Related Work

All intelligent thoughts have already been thought; what is necessary is only to try to think them again.

–Johann Wolfgang von Goethe (1749–1832)

Adaptive user interfaces and adaptive dialog systems have been a topic of research for decades and a significant number of prototypes and approaches have been developed. In this chapter, we present adaptive systems and adaptation approaches and discuss their relation to this work. Adaptation may be regarded as the application of artificial intelligence techniques to user interfaces. The aim of artificial intelligence is to create intelligent computer systems by imitating – and possibly surpassing – human intelligence. In doing so, artificial intelligence enables a computer to become a worthy opponent in playing games such as chess, to understand human speech, or to become an intelligent agent for tasks like planning appointments. When applied to user interfaces, artificial intelligence techniques are used to observe the user-system interaction and to perform improvements of the interface. Interactive systems that use artificial intelligence to anticipate user behavior and improve the user interface are called intelligent or adaptive interfaces. Adaptations enable a user to work with interactive systems more easily and conveniently.
Research on the adaptation of interactive systems to abilities, traits, and preferences of individual users started with investigation of user modeling. For instance, user modeling determines user preferences or assigns users to different groups. One example of an early user modeling system is the Grundy system (Rich, 1979), which describes the user by means of stereotypes. Based on these stereotypes, novels are recommended to the user. Subsequently, user modeling was applied to enable the adaptation of the interface to an individual user. One of the earliest investigations of the viability of adaptive interfaces is presented by Greenberg and Witten (1985). A user test with an adaptive menu-driven telephone book showed promising results and led to the development of further adaptive interactive systems. In the following decades, adaptivity has been applied to different kinds of interactive systems, such as office suites, hypertext systems, or speech dialog systems.

In this work, we address the adaptation of multimodal interactive systems in general. We present approaches for user modeling and adaptation that are applicable to a wide range of interactive systems. However, some of the adaptive interactive systems discussed in literature deal with very specific aspects of user interfaces or employ application-specific user modeling algorithms. Examples of such interfaces are an adaptive navigation engine (Bachfischer et al., 2007) or an adaptive restaurant guide (Langley, 1999). The user modeling algorithms and adaptations used in these interactive systems are not applicable to user interfaces in other domains. Instead, we discuss generic approaches that may be applied to different interactive systems. Adaptive systems also include recommender systems, for instance for online shops or movies based on films the user has watched before. An overview of recommender systems is given by Adomavicius and Tuzhilin (2005). A special case of recommender systems are collaborative filtering systems (Breese et al., 1998), which generate recommendations from preferences of other users. However, recommender systems are not the focus of this work. In the following, we provide an overview of different research areas in the domain of adaptive interactive systems and introduce examples.

This chapter is structured as follows. First, we give an overview of adaptivity in different kinds of interactive systems. Next, we describe different approaches for user modeling and introduce the concept of design patterns. Finally, a review of the use of ontologies and semantic technologies in interactive systems concludes the chapter.

2.1 Adaptive Interactive Systems

Adaptivity has been applied to a wide range of interactive systems, such as adaptive hypertext systems, adaptive user interfaces, or adaptive speech dialog systems. This section gives an overview of these different types of adaptive interactive systems.
2.1 Adaptive Interactive Systems

2.1.1 Adaptive Hypertext and Adaptive Websites

The adaptation of hypermedia systems and websites has been investigated extensively. Hypermedia systems provide interconnected content nodes to the user and the user moves between these nodes by means of hyperlinks. The content is composed of text fragments, graphics, and movies. Examples of hypermedia systems include learning systems, museum guides, or websites on the internet. Adaptive hypermedia systems generate custom versions of hypermedia pages for an individual user, for instance by adjusting content nodes to the knowledge level of the user or by sorting links according to the user’s interests. A user modeling component constructs a user model from an observation of visited nodes and time spent at each node. Based on this information, the user model identifies other content nodes the user might be interested in, for instance by recommending unknown nodes for relevant topics. Thereafter, different adaptation techniques generate a custom presentation for the user. Adaptive websites, a special case of adaptive hypertext systems, apply adaptation techniques to websites (Perkowitz and Etzioni, 1997). User modeling in adaptive websites is usually based on mining log files produced by a web server and employs algorithms such as cluster mining (Perkowitz and Etzioni, 1999). Since hypermedia systems are focused on content, user modeling algorithms primarily are concerned with the documents a user accesses and the adaptations alter the presentations of the content. The user interaction in interactive systems is richer and allows extracting more extensive information. For instance, hypermedia systems are usually limited to log data, whereas interactive systems observe the interaction of the user with each interface element, such as scrolling in a list or speech interaction.

Adaptations in adaptive hypermedia systems have been categorized into two groups, adaptive presentation techniques and adaptive navigation support (cf. De Bra et al., 1999a). Adaptive presentation performs adaptations of content nodes, such as showing or hiding nodes or selecting among different versions of a node. For instance, longer or additional text fragments are selected for starters, whereas only a short text is presented to experts. In addition, the adaptation may gray out known content fragments. Adaptive navigation support adapts the link structure between content nodes to reflect the knowledge and preferences of a user. For instance, links between content nodes are emphasized by reordering them to show important ones on the top or by annotating links with text or graphics. Links to nodes that are not interesting for the user are hidden by showing them as regular text or removed by not showing the link text at all. A comprehensive discussion of adaptive hypermedia technologies is presented by Brusilovsky (2001). It includes the taxonomy of adaptive hypertext technology given in Figure 2.1. The taxonomy includes adaptive presentation techniques and adaptive navigation support and divides these technologies into more specific adaptations. However, adaptations for hypertext systems may not be transferred directly to interactive systems. Adaptations can be applied to a wide range of different
interface elements rather than content nodes and links. User interfaces consist of components, such as buttons or speech input and output elements, which require different adaptations. Therefore, we present specific adaptations for interactive systems in this book. However, the advent of client-side dynamic web pages, which enable a web page to communicate with the server and update the page dynamically, blurs the differences between user interfaces and hypermedia systems. For instance, Schmidt et al. (2008) present an adaptation architecture for dynamic semantic web pages.

Different reference models have been defined in the domain of hypermedia, such as the Dexter reference model (Halasz and Schwartz, 1994). A reference model is an abstract definition of a domain and the concepts therein. It facilitates a general discussion of a domain, without restricting the validity to a single system. Different approaches extend the Dexter reference model for adaptive hypertext systems. The Adaptive Hypermedia Application Model (AHAM; De Bra et al., 1999b) adds three sub-models to the storage layer of the Dexter reference model. First, the domain model describes the content of the hypermedia system and comprises a list of text and multimedia fragments and links between these nodes. Second, the user model stores information about the knowledge level of different users, such as a list of visited content fragments. Third, the teaching model consists of a set of “pedagogical rules” and defines how adaptations are performed. The Munich reference model (Koch and Wirsing, 2002) is similar to AHAM, but uses the Unified Modeling Language (UML) as a formal foundation. This model also extends the storage layer of the Dexter reference model and introduces a domain model, a user model, and an adaptation model, which correspond to the
2.1 Adaptive Interactive Systems

respective models in AHAM. Based on these models, different operations, such as authoring, retrieval, or adaptation, specify the functionality of adaptive hypermedia systems. Other domains of adaptive systems may benefit from this research, which has produced sophisticated models for adaptive hypertext systems. In this work, we use an ontology for information representation, which may serve as a foundation for a reference model.

In the following, we present examples of adaptive hypermedia systems. ELM-ART (Weber and Brusilovsky, 2001) is an intelligent tutoring system for the LISP programming language. A traffic light metaphor provides adaptive navigation support: a green bullet in front of a link recommends a page for a user, whereas a red bullet indicates that the user’s level of knowledge is deemed insufficient for that page. The AVANTI project (Fink et al., 1996; Stephanidis et al., 1997) provides multimodal access to a tourist information system and considers the different needs of individual users. For instance, the AVANTI system provides laypersons with an explanation of technical terms, which is not necessary for expert users. One special focus group of the system are handicapped and elderly people. For instance, the tourist information contains more information on accessibility of sites when used by wheelchair users. A specific version of the text is served to blind people on a Braille keyboard. The AVANTI system supports adaptability, i.e., a manual adaptation by the user, as well as adaptivity, i.e., an automatic adaptation by the system. Adaptations are performed by means of adaptation rules.

Adaptive hypermedia techniques may also be applied to the semantic web, which extends regular hypertext with semantic annotations. These encode the meaning of the respective text sections and facilitate information extraction and reasoning. For example, the annotations clearly identify names of people or time designations in hypertext documents. Adaptive hypertext techniques have been applied to semantic websites. For instance, Dolog et al. (2003) present SIMPLE, an adaptation framework that is based on semantic web technologies and rule-based reasoning. For this purpose, a rule-based language called TRIPLE, which is presented in more detail in Section 2.4, defines adaptation rules. The approach presented in this work also relies on semantic web technology. However, instead of rules, we use a more abstract representation of adaptations. At the same time, the adaptations may be reused and tailored to the requirements of a specific interactive system.

2.1.2 Adaptive Graphical Interfaces

Graphical interfaces enable users to control a wide range of devices, such as personal computers, mobile phones, or personal navigation devices. Due to the high degree of deployment, significant research on adaptive user interfaces has been conducted in the domain of office suites, such as Microsoft Office. Since a lot may be learned from an investigation of unsuccessful approaches, we start this section with a discussion of unsuccessful adaptations and expose
their shortcomings. Thereafter, we introduce those adaptive interfaces that have proven successful.

One well-known use of adaptive technology is the Office Assistant, a cartoon style animated help agent introduced in Microsoft Office 2000 and removed again in Microsoft Office 2007. The agent observes user actions and provides help to the user. The technology is based on the Lumière research project (Horvitz et al., 1998), which employs explicitly modeled Bayesian networks to infer interesting help topics. However, different reasons caused the help agent to fail, such as a high degree of distraction and the selection of irrelevant help topics. Swartz (2003) presents a discussion on peoples’ attitudes towards anthropomorphic interface agents and identifies some of the problems related to the Office Assistant. For example, agents are not able to comply with human rules of etiquette. In general, interface agents are active and personalized collaborators that assist the user in an intelligent way with various tasks. Maes (1994) presents interface agents, which support the user with tasks such as handling incoming mail in an e-mail application, planning appointments in a calendar application, or filtering news in a news reader. However, Shneiderman (see Shneiderman and Maes, 1997) argues that direct manipulation interfaces correspond to how humans perceive and think better than intelligent agents. Direct manipulation describes interfaces in which elements on the screen are directly manipulated, e.g. with a mouse. The use of visualization techniques allows the user to deal with huge amounts of data.

Adaptive menus (see Figure 2.2) are a feature that was added to Microsoft Office and removed later. The menu hides rarely used menu entries and the user expands the full menu by clicking on small arrows at the bottom of the menu. However, if an item is not in the short menu, the user has to read both the short and the full menu, making the selection of items that are not on the short list very time-consuming. In addition, this kind of adaptive menu breaks with the usability principles of learnability, predictability and consistency by not leaving items in a place in which users always find them. Mitchell and Shneiderman (1989) obtained similar results in a comparison between static menus and dynamic menus. Dynamic menus placed frequently used items to the top of the list. An evaluation showed that dynamic menus did not improve the interaction and users preferred static menus. Therefore, if elements are removed from a list, the list element is more difficult to find, for instance because the user cannot remember positions in a list. These examples show a reluctance of users to adopt adaptive interfaces that do not adhere to usability principles. However, other research projects demonstrate that adaptive interfaces improve both performance and user satisfaction. In this work, we discuss usability principles for adaptive systems and present adaptations that comply with them.

Extensive research has been performed on adaptive menu selection. Two kinds of adaptations for list items were identified: changing the position by moving or duplicating menu items (spatial), and changing the appearance of menu items (graphical). One successful spatial adaptation of menus are split
menus, which place frequently used items into a separate section on top of the list. For example, Microsoft Word uses split menus in the font selection box. Frequently used fonts are shown in a separate section on top of the list. A study by Sears and Shneiderman (1994) shows that split menus, which move elements into the split area instead of copying them (see Figure 2.3(a)), accelerate list selections and users prefer split menus over non-adaptive menus. Findlater and McGrenere (2004) compared a static split menu, an automatically adaptive split menu, and an adaptable split menu, which lets users put items into the top section by hand. An evaluation revealed that the adaptable version was faster than the adaptive version and users preferred the former one. This suggests that users prefer customizable menus to adaptive menus. However, a study by Mackay (1991) showed that users do not perform manual customization unless the advantage of doing so is obvious. Moreover, many input devices limit the possibilities of complex customization, such as small mobile phone keypads or push rotary switches in the car. Gajos et al. (2006) investigated different adaptive versions of toolbars, which are common elements of graphical interfaces. For this purpose, three different adaptations were compared to a non-adaptive baseline. First, a split interface puts frequently used items into a separate toolbar. This corresponds to a split menu, which copies elements instead of moving them (see Figure 2.3(b)). Second, a moving interface moves frequently used elements from a popup menu into a list. Third, a visual pop-out interface highlights frequently used elements. An evaluation revealed that the split interface performed best, both yielding an improvement of the user performance and achieving the best ratings in the user satisfaction. The findings of Gajos et al. reveal that duplicating items rather than moving them improves the user performance.

Graphical adaptive menus sustain spatial consistency and only alter the appearance of menu entries. An adaptive emphasis is for instance accomplished by highlighting items that were predicted by a user modeling component. Figure 2.3(c) presents an example of an adaptation that highlights

![Fig. 2.2. Adaptive menu with manual expansion. Only frequently used items are visible when the menu opens and the expands the full menu by clicking on the arrows at the bottom.](image)
22  2 Related Work

Tsandilas and Schraefel (2005) present an evaluation that compares a list with highlighting to a list with highlighting and shrinking of text. Selection times are lower for the adaptive version with a constant font size and the authors reason based on a worst-case estimate that the adaptive versions should be faster than a non-adaptive baseline. Another comparison of different adaptive menus is presented by Park et al. (2007). They compared a traditional menu to an adaptable menu, which lets users change the sort order of the list, an adaptive split menu, and an adaptive menu that highlights the most frequently selected elements. Again, adaptable menus were most efficient and the users preferred them. While the adaptive highlight menu did not reduce the selection time, users preferred it to the traditional menu. Instead of highlighting menu entries, ephemeral adaptation (Findlater et al., 2009; see Figure 2.4) reduces the visibility of items that are not selected frequently and fades these items in quickly less than one second after the menu appeared. In doing so, it includes the temporal dimension. An evaluation revealed that menus with ephemeral adaptation reduce the selection time and users prefer them to other versions. Menu selection is an important task for any kind of interactive system. Therefore, the adaptations presented in this section are incorporated into a set of patterns we introduce in Section 4.3. Two of the systems created for the evaluation of the patterns use menu selection as their task.

Mixed initiative adaptation (Horvitz, 1999) represents a compromise between user-initiated customization and system-initiated adaptation. This approach reflects the user’s preferences by letting the user decide when to employ adaptations. Users prefer being in control, but do not personalize the system unless they see a clear advantage in doing so (cf. Mackay, 1991). MICA (Bunt et al., 2007) is a mixed-initiative adaptation framework that employs online GOMS analysis. This framework recommends frequently used interface elements and the user may place them into a personalized interface. The user

![Fig. 2.3. Three kinds of adaptive menus.](image-url)
decides if the recommended changes should be performed. A user evaluation showed that users prefer the mixed-initiative to a customizable interface, as long as the user modeling is accurate.

The adaptations presented in this section so far alter an existing interface. However, some adaptive systems generate the user interface according to the preferences or capabilities of an individual user. The SUPPLE system (Weld et al., 2003) adapts graphical interfaces to different requirements, for instance for desktop computers and mobile phones or pointer-based and touch panel-based devices (Gajos and Weld, 2004). In addition, a storyboard example is presented in which a printing dialog is adapted by adding frequently used options from sub-dialogs to the main dialog. An extended version, called SUPPLE++, supports users with motor impairments (Gajos et al., 2008). Some users have difficulties in moving the pointing device, whereas others have problems to click. Therefore, SUPPLE++ generates an interface that is tailored to the capabilities of an individual user by reducing either the distance the pointing device has to be moved or the number of clicks. An evaluation showed that the users strongly prefer the adaptive version. The most serious issue of generated interfaces is however the aesthetical appearance of the interface, which is less appealing than an interface designed by a human. In addition, safety requirements cannot be ensured with generated interfaces. Our approach does not employ generated interfaces, but improves interfaces created by a human designer. However, the “Alternative Elements” adaptation presented in Section 4.3.3 allows an interactive system to select among different alternatives provided by the developer.

2.1.3 Adaptive Speech Interfaces

Two human partners in a speech dialog adjust to each other, for instance by asking clarifying questions or raising their voices when they think the dialog partner does not understand them because their voice is too soft. Therefore, a

![Diagram](https://via.placeholder.com/150)

**Fig. 2.4.** Adaptive menu with ephemeral adaptation. Only the item that should be recommended to the user is visible at the beginning. Other items appear quickly thereafter.
natural and intuitive dialog between a machine and a person needs to be able to adapt to the user, rather than having a fixed dialog script. Consequently, adaptation in speech dialog systems is an important research topic. Different methods exist for implementing dialog control in speech-based interfaces. In this section, we introduce these methods, explain how adaptation is implemented within the individual approaches, and discuss the relation of these concepts to this work.

A first kind of dialog specification is based on a rigid dialog definition, which is followed closely by a dialog manager, such as state-based and form-based dialog systems (cf. McTear, 2004). State-based dialog systems define the dialog logic by means of state transition networks. These consist of states and transitions. Speech output by the dialog system is connected to states. Actions, such as speech input by the user, trigger transitions to other states. In doing so, the network defines all possible dialog paths. One example of an adaptive state-based dialog system is the TOOT system (Litman and Pan, 2002), a train information system that automatically adapts itself to the current user. For this purpose, it changes the dialog initiative and the confirmation style. The dialog system starts with a “user initiative”, in which the system asks open questions and leaves the initiative to the user, and “no confirmation” strategy, which does not confirm user input. Once problems are detected based on low speech recognition scores, the TOOT system switches to more conservative dialog strategies, such as “system initiative”, in which the dialog manager asks the user specific questions, and “explicit confirmation”, in which the dialog system confirms user input before advancing to the next question. User evaluations showed that an adaptive version of the TOOT system achieved a higher task success rate than a non-adaptive version. Hassel and Hagen (2006) follow a similar approach and present an adaptive voice-controlled infotainment system deployed in an automobile dashboard system. Speech output prompts are adapted to the expertise level of the current user. For instance, novice users receive a list of available commands, whereas the dialog system only plays short feedback tone for expert users. Table 2.1 presents an example of speech interactions with a novice and an expert user. In general, expert prompts are shorter than prompts for beginners. Rather than considering only speech input confidence scores, the dialog system by Hassel and Hagen uses more information to model the user’s experience, such as the number of help requests or timeouts. An evaluation showed that the adaptive prototype reduces the number of turns and the interaction time. The approach of selecting speech output prompts and input grammars depending on characteristics of the user-system interaction matches well with the framework presented in this work. A user modeling component derives information about the user from the user-system interaction, such as speech recognition errors. The adaptations presented in Chapter 4 include adaptations that select the appropriate speech input and output components for a user. Whereas state transition networks are rigid compared to other approaches, they are well suited for defining multimodal interactive systems and facilitate a uniform
### Table 2.1. An example of a speech interaction that is adapted to the experience of the user. (Source: Hassel and Hagen, 2006, page 2.)

<table>
<thead>
<tr>
<th>Novice</th>
<th>Expert</th>
</tr>
</thead>
<tbody>
<tr>
<td>user: Entertainment.</td>
<td>user: Entertainment.</td>
</tr>
<tr>
<td>system: Entertainment. You can say AM, FM, or CD.</td>
<td>system: Entertainment.</td>
</tr>
<tr>
<td>user: Choose CD.</td>
<td>user: Choose CD.</td>
</tr>
<tr>
<td>system: Say a CD number.</td>
<td>system: Number?</td>
</tr>
<tr>
<td>user: <em>Unintelligible.</em></td>
<td>user: <em>Unintelligible.</em></td>
</tr>
<tr>
<td>system: I could not understand you, repeat.</td>
<td>system: Pardon me.</td>
</tr>
</tbody>
</table>

definition of different modalities, such as speech and graphics. The reference implementation presented in this book employs a statechart formalism. It allows the development of graphical and speech-based interfaces with a common specification.

Form-based dialog systems also employ a rigid formalism. Forms define a number of slots that have to be filled by the user. The system directs the dialog, enabling the user to provide data for each of these slots. Form-based dialog systems offer more flexibility by letting the user decide whether to fill all slots with a single utterance or one slot at a time. Adaptation in form-based dialog systems is achieved by selecting among predefined components for speech input and speech output. Veldhuijzen van Zanten (1998) presents an example of an adaptive form-based dialog system, which extends the form-filling paradigm with hierarchical slots. Different kinds of questions are defined, such as high-level and low-level or open and closed questions. If speech recognition problems are discovered, the system moves to lower levels in the slot hierarchy, for instance by asking for values of individual slots instead of open questions. If the user provides more data in an utterance than is anticipated by the current hierarchy level (over-information), a higher level in the hierarchy is selected. The user model (Veldhuijzen van Zanten, 1999) consists of a set of flags for each slot that indicate the knowledge level of the user. VoiceXML\(^1\) is a form-based standard for defining voice interfaces. Niklfeld et al. (2001) present an adaptive architecture for multimodal dialog systems that is based on VoiceXML. Whereas the framework implemented within this work does not support the form-based approach, it may be integrated in addition to state-based dialogs. For this purpose, forms have to be attached to specific states in the dialog system.

Other approaches for the specification of dialog systems provide a higher degree of flexibility, for example the Information State Update (ISU; Larsson and Traum, 2000) approach. An information state “represents the information necessary to distinguish it from other dialogs” (Larsson and Traum, 2000, page 1) and a set of dialog moves trigger updates to the information state,

---

1 VoiceXML: [http://www.w3.org/Voice/](http://www.w3.org/Voice/)
for instance by means of update rules. In the TRINDI toolkit, a dialog move
engine computes the next action based on the current information state. In or-
der to enable adaptivity in the ISU non-adaptive approach, the TALK project
proposes an extension of an ISU-based dialog manager (Georgila and Lemon,
2004; Lemon et al., 2006) with reinforcement learning techniques to enable
the system to learn an optimized dialog strategy. Since the ISU approach is a
formalism for speech-based dialog systems and does not support multimodal
interaction, our framework does not support this approach.

Statistical dialog systems regard dialog design as an optimization prob-
lem by handling the dialog flow as a sequential decision process. For this
purpose, statistical algorithms, such as Markov decision processes (MDP),
learn optimal dialog strategies. Levin et al. (2000) and Scheffler and Young
(2002) first estimate the parameters of a simulated user from a dialog corpus
and then employ reinforcement learning to find the optimal dialog strategy
based on interactions of the simulated user with the dialog system. Different
techniques exist for collecting the dialog corpus of training sessions, for ex-
ample Wizard-of-Oz experiments (Rieser and Lemon, 2008) or training sessions
with a preliminary dialog system (e.g. in the NJFun dialog system, Singh
et al., 2002). An evaluation showed that the trained dialog strategy outper-
forms fixed strategies proposed in the literature. All possible states, i.e., all
possible values of the variables, form a state space and the dialog strategy
defines which action (e.g. which question to ask) should be taken in the cur-
rent state. A as a result, the dialog moves on to another state. The result
of the MDP training is a single optimized, yet non-adaptive dialog strategy.
When applying the learning algorithm at runtime, the dialog system adapts
to an individual user. For example, the CLASSiC project (Janarthanam and
Lemon, 2008; Rieser and Lemon, 2009) employs online learning of a statisti-
cal dialog system. Partially observable MDPs (POMDP) extend MDPs in
a way that they can handle both unobservable states and uncertainty (e.g.
the user’s beliefs) and enable the dialog manager to track all possible dialog
paths rather than just the most likely path. An application of POMDPs in
dialog systems was shown to create dialog strategies that perform better than
the ones created by regular MDPs (Young et al., 2010). Paek (2006) presents
a discussion of statistical dialog systems. The main advantage of statistical
dialog systems is the theoretical foundation, which most other dialog speci-
fications lack. In addition, the statistical approach handles uncertainty well.
On the other hand, Paek mentions the reluctance of application developers to
give up control over their application as the main disadvantage. Moreover, if
the results of statistical dialog systems are not always superior to handcrafted
systems, the handcrafted approach is likely to be followed. In addition, statis-
tical approaches are not suited for graphical or multimodal interactive systems
and thus aggravate the development of multimodal interactive systems by im-
peding a uniform approach. Since this book addresses multimodal interactive
systems, we do not include statistical dialog systems in the framework.
Agent-based software architectures employ autonomous software components, called agents, to solve complex problems through collaboration. Each of these intelligent agents implements a strategy or competence and contributes autonomously to solving specific tasks. The Interact project (Jokinen et al., 2002) is an example of agent-based dialog systems. It employs an architecture called Jaspis. The system consists of managers, agents, and evaluators. Managers take care of specific components, for instance a dialog manager or a presentation manager. Agents handle (possibly very specific) situations, such as speech recognition errors. Different agents implement different strategies for the same situation and evaluators select among the available agents. The advantages of the agent-based approach are a high degree of flexibility and the possibility to implement different strategies and select among them at runtime. The Jaspis architecture is cross-platform and distributed. However, agent-based architectures bring along a software overhead, which is not feasible with the limited computing resources of many interactive systems. Moreover, the behavior of agent-based systems is less predictable than the behavior of more rigidly defined dialog systems.

In addition to user behavior and traits like expertise, the modality of speech offers emotion as an additional channel. If a dialog system recognizes emotional cues, such as anger or impatience, in the user’s voice, the dialog strategy may be adapted accordingly. Examples of such systems are NIMITEK (Gnjatović and Rösner, 2008), which adapts spoken help messages to the current emotional state of the user, and PROBLEMO (Pittermann et al., 2007), an emotion-aware intelligent architecture for dialog systems. Emotions may be integrated into this framework by loading the results from the emotion recognition algorithms into the user model and using them as adaptation triggers. However, we focus on user behavior rather than other adaptation triggers in this book. Thus, we have presented different approaches for adapting speech dialog systems. The approaches that support multimodal interaction integrate well with our framework, whereas other may be added separately from other modalities. We present a set of adaptation patterns in Section 4.3 that include a discussion of speech dialog systems.

2.2 User Modeling for Adaptive Interactive Systems

An adaptive interactive system observes the user to find characteristics and preferences in the user-system interaction. These characteristics trigger adaptations, such as shortcuts for repeated actions or adaptive help. The process of observing a user and drawing conclusions is called user modeling. Adaptive interactive systems represent the user by means of a user model. In this section, we give an overview of user modeling approaches in the literature.

The user model obtains information about a user either explicitly by asking the user questions or implicitly by observing the user without interference (“loophole observation”). User-supplied information is for instance collected
by means of on-screen dialogs or questionnaires. However, user-supplied information is not reliable and users may be reluctant to provide information about them. In the remainder of this section, we address automatic user modeling that derives information from an observation of the user-system interaction. The information that a user model stores depends on the requirements of the adaptive interactive system and includes goals, knowledge, interests, traits, experience, and preferences.

One of the earliest user modeling systems is the Grundy system (Rich, 1979), which offers novel recommendations to a user. The system regards a user model as a “collection of good guesses about the user” (Rich, 1983, page 200) and includes information such as age and level of education. Based on this information, the current user is assigned to a predefined stereotype. Many user modeling approaches deal with specific domains and problems, such as modeling a user’s favorite TV program to be able to recommend interesting shows (e.g. Ardissono et al., 2004 or Bachfischer et al., 2007). In this section, we instead discuss reusable and generic user modeling algorithms and architectures.

2.2.1 User Modeling Algorithms

User modeling algorithms serve different purposes in user modeling systems, such as predicting user actions or modeling user preferences. The selection of these algorithms depends on the requirements of the interactive system and the adaptations. This section presents different kinds of algorithms that are suitable for adaptive interactive systems.

Sequence prediction algorithms (SPA) predict a future item based on past items. For instance, SPAs enable an adaptive interface to anticipate a user action based on previously observed actions and offer support accordingly. Davison and Hirsh (1998) present an SPA called Incremental Probabilistic Action Modeling (IPAM), which employs first-order Markov chains and reduces the influence of older data. In an evaluation with UNIX command line actions, a prediction accuracy of 40% was achieved. However, unlike many interactive systems, UNIX commands do not have a context, i.e., the user may enter all commands at all times. Hartmann and Schreiber (2007) present an SPA algorithm called FxL and compare it to different other SPAs, including IPAM. Different test sets are used, including the UNIX test set by Davison and Hirsh and an office suite test set. Prediction accuracy was limited to a range of 40% to 60%, but Hartmann and Schreiber reason that domain knowledge could improve the prediction accuracy. In this work, we present a sequence prediction algorithm for user actions with a similar background as IPAM. However, we employ domain knowledge in the form of a task model to filter out predictions that are not valid in the current context. For this purpose, we adapted a Markov chain-based algorithm for predicting link in websites presented by Sarukkai (2000) to user action prediction. We combined it with a task model to add domain information for better prediction results.
In doing so, the algorithm is optimized for the use in adaptive interactive systems.

Whereas SPAs predict a single item, sequence mining algorithms identify frequently occurring sequences. Adaptive interfaces use sequence mining for example for offering shortcuts for frequent action sequences. Mannila et al. (1997) present a sequence mining algorithm that retrieves frequent episodes from a sequence of events, for instance in user log files. However, the algorithm does not make a statement about the meaning of the discovered episodes. Liu et al. (2003) present an application of Mannila’s algorithm to adaptive user interfaces. The algorithm automatically detects repeated action sequences in a word processing software, such as repeatedly applying a certain combination of formatting options, and offers shortcuts to the user for this formatting. In Section 3.4.2, we present an adaptation of Mannila’s algorithm for predicting action sequences in adaptive interactive systems. This algorithm was used for the evaluation of an adaptive system (see Section 6.4.6).

In addition to user action prediction, the user’s goals and needs serve as adaptation triggers. One approach for modeling goals and needs are Bayesian networks, a graph-based model for probabilistic relationships between random variables. Horvitz et al. (1998) present an agent-based adaptive help system called Lumi`ere that employs Bayesian networks to describe the user’s experience and derive help messages for the current situation. Bayesian networks are explicit representations of a domain and a user’s knowledge. They facilitate probabilistic inference for determining the user’s needs and goals. Dynamic Bayesian networks allow incorporating temporal aspects by integrating the user’s interaction history. However, Bayesian networks have to be modeling largely by hand and thus add a considerable development effort in more complex domains. Bayesian networks may be integrated into the user model of our framework.

Different machine learning algorithms (see Witten and Frank, 2005) may be used for user modeling. Algorithms may be divided into supervised or unsupervised ones. Webb et al. (2001) discuss general requirements of machine learning algorithms for user modeling. For instance, these algorithms require a large collection of training data and labeled datasets that have been annotated before. Supervised machine learning algorithms, such as Markov models or neural networks (Mitchell, 1997), train models from labeled data and employ these models to classify unknown individuals by assigning them to a known class. In adaptive interactive systems, classification is for instance used for identifying users or assigning them to a group, such as beginner or expert. For example, Galassi et al. (2005) present an approach that uses Hierarchical Hidden Markov Models (HHMM) to train user profiles from recorded sessions and employ these models to identify users. Unsupervised machine learning algorithms do not rely on labeled data. Clustering algorithms divide a group of elements into a set of clusters. However, the properties shared by different elements in a group are not known. Clustering algorithms have for instance been used in adaptive web systems (Hamilton et al., 2001) to compute page
recommendations for new users based on interaction patterns of other users. However, this approach does not work for interactive systems, because interaction data from other users is usually not available. Zukerman and Albrecht (2001) provide an overview of predictive statistical models, such as Markov models and Bayesian networks. These algorithms may be implemented within the user modeling framework presented in this work. The selection of the algorithms depends on the requirements of a specific interactive system. We present a number of user modeling algorithms that we developed or adapted for adaptive interactive systems in Chapter 3.

2.2.2 Plan Recognition and Task Models

Plan recognition is the process of observing user actions and determining the goal a user tries to accomplish. A plan represents the order of actions required to reach that goal. Plans include uncertainty, since the actual behavior of users is to some degree unpredictable. Moreover, information such as the planner’s intent cannot be observed and may therefore only be inferred with a certain probability. Plans are very complex, especially when they describe “real world” problems, such as cooking. Many plan recognition systems rely on a plan library, which is created by hand with a significant effort. An early example of plan recognition systems called BELIEVER is presented by Schmidt et al. (1978). Carberry (2001) describes a number of problems that need to be addressed in conjunction with plan recognition. First, the input data is often noisy, i.e., the individual observations that are used to infer the user’s goal are uncertain. Second, the plan recognizer has to decide among competing hypotheses, e.g. when certain observations are part of different plans. Moreover, users tend to work on different tasks in parallel. Third, plan recognition algorithms have to scale up in large domains with plan libraries that contain numerous plans a user possibly works on.

In this work, we introduce an approach for adaptive interactive systems that employs a technique called task modeling (Paternò, 2001). It describes the user-system interaction by means of tasks, which define possible user actions without speculation about the user’s intentions. Task modeling is used in software engineering for design and evaluation. A detailed introduction to task modeling is given in Section 3.3. Instead, we employ task models at runtime to derive information about the user-system interaction, such as predicting future actions or identifying situations in which a user requires assistance. We present task models as a viable means for describing higher-level user behavior in adaptive interactive systems and deriving adaptation triggers from it.

Klug and Kangasharju (2005) employ task knowledge for supporting a user. Their interactive system observes the user’s activity by instantiating a task model at runtime and generates an improved user interface to better support the current task. In a similar approach, an intelligent classroom (Franklin et al., 2002) recognizes user actions by means of a plan-based action description and supports users in performing these actions, e.g. by advancing slides
2.2 User Modeling for Adaptive Interactive Systems

Mitrović et al. (2008) present an agent-based adaptive user interface that employs a task model notation based on ConcurTaskTrees (Paternò et al., 1997), similar to the approach presented in Section 3.3. Our approach uses the task model for instance for predicting user actions and detecting user problems.

2.2.3 User Modeling Architectures

In addition to the algorithms, the architecture of a user modeling system is an important aspect of the user modeling process. In early adaptive interfaces, user modeling used to be an integral component of these systems and could not be reused for other applications. Generic user models (Kobsa, 2001) instead allow the developer to reuse the user modeling architecture. In addition, generic user modeling systems may enable different applications to share a single user model and thus facilitate a reuse of user data. One example of a generalized user modeling system is the Doppelgänger user modeling system (Orwant, 1995). This system supports learning by means of different techniques, such as linear prediction and Markov models. However, custom learning techniques cannot be added and the accumulation of data is left to the application. The user modeling system developer for this work was created such that it allows the integration of a wide range of algorithms. A user modeling framework for intelligent learning environments that implements both supervised and unsupervised learning is presented by Amershi and Conati (2007). More recent developments in the domain of user modeling are ubiquitous and distributed user modeling systems (Heckmann, 2005). They combine information from different sources, such as mobile phones or portable computers. In addition, location awareness is an important topic for mobile devices (Sharifi et al., 2004). Our adaptation framework comprises a generic and extensible user modeling component. The framework may be extended with arbitrary user modeling algorithms and connected directly to observations from the user-system interaction. In doing so, the framework meets the requirements of different kinds of adaptive interactive systems.

Logic-based systems represent data in a way that allows inference of new knowledge by means of logical reasoning. One example of a user modeling system that employs a logic-based representation is the BGP-MS system (Kobsa and Pohl, 1995). It converts the internal representation to first-order logic and uses logic reasoning on the data. Rules draw additional inferences about the user. In a fashion similar to logic-based architectures, semantic user modeling systems employ a semantic representation using semantic web technology. The semantic web employs a description logic formalism to add semantic annotations to existing data, thus allowing a description of meaning as well as reasoning. An ontology defines the structure of the semantic data and often serves as a foundation for semantic user modeling architectures. Various ontologies have been proposed specifically for user modeling. For instance, Golemati et al. (2007) and Heckmann et al. (2005) demonstrate sophisticated
examples of such ontologies and store user modeling data in terms of ontologies. The semantic notation facilitates data sharing between different applications. Razmerita et al. (2003) present a general architecture for a user modeling server that is based on semantic technologies. However, semantic user models do not support complex derivations and specific data types (e.g. matrices for Markov chains). Since many algorithms employ such data types, we present a generic user modeling system that supports arbitrary types efficiently. In order to connect the user model to a semantic description of the interactive system, we use a bridging component.

2.3 Design Patterns for Adaptations

Every domain has experts who have collected valuable experience about successful solutions for recurring problems. Sharing expert knowledge helps other people working in the same domain to straightforwardly select suitable solutions. Design patterns are an approach to record experience by writing down recurring problems in a specific domain and successful solutions for these problems. In doing so, patterns serve as a means for communicating best practice. Design patterns have their origins in architecture. In the seminal 1977 book “A Pattern Language”, Alexander et al. (1977) introduce a set of 253 architectural patterns. Each of these patterns describes an architectural problem and presents a proven solution to this problem. Patterns are defined as follows:

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. (Alexander et al., 1977, page x)

All these patterns employ the same structure, which is often referred to as the Alexandrian form. The description is narrative and uses few subheadings. After the heading, the context of the pattern is introduced followed by a problem statement and an elaborate discussion of the problem. Thereafter, a concise solution statement and a detailed discussion of the solution, including a small drawing, present the solution. For example, the “Your Own Home” pattern (Alexander et al., 1977, page 392) states that people are not happy in a home they do not own. The recommended solution is to enable people to own the house they live in, which they arrange according to their wishes and expectations.

The concept of design patterns was adopted for the domain of software engineering and design patterns have become an established method in software engineering. The most well-known set of design patterns for software engineering is presented in the book “Design Patterns: Elements of Reusable Object-Oriented Software” by Gamma et al. (1995), often referred to as the “gang of four” patterns. These design patterns describe problems occurring
frequently during (object-oriented) software development and present successful solutions. For instance, the “Singleton” design pattern proposes a method to ensure that only a single instance of a specific class exists, which different parts of a software application share. Design patterns have been widely adopted and are part of many students’ curricula. Another pattern collection called “Pattern-Oriented Software Architecture” (POSA; Buschmann et al., 1996) presents a set of patterns for software architecture, which deal with a more high-level view on software design. Beyond these patterns, numerous pattern collections have been presented at workshops like “Pattern Languages of Programming” (PLoP), such as patterns for parallel programming (Ortega-Arjona, 2009) or patterns for creativity (Georgiakakis and Retalis, 2009).

A pattern language is a collection of patterns that describe best practice for a specific domain. The patterns cover the most common problems of this domain. An interlinking further connects the patterns to each other. While many patterns do not intersect, some patterns may address overlapping problems. In this case, the context of the pattern determines which pattern to use best. In the following, we introduce the concept of patterns and review patterns from the literature, both in the area of interface patterns and formalization of design patterns.

### 2.3.1 Interface and Adaptation Patterns

Whereas design patterns are mostly used in the domain of software engineering, patterns have also been written for the domain of user interface design and interaction design. Borchers (2001) introduces a pattern language for interactive exhibits. For instance, the “Attract Visitor” pattern emphasizes the importance of having an exhibit that is interesting enough to attract a visitor’s attention. Van Duyne et al. (2006) present a pattern language for websites. These patterns are structured into groups such as genres of sites, content, or navigation. For example, the “Sitemap” pattern proposes a single place called sitemap that may be used to access all pages of a website.

Different pattern catalogs have been defined for user interface design. Van Welie and van der Veer (2003) have compiled extensive pattern collections of reusable interface design knowledge, which may be used by designers and developers for creating graphical interfaces. Similarly, Tidwell (2005) presents an extensive structured catalog of 94 interface design patterns. These patterns cover a wide range of topics, from the general structure of graphical applications to form input to aesthetics. For example, the “Extras On Demand” pattern advises a designer to offer a limited set of options in an interface and to add a button that opens a larger set of options for advanced users. The “Cancelability” pattern recommends offering a way to the user for cancelling time-consuming operations. Many interface designs pattern are very specific and describe a single interface element. While some of the patterns describe dynamic behavior, such as “Responsive Enabling” or “Smart Menu Items”, these patterns do not address adaptive user interfaces. Dearden and Finlay
Patterns have also been used to describe adaptive hypertext systems. A basic set of abstract adaptation patterns was presented for adaptive hypertext systems by Danculovic et al. (2001), introducing “Link Personalization”, “Content Personalization”, “Structure Personalization”, and “Remote Personalization”. For example, the “Link Personalization” pattern recommends altering the link structure of a website to better reflect a user’s needs. The “Content Personalization” pattern discusses offering personalized content nodes. The patterns are more general in nature than interface design patterns and the number of patterns is significantly lower. These adaptation patterns were extended by Koch and Rossi (2002) with more detailed patterns, such as “Adaptive Anchor Annotation” or “Adaptive Sorting of Anchors”. The “Adaptive Anchor Annotation” pattern discusses how adding annotations to hypertext links lets the user better estimate the usefulness of a link. Another pattern called “Adaptive Sorting of Anchors” recommends sorting anchors in a way that more interesting anchors are shown first. As discussed in Section 2.1, adaptive hypertext systems differ from other interactive systems. For instance, the adaptations alter interface elements rather than links between nodes or text. Therefore, we present a distinct set of adaptation patterns for interactive systems in Chapter 4.

2.3.2 Formalization of Design Patterns

Design patterns use a textual and narrative form. For example, the patterns collections by Gamma et al. (1995) and Buschmann et al. (1996) have been published as books to be read by programmers. Therefore, humans read the pattern descriptions, but a computer cannot process them automatically. Approaches for the formalization of patterns aim to increase the utility of patterns by representing patterns using a well-defined structure and vocabulary. In doing so, they provide a standardized and machine-processable representation. The formalization of patterns takes place on different levels of abstraction. First, a formalized pattern format extends the narrative description with special markings to label sections of a pattern description. For instance, all patterns may share the “motivation” and “solution” sections. This machine-readable structure ensures consistency and enables referencing between different pattern collections. The Pattern Language Markup Language (PLML; Fincher et al., 2003) follows this approach and provides an XML document type definition (DTD) for specifying patterns. PLML is however a very high-level definition for describing textual pattern collections in a uniform way to enable interchange.

A more formal notation of patterns may serve as a basis for intelligent tool support. For instance, tools may provide support for refactoring existing projects according to patterns (e.g. Zannier and Maurer, 2003). Other approaches employ formal languages to specify patterns, including the semantics
of the pattern. Mikkonen (1998) discusses an approach for formalizing patterns based on a custom notation for defining objects formally, with the focus being on the temporal behavior of design patterns. Hallstrom and Soundarajan (2008) present an approach for ensuring implementation correctness and facilitating reasoning about patterns. The approach is based on a pattern contract formalism with pre and post conditions. However, approaches that rely on formal languages have not found wide-spread use due to their high complexity (Zdun, 2007).

Ontologies are formal representations of a domain and the contained concepts. The Web Ontology Language (OWL; Smith et al., 2004) is a language for defining ontologies. Different approaches use OWL to define design patterns. For instance, Dietrich and Elgar (2005) present a formal pattern description based on OWL that describes the structure of design patterns. The formalized patterns are used for scanning source code for design pattern usage and thus assist in the task of documenting source code. Henninger and Ashokkumar (2006) propose a meta-model for software patterns based on an OWL infrastructure for applying patterns in the software development process. This model conceptually builds on PLML, but extends it considerably by including a description logic representation of the patterns. For this purpose, pattern attributes, such as “hasProblem”, “hasContext”, or “hasSolution”, are defined by means of OWL restrictions. Henninger presents interface patterns as an example of this approach. A tool called BORE supports interface development by offering context-sensitive information. Moreover, a formal framework for creating an interconnected pattern language for interactive systems is provided. Our approach provides tool support for adaptation patterns and includes a semantic description of parts of the patterns based on OWL. However, it does not fully formalize the semantics of patterns in OWL. Instead, OWL serves as a common formalization of different aspects of adaptive interactive systems and enables the use of semantic techniques, such as reasoning.

Model-based development (Schmidt, 2006) is based on an abstract model (called platform-independent model or PIM), which is transformed into a concrete model (called platform-specific model or PSM). For instance, Petrasch (2007) presents how to apply formalized interface patterns to model-based user interface development. The more specific patterns are the better they may be incorporated into the model-based development process. We present an integration of a set of adaptation patterns into a model-based adaptation framework. The model-based system description is automatically transferred to an OWL-based semantic layer.

2.4 Semantics in Interactive Systems

Semantic interactive systems rely on an ontology for a description of the domain and other topics, such as the interactive system or the user. Ontologies
create a formalization of a domain and describe all concepts contained in this domain. For instance, the domain of cooking consists of ingredients and recipes. Both concepts (e.g., vegetables as a group of ingredients) and instances (e.g., a tomato as a special kind of ingredient) form the ontology. The ontology is either created manually, for instance by using semantic editors such as Protégé-OWL (Knublauch et al., 2004), or by mining data automatically, for instance by means of ontology mining (e.g., Buitelaar and Ramaka, 2005). The Web Ontology Language (OWL; Smith et al., 2004) represents a common notation for the definition of ontologies. A more detailed introduction to OWL is given in Chapter 5.

In this section, we discuss the use of semantic technologies in interactive systems. On the one hand, domain knowledge encoded in ontologies allows reasoning on this data, for instance in speech-based dialog systems. On the other hand, ontologies enrich the dialog logic of interactive systems. After a review of the use of ontologies in different research projects, we investigate different ontology-based adaptation architectures.

### 2.4.1 Ontologies in Dialog Systems

A formalized representation of the domain of an interactive system facilitates different applications in dialog systems. On the one hand, a generic dialog engine accesses the domain knowledge when processing user input. On the other hand, a single ontology serves as a unified knowledge representation for different components of an interactive system. Milward and Beveridge (2003) present an approach for replacing hand-crafted dialog design with a generic dialog engine and ontological domain knowledge, which is used for different purposes. For instance, the order in which the dialog system asks the user questions is improved by incorporating the ontology. Other applications of this ontology are in speech recognition and interpretation. In addition, they present a system that supports general practitioners in the decision whether a patient should be referred to a cancer specialist. For this purpose, the system integrates medical domain knowledge encoded in an ontology.

Due to the broad area of application, ontologies have been used in multimodal systems for other purposes than dialog management. In the following, various uses of ontologies in three research projects are presented. The SmartKom project (Wahlster, 2003) created a “dialog shell” for applications that employ multimodal interaction between users and interactive systems, supporting modalities such as speech and gesture for input and speech and graphical for output. The SmartKom system employs an extensive unified ontology (Gurevych et al., 2003), which is utilized by different components of the system. Prior systems instead often used different knowledge bases for different purposes. Porzel et al. (2003) present several applications of the ontology, such as multimodal fusion, semantic coherence scoring (interpretation of the ASR results), and computing dialog coherence (interpretation in context). In addition, the ontology is used for dialog management. For this purpose, a
plan language models the actions necessary to carry out processes that are defined by the ontology. The SmartWeb project (Wahlster, 2007) builds on the SmartKom project and provides multimodal access to the semantic web. As an example application, access from a mobile device to information in the domain of a soccer championship is presented. The SmartWeb Integrated Ontology (SWIntO; Sonntag et al., 2007 and Oberle et al., 2007) forms the foundation of the dialog system. Factual knowledge is directly encoded into the ontology and may be queried by the user. Moreover, a discourse ontology specifies different kinds of multimodal interactions. In addition, semantic web services are connected to the ontology (e.g. asking for available activities). The different components of the dialog manager communicate via ontology instances. Thus, the SmartWeb project uses the ontology as a unified knowledge representation for different tasks.

The TALK research project addresses adaptive multimodal interactive systems and includes different components. First, the SAMMIE dialog manager (Becker et al., 2006), an in-car dialog system for an MP3 application, uses an ontology to model both domain knowledge as well as possible tasks. The tasks are automatically converted into a format that a plan-based discourse manager processes. A different approach was followed in the intelligent home scenario (Amores et al., 2006). A domain ontology is used by a knowledge manager component in a dialog system to provide domain reasoning. For instance, in a home automation setting for handicapped people, a query like “are there red lamps in the house” is answered using a query to the knowledge manager, which computes the requested information and offers the results to the user. The approach presented in this work also employs a single ontology, which comprises information about the interactive system, the user, and the domain. However, the adaptations are defined explicitly and are not derived by means of ontology reasoning.

### 2.4.2 Architectures for Adaptive Interactive Systems

In addition to serving as a unified knowledge representation, ontologies have been used for modeling interactive systems. For instance, Obrenović et al. (2003) employ an ontology in addition to UML in the development process of multimodal interactive systems. The high-level model description is only available at design time to provide development support and for platform mapping. However, it is not available at runtime as a knowledge representation for the interactive system itself. Our framework instead uses the model both at design time and runtime. Aragones et al. (2007) present a semantic adaptation framework, called ACUITy. A controller component mediates between a UI engine and an ontology, which comprises information on the user, the user interface, and the domain. An interface is generated from an ontology-based definition of the application. Custom-tailored interfaces are created by incorporating information about past interactions.
Sophisticated adaptation architectures have been presented in the domain of adaptive hypertext systems. Dolog and Nejdl (2003) and Henze et al. (2004) present an approach that transfers techniques from adaptive hypermedia to the semantic web. A first-order logic language adds semantic information to documents and resources, such as topics of a document and dependencies between concepts. For instance, concept B discussed in document X may require prior knowledge of concept A. A rule language called TRIPLE implements reasoning rules, for instance for deriving a set of examples for a specific topic. Since the user modeling is also defined by means of semantic triples, reasoning may be used to extract a custom set of documents or resources for a user, for instance a set of documents that discusses knowledge required as a prerequisite for the current document.

In a similar fashion, the ODAS domain ontology presented by Tran et al. (2008) is used in conjunction with adaptation rules in an adaptive hypertext portal. Tran et al. reason that rules ensure a better transparency and controllability for developers than statistical adaptation methods. The ontology provides a knowledge foundation for adaptation rules and contains different models, such as a system model, a task model, and a resource model. Carmagnola et al. (2005) present a different semantic adaptation framework for hypertext systems. It relies on different planes for the ontological representation, a specific type of knowledge, such as the user, the user’s actions, the domain, or the context. Rules in the SWRL notation define intersections of the ontological planes, since rules combine the information from these planes. In doing so, rules implement both user modeling and adaptations. User modeling rules infer information about the user and adaptation rules perform adaptations, such as removing links or adding explanations to links. For example, an adaptation rule may emphasize items for older users (“user model” plane) in the night (“context” plane) on a PDA device (“device” plane), thus exploiting information from different planes. However, these architectures do not represent adaptations in an intuitive and reusable way. Writing semantic rules requires special knowledge of semantic technologies. The adaptation framework presented in this work also relies on a unified semantic information representation, on top of which adaptations are defined. Adaptations are triggered by information from the user model, such as a prediction of the next user action. However, adaptations are defined by means of adaptation patterns to facilitate reuse. The ontology does not cover every aspect of the system, but only information required for deciding which adaptations to apply.

### 2.5 Discussion

In this chapter, we reviewed related work for adaptive interactive systems and user modeling. The chapter started with a review of different kinds of adaptive

---

2 SWRL: A Semantic Web Rule Language Combining OWL and RuleML: http://www.w3.org/Submission/SWRL/
interactive systems, such as adaptive websites, graphical user interfaces, and speech dialog systems. Next, user modeling was introduced as a prerequisite for adaptations. Both different user modeling algorithms, such as machine learning or task modeling, and user modeling architectures were presented. Thereafter, the concept of design patterns was introduced and different design pattern collections for graphical user interfaces and adaptive hypertext systems were presented. Different approaches for formalizing patterns were introduced. Finally, the use of ontologies in interactive systems and semantic adaptation architectures was presented.

However, when addressing adaptive interactive systems, such as digital TV systems or automotive dashboard systems, the approaches presented in the literature cannot be directly transferred, since adaptive interactive systems have different requirements with regard to user modeling and adaptations. For example, hypertext systems adapt documents rather than interface elements. In addition, many approaches in the literature present approaches that are limited to specific domains or types of interactive systems. Therefore, we present an approach for user modeling in adaptive interactive systems in the following chapters and define adaptations for interactive systems in a general and reusable way by means of multimodal adaptation patterns. In addition, we present a generic adaptation framework. The framework employs semantic technologies, but offers a reusable and abstract adaptation definition.
Adaptive Multimodal Interactive Systems
Bezold, M.; Minker, W.
2011, X, 184 p., Hardcover
ISBN: 978-1-4419-9709-8