm for computing content support based on direct relations and rules. Algorithm 7 shows the simplified pseudo-code of the algorithm.

At first, we only consider the content support based on direct relations. For that purpose, we identify all context nodes which are directly linked to the activity a_i for which content support should be provided. In our example in Figure 7.8, this is only the context node of type "Calendar Entry". We query the *Current Context* and the *Context Server* for all context objects which match the type and the filter of these context nodes. Figure 7.10 illustrates the calculation of content support for our example: the context objects O_1 and O_3 are provided by the *Context Server* and O_2 by the *Current Context*. A content support element is constructed from each of the returned context objects which consists of values for all interaction elements which are directly related to the respective context node (step (2) in Figure 7.10). In our example, the resulting content support elements C_1-C_3 consist of values for the "date" and "time" interaction elements.

If the different relations between context node and activities vary in their confidence, several content support elements are created: At first a content support element considering only the relation (or relations) with highest confidence, then another content support element by including the relation (or relations) with the next highest confidence and so on.

Next, we augment these recommendations by applying rules on them (step (3) in

Algorithm 7 Content Support based on relations

Purpose: Compute content support based on relations for activity a_i
Require: Activity a_i

$CS = \{\}$
//Consider direct relations (step (1) and (2))
$ContextNodes$ ={all context nodes directly related to a_i}
for all $ContextNode \in ContextNodes$ **do**
  for all O_i provided by *Current Context* and the *Context Server* that match type and filter of $ContextNode$ **do**
    compute $C \subseteq O_i$ consisting of suggestions for all activities directly related to the $ContextNode$
    $CS = CS \cup \{C\}$
  end for
end for

//Consider rules (step (3))
repeat
  $CS' = \{\}$
  for all $r \in$ {context-aware rules of $ContextNodes$}$\cup$ {global rules of the current state} **do**
    for all $C \in CS$ **do**
      if C and information entered in UI match antecedent of r **then**
        $C' = C \cup$ consequent of r
        $CS' = CS' \cup \{C'\}$
      end if
    end for
  end for
  $CS = CS \cup CS'$
until $CS' = \emptyset$
return CS

Figure 7.10). For each rule, we check whether all of its conditions are met either by the attributes of the corresponding context object (for the `<contextAttribute>=<value>` conditions), by the currently entered values in the UI, or by the other elements in the content support element (for the `<interactionElement>=<value>` conditions). For example, in Figure 7.10 only the content support element C_3 satisfies the condition of rule r_2. For all rules with satisfied conditions and whose conclusions do not conflict with the already defined values of the content support element, a new content support element is created containing all the information from the original content support element augmented with the conclusions of the rule. In our example, a new content support element C_4 is created which contains all the information of C_3 augmented with the conclusion `to="Frankfurt"`, i.e. with the additional element `(to,"Frankfurt")`. After new content support elements are created, we again check

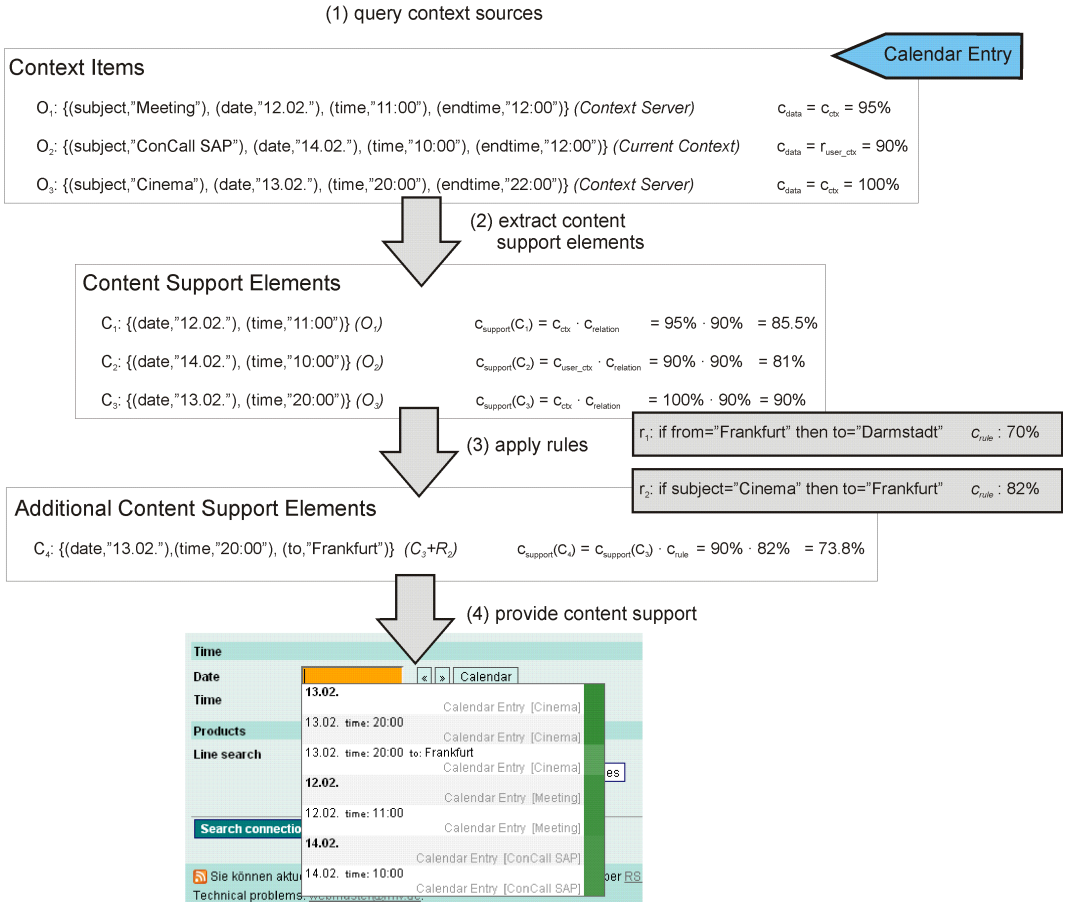


Figure 7.10. Example calculation for content support

whether the conditions of further rules are satisfied and repeat this process until no new content support elements can be found.

This augmentation of content support elements is also performed for all other content support sources (*Previous Usage* and *Semantics*) by applying the global rules. For example, if a content support element `C={(from,"Frankfurt")}` is computed based on previous usage for our example in Figure 7.8, the global rule `if from="Frankfurt" then to="Darmstadt"` is satisfied and a new content support element `C'={(from,"Frankfurt"), (to="Darmstadt")}` is created.

7.4.5. Confidence in Content Support based on Relations

As stated in Equation (7.1), the confidence $c_{support}$ for each content support element is computed as

$$c_{support} = c_{data} \cdot c_{relation}$$

The confidence c_{data} for each context object is provided by its context source, i.e. the confidence in the context object if it is provided by the *Context Server* and its

relevance if it is gathered from the *Current Context*:

$$c_{data} = \begin{cases} c_{ctx} & \text{if context object provided by } \textit{Context Server} \\ r_{user\_ctx} & \text{if context object provided by } \textit{Current Context} \end{cases}$$

For direct relations, $c_{relation}$ is given by the confidence in the link between context and interaction element (90% for the example in Figure 7.8).

The confidence in a content support element C' which is derived from applying a rule r on a content support element C is defined as:

$$c_{support}(C') = c_{support}(C) \cdot c_{rule}(r)$$

c_{rule} denotes the confidence value which is associated with rule r as described in Section 7.4.3.

7.4.6. Summary

In this section, we presented how relations between context and application elements can be learned from observation. We showed how the Apriori algorithm for mining association rules has to be extended for context-aware IUIs. We evaluated which interestingness measure is best suited for determining relevant rules for our case. Further, we presented how (learned or modeled) direct relations and rules can be used for providing content support. To our knowledge this is the first approach which is able to consider arbitrary context information for providing content support.

7.5. Chapter Summary

In this chapter, we presented three approaches how content support can be generated to facilitate entering information for the user. At first, we showed how this support can be provided based on previous usage of the application. Then we presented a novel approach which utilizes the semantics of UI and context information, which is contained in their textual descriptions, to determine which context information can be suggested as input to the user. The algorithm does not require any pretagging or training data. Thus, it can be applied for generating content support for any application right at the first usage. Finally, we presented how modeled relations between context and application elements can be used for providing content support and how new relations can be learned from observation.

8 Navigation Support

AUGUR is able to support the user's navigation in three different ways: (i) guiding her through an application, (ii) providing navigation shortcuts to other applications, and (iii) generating a reduced version of a UI for ubiquitous usage. In this chapter, we present the algorithms for these three navigation support types. For guiding the user, we need an algorithm to predict the next relevant interaction element. For that purpose, we developed a novel sequence prediction algorithm FxL (Section 8.1). In Section 8.2, we present how navigation shortcuts triggered by context events are realized in AUGUR and how new shortcuts are learned from observing the user. Section 8.3 introduces a novel approach for generating a scaled down version of a UI adapted to the user's current interactions.

8.1. Guidance

Guidance aims at helping the user navigate through an application by highlighting the next relevant interaction element (see Figure 8.1 for an example). This is especially useful when navigating in large menu structures or for novice users. It reduces the cognitive load of the user by helping her to focus on the important parts of the interface. For being able to guide the user, we need an algorithm that can predict the next user action, a so-called sequence prediction algorithm (SPA). The prediction is thereby based on the user's past behavior. In order to guide novice users, the prediction can be based on the past behavior of an expert user.

In this section, we give an overview of statistical methods for sequence prediction that can be applied without any prior task knowledge. We give formal definitions of state-of-the-art SPA algorithms, and then present our own approach. We show that it outperforms existing approaches regarding its prediction accuracy. Further, we compare the best performing algorithms with respect to the required memory and processing time, as those factors are of great importance when used on small mobile devices, and show that the FxL algorithm is also superior to related approaches in this respect.

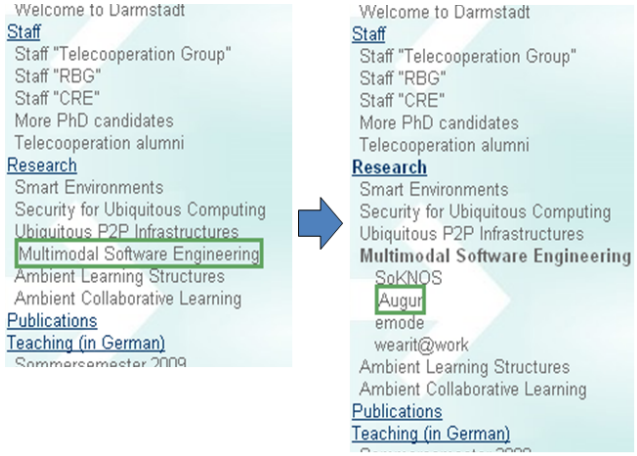


Figure 8.1. Example for guiding the user in a menu structure

8.1.1. Sequence Prediction

Algorithms for sequence prediction were mainly developed in the area of data compression. IUIs also make use of such predictions for facilitating interaction (e.g. in an intelligent home environment (Gopalratnam and Cook, 2007)) or for assisting the user by giving explanations (e.g. adaptive tutoring system (Künzer et al., 2004)).

The problem of predicting the next symbol (representing a user action) in an input sequence can be formally defined as follows:

> ***Sequence Prediction Algorithm (SPA)**: Let Σ be the set of possible input symbols and let $A = a_1 \dots a_n$, with $a_j \in \Sigma$, be a sequence of input symbols of which the first i symbols, that is $a_1 \dots a_i$, have already been observed. An SPA decides at first whether it is able to make a prediction and if so returns the probability for each symbol $x \in \Sigma$ for x being the next element in the input sequence. These values define a conditional probability distribution P over Σ, where $P(x|a_1 \dots a_i)$ is the probability for the singleton subset of Σ containing x.*

For applying an SPA for guiding the user, it has to satisfy certain requirements. At first, it has to meet the requirements that we identified regarding the competence of an IUI (see Section 2.3.1):

- **R1$_{SPA}$ (few usage data)**: An SPA for context-aware IUIs should be able to cope with few usage data.
- **R2$_{SPA}$ (changing user behavior):** An SPA for context-aware IUIs should adapt to changing user behavior.

Further, the algorithm needs a high applicability. For example, in a mobile setting, it is not feasible to display all available interaction elements on the small screens

of mobile devices. Thus, we want to reduce complex interfaces to the interaction elements that are most relevant. In such a setting, we need sequence prediction algorithms with a high applicability. Thus, we formulate $R3_{SPA}$ as follows:

- **$R3_{SPA}$ (high applicability)**: An SPA for context-aware IUIs should have high applicability values.

Of course, the algorithms should also provide very accurate results

- **$R4_{SPA}$ (high accuracy):** An SPA for context-aware IUIs should provide high accuracy values.

Finally, we also want to be able to run the algorithms on mobile devices with memory and power restrictions.

- **$R5_{SPA}$ (resource-conserving)**: An SPA for context-aware IUIs should be fast and require few memory.

8.1.2. Existing Algorithms

In this section, we describe the four most prominent existing approaches of SPAs. There are two ways of calculating the probabilities: on-demand or live. Algorithms using the former method maintain a data structure to compute the probabilities. They update the data structure after each symbol in the input sequence, whereas the live algorithms directly update the probability distributions. In real-life settings, the set of possible input symbols Σ may be unknown. We assume that the probability of an unseen element is 0.

IPAM

The IPAM algorithm by Davison and Hirsh (1998) employs a first order Markov model, i.e. it bases its predictions only on the last symbol in the input sequence. IPAM maintains a list of unconditional probabilities $P_{ipam}(x)$, $x \in \Sigma$ and a table of conditional probabilities $P_{ipam}(x|y)$, $x, y \in \Sigma$. The table of conditional probabilities is directly used as probability distribution for predicting the next symbol, i.e. $P(x|a_1 \dots a_i) = P_{ipam}(x|a_i)$. This live algorithm updates its probability distribution as follows after observing the symbol a_{i+1} in the input sequence:

$$P'_{ipam}(x|a_i) = \begin{cases} \alpha\ P_{ipam}(x|a_i) + (1-\alpha) & \text{if } x = a_{i+1} \\ \alpha\ P_{ipam}(x|a_i) & \text{otherwise} \end{cases}$$

The new probabilities $P'_{ipam}(x|y)$ are computed for all $x \in \Sigma$ from $P_{ipam}(x|y)$ using an aging factor α and thus meeting requirement $R2_{SPA}$. The probabilities $P_{ipam}(x|y)$ for $y \neq a_i$ are not updated and the same equation holds for the unconditional probabilities. If a symbol y was observed for the first time, its conditional probability distribution is initialized with $P_{ipam}(x|y) = P_{ipam}(x)$ for all $x \in \Sigma$ and its unconditional probability is initialized with 0 before updating the probabilities as above.

ONISI

Gorniak and Poole (2000) argue that the last action does not provide enough information to predict the next action. Hence, they build an on-demand prediction model called ONISI that employs a k-nearest neighbors scheme. They consider not only the actions performed by the user, but also the corresponding user interface states. They compute a probability distribution according to the k longest sequences in the interaction history that match the immediate history. Gorniak and Poole found that $k = 5$ was sufficient to gain optimal results in their example of a Web application (for learning AI concepts). However, some actions are strongly correlated to the state in which they are performed, but do not belong to long sequences. To account for this fact, Gorniak and Poole weigh off the probability distributions determined by the current state and by the longest sequences. For that purpose they use a weighting factor β.

$$P_{onisi}(x|(s_1, a_1) \dots (s_i, a_i)s_{i+1}) =$$

$$\beta \frac{l(s_{i+1}, x)}{\sum_y l(s_{i+1}, y)} + (1 - \beta) \frac{fr(s_{i+1}, x)}{\sum_y fr(s_{i+1}, y)}$$

Thereby, $l(s, a)$ returns how often the state action pair (s, a) occurred after the longest k sequences that match the recent interaction history. In contrast, $fr(s, a)$ reflects how often action a occurred in the interface state s.

Jacobs Blockeel

Jacobs and Blockeel (2002) claim that the longest sequence in the interaction history which matches the immediate interaction history is not always the best choice for determining the probabilities of the next symbol, and that the ideal length cannot be known in advance. Their live algorithm builds upon IPAM, but allows longer premises in the table of conditional probabilities. For that purpose, they add a step that is performed only after a correct prediction was made. If the algorithm made a correct prediction for a_{i+1} after observing the sequence $A = a_1 \dots a_i$, new entries for every C that is suffix of A with $P_{jb}(x|C) > 0$ are added to the probability table. Let L be the longest suffix of the concatenation $C \circ a_{i+1}$ for which already an entry $P_{jb}(x|L) > 0$ exists. This probability distribution is taken as the best estimation of the new probabilities. Let P_{jb} be the probabilities that result after the IPAM update step. Next the probabilities for all new premises $C \circ a_{i+1}$ are computed as

$$P'_{jb}(x|C \circ a_{i+1}) = P_{jb}(x|L).$$

So the algorithm does not rely on a fixed order Markov model, but uses a mixed order approach to compute the probability distribution of the next element. Using the above equations, the sum of the probabilities over Σ is not always 1 and a normalization has to be performed.

ActiveLeZi

Another on-demand algorithm that considers several Markov models is ActiveLeZi (Gopalratnam and Cook, 2007). It stores the frequency of input patterns in a trie according to the compression algorithm LZ78, but uses a variable length window of previously-seen symbols. The size of the window grows with the number of different subsequences seen in the input sequence. When a new sequence is observed, which is not yet stored in the database, the window size is increased accordingly if the sequence length exceeds the size of the current window. Let suf_l be the suffix of length $l+1$ of the immediate interaction history A, that is $a_{i-l} \dots a_i$. The probabilities are recursively defined as follows:

$$P^0_{alz}(x|A) = \frac{fr(x)}{\sum_{y \in \Sigma} fr(x \circ y)}$$

$$P^l_{alz}(x|A) =$$
$$\frac{fr(suf_l \circ x)}{fr(suf_l)} + \frac{fr(suf_l) - \sum_{y \in \Sigma} fr(suf_l \circ y)}{fr(suf_l)} P^{l-1}_{alz}(x|A)$$

Thereby, $fr(x)$ returns the frequency of the input pattern x as stored in the trie. The probability distribution that is finally returned by ActiveLeZi is P^k_{alz} where k is the current size of the window.

8.1.3. FxL

We developed an algorithm using a mixed-order Markov model which is able to cope with changing user behavior. The presented algorithm builds on the KO algorithm (Künzer et al., 2004), but extends it with aging to meet $R2_{SPA}$. Further, the KO algorithm does not achieve high applicability values as it takes only frequencies of subsequences into account that have a minimal support in the input sequence, thus violating $R4_{SPA}$.

The FxL algorithm operates on the *Usage Model* as presented in Section 6.2.1. The *Usage Model* stores an n-gram trie with frequencies $fr(a_1 ... a_i)$ of the different input sequences $a_1 ... a_i$. $fr(a_1 ... a_i)$ reflects how often a sequence $a_1 ... a_i$ has already been observed, whereby the influence of older sequences is decayed with an aging factor α. To reduce the amount of data that needs to be stored and processed, only sequences of a length up to a specified value k are taken into account.

For each symbol $x \in \Sigma$, we compute a $score(x)$ by adding the absolute frequencies of x succeeding any suffix (up to length $k-1$) of $a_1 ... a_i$. As the longer suffixes yield more reliable results than the shorter ones, the frequencies are assigned a weight which corresponds to the length j of the suffix that is considered. Thus, the score is computed as follows:

$$score(x) = \sum_{j=1}^{k-1} j \cdot fr(a_{i+1-j} ... a_i \circ x)$$

We call this approach **FxL** as the score for a symbol is calculated by multiplying the frequency (F) of the symbols with the length (L) of their preceding suffixes. As the scores for the symbols can sum up to a value greater than 1, they have to be normalized to result in probability values $P(x|a_1...a_i) \in [0,1]$:

$$P(x|a_1...a_i) = \frac{score(x)}{\sum_{y \in \Sigma} score(y)} \tag{8.1}$$

8.1.4. Confidence in Guidance

The confidence $c_{support}$ in guidance support based on a sequence prediction algorithm, is given by its probability, i.e. for using FxL

$$c_{support}(x) = P(x|a_{i+1-k}...a_i)$$

8.1.5. Evaluating Sequence Prediction Algorithms

In order to determine which of the presented algorithms is best suited for guiding the user in context-aware IUIs, we already compared the algorithms regarding whether they are able to adapt to changing user behavior ($R2_{SPA}$). Further, we have to evaluate their performance regarding the amount of required training data ($R1_{SPA}$), the applicability ($R3_{SPA}$) the accuracy ($R4_{SPA}$) of the algorithm, and the required resource consumption ($R5_{SPA}$). In this section, we point out how the accuracy and applicability of an algorithm can be computed and which factors, beside the available data, has to be taken into account.

In the literature, two measures are used to compute the accuracy of an SPA algorithm: prediction accuracy pr_{ac} and prediction probability pr_p. Albrecht et al. (1998) state that the prediction probability provides finer-grained information about the performance of an algorithm than the prediction accuracy, but that both measures produce generally consistent assessments. Both measures are computed by assigning a score to every prediction made by the algorithm and averaging over the number of predictions made by the algorithm, but they differ in the way the score is computed. pr_{ac} and pr_p is defined as follows for the input sequence $a_1...a_n$:

$$pr_x(a_1...a_n) = \frac{1}{m} \sum_{i=0}^{n-1} eval_x(a_1...a_i, a_{i+1})$$

where m ($\leq n$) is the number of predictions made and $eval_x$ is the evaluation function that returns the value for a single prediction (it returns 0 if no prediction was made by the algorithm). The $eval_x$ function thereby differs for the two metrics:

- The prediction accuracy $eval_{ac}$ considers only the symbols which are predicted with maximal probability. We define the set $\hat{A}$ as the set of all these symbols: $\hat{A}_{i+1} = \{x \in \Sigma : \forall y \in \Sigma . P(x|a_1 \dots a_i) \geq P(y|a_1 \dots a_i)\}$. The $eval_{ac}$ function for the prediction accuracy then returns a value reflecting whether the actual

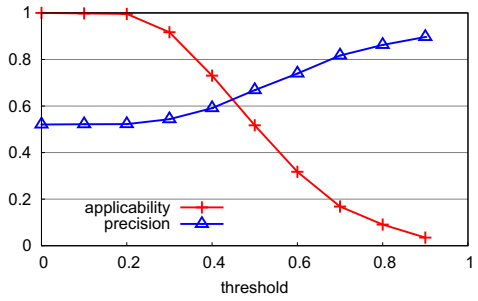


Figure 8.2. Applicability and precision for different confidence thresholds[24]

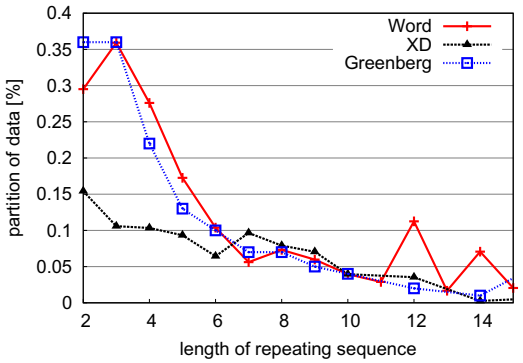


Figure 8.3. Distribution of repetitive sequences for the three real datasets used in the evaluation

next symbol a_{i+1} is among these values $\hat{A}_{i+1}$. Thus, $eval_{ac}$ is 0 if $a_{i+1} \notin \hat{A}_{i+1}$ otherwise it is computed as $eval_{ac}(a_1...a_i, a_{i+1}) = \frac{1}{|\hat{A}_{i+1}|}$.

- In contrast, the $eval_p$ function for the prediction probability rates with which probability the correct symbol was suggested, no matter if it has been assigned a maximal probability or not: $eval_p(a_1...a_i, a_{i+1}) = P(a_{i+1}|a_1...a_i)$.

The applicability ap of an algorithm is defined as the ratio of input symbols the algorithm was able to make a prediction for: $ap = m/n$.

As the algorithms are often evaluated on datasets containing the input sequences of several users, the results have to be averaged over all users. This can be done by computing the average over the results of all users weighing every user equally, independent of the length of the corresponding sequence (*macro-average*), or by averaging over all data (*micro-average*), emphasizing frequent users.

In order to judge the accuracy and applicability of the algorithm in different conditions, we need to evaluate the algorithms depending on several parameters influencing their performance. The most important parameters are:

1. **dataset size available for training** (referring to $\mathrm{R1}_{SPA}$)

2. **distribution of repetitive sequences**: We call a sequence *repetitive* if it is not part of a longer repetitive sequence and has a minimal support of s, i.e. it occurs at least s times in the corresponding dataset. The parameter s can be a constant value or defined relatively to the dataset size.

3. **noise in the repetitive sequences:** Noise is introduced in a repetitive sequence if the user alters the sequence of actions. However, to measure noise the repetitive sequences have to be known in advance.

To facilitate the comparison of the different algorithms, it is useful to compile the resulting performance metrics (pr and ap) in a single metric. One possibility is to combine the applicability ap and prediction quality pr in a single measure, comparable to the F-measure for precision and recall values. As one of our requirements is high applicability, we decided to use a fixed applicability value to compare the results regarding their prediction quality, and only use ap as additional parameter. For measuring the prediction accuracy and prediction probability for a given applicability, several (ap, pr) pairs have to be computed to infer the pr value corresponding to the specified ap value from them. The (ap, pr) pairs can be obtained by evaluating the algorithm using several thresholds for the reported predictions. This means that only predictions are taken into account which have a probability above this threshold. We assumed (and also found) that a higher threshold leads to a decrease in applicability and to an increase in the prediction accuracy of the algorithm (see Figure 8.2). The required pr value for the desired applicability is finally computed by linear interpolation of the (ap, pr) values. This approach is thus independent of the actual threshold used and allows for a fair comparison between the algorithms.

8.1.6. Evaluation

For the evaluation of the SPAs, we used three datasets (Word, XD, and Greenberg) containing real usage data. The Word dataset contains logs of MS Word usage[25] described in (Linton et al., 2000). The Greenberg dataset (Greenberg, 1988) is widely used in the literature, and contains more than 225,000 UNIX commands from 168 users divided into four categories depending on their computer experience. The CrossDesktop[26] dataset (abbreviated XD) contains log data from a Web application for managing files and emails. It contains about 200,000 requests from 37 users, where the usage varies heavily between the different users. Figure 8.3 shows the sequence distribution for the real datasets Word, XD, and Greenberg used in our experiments (whereby we used a minimal support s of 5). Additionally, we generated synthesized datasets to evaluate the algorithms with respect to the parameters "distribution of repetitive sequences" and "noise in repetitive sequences" that cannot be varied in datasets containing real usage data.

We chose to macro-average the results for the different users because in our application domain it is more important to optimize the results for the average than for the frequent user. We only used prediction accuracy pr_{ac} as evaluation metric for presented results, because we are more interested in the prediction with the highest probability than in its actual probability value. However, we yield consistent results if we use prediction probability pr_p, or if we micro-average the results. As discussed in Section 8.1.5, we analyze the performance of the algorithms regarding

[24] The precision values for the threshold 100% are not shown because hardly any predictions are made in this case. For that reason, the corresponding precision value is very unreliable and fluctuates heavily.

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

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

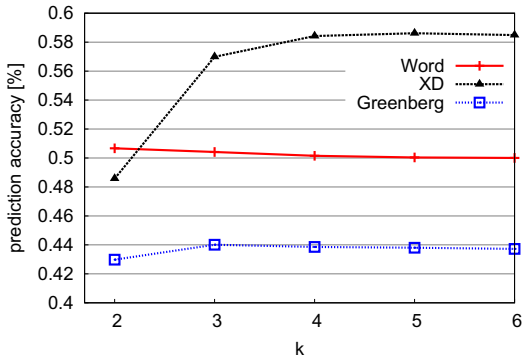


Figure 8.4. Evaluation over the maximum length k of considered suffixes for the three real datasets (without training data)

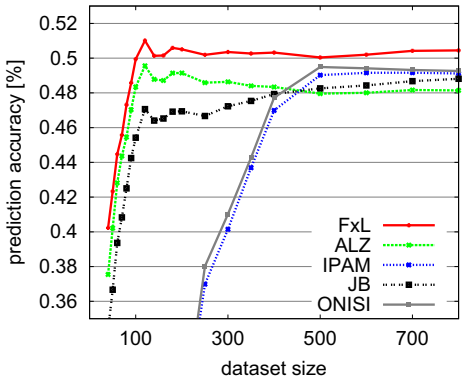
the four parameters (i) dataset size available for training, (ii) distribution of repetitive sequences, (iii) noise, and (iv) applicability. In the presented results, we used applicability levels of 90% if not stated otherwise.

Algorithms For most algorithms, we used the configuration which was empirically determined by the authors, as we found that this led to good results on a small training dataset. This means, we used $\alpha = 0.8$ for IPAM, $\beta = 0.9$ and $k = 5$ for ONISI, and $\alpha = 0.9$ for FxL. For the FxL algorithm, we use $k = 5$ as the maximum length of considered suffixes, but we found that the algorithms already reach their optimum with $k = 3$ or $k = 4$ (as can be seen in Figure 8.4) and stay at that level for greater k. For the evaluation, we have to limit the ONISI algorithm to a single interface state, as the real datasets used in the experiments have no interface states and no restriction for the order of actions.

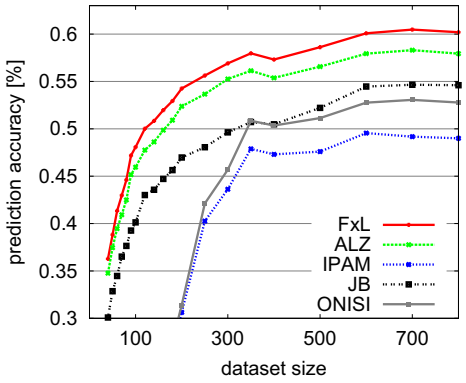
Dataset size At first, we calculated the average prediction accuracies for the different algorithms depending on the dataset size. Thereby, we included all predictions in the performance evaluation (no training data). As the results in Figure 8.5 show,

Algorithm	**Prediction Accuracy [%]**		
	Word	XD	Greenberg
FxL	**50.0**	**58.6**	**43.8**
ActiveLezi	47.9	56.7	41.8
IPAM	49.0	47.6	38.7
Jacobs Blockeel	48.3	52.2	41.9
ONISI	49.7	51.0	38.5

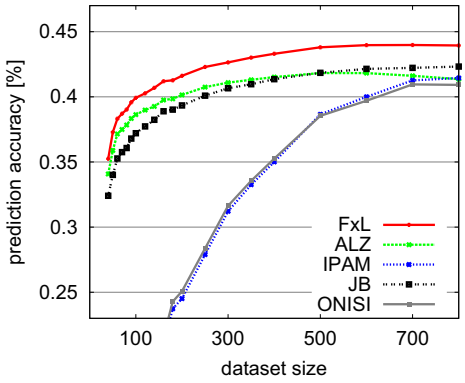
Table 8.1. Macro-average prediction accuracy pr_{ac} for an applicability of 90% and 500 symbols (without training, best values in bold and blue)



(a) Word dataset



(b) XD dataset



(c) Greenberg dataset

Figure 8.5. Performance on the different dataset with respect to dataset size

FxL performs best on all datasets followed by ActiveLeZi (ALZ). The mixed-order Markov models (FxL, ALZ, and Jacobs Blockeel (JB)) outperform the other approaches on all datasets. Their accuracies increase with the dataset size, and reach a relatively stable level when considering more than 500 symbols. Table 8.1 list the prediction accuracy for all algorithms for 500 symbols.

Applicability We calculated prediction accuracy and applicability pairs or the different datasets as described before. Thereby, we used 20% of the data for training. Figure 8.6 shows the result for the XD dataset. The charts for the Greenberg and Word dataset are very similar to the chart for the XD dataset. For that reason, they are omitted in this thesis. The figure shows that the prediction accuracy pr_{ac} decreases with growing applicability as was expected. FxL again outperforms the other algorithms followed by ActiveLeZi.

Distribution of repetitive sequences In Section 8.1.5, we already pointed out that the three datasets differ in the underlying sequence distributions. To analyze the algorithms regarding this parameter, we created synthetic datasets varying the sequence distribution with a dataset size of 500 symbols. Figure 8.7 shows the average prediction accuracies over 80 randomly generated datasets. Thereby, the x-coordinate represents the sequence lengths that were repeated most often in the dataset. The result is similar to the previous ones: The algorithms FxL and ActiveLeZi outperform the other algorithms, and the mixed-order algorithms perform better than the other ones.

Noise At last, we have to consider the parameter noise for the evaluation. Thus, we also created synthetic datasets that vary in their noise level. We used three operators to insert noise: an element from the original sequence could be (i) left out, (ii) repeated up to 5 times, or (iii) exchanged with the following element. The versions of the dataset differed with respect to the probability with which a noise operator was applied on each element in the dataset. The results in Figure 8.8 show the decay in the algorithms' performances with rising noise levels and confirm our former results: FxL and ActiveLeZi perform better than the other algorithms.

Storage and Computational Costs As stated in $R5_{SPA}$, we also have to consider the storage and computational costs of the algorithms. We compared these factors for ActiveLeZi and FxL, as they provided the best results concerning the accuracy under the different conditions. The storage costs of ActiveLeZi grow with the dataset size, whereas the storage costs for the FxL algorithm are limited by the specified k and the amount of possible user actions ($|\Sigma|$). Table 8.2 lists the average costs of the two algorithms which performed best on the three real datasets, i.e. FxL and ActiveLeZi.

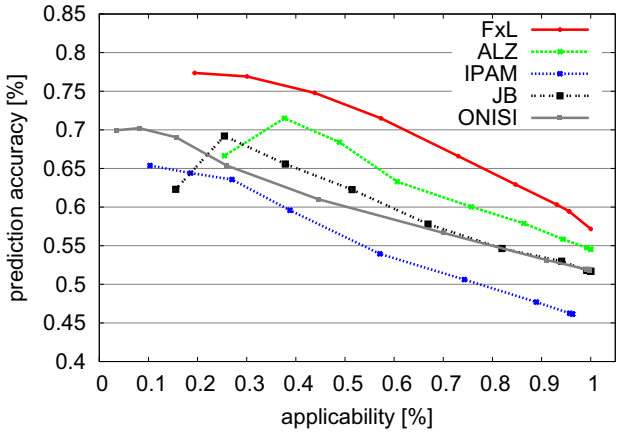


Figure 8.6. Prediction accuracy of the XD dataset for different levels of applicability when using 20% training data

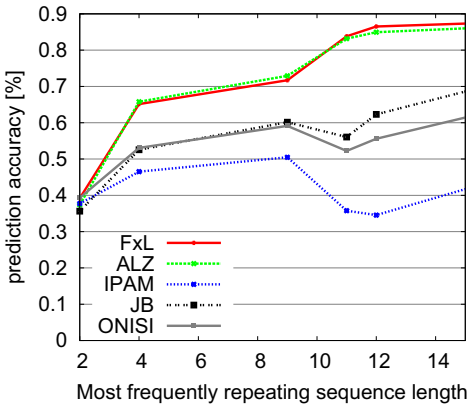


Figure 8.7. Performance regarding sequence length

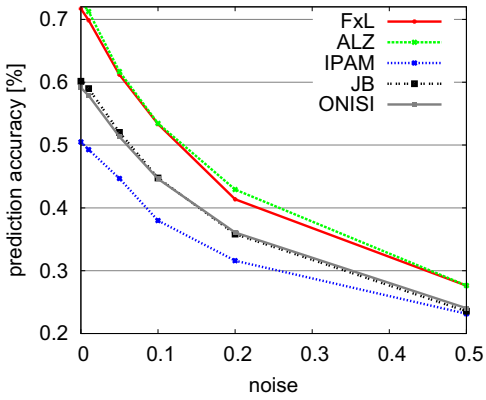


Figure 8.8. Performance of the algorithms regarding different noise levels

Thereby, "stored keys" reflects the keys contained in the main data structure used by the algorithm. We list values for the FxL algorithms with $k = 3$ (FxL:3) and $k = 6$ (FxL:6). "Computing time [ms]" refers to the time which is needed in average for a single prediction for the whole dataset. The results show that the FxL algorithm uses less resources and can perform the computation more quickly than ActiveLeZi.

		Greenberg	XD	Word
prediction accuracy	FxL	46.2%	59.7%	51.4%
	ALZ	42.7%	56.0%	48.5%
stored keys	FxL:3	258	227	111
	FxL:6	2061	1599	949
	ALZ	2706	2743	1955
computing time [ms]	FxL	0.58	0.89	0.86
	ALZ	1.50	4.87	3.18

Table 8.2. Average performance values for the three datasets containing real usage data. The prediction accuracy refers to the overall accuracy on the datasets with 90% applicability and when using 20% of the data for training.

8.1.7. Summary

In this section, we compared several sequence prediction algorithms which are required for guiding the user in context-aware IUIs. We presented a novel algorithm called FxL based on mixed-order Markov models which uses aging to take changing user behavior into account, thus satisfying $R2_{SPA}$. We showed that FxL outperforms the other approaches with respect to its accuracy ($R4_{SPA}$) under various conditions (varying applicability, dataset size, sequence distribution, and noise level). FxL further provides the best accuracy values while maintaining high applicability ($R3_{SPA}$). The algorithm shows the best results even for small dataset sizes ($R1_{SPA}$). We further compared the computational time and memory consumption ($R5_{SPA}$) of FxL with the one of its closest competitor, ActiveLeZi. We showed that FxL is faster and more resource-conserving than ActiveLezi, and thus the ideal candidate for guiding users in context-aware IUIs.

8.2. Navigation Shortcuts

The goal of navigation shortcuts is to support the user in switching to an application that provides information which might be of interest for a user in a given situation (see for example Figure 8.10). As discussed in Section 4.3.3, a navigation shortcut can be triggered

- by **external events**, e.g. if someone calls, the user might want to switch to the contact page of this person, or
- by **information present on the current UI**, e.g. if an address is shown, we can suggest a shortcut to a map application for looking up this address.

The relations between context information and UIs are modeled via data relations between context and state nodes in the application model. An example of such an application model can be found in Figure 8.9. As soon as an event of the given context type (here "Phone Call") occurs, a navigation shortcut to the UI associated with the corresponding state node is provided to the user. In case the context node specifies filters, the navigation shortcut is only triggered if the event matches the filters. For example, if a navigation shortcut should only be provided for incoming phone calls from SAP.

8.2.1. Learning Navigation Shortcuts

For learning which navigation shortcuts should be provided to the user when an event occurs, we need a log L_{event} to store the events and all navigation shortcut candidates. The log L_{event} contains a set of eventlogs $E_i = \{e_i, s_j, s_k ...\}$ with e_i being the context type of an event (e.g. "Phone Call") and s_i the id of a navigation shortcut candidate, i.e. the id of a UI (represented as the id of its state in the application model). We consider all UIs as navigation shortcut candidates to which the user navigated after the event was raised, because these UIs are likely to be in a causal relation to the event. As the probability of a causal relation between an

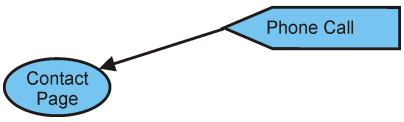


Figure 8.9. Example ATML model which contains a data relation to be used for providing navigation shortcuts

Figure 8.10. Example of a navigation shortcut

event and the navigation to a UI decreases over time, we limit the learning process to the next n UIs which are visited by the user within m minutes. For the AUGUR prototype, we empirically determined $n = 3$ and $m = 5$ to be suitable values.

We define the confidence in a relation between an event e and a UI s as:

$$c_{relation}(e, s) = \frac{|\{X \in L_{event} : \{e, s\} \subseteq X\}|}{\max\{|L_{event}|, min_{navigation}\}}$$

Thereby $min_{navigation}$ is used to adjust the influence of relations with a low (absolute) support. In case of a high $min_{navigation}$ value, an event-state combination has to be observed rather often to be suggested as navigation shortcut to the user.

For learning relations, AUGUR determines all $\{e, s\}$ combinations with $c_{relation}(e, s) > t_{highlight}$. All combinations with a confidence below this threshold would never be considered for a navigation shortcut as their confidence is too low. Thus, they do not need to be included in the application model. For each event-value pair, AUGUR checks whether the corresponding relation is already contained in the application model. If this is the case, AUGUR updates the confidence in the relation accordingly, otherwise it adds a new relation to the application model.

8.2.2. Confidence in Navigation Shortcuts

The confidence in the navigation shortcut support $c_{support}$ is computed analogue to $c_{support}$ for the content support (see Section 7) as $c_{support} = c_{data} \cdot c_{relation}$, because it also depends on the confidence in the event (c_{data}) and in the relation ($c_{relation}$). As the data is always provided by the *Context Server*, $c_{data} = c_{ctx}$. If the relation between context and UI was modeled by the end-user or an application developer, $c_{relation} = 1$. The confidence in learned relations is determined by the learning algorithm as described in the previous section. Thus,

$$c_{support} = \begin{cases} c_{ctx} & \text{if relation between UI element and context is modeled} \\ c_{ctx} \cdot c_{relation} & \text{if relation between UI element and context is learned} \end{cases}$$

8.2.3. Summary

In this section, we presented how context is used by AUGUR to provide navigation shortcuts. We further pointed out how new relations between context and applications can be learned to provide new navigation shortcuts.

8.3. Interface Adaptation

The interaction costs for mobile usage of applications are usually much higher than for traditional desktop settings. One possibility to increase the usability in these settings is to decrease the required amount of interactions, i.e. key presses or scroll movements to fulfill a task with the application and thus to facilitate the navigation. This can be achieved by reducing the UI to the most important parts. To identify the most relevant interaction elements, we developed a novel algorithm FxL* which predicts the most relevant interaction elements based on the environmental context (i.e. the device used) and the user context (i.e. how she usually interacts with an application).

In the following, we at first list the requirements for a context-aware adaptation process (Section 8.3.1), and give an overview of the state-of-the-art approaches for adapting UIs in Section 8.3.2. In Section 8.3.3, we present the overall adaptation process applied by AUGUR. Section 8.3.4 introduces the FxL* algorithm for computing the relevant interaction elements for that purpose, based on the device used and on the user's current situation (i.e. the user context). In Section 8.3.5, we compare this approach to approaches that do not take the user or her current situation into account.

8.3.1. Requirements

For context-aware interface adaptation, the same requirements apply as identified in Section 3.1, i.e.

R1 (Awareness of user context and environmental context) An ideal context-aware interface adaptation should be able to consider all kinds of user and environmental context for adapting the user interface in the optimal way. Thereby, device information is one of the most important environmental context information that needs to be considered.

R2 (Cope with error-prone context) An ideal context-aware interface adaptation shall be able to cope with the error-prone nature of context information.

R3 (Application-independence) An ideal context-aware interface adaptation should be able to facilitate the interaction for all kinds of applications.

R5 (Support for legacy applications) An ideal context-aware interface adaptation should be able to provide support for existing applications and gather information from the user's interaction with them.

R6 (Involving end-user) An ideal context-aware interface adaptation should enable the end-user to specify additional information which need to be considered in the adaptation process.

R7 (Learning capabilities) An ideal context-aware interface adaptation should learn how to adapt the user interface from observing the user's interactions.

The only requirement which does not need to be addressed for interface adaptation is R4 (*Support across application boundaries*), because there is no need to transfer information between applications for adapting the interface.

8.3.2. Related Work

The most common approach to generate adapted UIs is the usage of a **platform independent UI description language**. This abstract UI description contains all information necessary to transform it to a concrete UI for a given device.

Another approach is to specify **adaptation rules** for concrete UIs. For example, to increase the size of all buttons, or to display only the five most relevant interaction elements.

In the following, we give an overview of the most important related work in these two areas. Table 8.3 summarizes which requirements are satisfied by the different approaches. Thereby, simple user profiles are not considered as user context.

Platform independent languages

The goal of automatic device adaptation using platform independent languages is to save development costs by automatically transforming a single UI description into concrete UIs on a multitude of platforms. Abrams et al. (1999), Puerta and Eisenstein (2002), and Ziegert et al. (2004) propose notations that need to be instantiated into a concrete UI for each target platform[27].

XWeb (Olsen et al., 2000) introduces a device independent representation for UI instances to reduce the modeling effort. In addition to device independence, Calvary et al. (2003) aim at modality independence. For that reason, they introduce further layers of abstraction. Most of these languages support the usage of environmental context information for choosing the most appropriate UI representation (Limbourg and Vanderdonckt, 2004; Vanderdonckt et al., 2008). However, the user context is restricted to static user profiles like the user's skills and preferences (R1: -). We subsume all these approaches under the term **MDA** (model-driven architectures) in Table 8.3.

A different approach is taken by the **SUPPLE** system. Gajos and Weld (2004) consider interface adaptation as an optimization problem. For that purpose, they take a functional UI description, a user model as well as the device context into account (R1: +). The adaptation is performed by optimizing a cost function, which is based on the predicted average costs for interacting with a UI on a device for a given user. These interaction costs are thereby learned from observing the user (R7: +).

As the platform independent languages only require a model of the application for the adaptation process, these approaches can be applied to different applications (R3: +). However, they introduce a high modeling effort. Further, existing applications usually do not provide a separate abstract UI description which can be used

[27] See Souchon and Vanderdonckt (2003) for a more detailed survey of such languages.

		Awareness of user context* (R1)	Awareness of environmental context (R1)	Cope with error-prone context (R2)	Application-independence (R3)	Support for legacy applications (R5)	Involving end-user (R6)	Learning capabilities (R7)
Platform independent languages	*MDA*		●		●			
	SUPPLE	●	●		●			●
Adaptation rules	Highlight				●	●	●	
	Palio		●		●	●		
	MICA	●		○				●
	UIDE	●		○				●
	Findlater	●		○				●
	Smyth	●			●	●		●

Table 8.3. Overview of the state-of-the-art approaches for interface adaptation with respect to the identified requirements (* using static user profiles is not considered as user context, ○: limited support, *MDA* subsumes several systems with same characteristics)

to create a UI for interacting with the application. Thus, platform independent languages cannot be applied to existing applications without modifying the application itself (R5: -).

Adaptation rules

In contrast to platform independent languages, systems that base on adaptation rules operate on the concrete (platform-dependent) user interface level, and apply adaptations on it. The **Highlight** system by Nichols et al. (2002) enables the end-user to specify how the UI of a web application should be reduced to a mobile version. For that purpose, the user can choose which parts of the UI should be displayed. However, this approach does neither consider the environmental context (especially the device context) nor the user context (R1: -). However, it enables the end-user to control the adaptation (R6: +).

The **PALIO** framework (Stephanidis et al., 2004) is targeted to information seeking tasks. It automatically adapts the content and the UI to the user and device context. However, the user context is again restricted to simple user profiles (R1: +). The PALIO framework transforms the UI into an XML representation (if the UI is not already available as XML representation). Thus, it is application-independent (R3: +). Adaptation rules are then applied to this representation. The UI is transformed into a representation understandable by the user's device (e.g. HTML, WML) and sent to the user.

Several approaches exist that consider also user context information beyond profile information. However, they only focus on desktop settings or on a few predefined devices, thus they are not able to incorporate the environmental context, especially not the device context (R1: -). For example, **MICA** (Mixed-Initiative Customization Assistance) (Bunt et al., 2007) relies on an algorithm to suggest customizations of the UI based on an automatic analysis of interaction costs. The user can then easily switch between the full UI version and her personalized version. **UIDE** (Sukaviriya and Foley, 1992) suggests the reordering of menu items based on evaluating possible arrangements against a keystroke performance model considering the user's previous interactions.

Findlater and McGrenere (2008) add a personalized section to the menubar of Microsoft Office that contains only options that are most relevant given the user's dynamic properties (e.g. the most recently used options). However, as they only used two predefined devices to compare the users' performance for this kind of adaptation for desktop and for mobile devices, their approach does not automatically adapt to the currently used device (R1: -). However, like the two previous approaches, it enables the user to ignore the provided interface adaptations, thus accounting at least a little for the unreliability of the context information (R2: o). Further, these approaches depend on a specific application (R3: -).

Smyth and Cotter (2002) optimize the navigation by adapting the navigation structures for wireless portals, especially for WAP portals. Thereby, they restrict themselves to hierarchical applications, where every point in the application can only be reached in a single way. However, these approaches in turn only optimize for one device configuration. Thus they do not adapt to the environmental context (R1: -).

Summary

Platform Independent Languages cannot be applied for AUGUR, as we want to provide support for arbitrary existing applications (R5 - *Support for legacy applications*). Existing applications usually do not provide a separate abstract UI description which can be used to create a UI for interacting with the application. For that reason, we apply *Adaptation Rules* on concrete UI instances of the applications.

The user context used by existing approaches is often limited to static user properties, e.g. motor skill level or short sightedness. To our knowledge, there is no approach that combines the adaptation to dynamic user context (like the user's cur-

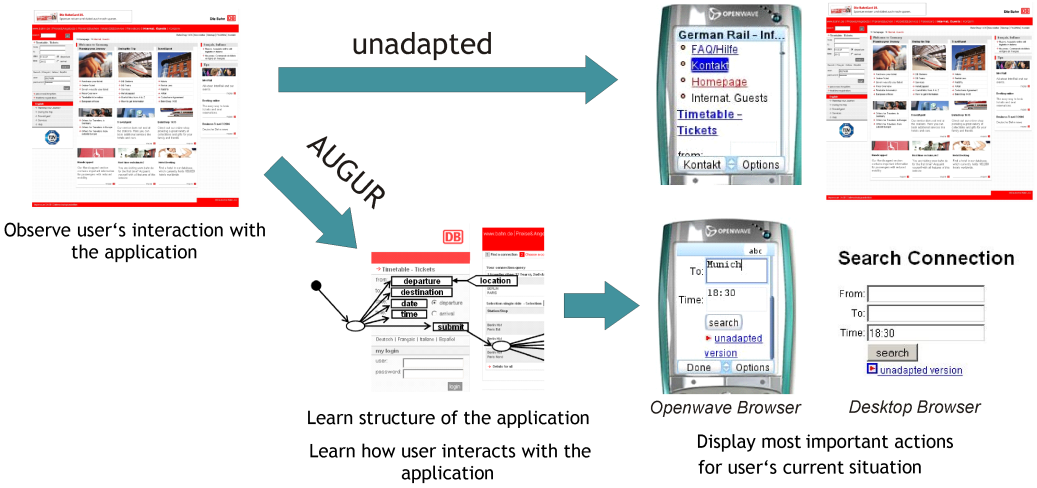


Figure 8.11. Interface adaptation process of AUGUR

rent situation or which action she has executed at last) with the adaptation to the device used. However, we state that considering the dynamic user context can enhance the adaptation process. This is also confirmed by our evaluation in Section 8.3.5. Especially for small screen devices, adaptation to the user's dynamic context is beneficial as was shown by Findlater and McGrenere (2008).

8.3.3. Adaptation Process

In this section, we present how AUGUR generates a reduced version of the UI which is adapted to the user's current needs. From observing the user, AUGUR has learned how the user interacts with an application and has stored this information in the usage models. This model is used for determining which elements are most relevant for the user in her current situation. For that purpose, we developed the FxL* algorithm as presented in the next section. It also considers the currently used device by adjusting the amount of presented interaction elements to the available screen size. However, there are also non-interactive elements that should be presented to the user, e.g. the departure times for trains in our train booking example. This information cannot be inferred from the usage model of the application. For that reason, we introduced the *UIContent* nodes in the application modeling language (see Section 6.3) to annotate relevant non-interactive elements. On the one hand, this enables the end-user to influence the adaptation process (R6: +). On the other hand, it introduces additional modeling effort, because 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 feedback of the application 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.

For generating the adapted UI, AUGUR at first determines which non-interactive elements should be contained in that UI based on the application model. Then, AUGUR calculates the most relevant interactive elements for the remaining space using the FxL* algorithm. Further, an interaction element is added which enables the user to trigger the calculation of the next interaction elements if not all relevant interaction elements for a UI can be displayed at the same time. Finally, the automatically generated UI always includes a link back to the non-adapted version of the application. Thus, it does not hamper the normal usage of an application and copes with the uncertainty of predictions based on context information in a limited way (R2: o).

All the computed elements are then presented to the user. Thereby, AUGUR supports to render the UI in different representation languages like HTML or VoiceXML depending on the interaction device used.

Figure 8.11 shows the overall adaptation process applied by AUGUR. The upper part shows the original Web interface of the application, and how it is rendered on mobile devices. Most interaction elements, including the input elements for looking up a train connection, get more difficult to reach. The lower part shows how AUGUR has tailored the UI of this application to the needs of an exemplary user which heavily reduces the interaction costs for searching for a train connection.

8.3.4. FxL*: Prediction Algorithm

In Section 8.1, we presented our sequence prediction algorithm FxL which is able to predict the next element in a sequence. However, for reducing the UI to the most relevant functionality it is not sufficient to predict only the next action, but we need to know the next n actions the user will most probably perform. Thereby, n depends on the available display size and on how much additional information is presented, i.e. how many interaction elements can be displayed. For example, if the user always interacts with an application by performing the actions `a`, `b`, and `c` in that order –abbreviated as `abc`– FxL can only predict that the next action will be `a`. For that reason, we extend the FxL algorithm as described in the following. The (simplified) pseudo-code of the resulting algorithm called **FxL*** is illustrated in Algorithm 8.

We explain its behavior with a simple example (see Figure 8.12): The original UI displays the interaction elements `a`,`b`,`c`, and `d`, and the user only has a very small screen device for displaying $n = 2$ elements. The user trace is `efd` (`e` and `f` are interaction elements of the preceding UI). This example is shown in Figure 8.12. To determine which interaction elements should be displayed to the user, we at first use FxL to compute the most probable next actions $x_1, \ldots, x_n$ for the recent interaction history $a_1 \ldots a_i$. In our example, the most probable next actions are `a` (70%), `c` (20%) and `b` (10%). For every possible next action x_j, we apply FxL again on every sequence $a_1 \ldots a_i x_j$. The resulting probabilities are multiplied by $P(x_j|a_1 \ldots a_i)$, i.e. the probability for x_j succeeding $a_1 \ldots a_i$, as the probability of an action cannot exceed the probability of its preceding action. In our example, we apply FxL for the sequence `efda` which returns `a` with probability 30% and `b` with a probability of 70%. Thus,

Algorithm 8 FxL*

Purpose: Calculates the probabilities for all possible actions in the global variable Q, if n interaction elements can be displayed
The global variable p_{min} thereby denotes the probability of the n^{th} probable action, i.e. the action with the minimal probability that would currently be displayed.

Input: $a_1 ... a_i$ sequence of most recent actions
p parent probability (initialized with 1)
n number of elements to be displayed

procedure FxL* $(a_1 \dots a_i, p, n)$:
  $P(x|a_1 \dots a_i) \Leftarrow (a_1 \dots a_i)$
  for all x **do**
    $q(x) \Leftarrow P(x|a_1 \dots a_i) \cdot p$
    $Q(x) \Leftarrow \mathbf{max}(Q(x), q(x))$
    update p_{min}
    if $q(x) > p_{min}$ **then**
      FxL*$(a_1 \dots a_i x,\ q(x), n)$
    end if
  end for
end FxL*

the probability that action `a` is performed as second activity is $30\% \cdot 70\% = 21\%$ and for action `b` the probability is $70\% \cdot 70\% = 49\%$. 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.

a	b	c	d
70%	10%	20%	0%
70%	**49%**	20%	0%

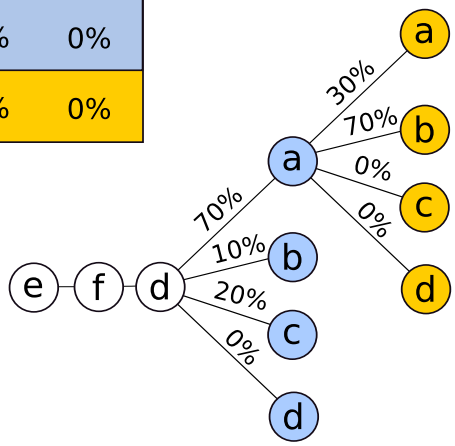


Figure 8.12. Example calculation of FxL* if two elements can be displayed. As result the interaction elements a and b would be presented.

In our example, we do not update the probability of `a` as $21\% < 70\%$, but we update the probability of `b` to 49%. This process is repeated until no action can be found which has a higher probability than the elements that would be currently displayed. In our example, the elements `a` and `b` would be displayed. As no action with a higher probability can be found, the algorithm terminates.

8.3.5. Evaluation

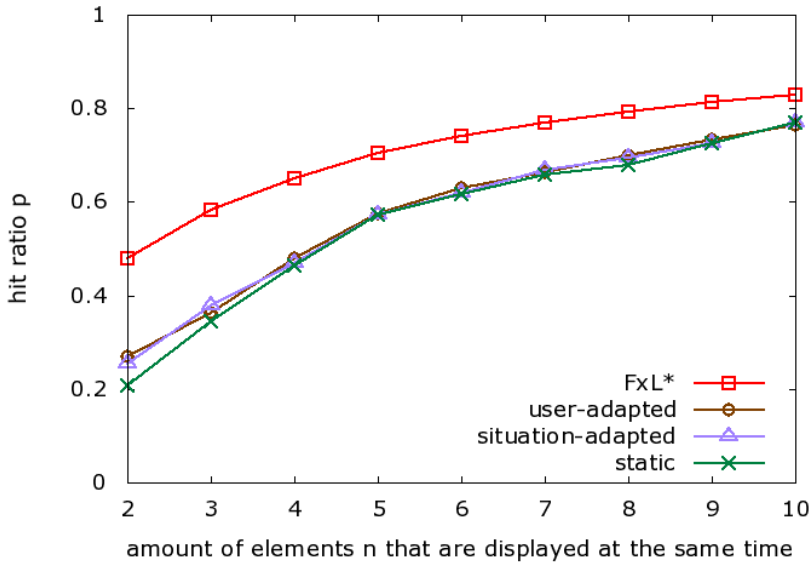
To evaluate our interface adaptation approach, we compared it to other strategies, which do not adapt to individual users or to their current situation. We compared four adaptation strategies:

- The *static* strategy displays the interaction elements which were most frequently used in average over all users. This strategy is optimal for a non user-adapted interface, as it optimizes for the average user for all situations.
- The *user-adapted* strategy takes the individual user into account, displaying the actions the user has used most often. However, the user's current situation reflected by the immediate interaction history is not considered.
- The *situation-adapted* strategy in contrast considers only the interaction history ignoring the individual user. In this strategy, we compute a single usage model for all users, and apply the FxL* algorithm to predict the next actions for each user.
- The *FxL** strategy combines user- and situation-awareness by applying FxL* on the interaction model learned for each individual user.

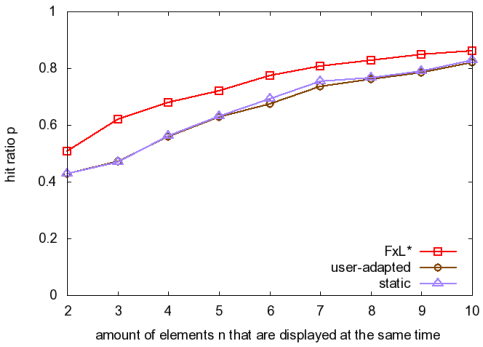
We applied every strategy to the three sets of real usage data that we also used in Section 8.1.6: The Greenberg dataset (Greenberg, 1988) containing UNIX commands, the CrossDesktop dataset (abbreviated XD) with log data from a web application for managing files and emails[28], and the Word dataset containing logs of MS Word usage[29]. Device context was modeled by varying the amount of elements n that can be displayed at the same time (corresponding to different display sizes). As dependent variable, we counted how often the action that was actually performed next (given by the usage log), was found among the elements that were currently displayed according to the adaptation strategy applied. The elements that are displayed were recalculated whenever an action was requested that was not present among the current elements.

All strategies operate incrementally and update their usage model after each observed action. In Figure 8.13, the macro-averaged results over all user traces of the three different datasets are shown[30]. Table 8.4 summarizes the results for $n = 5$.

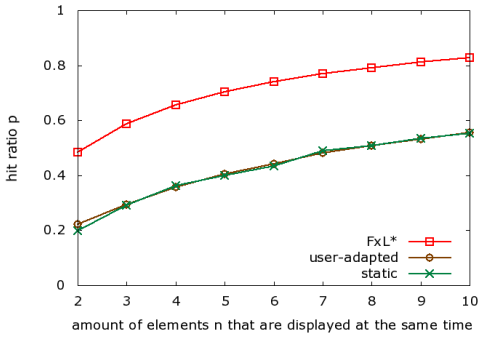
[28] http://www.crossdesktop.com
[29] http://www.cs.rutgers.edu/ml4um/datasets/
[30] We use the macro-average as we do not want to emphasize frequent users.



(a) Word dataset



(b) XD dataset



(c) Greenberg dataset

Figure 8.13. The evaluation results for the (a) Word, (b) XD, and (c) Greenberg dataset. For the ease of readability, we omitted the results for the *situation-adapted* strategy in (b) and (c) as they run very similar to the *static* strategy. The y-axis represents the ratio of how often the next user action was present on the currently displayed adapted UI with the given adaptation strategy.

Adaptation Strategy	Prediction Accuracy [%]		
	Word	XD	Greenberg
FxL*	**70.4**	**72.0**	**70.5**
user-adapted	57.2	62.9	40.4
situation-adapted	57.2	63.0	40.6
static	57.5	63.2	39.8

Table 8.4. Evaluation results for $n = 5$ (best values in bold and blue)

In the Word dataset, the *static* strategy performs only slightly worse than the *user-adapted* and *situation-adapted* strategy. In the other two datasets, we could not even detect a difference between the *user-adapted* and *situation-adapted* strategy. This indicates that the frequently used actions for the given datasets are the same for most users, but that there are no global usage patterns which are valid for all users. For the ease of readability, we therefore omitted the results for the *situation-adapted* strategy in the graphs. The user- and situation-adapted *FxL** strategy clearly outperforms the three other strategies (the difference in the hit ratio ranges for $n \in [2, 10]$ from 6.0% to 27.1% for the Word dataset, from 25.3% to 30.2% for the Greenberg dataset, and from 3.2% to 15.2% for the XD data). The hit ratio for the *FxL** strategy ranges from about 47% to 86% for all datasets (Word: 48% to 83%, Greenberg: 47% to 83% and XD: 50% to 86%). This shows that it is important to take the user and her current situation into account in order to generate an adapted UI.

The actual benefit of user-adapted UIs over its unadapted counterparts can be estimated by comparing the interaction costs for the two UIs. The interaction costs comprise the amount of clicks, navigational movements, and keys that need to be pressed in order to interact with an element. If the action the user wants to perform next, is part of the currently displayed UI in p percent of all cases, the user has to switch to the unadapted UI in $1 - p$ percent of all cases. We define that the average costs for using one of the n elements in the adapted UI and for switching to the unadapted UI are c_a. We further define that the average costs for selecting an action in the unadapted version are c_u. The values of c_u and c_a depend on the individual user. The benefit b of using the adapted UI can be calculated as difference between the average costs for interacting with the unadapted UI (c_u) and the average costs for the adapted UI. Thereby, the average costs for interacting with the adapted UI is c_a in p percent of all cases, and $c_a + c_u$ in all other cases, i.e. c_a for selecting the link to the unadapted version and c_u for performing the operation in the unadapted UI. Thus, the benefit of using an adapted UI is defined as

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

This means that using an adapted UI is beneficial (i.e. $b > 0$) if $c_a/c_u < p$. For example the adaptation with FxL* in the Word dataset is beneficial for displaying four elements, if $c_a/c_u < 0.62$, i.e. if $c_u > 1.61 \cdot c_a$. This value can easily be reached,

as the average interaction costs c_a for interacting with one of four elements is much lower than the average costs c_u for interacting with one of more than 100 elements (or even 1000 for the Greenberg dataset).

8.3.6. Summary

In this section, we presented a novel approach to generate UIs that are adapted to individual users and their current situation. In the evaluation, we proved our claim that it is beneficial to adapt to the user and her current situation to provide the best UI. As our approach also takes the device used into account, it considers the user context as well as the environmental context (R1: +). To our knowledge, it is the first approach based on adaptation rules which considers all of these factors. AUGUR accounts for the unreliability of context information used in the adaptation process in a limited way: AUGUR enables the user to switch back to the unadapted version at all times (R2: o). The proxy-based architecture of AUGUR enables it to generate adapted UIs for all kinds of existing form-based applications (R3: +, R5: +). The end-user can influence which elements are presented in the UI by stating this information in the corresponding application model (R6: +). Finally, AUGUR learns which elements are most relevant for the user from observing her interactions (R7: +). Table 8.5 summarizes the requirements met by AUGUR.

However, the presented approach is just a first step towards generating reduced UIs which are adapted to the user's current situation. We only adapt which elements should be presented to the user and not how they should be represented. For example, a voice user interface poses different requirements on the representation than a graphical user interface. To provide an ideal UI, these requirements have to be taken into account and the representation of the UI has to be adapted accordingly.

8.4. Chapter Summary

In this chapter, we presented algorithms for generating navigation support based on context information. We introduced a novel algorithm for guiding the user through an application and showed that it outperforms state-of-the-art approaches with respect to accuracy and computational costs. Further, we illustrated how context information can be used for triggering navigation shortcuts. Finally, we described a novel approach for generating a reduced UI containing only the most important elements. In contrast to existing approaches, it is thereby able to consider the user's current situation as well as the device used.

		Awareness of user context* (R1)	Awareness of environmental context (R1)	Cope with error-prone context (R2)	Application-independence (R3)	Support for legacy applications (R5)	Involving end-user (R6)	Learning capabilities (R7)
Platform independent languages	*MDA*		●		●			
	SUPPLE	●	●		●			●
Adaptation rules	Highlight				●	●	●	
	Palio		●		●	●		
	MICA	●		○				●
	UIDE	●		○				●
	Findlater	●		○				●
	Smyth	●			●	●		●
	AUGUR	●	●	○	●	●	●	●

Table 8.5. Comparison of AUGUR with state-of-the-art approaches for interface adaptation with respect to the identified requirements (* using static user profiles is not considered as user context, ○: limited support, *MDA* subsumes several systems with same characteristics)

9 User Study

In this chapter, we describe the setup and the results of a user study that we performed for evaluating the impact of context-aware interaction support as presented in this thesis on the usability of an application. For that purpose, we gathered data for measuring all three aspects of usability as defined in ISO 9241 (2000): the user's efficiency, effectiveness, and user satisfaction. We put a special focus on whether erroneous support has adverse effects on these factors, as context information is usually very error-prone.

In Section 9.1, we describe the experimental setup, and in Section 9.2 the gained results. In Section 9.3, we summarize the results, and draw conclusions for the further development of context-aware Intelligent User Interfaces.

9.1. Experiment

For evaluating the effects of context-aware interaction support, we use three settings:

- **no support**: no context-aware interaction support - the baseline UI
- **correct**: correct interaction support is provided by the UI
- **erroneous**: erroneous interaction support is provided by the UI to reflect the worst case of using error-prone context information

The participants of the study were asked to perform the same task twice with different settings, so we performed both, in-between and within subject testing of different settings. Thereby, one setting was always the *no support* setting, because we wanted to avoid that the users build up a mental model of the support (i.e. all suggestions are correct/wrong) which is not valid for the second try.

In the user study, we focused on the *Guidance* and *Content Support* interaction support types. Thereby, *Guidance* was only evaluated in form of highlighting interaction elements and not in the form of automatically following links. *Content Support* was investigated with both possible representations: as suggestions and as automatically filling data.

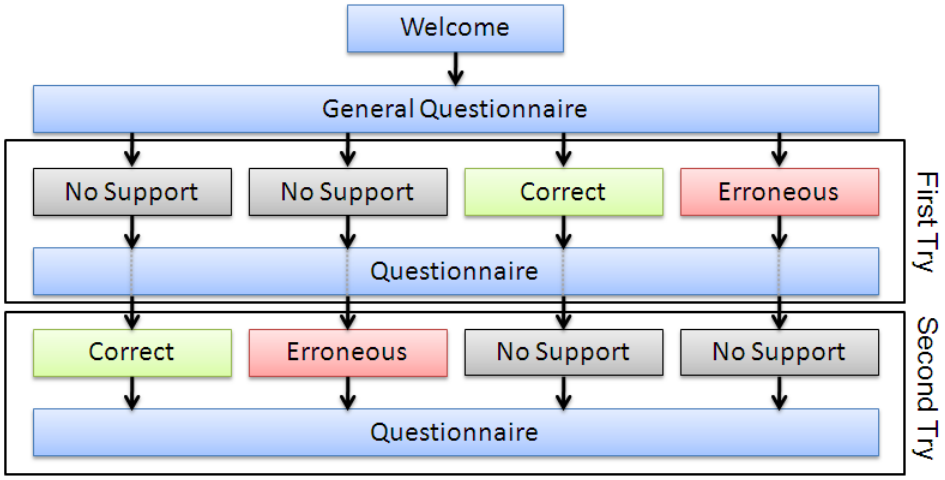


Figure 9.1. Procedure of the user study

We chose the example task of booking a train ticket for a certain day and time. We wanted to make sure that the users were unfamiliar with the specific look of the UI. For that reason, we chose to create our own mock-up of a ticket booking application (see Figures 9.3 and 9.6), instead of using an existing application. To reduce the influence of other factors beside the interaction support itself, we used a simplified version of AUGUR: It was not controllable by the user, did not learn, and did not display the knowledge provenance or any hints on the confidence in the interaction support.

Procedure The whole study was performed in German. Figure 9.1 provides an overview of the procedure of the experiment. At first, a short welcome page explains the task. The participants also had to answer some initial questions about general internet experience and experience with online ticket booking tasks. Next, the user had to complete the ticket booking task twice with different settings. To avoid distortion of the results for the second try, we used the HTML directive `autocomplete off` in all form fields, disabling the browsers built-in filling function. After completing the task, the users were asked to fill out a small questionnaire asking them to rate how helpful and how disturbing they perceived the UI on a Likert scale from 0 to 5. They were also asked to rate if they would like to work with such a UI again, and how disturbing they think such a UI would be in the long run. The questionnaire after performing the task in a try with *no support* did not contain the questions about the disruptiveness and helpfulness of the UI, because it confused participants in the pre-study. They found these questions difficult to answer if no additional support was provided. The German questions used in the questionnaires can be found in Appendix A.2.

Figure 9.2. Reminder for the task in German. English translation: Your task, from: Darmstadt, to: Frankfurt, date: 26.07.2007, departure: 18:30 or later, Bahn Card: BC50 2nd class, BC number: 912837465, Ticket: 1 grown-up 2nd class

Task The user had to enter data into three web forms for booking a train ticket. The data which is needed for successful completion of the task was displayed next to the form as an image (see Figure 9.2) to prevent the users from using copy and paste. The first form required to enter the place of departure and the destination into two text entry fields, to choose the year, month, day and time of travel, and whether to treat this time as arrival or departure time into six drop down boxes (see

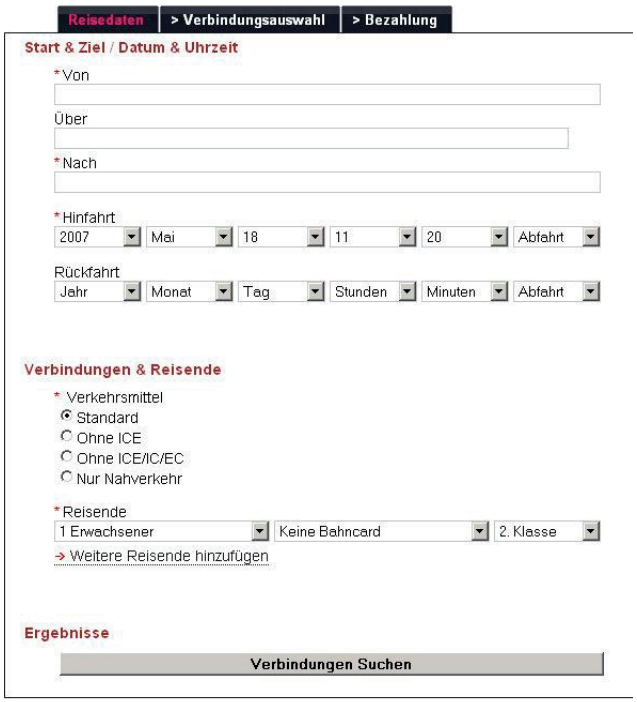


Figure 9.3. First form of the task

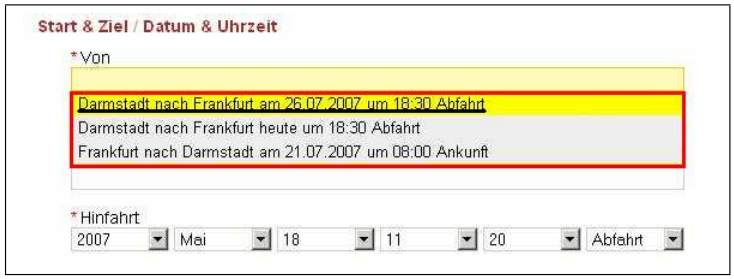


Figure 9.4. Suggesting values for several input fields at once (A1: suggested content support)

Figure 9.3). In the middle of the form, the user could select her means of travel, but no change was required there for completing the task. Finally, the user had to choose a specific discount option from another drop down menu. In the *correct* and *erroneous* setting the UI provided the following interaction support:

A1 *Suggest content support:* Suggest values for the upper part of the form as shown in Figure 9.4. In the *erroneous* setting, the suggested values did not completely fit to the task set for the user.

A2 *Guidance*: Highlight the drop down element for the BahnCard (frequent traveler card for the Deutsche Bahn) shown in Figure 9.5 as next action in the *correct* setting.

A3 *Guidance*: Highlight the button for proceeding to the next step. In the *erroneous* setting, another button was highlighted.

In the next form, the user was asked to select the correct train connection. In the *correct* and *erroneous* setting, the text passage "no online booking" was highlighted, when the user tried to click on a connection that was marked as "no online booking" (the first, second, and forth presented connection). We used this to investigate whether additional help could facilitate the usage. However, we did not further head in this direction (and also did not find any significant results). For that reason, we will not go into details for this part.

In the third form (see Figure 9.6), the user had to enter her BahnCard number. Thereby, she was supported in the *correct* and *erroneous* setting by the following interaction support:

Figure 9.5. Example of facilitating the interaction by highlighting the next step (A2: guidance)

Figure 9.6. Third form of the task (A6: automated content support)

A4 *Automated Content Support*: Automatically fill the BahnCard number and highlight the field, whereby the number filled in the *erroneous* setting was incorrect.

9.1.1. Technical Setup

The train booking application was realized as Web application. The context-aware interaction support as well as the logging was integrated via JavaScript files as described in Section 5.3. The logfile stored accurate timing data, when the interaction with an interaction element started. The data from the questionnaires was processed and stored in a database by a script on the web server. The whole study was controlled by a single script that randomly assigned settings to users. The participants could perform the study remotely, they just needed to direct their browser to a short URL which we sent to them. More details about the setup of the user study can be found in (Schreiber et al., 2008).

In total, we collected data from 40 participants. Most of them were computer science students or faculty members, but also persons without computer science background participated.

9.2. Results

We now present the results of our evaluation study aimed at assessing the impact of context-aware interaction support on the usability of a UI. Before subjecting the data to statistical analysis, we took care in filtering outliers and strange data. For example, two participants used a version of the Safari web browser, in which the injected JavaScript files did not work as expected. For that reason, we excluded their data for the *correct* and *erroneous* setting.

For analysing the data, we used One-way ANOVA with the **measured times for completing the action** and **error rates** as dependent variables, and **try** and **setting** as factors. We verified the homogeneity of variances using the Levene test and used the Welch test instead of normal ANOVA in cases where we could not rely

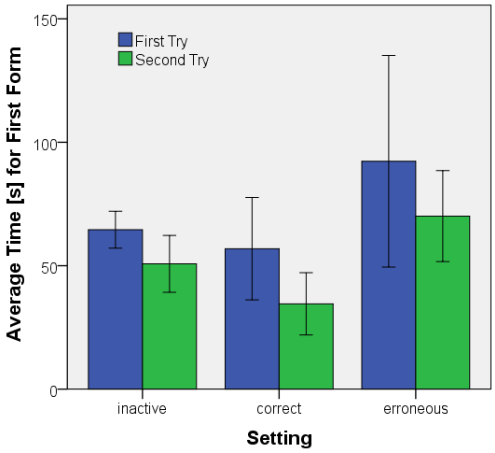


Figure 9.7. Learning effect for the different settings for the first form (with 95% confidence interval)

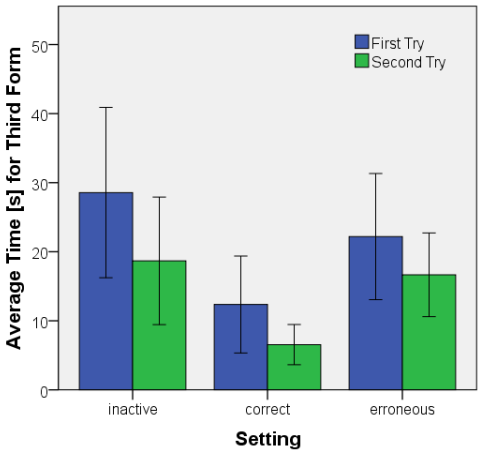


Figure 9.8. Learning effect for the different settings for the third form (corresponding to A4) (with 95% confidence interval)

on homogeneity of variances. We report significance at a level of 0.05 as commonly used for usability studies.

In the following, we describe the obtained results regarding the three aspects of usability: efficiency, effectiveness, and satisfaction.

9.2.1. Efficiency

To measure the efficiency of the UI, we analyzed the measured total time for all three forms of the study as well as for the subparts of the first form corresponding to the different actions. An overview of the collected timing data is shown in Table 9.1. We did not include the data for A2 as it was difficult to distinguish between the end of A1 and the beginning of A2 based on the HTML events, and thus did not provide reliable results.

The effects of try and setting on the time needed for completing the first and third form of the task can be seen in Figures 9.7 and 9.8, respectively. It shows that there is a significant difference between the first and second try for each form, as the users learned how to interact with the application. For that reason, we present in the following only the results of a single try (the second one). However, the overall results and the results for the first try yielded comparable results if not stated otherwise.

Suggested Content Support

For *correct* suggestions of values in action A1, a significant benefit ($F(2, 51) = 4.441$, $p < .05$) compared to *no support* could be observed as shown in Figure 9.9. This is the case although the participants were not familiar with context-aware interaction

Setting	Try	1st Form Total	σ	A1	σ	A3	σ	3rd Form Total / A4	σ
no support	1	64.6	15.5	47.5	14.1	8.1	5.4	28.6	25.6
	2	50.7	24.6	40.6	15.6	**3.2**	2.2	18.7	19.7
correct	1	**56.9**	24.8	**29.6**	16.4	**7.6**	7.0	**12.4**	8.4
	2	**34.6**	20.8	**20.2**	18.3	4.5	2.8	**6.5**	4.8
erroneous	1	92.3	74.2	56.7	25.0	9.0	6.2	22.2	15.1
	2	70.1	19.9	58.6	25.1	7.4	2.3	16.7	6.6

Table 9.1. Recorded timing data. All values in seconds (best values in bold and blue)

support. As expected, the time for completing this activity with a *correct* suggestion is shorter than the time in the *no support* setting, while the time in case of a *erroneous* suggestion is slightly increased.

Guidance

The times required for the action related to the guidance action A3 are very small (see Figure 9.10), vary heavily, and are not consistent over the first and second try (see Table 9.1). For those reasons, we cannot draw any meaningful conclusions from the obtained results, and cannot note any significant differences between the three settings. We assume that the highlighting was not sufficiently attracting the attention of the users.

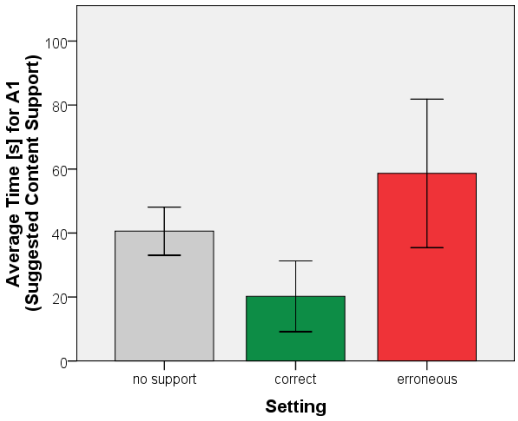


Figure 9.9. Average time for the activity related to *Suggested Content Support* (A1) with 95% confidence intervals

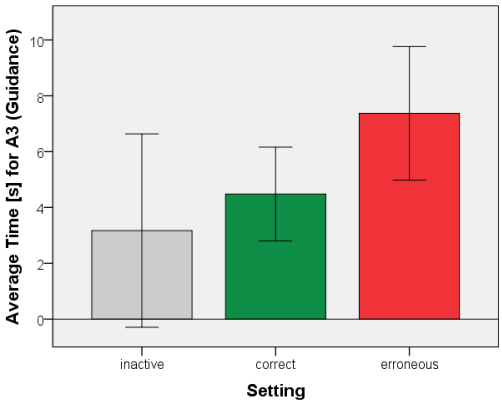


Figure 9.10. Average time for completing the activity related to *Guidance* (A3) with 95% confidence intervals

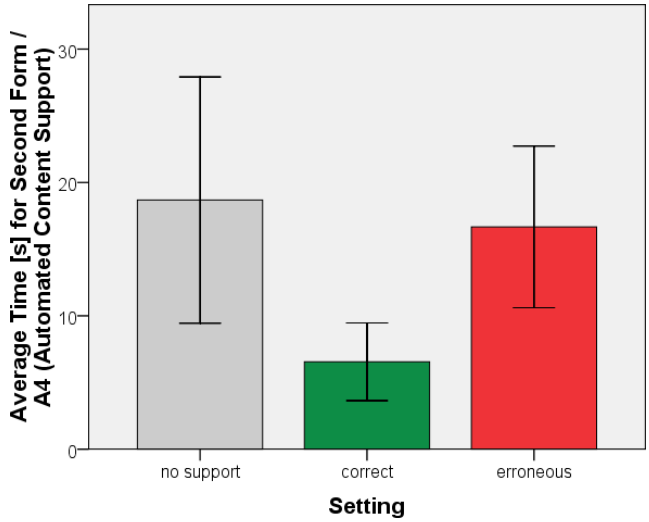


Figure 9.11. Average time for completing the activity related to *Automated Content Support* (A4) with 95% confidence intervals.

Automated Content Support

In the third form, the "BahnCard number" field was filled with the correct number in the *correct* setting and with a wrong number in the *erroneous* setting. Figure 9.11 shows that filling in a wrong number does not seem to have a negative effect on the efficiency. A possible explanation is that automatically filling data in interaction elements leads to highlighting the element, thus provides guidance support. Additionally, the inserted number serves as an example of which kind of data needs to be entered.

9.2.2. Effectiveness

The effectiveness was measured as the amount of errors committed in the different settings. In the erroneous setting, a wrong decision in A1 could easily lead to a higher number of errors (3 to 8), as multiple input elements are filled at once. An overview of the distribution of errors (again only for the second try) can be found in Figure 9.12. It shows the relative amount of participants with the according number of errors for each setting. On average there were only few errors at each form, so no statistical significance for the influence of the setting or the try on the number of errors was found. This might indicate that there is no significant malus in effectiveness even for erroneous interaction support.

9.2.3. Satisfaction

The user satisfaction was evaluated with the questionnaires. The results can be seen in Figures 9.13 and 9.14. The participants were asked to rate the support of the UI

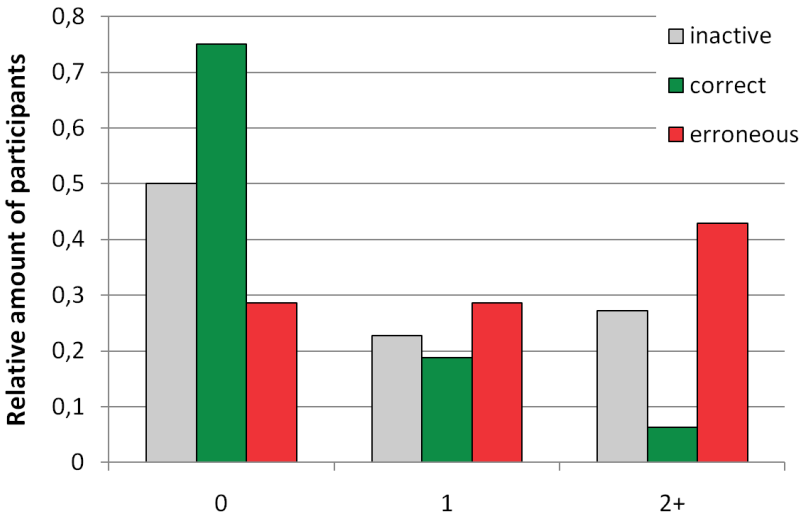


Figure 9.12. Distribution of the amount of errors for each setting

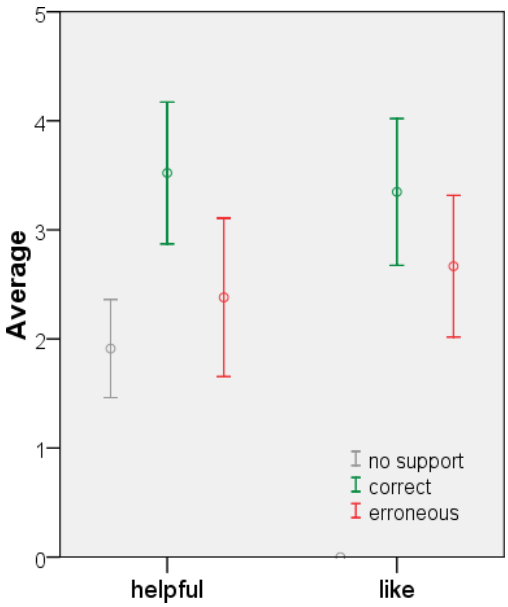


Figure 9.13. User ratings on a 0-5 Likert scale (best: 5) whether the users perceived the UI as *helpful* and whether they would *like* to see more UIs with interaction support

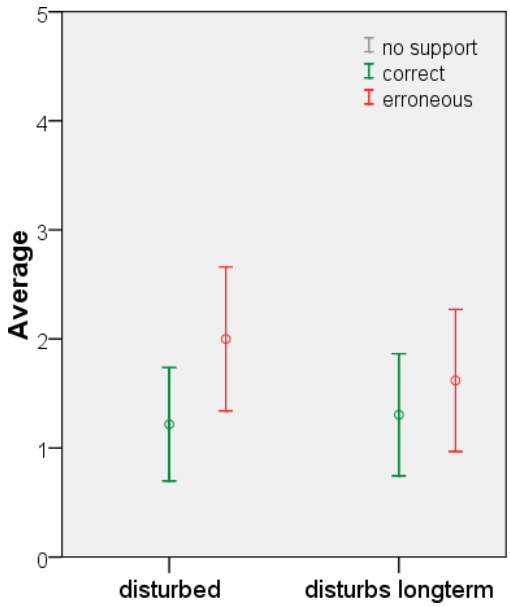


Figure 9.14. User ratings on a 0-5 Likert scale (best: 0) whether the users perceived the interaction support as disruptive (*disturbed*) and whether they think they would perceive it as disruptive on the long run (*disturbs longterm)*

("How much did the system support you in fulfilling your task?") on a Likert scale from 0 to 5, whereby 0 means "not at all" and 5 "very helpful". We found that the UI which provided *correct* context-aware interaction support is perceived as more helpful than the *no support* UI with statistical significance ($t(54) = 4.8, p < 0.05$). Our data also indicates that the interaction support is even perceived as more helpful when *erroneous* data is used ($t(40) = 2.34, p < 0.05$).

The question, "Would you like to see more applications with this kind of assistance?" was again answered on a Likert scale from 0 "not at all" to 5 "very much", the results for *correct* ($n = 23, M = 3.35, SD = 1.56$) and *erroneous* support ($n = 21, M = 2.67, SD = 1.43$) show that interaction support is perceived positively by the users. Here again, the rating of the users in the *erroneous* setting was not significantly lower than in the *correct* setting. This shows that users are not irritated or frustrated by wrong suggestions in the way we provided them.

The next question asked was "How disturbing did you find the assistance of the application?" (0 meaning "not at all disturbing" and 5 "very disturbing"). This question was only answered by the participants in the *correct* and *erroneous* setting, as participants of the pre-study in the *no support* setting were confused by this question as they apparently did not receive any assistance. Participants felt only mildly disturbed for both *correct* ($n = 23, M = 1.22, SD = 1,20$) and *erroneous* ($n = 21, M = 2.00, SD = 1.45$) support. Our next question targeted the users' judgment whether the assistance would disturb in the long run. The scale ranged from "not at all disturbing" to "very disturbing" like for the previous question. The data shows that users rated the disruptive effect as rather low, even for the erroneous support (for *correct* support: $n = 23, M = 1.30, SD = 2.30$, for *erroneous* support: $n = 21, M = 1.62, SD = 1.43$).

9.3. Chapter Summary

In this chapter, we reported the results of a user study that we performed for evaluating the effects of context-aware interaction support. We found significant benefits in efficiency for correct interaction support. We could show that the efficiency and the effectiveness of the users stayed stable or did not decrease significantly, when erroneous interaction support was provided. Thus, carefully planned context-aware interaction support is indeed beneficial, even though it may be erroneous sometimes.

10 Conclusion

The costs for interacting with modern applications are steadily increasing, due to (i) the increase in the amount of provided functionality, which often leads to a decrease in the usability of the application, and (ii) the increasing usage in mobile and ubiquitous settings where we have to deal with higher interaction costs and reduced attention of the user. This problem can be tackled by intelligent interaction support, e.g. by guiding the user through an application, adapting the user interface, or facilitating the input of data. In order to get a better understanding of the user's needs and thus to be able to support the user in an optimal way, we have to take her context into account. The relevant context can be gathered from sophisticated user models (user context) as well as from the environment (environmental context). However, existing approaches only make use of either the user context or the environmental context. In this thesis, we presented a novel approach (called AUGUR) which considers the user as well as the environmental context to support the user's interactions. AUGUR is also able to provide content support and navigation support for different applications.

In this chapter, we first summarize the main contributions of this thesis, and how it fulfills the requirements identified for context-aware IUIs (Section 10.1). In Section 10.2, we revisit the challenges identified for developing IUIs in general, and point out how they are addressed by our approach. In Section 10.3, we discuss issues for future research.

10.1. Contributions

Overall concept The overall concept of AUGUR is able to consider user as well as environmental context for supporting the user in entering data and in navigating. For that purpose, it incorporates a context server for providing environmental context information, a user model for modeling the user's behavior, and a component for storing the *Current Context,* which keeps track of all context information that is currently relevant for the user. To determine the relevance of the context information, the *Current Context* mimics human memory based on the cognitive architecture ACT-R.

However, when using context information for providing support, we have to con-

sider its error-prone nature. For that purpose, AUGUR provides different levels of proactive support which differ in their obtrusiveness and in how much they reduce the required interaction costs. For example, automatically filling data is perceived as very obtrusive by most users, but can dramatically reduce the required interaction costs. In choosing the appropriate level of proactivity, AUGUR takes the user's preferences as well as the confidence in the provided support into account.

AUGUR relies on application models for providing support. The application models store the relations between context elements and application elements. They are used for determining which context information can be suggested as input for an interaction element and which context events should trigger navigation shortcuts to other applications. These models can be defined by the application developer, and augmented by the end-user. AUGUR is able to automatically extend the model with new relations learned from observing the user's interactions. Thus, AUGUR combines the three main approaches for building Intelligent User Interfaces (*knowledge-based*, *end-user programmed*, and *learning* Intelligent User Interfaces). This enables AUGUR to adapt its support to the user's needs (either by explicit advice or by learning from the user's interactions). At the same time, AUGUR is still able to benefit from rich application models provided by an application developer. The support improves over time, as more relations between context elements and application elements are learned.

The approach taken by AUGUR is application-independent. It uses a proxy-based architecture which enables it to keep track of all user events, because they are routed through AUGUR. The proxy-based architecture also allows AUGUR to augment the UI of the application to provide interaction support for all kinds of (form-based) applications and across application boundaries.

We evaluated the effects of context-aware interaction support in a user study. The results show that correct interaction support can significantly increase the usability of an application. As context information is often error-prone, we also analyzed the results of erroneous support on the usability. We found that erroneous support has little or no adverse effect on the usability.

ApplicaTion Modeling Language (ATML) For modeling the relations between context elements and application elements, we developed a new application modeling language called ATML. It overcomes the shortcomings of existing modeling languages that are not able to model relations to context information or to existing UI elements to support legacy applications. ATML uses rules to specify the relations between context elements and application elements. In the AUGUR prototype, we implemented an editor for the application models which enables the end-user to easily add new relations, or inspect and change relations that were automatically learned by AUGUR from observing the user.

Semantic Mapping In order to be able to support the user in entering data, we developed a semantic mapper which compares the context information present in the

Current Context to the information required by the UI. Based on semantic similarity measures it can determine which context information is relevant for the current UI and suggest it as input to the user. In this way, the mapping process does not require any premodeled information. The comparison relies on computing the similarity between the textual descriptions of UI elements (e.g. their labels or tooltips) and context information. We showed that considering all information available about the elements provides better results than relying only on the labels of interaction elements, as used by most existing approaches. Furthermore, we showed that it is beneficial to combine string-based and semantic similarity measures, and to learn new synonyms from observing the user's interactions.

As the labels of UI elements cannot easily be determined for Web applications, we developed and implemented an algorithm (called *LabelFinder*) that determines the label of an element from the visual representation of the UI. We showed that it outperforms state-of-the-art approaches.

FxL algorithm We developed the FxL sequence prediction algorithm to predict the next relevant interaction element. This enables AUGUR to guide the user through an application. We showed that the FxL algorithm outperforms state-of-the-art sequence prediction algorithms with respect to accuracy and required resource consumption.

Interface Adaptation using FxL* algorithm The FxL* algorithm is able to predict the set of most relevant interaction elements. It is used to generate a reduced version of a UI, which is adapted to the user's current situation and to the output device used. For determining the set of relevant elements, the FxL* algorithm repeatedly applies the FxL algorithm, which only predicts the next step. We showed that considering the user's current situation, i.e. her last actions, can dramatically reduce the interaction costs for the user.

AUGUR prototype We implemented AUGUR in a working prototype. The prototype is able to provide support for all kinds of form-based Web applications. It supports the user in entering data and in navigating through the application. AUGUR is thereby also able to cope with highly dynamic Web applications using AJAX.

To sum up, the approach presented in this thesis is able to consider both types of context: User context and environmental context (R1 *Awareness of user context and environmental context*). It is also able to adapt the provided support to the confidence of the context information (R2 *Cope with error-prone context*). Our approach is applicable to various (form-based) applications (R3 *Application-independence*, R5 *Support for legacy applications*) and across application boundaries (R4 *Support across application boundaries*). In addition, AUGUR enables the end-user to augment and inspect the models used for providing support (R6 *Involving end-user*). It is also able to enhance the provided support by learning from the user's interactions (R7 *Learning*

capabilities). Thus, AUGUR meets all the requirements identified for context-aware IUIs as summarized in Table 10.1.

10.2. Revisiting Challenges

In this section, we review the main challenges which need to be addressed for developing IUIs as identified in Section 2.3.1. This comprises the *presentation* and *competence* of the provided support, and how to induce user *trust* in an IUI. We below point out how the different challenges are targeted by the approach presented in this thesis.

Presentation The first challenge for the presentation is the *interaction design* of the support: It should (i) not hamper the normal usage of the applications, (ii) provide some kind of forgiveness, and (iii) fail softly. In AUGUR, all these three requirements are met: AUGUR is designed as an overlay to existing applications without affecting the actual functionality of the application, thus not hindering the usage of the application. When AUGUR generates a reduced UI for an application, it enables the user to easily access the full functionality, and thus still supports the normal usage. The interaction support is furthermore designed in a way that can be easily undone if it is wrong. If AUGUR provides erroneous suggestions, they can easily be ignored. Finally, partly erroneous content support can also move the user closer to her goal as it allows at least parts of the required data to be filled in.

The second challenge is *unobtrusiveness*. The interaction support in AUGUR is designed in a way that does not disrupt the workflow of the user. For providing navigation support it however has to make the user aware of the suggested link or interaction element. For that reason, the provided support briefly flashes to make the user aware of it, but that can easily be ignored. The user can also adjust the proactive behavior of AUGUR by defining thresholds for the different levels of proactivity.

The third main challenge is to make the support *adaptive*: Adaptive to the user, the device, and the current context. For that reason, AUGUR maintains a usage model for predicting the user's next actions and to generate reduced UIs that best fit the user's needs. Furthermore, the user can adjust the application models to her needs, and AUGUR also augments these models with data learned from observing the user. AUGUR also considers the currently used device when generating a reduced UI by adapting the amount of displayed information and interaction elements to the available screen size. Finally, AUGUR is aware of the current user and environmental context and adapts the provided support (i.e. the navigation shortcuts and the content support) accordingly.

Competence Regarding the competence of an IUI, we have to face two main challenges: little usage data and changing user behavior. The algorithms we presented for the different support types are designed to cope with *little usage data*; the content support based on semantic comparison even needs no usage data at all. The

		Content support	Navigation shortcuts	Guidance	Awareness of user context (R1)	Awareness of environmental context (R1)	Cope with error-prone context (R2)	Application-independence (R3)	Support across application boundaries (R4)	Support for legacy applications (R5)	Involving end-user (R6)	Learning capabilities (R7)
Knowledge-based	COLLAGEN	•		•				•				○
	AGUSINA	•		•	•			•		•		•
	CyberDesk		•			•		•	•	•		
	onCue		•					•	•	•		
	Miro		•					•	•	•	•	
	ActiveBadge					•	•					
End-user programming	Citrine	•						•		•	•	
	Context Rules					•				•		
Learning	CAP	•			•						•	•
	LookOut	•			•		•					•
	CMRadar	•			•							•
	PTIME	•			•						•	•
	Maxims	•			•		•				•	•
	CAIA	•			•							•
	Chusho	•						•		•	•	•
	VIO	•						•				•
	Folder Predictor		•		•							•
AUGUR		•	•	•	•	•	•	•	•	•	•	•

Table 10.1. Comparison of AUGUR with state-of-the-art approaches (○: is able to learn task models from usage sequences, *Context Rules* subsumes several systems with same characteristics)

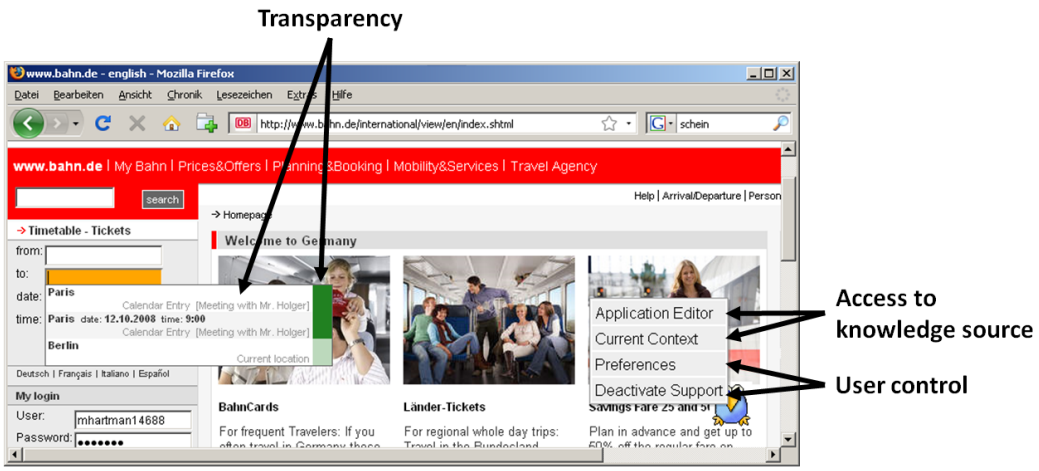


Figure 10.1. Realization of trust issues in the UI design of the AUGUR prototype

algorithms are also able to adapt its support to *changing user behavior*, i.e. they constantly adapt to the user's current behavior. The guidance feature further uses aging to reduce the influence of older usage traces. Furthermore, the user is always able to adapt the behavior to her needs using the integrated ATML editor.

Trust To induce user's trust in an IUI, its behavior has to be controllable, intelligible, and to maintain the user's privacy. Figure 10.1 shows how some of these challenges are reflected in the UI design of the AUGUR prototype.

For enabling the user to *control* the behavior of the IUI, AUGUR allows the user to override or ignore erroneous behavior, i.e. erroneous highlighting or suggestions can be simply ignored and the user can override erroneous data which was automatically filled in by AUGUR. Moreover, the user can influence the interaction support provided by AUGUR. She does this by modifying the underlying application models or altering the proactive behavior of AUGUR by adjusting the confidence thresholds accordingly.

In order to make an IUI *intelligible* to the user, its actions should be transparent and the user should be able to access the knowledge sources to gain a better mental model of the IUI. AUGUR realizes this by providing feedback about its confidence in the different support actions and about the knowledge provenance of the support (see Figure 10.1). The user can always access and inspect the underlying application models and the data in her current context using the integrated editors. This should enable the user to build a valid mental model of the IUI, which then results in appropriate expectations of the support which can be provided by AUGUR.

Finally, *privacy* is not an issue for AUGUR, as AUGUR does not exchange information with other IUIs. Furthermore, it can be run on the personal computer of a user, thus giving the user full control of her data.

10.3. Outlook

In this section, we point out in which ways AUGUR could be enhanced and suggest some future research directions.

Enhancements to AUGUR AUGUR is just the first step for using context information for facilitating data entry and navigation. There are many ways in which the provided support could be enhanced. For example, AUGUR needs to be aware of which information presented by a UI is relevant for the user, e.g. the departure times of trains, to include it in the reduced version of the UI (see Section 6.3). However, these non-interactive elements cannot be easily inferred from observing her interactions as the user is usually only reading the information without interacting with it. For that purpose, AUGUR requires that the relevant non-interactive elements are modeled in the corresponding application model. Automatically identifying these elements could decrease the required modeling effort. This could be realized for example by observing the user's gaze or by inferring this information from the scroll and mouse movements of the user.

Furthermore, AUGUR only supports the usage of simple rules for describing the relations between context and application elements. This could be extended by supporting more complex relations, e.g. using variables, mathematical or string operators. This would enable us to state relations to provide content support which suggests `firstName+" "+lastName` for an interaction element.

The application models could also be automatically enhanced by identifying similar applications and thus transferring the knowledge from one application model to another. For example, the application model for Deutsche Bahn contains relations from a calendar entry to the "to", "date", and "time" input elements. If the user is interacting with a similar application that contains the same input elements, AUGUR could infer that the calendar entry might also be relevant for this application and suggest it as input to the user.

AUGUR could also make use of the application models of other users. Relations contained in the application models of many users could be recommended to the user or automatically incorporated. Exchanging the data with other users, can also be applied for enhancing the guidance and interface adaptation. This is especially useful if the user has not frequently used the application, which means that AUGUR has only a limited and unreliable usage model of her behavior.

At the moment, AUGUR does not consider the goals and intentions of a user. If AUGUR is able to automatically recognize the user's intention and the application models state which goals they can fulfill, AUGUR could apply planning algorithms to guide the user to meet the desired goal.

Using context for other types of Intelligent User Interfaces In this thesis, we focused on using context information for facilitating data entry and navigating. However, context information can also be applied to all the other aspects of Intelligent User Interfaces. Help texts can be automatically adapted to the current context, for

example "if you want to use the printer to your right, you have to choose Epson220". This facilitates the understanding of the help texts by the user. Furthermore, it allows the help texts to be reduced to the parts really relevant for the user's current context. For example, cooking instructions could highlight the step which is relevant for the current stage of the cooking process.

Programming by demonstration systems can also benefit from incorporating the user's current context. It can be used to better understand the user's actions and to generalize from observations. For example, if the user enters her current location as "Darmstadt", a context-aware programming by demonstration system could be able to provide a generalized macro that always uses her current location as input.

Intelligent user interfaces for facilitating the information retrieval can also benefit from context information. *Recommender systems* can take the current context into account to adapt the recommendations. For example, if the user is searching for a movie and many friends are present, a recommender system might not want to suggest a romantic movie.

Ubiquitous use of Intelligent User Interfaces Most Intelligent User Interfaces for providing content and navigation support focus on desktop applications. However, providing proactive support in ubiquitous settings raises many new challenges: How should the support be represented in these settings? For example, if voice output is used, automatically suggesting a value is probably preferable to reading a lengthy list of suggestions to the user. Another example is the interaction with smart products, i.e. physical products that are able to communicate with the user but usually provide only very limited input and output capabilities. How should they proactively approach the user to guide her through a task? How can the user provide feedback in the most natural way? For those reasons, the provided support needs to be adapted to the available input and output capabilities.

Furthermore, the amount of available context information in ubiquitous settings is often much higher than in desktop settings. This leads to the problem of identifying the relevant context information. To decrease the amount of context information that needs to be considered, we could use a common context ontology that states relations between different context elements and between context elements and tasks. However, this induces a high modeling effort. For automatically gathering a context ontology, we could rely on semantic analysis of texts describing tasks. If some context information is often mentioned in conjunction with a task or other context information, they might be related to each other.

Another problem that arises in ubiquitous computing settings is that users are often involved in several tasks at the same time. For that reason, they often do not pay full attention to the Intelligent User Interface. The proactive support should thus also consider the interruptibility of a user. For example, it could defer proactive suggestions until the user turns her attention back to the Intelligent User Interface. Which level of proactivity is chosen should also consider the interaction costs of the user. For example, if the user interacts with a mobile phone for entering data, she

might prefer that the data is automatically entered even though it is more obtrusive than only suggesting data.

A Appendix A

A.1. DTD for ATML

```
<!ELEMENT atmlModel (states | activities | wrappingNodes |
                                                    relations)*>
<!ATTLIST atmlModel name CDATA #IMPLIED>
<!ELEMENT states (state)*>
<!ELEMENT state (#PCDATA | rule)*>
<!ATTLIST state
    id      ID      #REQUIRED
    ref     CDATA   #REQUIRED
    name    CDATA   #IMPLIED
>
<!ELEMENT activities (activity)*>
<!ELEMENT activity (#PCDATA)>
<!ATTLIST activity
    id          ID              #REQUIRED
    ref         CDATA           #REQUIRED
    label       CDATA           #IMPLIED
    automate (true|false) #IMPLIED
>
<!ELEMENT wrappingNodes (#PCDATA | context | uiContent)*>
<!ELEMENT context (#PCDATA | filter | rule)*>
<!ATTLIST context
    id    ID    #REQUIRED
    type CDATA #REQUIRED
>
<!ELEMENT uiContent (#PCDATA)>
<!ATTLIST uiContent
    id              ID              #REQUIRED
    ref             CDATA           #REQUIRED
    userRelevant (true|false) #IMPLIED
```

```
>
<!ELEMENT relations (relation)*>
<!ELEMENT relation (#PCDATA)>
<!ATTLIST relation
    id      ID                          #REQUIRED
    type (CONTROL|DATA|UICONTENT) #REQUIRED
    from    IDREF                       #REQUIRED
    to      IDREF                       #REQUIRED
    attribute     CDATA         #IMPLIED
    isSuggested (true|false) #IMPLIED
    observed      CDATA         #IMPLIED
>
<!ELEMENT filter (#PCDATA)>
<!ATTLIST filter
    attribute CDATA #IMPLIED
    operation CDATA #IMPLIED
>
<!ELEMENT rule (#PCDATA|if|then)*>
<!ATTLIST rule
    confidence    CDATA         #IMPLIED
    isSuggested (true|false) #IMPLIED
>
<!ELEMENT if (#PCDATA)>
<!ATTLIST if
    id    IDREF         #REQUIRED
>
<!ELEMENT then (#PCDATA)>
<!ATTLIST then
    id    IDREF         #REQUIRED
>
```

A.2. Questionnaires of the User Study

A.2.1. General Questionnaire

Vorerfahrung
Bitte beantworten Sie vorab einige Fragen zu Ihrer Vorerfahrung und zur statistischen Auswertung:

Ihr Geschlecht:
○ weiblich ○ männlich

Haben Sie Informatik oder ein verwandtes Fach studiert bzw. studieren es noch:
○ ja ○ nein

Wie häufig nutzen Sie das Internet?
(höchstens einmal im Monat) 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ (täglich)

Haben Sie die Internetseite der Deutschen Bahn bereits genutzt, um nach Verbindungen zu suchen?
(noch nie) 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ (sehr häufig)

A.2.2. Questionnaire regarding Support

Wie wurden Sie bei der Erfüllung Ihrer Aufgabe durch die Anwendung unterstützt?
(schlecht) 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ (sehr gut)

Wie störend fanden Sie die Hilfestellung der Anwendung?
(nicht störend) 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ (sehr störend)

Wie störend glauben Sie wäre die Hilfestellung der Anwendung auf lange Sicht?
(nicht störend) 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ (sehr störend)

Würden Sie Anwendungen mit dieser Art von Unterstützung gerne einsetzen?
(überhaupt nicht) 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ (sehr gerne)

Haben Sie irgendwelche generellen Kommentare zur gesehenen Anwendung?

List of Figures

List of Tables

List of Algorithms

Bibliography

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., and Shuster, J. E. (1999). UIML: An Appliance-Independent XML User Interface Language. In *Proceedings of the 8th International Conference on World Wide Web*, page 1695–1708, Toronto, Canada. Elsevier North-Holland, Inc.

Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago de Chile, Chile. Morgan Kaufmann Publishers Inc.

Aitenbichler, E., Kangasharju, J., and Mühlhäuser, M. (2007a). MundoCore: A Light-Weight Infrastructure for Pervasive Computing. *Pervasive Mobile Computing*, 3(4):332–361.

Aitenbichler, E., Lyardet, F., and Mühlhäuser, M. (2007b). Designing and Implementing Smart Spaces. *CEPIS Upgrade*, VIII(4):31–37.

Albrecht, D. W., Zukerman, I., and Nicholson, A. E. (1998). Bayesian Models for Keyhole Plan Recognition in an Adventure Game. *User Modeling and User-Adapted Interaction*, 8(1-2):5–47.

Amandi, A. and Armentano, M. (2004). Connecting Web Applications with Interface Agents. *International Journal of Web Engineering and Technology*, 1(4):454–470.

Anderson, J. R. and Lebiere, C. J. (1998). *The Atomic Components of Thought*. Lawrence Erlbaum.

Apple (2008). Apple Human Interface Guidelines. `http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html`.

Armentano, M. and Amandi, A. (2003). Agents Detecting User's Intention. In *Proceedings of the Argentine Symposium on Artificial Intelligence (ASAI)*, electronic proceedings.

Asuncion, A. and Newman, D. (2007). *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A Survey on Context-Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

Bao, X., Herlocker, J. L., and Dietterich, T. G. (2006). Fewer Clicks and Less Frustration: Reducing the Cost of Reaching the Right Folder. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*, pages 178–185, Sydney, Australia. ACM.

Barkhuus, L. and Dey, A. (2003). Is Context-Aware Computing Taking Control Away from the User? Three Levels of Interactivity Examined. *Lecture notes in Computer Science*, 2864:149–156.

Bellotti, V. and Edwards, K. (2001). Intelligibility and Accountability: Human Considerations in Context-Aware Systems. *Human-Computer Interaction*, 16(2):193–212.

Berry, P., Peintner, B., Conley, K., Gervasio, M., Uribe, T., and Yorke-Smith, N. (2006). Deploying a Personalized Time Management Agent. In *Proceedings of the fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1564–1571, New York, NY, USA. ACM.

Brijs, T., Vanhoof, K., and Wets, G. (2003). Defining Interestingness for Association Rules. *International Journal of Information Theories and Applications*, 10(4):370–376.

Buchanan, G., Farrant, S., Jones, M., Thimbleby, H., Marsden, G., and Pazzani, M. (2001). Improving Mobile Internet Usability. In *Proceedings of the 10th International World Wide Web Conference*, pages 673–680, Hong Kong, Hong Kong. ACM.

Buchholz, T., Küpper, A., and Schiffers, M. (2003). Quality of Context: What it is and why we need it. In *Proceedings of the 10th HP-OPenView Workshop*, Geneva, Switzerland.

Budanitsky, A. and Hirst, G. (2006). Evaluating WordNet-Based Measures of Semantic Distance. *Computational Linguistics*, 32(1).

Bunt, A., Conati, C., and McGrenere, J. (2007). Supporting Interface Customization Using a Mixed-Initiative Approach. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, pages 92–101, Honolulu, Hawaii, USA. ACM.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15(20):289–308.

Chen, G. and Kotz, D. (2000). A Survey of Context-Aware Mobile Computing Research. Technical report, Dartmouth College.

Chusho, T., Fujiwara, K., and Minamitani, K. (2002). Automatic Filling in a Form by an Agent for Web Applications. In *Proceedings of the 9th Asia-Pacific Software Engineering Conference*, pages 239–247. IEEE Computer Society.

Cohen, W., Ravikumar, P., and Fienberg, S. (2003). A Comparison of String Metrics for Matching Names and Records. In *Proceedings of the KDD Workshop on Data Cleaning and Object Consolidation.*

Cook, R. and Kay, J. (1994). The Justified User Model: A Viewable, Explained User Model. In *4th International Conference on User Modeling*, pages 145–150, Hyannis, MA.

Cypher, A. (1991). EAGER: Programming Repetitive Tasks by Example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Reaching Through Technology*, pages 33–39, New Orleans, LA, USA. ACM.

Davison, B. D. and Hirsh, H. (1998). Predicting Sequences of User Actions. In *Proceedings of the AAAI-98/ICML-98 Workshop on Predicting the Future: AI Approaches to Time-Series Problems*, pages 5–12. AAAI Press.

Dent, L., Boticario, J., Mcdermott, J., Mitchell, T., and Zabowski, D. (1992). A Personal Learning Apprentice. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 96–103, San Jose, CA, USA.

Dey, A., Sohn, T., Streng, S., and Kodama, J. (2006). iCAP: Interactive Prototyping of Context-Aware Applications. *Lecture Notes in Computer Science*, 3968:254–271.

Dey, A. K. (2001). Understanding and Using Context. *Personal Ubiquitous Computing*, 5(1):4–7.

Dey, A. K., Abowd, G. D., Pinkerton, M., and Wood, A. (1997). CyberDesk: A Framework for Providing Self-Integrating Ubiquitous Software Services. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 75–76, Banff, Alberta, Canada.

Dey, A. K. and Newberger, A. (2009). Support for Context-Aware Intelligibility and Control. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, pages 859–868, Boston, MA, USA. ACM.

Dix, A., Beale, R., and Wood, A. (2000). Architectures to Make Simple Visualisations Using Simple Systems. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 51–60, Palermo, Italy. ACM.

Dix, A., Catarci, T., Habegger, B., Ioannidis, Y., Kamaruddin, A., Katifori, A., Lepouras, G., Poggi, A., and Ramduny-Ellis, D. (2006). Intelligent Context-Sensitive

Interactions on Desktop and the Web. In *Proceedings of the International Workshop in Conjunction with AVI 2006 on Context in Advanced Interfaces*, pages 23–27, Venice, Italy. ACM.

Dragunov, A., Dietterich, T., Johnsrude, K., Mclaughlin, M., Li, L., and Herlocker, J. (2005). TaskTracer: A Desktop Environment to Support Multi-Tasking Knowledge Workers. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, pages 75–82. ACM Press.

Eyharabide, V. and Amandi, A. (2005). Automatic Task Model Generation for Interface Agent Development. *Inteligencia Artificial*, 9(26):49–57.

Faaborg, A. and Lieberman, H. (2006). A Goal-Oriented Web Browser. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 751–760, Montreal, Quebec, Canada. ACM.

Faulring, A. and Myers, B. A. (2005). Enabling Rich Human-Agent Interaction for a Calendar Scheduling Agent. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, pages 1367–1370, Portland, OR, USA. ACM.

Fellbaum, C. (1998). *WordNet An Electronic Lexical Database*. MIT Press, Cambridge, MA.

Findlater, L. and McGrenere, J. (2008). Impact of Screen Size on Performance, Awareness, and User Satisfaction with Adaptive Graphical User Interfaces. In *Proceeding of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems*, pages 1247–1256, Florence, Italy. ACM.

Gabrilovich, E. and Markovitch, S. (2007). Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1606–1611, Hyderabad, India.

Gajos, K. and Weld, D. S. (2004). SUPPLE: Automatically Generating User Interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, pages 93–100, Funchal, Madeira, Portugal. ACM.

Gajos, K. Z., Czerwinski, M., Tan, D. S., and Weld, D. S. (2006). Exploring the Design Space for Adaptive Graphical User Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 201–208, Venezia, Italy. ACM.

Garland, A., Ryall, K., and Rich, C. (2001). Learning Hierarchical Task Models by Defining and Refining Examples. In *Proceedings of the 1st International Conference on Knowledge Capture*, pages 44–51, Victoria, British Columbia, Canada. ACM.

Glass, A., McGuinness, D. L., and Wolverton, M. (2008). Toward Establishing Trust in Adaptive Agents. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, pages 227–236, Gran Canaria, Spain. ACM.

Gopalratnam, K. and Cook, D. J. (2007). Online Sequential Prediction via Incremental Parsing: The Active LeZi Algorithm. In *IEEE Intelligent Systems*, volume 22, pages 52–58, Los Alamitos, CA, USA. IEEE Computer Society.

Gorniak, P. and Poole, D. (2000). Predicting Future User Actions by Observing Unmodified Applications. In *Proceedings of the 17th National Conference on Artificial Intelligence*, pages 217–222.

Greenberg, S. (1988). *Using Unix: Collected Traces of 168 Users. Research Report 88/333/45.*

Gurevych, I., Müller, C., and Zesch, T. (2007). What to be? - Electronic Career Guidance Based on Semantic Relatedness. In *Proceedings of 45th Annual Meeting of the Association for Computational Linguistics*, pages 1032–1039.

Hartmann, M. (2009). Challenges in Developing User-Adaptive Intelligent User Interfaces. In *Proceedings of the 17th Workshop on Adaptivity and User Modeling in Interactive Systems*, pages 6–11, Darmstadt, Germany.

Hartmann, M. and Austaller, G. (2008). Context Models and Context-awareness. In Mühlhäuser, M. and Gurevych, I., editors, *Ubiquitous Computing Technology for Real Time Enterprises*, pages 235–256. Information Science Reference.

Hartmann, M. and Mühlhäuser, M. (2009). Context-Aware Form Filling for Web Applications. In *Proceedings of the Third IEEE International Conference on Semantic Computing ICSC*, Berkeley, CA, USA.

Hartmann, M. and Schreiber, D. (2007). Prediction Algorithms for User Actions. In Hinneburg, A., editor, *Proceedings of Lernen Wissen Adaption, ABIS 2007*, pages 349–354.

Hartmann, M. and Schreiber, D. (2008). Proactively Adapting Interfaces to Individual Users for Mobile Devices. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 300–303.

Hartmann, M. and Schreiber, D. (2009). AUGUR: Interface Adaptation for Small Screen Devices. In *Advances in Ubiquitous User Modeling*, pages 94–110. Springer.

Hartmann, M., Schreiber, D., and Kaiser, M. (2007). Task Models for Proactive Web Applications. In *Proceedings of WEBIST 2007*, page 150–155. INSTICC Press.

Hartmann, M., Schreiber, D., and Mühlhäuser, M. (2008a). Tailoring the Interface to Individual Users. In *5th International workshop on Ubiquitous User Modeling at IUI'08*, New York, NY, USA. ACM.

Hartmann, M., Schreiber, D., and Mühlhäuser, M. (2009). Providing Context-Aware Interaction Support. In *Proceedings of Engineering Interactive Computing Systems (EICS)*, pages 123–132. ACM.

Hartmann, M., Zesch, T., Mühlhäuser, M. M., and Gurevych, I. (2008b). Using Similarity Measures for Context-Aware User Interfaces. In *Proceedings of the 2nd International Conference on Semantic Computing*, page 190–197. IEEE.

He, H., Meng, W., Yu, C., and Wu, Z. (2004). Automatic Integration of Web Search Interfaces with WISE-Integrator. *The VLDB Journal*, 13(3):256–273.

Hermens, L. A. and Schlimmer, J. C. (1994). A Machine-Learning Apprentice for the Completion of Repetitive Forms. *IEEE Expert: Intelligent Systems and Their Applications*, 9(1):28–33.

Horvitz, E. (1999). Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: The CHI is the Limit*, pages 159–166, Pittsburgh, Pennsylvania, United States. ACM.

Humble, J., Crabtree, A., Hemmings, T., Åkesson, K., Koleva, B., Rodden, T., and Hansson, P. (2003). "Playing with the Bits" User-Configuration of Ubiquitous Domestic Environments. In *UbiComp 2003: Ubiquitous Computing*, pages 256–263.

Höök, K. (2000). Steps to take before Intelligent User Interfaces become real. *Journal of Interacting with Computers*, 12(4):409–426.

Höök, K., Karlgren, J., Wærn, A., Dahlbäck, N., Jansson, C., Karlgren, K., and Lemaire, B. (1996). A Glass Box Approach to Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 6(2):157–184.

ISO 9241 (2000). *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs).*

Jacobs, N. and Blockeel, H. (2002). Sequence Prediction with Mixed Order Markov Chains. In *Proceedings of the Belgian/Dutch Conference on Artificial Intelligence*.

Jameson, A. (2001). Modelling both the context and the user. *Personal and Ubiquitous Computing*, 5(1):29–33.

Jameson, A. (2007). Adaptive Interfaces and Agents. In Jacko, J. A. and Sears, A., editors, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, pages 305–330. Lawrence Erlbaum Associates, Inc.

Jameson, A. and Schwarzkopf, E. (2002). Pros and Cons of Controllability: An Empirical Study. In *Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 193–202. Springer-Verlag.

Jaro, M. A. (1995). Probabilistic Linkage of Large Public Health Data Files. *Statistics in Medicine*, 14:491–498.

Kalfoglou, Y. and Schorlemmer, M. (2005). Ontology Mapping: The State of the Art. In *Semantic Interoperability and Integration*, Dagstuhl Seminar Proceedings.

Kaljuvee, O., Buyukkokten, O., Garcia-Molina, H., and Paepcke, A. (2001). Efficient Web Form Entry on PDAs. In *Proceedings of the 10th International Conference on World Wide Web*, pages 663–672, New York, NY, USA. ACM.

Kay, J. (2001). Learner Control. *User Modeling and User-Adapted Interaction*, 11(1-2):111–127.

Korpipää, P., Malm, E., Salminen, I., Rantakokko, T., Kyllönen, V., and Känsälä, I. (2005). Context Management for End User Development of Context-Aware Applications. In *Proceedings of the 6th International Conference on Mobile Data Management*, pages 304–308, Ayia Napa, Cyprus. ACM.

Kozierok, R. and Maes, P. (1993). A Learning Interface Agent for Scheduling Meetings. In *Proceedings of the 1st International Conference on Intelligent User Interfaces*, pages 81–88, New York, NY, USA. ACM.

Künzer, A., Ohmann, F., and Schmidt, L. (2004). Antizipative Modellierung des Benutzerverhaltens mit Hilfe von Aktionsvorhersage-Algorithmen. *MMI-Interaktiv*, (7):61–83.

Langley, P. and Fehling, M. (1996). The Experimental Study of Adaptive User Interfaces. Technical report 98-3, Institute for the Study of Learning and Expertise, Palo Alto, CA.

Lashkari, Y., Metral, M., and Maes, P. (1994). Collaborative Interface Agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA. AAAI Press.

Lieberman, H., Liu, H., Singh, P., and Barry, B. (2004). Beating Common Sense Into Interactive Applications. *AI Magazine*, 25(4):63–76.

Limbourg, Q. and Vanderdonckt, J. (2004). USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In *ICWE Workshops*, pages 325–338.

Linton, F., Joy, D., Schaefer, H., and Charron, A. (2000). OWL: A Recommender System for Organization-Wide Learning. *Educational Technology & Society*, 3(1).

Little, G., Lau, T. A., Cypher, A., Lin, J., Haber, E. M., and Kandogan, E. (2007). Koala: Capture, Share, Automate, Personalize Business Processes on the Web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 943–946, New York, NY, USA. ACM.

Liu, H. and Davenport, G. (2004). ConceptNet: A Practical Commonsense Reasoning Toolkit. *BT Technology Journal*, 22(4):211–226.

Maes, P. (1994). Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):30–40.

Malaka, R. (2008). Intelligent User Interfaces for Ubiquitous Computing. In Mühlhäuser, M. and Gurevych, I., editors, *Ubiquitous Computing Technology for Real Time Enterprises*, page 470–486. Information Science Reference.

Maybury, M. T. and Wahlster, W., editors (1998). *Readings in Intelligent User Interfaces*. Morgan Kaufmann Publishers Inc.

Modi, P. J., Veloso, M., Smith, S. F., and Oh, J. (2005). CMRadar: A Personal Assistant Agent for Calendar Management. In *Agent-Oriented Information Systems II*, pages 169–181.

Monge, A. E. and Elkan, C. P. (1996). The field matching problem: Algorithms and applications. In *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270.

Myers, B. A. (2007). A User Acceptance Equation for Intelligent Assistants. In *AAAI 2007 Spring Symposium on Interaction Challenges for Intelligent Assistants*.

Mühlhäuser, M. and Hartmann, M. (2009). Interacting With Context. In *Worshop on Quality of Context QuaCon'09*.

Nichols, J., Myers, B. A., Higgins, M., Hughes, J., Harris, T. K., Rosenfeld, R., and Pignol, M. (2002). Generating Remote Control Interfaces for Complex Appliances. In *Proceedings of UIST '02*, page 161–170. ACM Press.

Norman, D. A. (1994). How might people interact with agents. *Communications of the ACM*, 37(7):68–71.

Olsen, J. D. R., Jefferies, S., Nielsen, T., Moyes, W., and Fredrickson, P. (2000). Cross-Modal Interaction Using XWeb. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 191–200, San Diego, California, United States. ACM.

Orwant, J. (1994). Heterogeneous Learning in the Doppelger User Modeling System. *User Modeling and User-Adapted Interaction*, 4(2):107–130.

Pantic, M., Sebe, N., Cohn, J. F., and Huang, T. (2005). Affective Multimodal Human-Computer Interaction. In *Proceedings of the 13th annual ACM international conference on Multimedia*, page 669–676.

Paterno, F., Mancini, C., and Meniconi, S. (1997). Engineering Task Models. In *Proceedings of ICECCS '97*, page 69, Washington, DC, USA. IEEE Computer Society.

Paterno, F., Santoro, C., Mantyjarvi, J., Mori, G., and Sansone, S. (2008). Authoring Pervasive Multimodal User Interfaces. *International Journal of Web Engineering and Technology*, 4(2):235–261.

Puerta, A. and Eisenstein, J. (2002). XIML: A Common Representation for Interaction Data. In *Proceedings of the 7th international conference on Intelligent user interfaces*, page 214–215, San Francisco, California, USA. ACM.

Qiu, Y. and Frei, H. (1993). Concept Based Query Expansion. In *Proceedings of the ACM International Conference on Research and Development in Information Retrieval*.

Raghavan, S. and Garcia-Molina, H. (2001). Crawling the Hidden Web. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 129–138, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Rahm, E. and Bernstein, P. A. (2001). A Survey of Approaches to Automatic Schema Matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350.

Rich, C., Sidner, C., Lesh, N., Garland, A., Booth, S., and Chimani, M. (2006). DiamondHelp: A New Interaction Design for Networked Home Appliances. *Personal Ubiquitous Computing*, 10(2-3):187–190.

Rich, C. and Sidner, C. L. (1997). COLLAGEN: When Agents Collaborate with People. In *Proceedings of the first international conference on Autonomous agents*, pages 284–291, Marina del Rey, California, United States. ACM.

Rich, C., Sidner, C. L., and Lesh, N. (2001). COLLAGEN: Applying Collaborative Discourse Theory to Human-Computer Interaction. *AI Magazine*, 22(4).

Robinson, R., Henricksen, K., and Indulska, J. (2007). XCML: A Runtime Representation for the Context Modelling Language. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 20–26. IEEE Computer Society.

Rukzio, E., Noda, C., Luca, A. D., Hamard, J., and Coskun, F. (2008). Automatic Form Filling on Mobile Devices. *Pervasive Mobile Computing*, 4(2):161–181.

Salton, G. and McGill, M. J. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.

Satyanarayanan, M. (2001). Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*.

Schilit, B., Adams, N., and Want, R. (1994). Context-Aware Computing Applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, page 85–90. IEEE Computer Society.

Schmid, H. (1995). Probabilistic Part-of-Speech Tagging Using Decision Trees. In *International Conference on New Methods in Language Processing*.

Schreiber, D., Hartmann, M., Flentge, F., Mühlhäuser, M., Görtz, M., and Ziegert, T. (2008). Web Based Evaluation of Proactive User Interfaces. *Journal on Multimodal User Interfaces*, 2(1):61–72.

Schreiber, D., Hartmann, M., Flentge, F., Mühlhäuser, M., Ziegert, T., and Görtz, M. (2007). Web Based Evaluation of Proactive Multimodal User Interfaces. In *Proceedings of International Workshop on Usability of User Interfaces*.

Sikora, C. and Swan, R. (1998). Perceived Usability and System Complexity. *Asia-Pacific Computer and Human Interaction*, pages 76–81.

Smyth, B. and Cotter, P. (2002). The Plight of the Navigator: Solving the Navigation Problem for Wireless Portals. In *Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 328–337.

Souchon, N. and Vanderdonckt, J. (2003). A Review of XML-Compliant User Interface Description Languages. In *DSV-IS*, pages 377–391.

Stephanidis, C., Paramythis, A., Zarikas, V., and Savidis, A. (2004). The PALIO Framework for Adaptive Information Services. In *Multiple User Interfaces*. John Wiley & Sons, Ltd.

Strang, T. and Linnhoff-Popien, C. (2004). A Context Modeling Survey. In *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp*.

Stumpf, S., Bao, X., Dragunov, A., Dietterich, T. G., Herlocker, J., Johnsrude, K., Li, L., and Shen, J. (2005). Predicting User Tasks: I Know What You're Doing! In *Proceedings of the 20th National conference on Artificial Intelligence, Workshop on Human Comprehensible Machine Learning*.

Stylos, J., Myers, B. A., and Faulring, A. (2004). Citrine: Providing Intelligent Copy-and-Paste. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, pages 185–188, New York, NY, USA. ACM.

Sukaviriya, P. and Foley, J. (1992). Built-In User Modelling Support, Adaptive Interfaces, and Adaptive Help in UIDE. Technical report, http://smartech.gatech.edu/dspace/bitstream/1853/3680/1/92-25.pdf.

Susan, B. and McEvoy, A. T. (2003). An Intelligent Learning Environment with an Open Learner Model for the Desktop PC and Pocket PC. *Artificial Intelligence in Education: Shaping the Future of Learning through Intelligent Technologies*, pages 389–391.

Tan, P., Kumar, V., and Srivastava, J. (2002). Selecting the Right Interestingness Measure for Association Patterns. In *Proceedings of the eighth ACM SIGKDD*

international conference on Knowledge discovery and data mining, pages 32–41, Edmonton, Alberta, Canada. ACM.

Tiernan, S. L., Cutrell, E., and Czerwinski, M. (2001). Effective Notification Systems Depend on User Trust. In *Proceedings of Human-Computer Interaction–Interact*, pages Tokyo, Japan, 684–685.

UIUC (2003). The UIUC Web Integration Repository. Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository.

Vanderdonckt, J., Mendonca, H., and Massó, J. P. M. (2008). Distributed User Interfaces in Ambient Environment. In *Constructing Ambient Intelligence*, pages 121–130.

Wahlster, W. (1988). Distinguishing User Models from Discourse Models. *Computational Linguistics*, 14(3):101–103.

Want, R., Hopper, A., Falcão, V., and Gibbons, J. (1992). The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102.

Wexelblat, A. and Maes, P. (1997). Issues for Software Agent UI. *Unpublished Manuscript*.

Winkler, W. E. and Thibaudeau, Y. (1991). An Application of the Fellegi-Sunter Model of Record Linkage to the 1990 U.S. Decennial Census. Statistical Research Report Series RR91/09, Washington, D.C.

Wu, W., Yu, C., Doan, A., and Meng, W. (2004). An interactive clustering-based approach to integrating source query interfaces on the deep Web. In *Proceedings of SIGMOD*, page 95–106.

Zesch, T., Müller, C., and Gurevych, I. (2008). Extracting Lexical Semantic Knowledge from Wikipedia and Wiktionary. In *Proceedings of LREC*.

Zhang, Z., He, B., and Chang, K. C. (2004). Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 107–118, New York, NY, USA. ACM.

Ziegert, T., Lauff, M., and Heuser, L. (2004). Device Independent Web Applications - The Author Once - Display Everywhere Approach. In *ICWE*, pages 244–255.

Zimmerman, J., Tomasic, A., Simmons, I., Hargraves, I., Mohnkern, K., Cornwell, J., and McGuire, R. M. (2007). Vio: A Mixed-Initiative Approach to Learning and Automating Procedural Update Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1445–1454, New York, NY, USA. ACM.

Wissenschaftlicher Werdegang der Verfasserin[31]

10/2000 – 02/2006	Studium der Informatik an der Universität Darmstadt Nebenfach Psychologie Abschluss: Diplom-Informatiker Diplomarbeitsthema: *Task-switching in Audio-based Systems*
seit 02/2006	Wissenschaftliche Mitarbeiterin im Fachbereich Informatik an der Technischen Universität Darmstadt

[31] Gemäß §20 Abs. 3 der Promotionsordnung der TU Darmstadt