Adaptation Agents: Providing Uniform Adaptations in Open Service Architectures

Markus Bylund, Annika Waern

Swedish Institute of Computer Science, Stockholm, Sweden

Abstract. We present the notion of adaptivity agents, agents that can be used to adapt an open range of services to a common user model, using a common adaptation scheme. The idea is illustrated with a specific adaptation agent, and exemplified with the KIMSAC system, an agent-based system for an open range of public information services to citizens.

The usage of adaptation agents has several advantages to existing work on user-adaptive systems. The user model is shared between services, allowing services to make use of each other's inferences about the user. The adaptation agent is intended to be owned by the user, who can inspect and control its content and decide how information can be distributed to services. This way, an important privacy issue for open service architectures is addressed. The adaptation agent also provides a common model of user adaptations to services. This way, the user needs not learn a new interaction model each time a service is added.

The functionality of the adaptation agent is determined by its *service contract* with services. The service contract specifies which adaptations will be made, and what information needs to be exchanged between agents to achieve these adaptations. We provide a first step towards a truly open service environment, by providing an analysis of the ontologies needed for communication between services and adaptation agents.

1 Introduction

The rapid development of Internet services gives rise to new challenges for user adaptive systems. Internet allows a user to access an open set of services, each with different characteristics. Each of these has a design of its own, which could include a service specific way to model users and adapt to their characteristics.

This potential diversity in user modeling characteristics pose several problems to the user. Firstly, users have to learn all ways that services are designed, and understand something about how they adapt. The individual user would be better served if all services would adapt in a similar way to the same user characteristics, or in other words, use the same adaptation model. It is even possible to envision a system, where services interact with the user through an interaction schema that the user can control, tailoring it to his or her own preferences.

Secondly, if each service maintains inferences about the user's characteristics (i.e. a user model), they must all acquire the information needed from the user. The method of acquiring the user model may vary greatly between services. Some may acquire the model automatically from the user's behavior, some may require an initialization procedure, and some may allow the user to inspect and control the model in some form that is entirely unique for the service in specific. The individual user would be better served if all services could refer to a common user model, one that the user could inspect and control in one place for all services. Also, perhaps most important of all is that it is natural to let the user own this model, not the services.

Most existing examples of user adaptive systems are entirely application specific. The adaptation models are almost invariably so, and even if there exist some general approaches for user modeling of preferences, expertise, and user task, (see e.g. BGP-MS [KOB94], Kautz [KAU86]), it turns out that practical examples often use hand crafted user modeling schemas, tailored to the specific application. As an example, compare the early GRUNDY system [RIC79] to the educational hypertext system InterBook [BRU97] and the adaptive hypertext manual PUSH [HOO96a]. All these systems provide interesting and useful adaptations, but they use completely different user modeling

schemas: in Grundy user preferences, in InterBook domain knowledge, and in PUSH the current user task.

There are some notable exceptions. Using a user modeling shell such as BGP-MS [KOB94], several services could share a common user model. BGP-MS can be used to store information about user task and user expertise in a uniform way. It can also be used to store some information about when the user model should trigger adaptations. The BGP-MS shell has a very wide range of applications, as it essentially can encode arbitrary information about the user's knowledge and tasks, as well as store general inference rules expressed in first order logic. However, it gives very little support in the development of a particular application. The rules of inference and adaptation principles all have to be entered anew for each novel application.

Another exception is the learning user model used by the Firefly[1] service [LAS94]. This user model implements a common user and user adaptation model that can be used for a number of services. This model suffers from the opposite problem: the model is very limited and supports only services of a very specific type. The same is true for InterBook, which is not one specific application but a generic tool for constructing educational adaptive hypermedia, it is very difficult to see how it can be used outside this domain.

In this paper, we describe how agent technology can be used to define adaptivity agents. An adaptivity agent maintains a user model, and uses it to adapt a well-defined part of the user-system interaction to the individual user. Adaptivity agents can be designed to be specific to a small range of services, or more general, depending on the context in which they are intended to be used. We will focus on adaptive *help* agents, agents that supply system and domain oriented help in a personal manner. The adaptivity agent can also act in the role of a user model agent, supplying user information to services that want to adapt to individual users.

1.1 Open Service Architectures

In this paper, we assume that the adaptivity agent is acting in an open service agent architecture. There are a number of properties that we impose on this architecture:

- Service agents are essentially benevolent; they are dedicated to providing a service to the user, not to cheat nor lie to them.

- Some agents collaborate to achieve their task, which means that agents must be able to communicate with each other. In particular, service agents collaborate with the adaptation agent.

- Service agents can communicate directly with the user, they need not help from the adaptation agent to do this.

Note that agents in this domain of application are very coarse-grained: each agent is both responsible for a set of domain specific tasks, and for carrying out such dialogue with the user that is necessary to achieve this task. This can be compared with other work on agent architectures, such as the OAA architecture [CHE97] where agents need not supply both a domain service and an interaction model.

---

[1] Firefly Inc. can be found at http://www.firefly.com

2 An Architecture for Adapting Help

The adaptation agent lives in an environment with two other types of agents: service agents and a presentation system agent (see *Figure 1*).

Service Agents

Service agents (SA) execute services to a user. This requires a user dialogue, and each service agent is capable of carrying out such a dialogue. Some services will also access external sources that are not available to other agents (such as legacy databases), and service agents may communicate with each other to realize a service. Service agents do not have to know exactly how to present information to a user and interpret user input, this job is delegated to the separate presentation
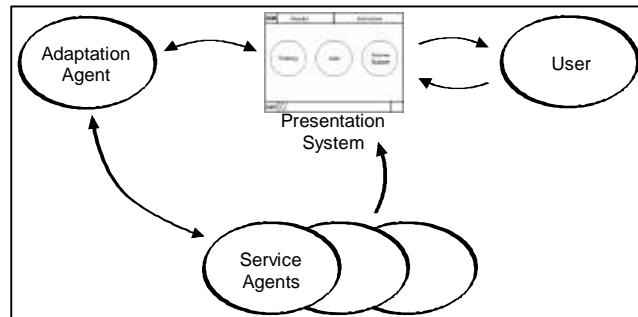
Figure 1. A schematic graph of an open agent architecture with a centralized adaptation agent.

system agent. Instead, service agents use a specific interaction ontology (interaction entities) to describe to the presentation system *what* should be presented, and how user responses should be interpreted. An example of such interaction ontology has been developed within the KIMSAC project [CHA96]. An interaction entity normally describes one round of user interaction. Namely, what the system should present and how the user can respond to this, but it can also describe longer sequences of user interaction that can be executed locally within the presentation system.

Presentation System Agent

The presentation system agent (PS) renders presentations and sends back user responses to agents. The major role of the presentation system is to generate presentations in multimedia and interpret user input. Messages going back and forth between the PS and agents are distributed in interpreted form, in KIMSAC in KQML with KIF as content language [KQML92, KIF92]. The PS can be arbitrarily complex; it can for example require multimodal interpretation and generation of text and language. In KIMSAC, we have developed a PS that realizes multimodal interaction by combinations of multimedia assets, which are pre-canned and loaded into the presentation system from a multitude of databases. Examples of such assets are specially designed buttons, forms etc., and also sound files, HTML pages, video clips and even a video conferencing channel.

Adaptivity Agent

The adaptivity agent acts as a mediator between user and service agents. The main role of the adaptivity agent is to supply a uniform adaptation model to several services. This requires that it takes on two tasks: user adaptation and user modeling. It must be able to influence the user interaction, either by adding interaction entities of its own, or by telling service agents to generate certain adaptations. It must also maintain a user model, and be able to distribute information about it to service agents. For privacy reasons, the user should own the user model; the user should have the final word about what information can be distributed to which service. The user must also be able to inspect and control the user model. Since the user model is owned by the user, it is natural to view the adaptation agent as an embryo of a user agent; an agent that represents the user in the service environment, and one that can negotiate service contracts with service agents.

Help Agent - a Special Case

In this paper, we restrict ourselves to describing a specific application of the adaptivity agent; *adaptive help agents*. The reason for this is that help agents realize an adaptation model that is domain independent, which for that reason can be applied to a very wide range of services. The basic idea is

that help is supplied by a separate entity in the interface, ideally a personal help agent. Certain communication resources, such as areas on the screen and/or the sound channel, are reserved for help functionality. The rest of the interface is not affected by the adaptivity agent, and service agents can use it in a completely service specific manner.

Other types of adaptivity agents are also possible, but in most cases they will need to be more restricted to specific types of services. Such agents are less fit to function as user agents, their role is rather to serve as a common user modeling resource for a set of similar services. One example is HyperBook [BRU97], which could be reconstructed to provide adaptations to an open set of HyperBook courses, using one and the same user model to all of them. Hyperbook mainly adapts the link structure between text pages in a restricted way, but also provides some very simple in-page adaptations. As the adaptations are uniform, an explicit adaptation model could describe them, and the user agent could inform individual services of how they should adapt their presentation. The advantage would be that the user model could be extended to carry over between several courses, rather than apply only to one course. For example, the student who already covered analogue electric technology has some knowledge that carries over to the course in digital technology.

2.1 Setup and Execution: Two Phases of Interaction

To achieve a truly open service architecture, the user agent and service agents need at least two phases of interaction: a setup phase and the actual execution phase. During the setup phase, the agents register with each other, and determine how (and if) they can collaborate. We could see this as agents agreeing on a *service contract*; determining what tasks each of them will perform, what information they need to communicate, and in what form. The contract must include:

- What is to be adapted?

- What should adaptations look like and what user characteristics should influence the adaptivity?

- How do agents interact to achieve adaptations?

- What information about the user and/or adaptations will the user agent need to inform service agents about, and vice versa?

- Are there domain specific rules of inference that the adaptation agent should use in its user model?

An example of a service contract could be an adaptive glossary. Suppose that a glossary is constructed, in which each term can be explained in different ways, depending on which other terms a user masters. The service contract could then be:

- The purpose is to obtain an adaptive glossary.

- The glossary should show different texts for novice and expert users.

- Glossary entries are shown on user initiative; service agents pick up the actual text, and the adaptivity agent supplies information about whether the user is a novice or expert.

- During execution, service agents will need to tell the adaptivity agent when a user reads a specific term. The adaptivity agent will need to tell service agents whether a user is novice or expert on a specific term.

- During initialization, service agents also tell the adaptivity agent about terms they will use, and relationships between terms. The adaptivity agent uses this information to set up its user model inferences.

The service contract can be partly built into the system design. This simplifies setup, at the expense of a less generic adaptivity support. This is the case in our example domain, the KIMSAC system, where most service contracts are built into the system design.

2.2 Ontologies

The service contract determines what information can be transferred between agents during execution. In this section, we aim to scope the kind of information that service agents and adaptivity agents may need to communicate. Outlining four ontologies, which may need to be shared between agents, does this. Note that all aspects of these ontologies are not always needed; in an actual system the service contract may be much more limited than what is described here. For example, the KIMSAC system uses only one aspect of the user model ontology, user expertise, and it does not distinguish between the adaptivity ontology and the interaction schema ontology, as each adaptation use exactly one interaction schema.

The User Model

As discussed earlier, the adaptation agent must maintain a user model, using some built-in inference schema. Obviously, the adaptation agent will need to be able to talk about this model: service agents may request information about its content, and they may also inform the adaptation agent about things they find out about the user from interacting with them.

The content of the user model can deal with one or more of four categories of user information [HOO96b]: user expertise, user cognitive or physical characteristics, user task, and user preferences.

- *User expertise.* There are two main aspects of user expertise: the user's expertise in using the system or a specific service, and the user's domain expertise. These are interrelated, since a user who uses a specific service cannot avoid learning something about its domain. But they are not the same, several services can deal with the same domain, and services can be similar enough to allow users to be experts on a new service without knowing anything about its domain. User domain expertise is often modeled in educational systems [BRU97], whereas the user's expertise in using the system is useful to model in help applications [OPP94].

    The user's level of expertise is a semi-stable property of the user, which is fairly easy to obtain information about, either by monitoring their usage of the system or asking them to rate their own level of expertise. Much work on user modeling deals only with modeling the user's expertise.

- Some aspects of *the user's Cognitive Abilities.* There are several factors of the user's cognitive abilities that can be useful to adapt to. Benyon [BEN93] showed in an early work that the selection of an optimal interaction schema could vary with the user's *spatial ability.* More obviously, systems can adapt to the reading abilities of users, or to the development stage of children.

    It is much harder to model cognitive abilities than it is to model user expertise. In particular, the system cannot ask users about it since they often won't know the answer. On the other hand, the user's cognitive abilities are fairly stable characteristics that can be estimated over a long period of interaction.

- *User's physical abilities.* These are seldom modeled, but become crucial in interfaces for physically challenged users. Inferences are really not needed to be made, users are perfectly able to state their physical abilities and disabilities, and the properties are usually quite stable.

- *The user's task* can be modeled at several levels, ranging from a guess at the immediate next action, to the user's overall goal with using the system. Help systems often maintain some kind of model of the user's task [BRE90].

    The user's task is the most volatile part of the user model, and it is sometimes not considered a part of the user model at all, but as a part of the context, identical to the system's task. This is an oversimplification though, as there may be a number of reasons for the user to use the same system service. It can sometimes be very hard to obtain a good model of the user's task [WAE96]. There exist a number of ways to infer the user's task from their actions [KAU86, WAE97], and it is also

straightforward for users to explicitly state their goals and plans - this occurs naturally in task-oriented dialogues, for example [GRO86, POL86].

■   *The user's explicit preferences.* Modeling user preferences is the simplest form of user model. Preferences are to a large extent domain specific (the fact that somebody likes sport, for example), but they can also be influenced by other user characteristics. For example, a person that prefers long explanations may be a novice, and a person who wants all texts read out may have a low reading ability (or poor eyesight).

Services based on modeling user preferences have shown quite effective, some notable examples are Firefly [LAS94] and the early GRUNDY system [RIC79]. Preferences can to some extent be inferred implicitly from what users choose to read or which services they choose to use, and users can also explicitly state their preferences.

System Context and Domain

When adapting help in a system where services can be added and removed in an open manner, it is crucial to keep track of what the user is currently doing. Important information includes the system context and the service that the user is currently working with.

The system context maps directly to a low-level generalization of a user task with the system, the user's current 'action': choosing service, specifying search parameters, browsing search results, etc. One important characteristics of the system context is that it should be independent of domain, so that the same set of actions could occur no matter which services the user is choosing from. In a system where multiple services share the same user interface (UI), the context may be mapped more or less directly to it (see section 3.4 on the KIMSAC system for an example). This implies that by knowing the state of the UI, one also knows something about the user's current task.

In addition to the domain independent context, it is obviously useful to adapt help to the domain that the user is working in. Specifying a system wide ontology for discussing services and domains can do this. Using this ontology, the agent that controls the interaction with the user is free to define which domain the user is working in at the moment. Service agents can have interests in multiple domains, but must always specify a main domain of interest.

By sharing these two ontologies, i.e. the system context and domain, the user's current actions with the system are quite precisely defined, even in very complex and large systems with lots of functionality and many service domains.

Interaction Schemas

As mentioned in section 2.1, one part of the service contract that agents must agree upon before adaptations can be made, is how agents need to interact in order to achieve adaptations. Note that this agreements needs not necessarily be taken during system initialization; it could very well be done during the design phase of the system.

Based on work by Kuhme et al. [KUH92], we divide the process of providing the user with adapted help into four phases:

■   *Initiative* – describes the entity that is responsible for taking the initiative to an adaptation, and to which entity this initiative is forwarded.

■   *Proposal* – describes the entity that proposes which adaptation should be made.

■   *Decision* – describes the entity that finally decides upon an adaptation.

■   *Execution* – describes the entity that is responsible for executing the selected adaptation, and towards which (possibly the same) entity this adaptation should be directed.

While Kuhme et al. only had to take into consideration two entities, the user and the system, we have expanded the model to allow for a whole family of systems (representing a set of service agents), and to include the adaptation agent as a separate entity. Following are three examples of interaction schemas that have been found particularly useful when adapting help.

Figure 2 shows an interaction schema that can be used for adapting active[2] system help to the user's expertise, given that the help text is only dependent on context and not on domain. As the service agents are responsible for normal interaction, they also hold the initiative to provide the user with system help. The adaptation agent receives the request for an adaptation to be made (e.g. by a message signaling a context switch). If the adaptation is dependent solely on the current context and user expertise, the contents of help texts like this can be compiled during the system design phase. In this case, the adaptivity agent performs both selection and decision. Finally, it executes the adaptation by sending the adapted help text to the presentation system, which presents it to the user.
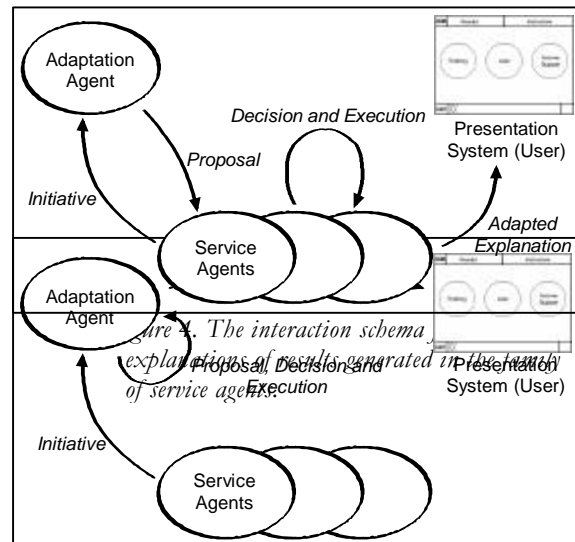

Figure 2. Interaction schemas for active system help.

Figure 3 shows an interaction schema for passive, domain and context dependent, help texts. Since this example shows the passive help mode, the user takes the initiative. The adaptation agent maintains a list of all service agents that are prepared to provide help in different situations (defined by context and domain). Upon the initiative from the user, the adaptation agent sends a request to each and every such agent, together with information specifying the user's level of expertise and possibly also other requirements on the help text (e.g. a request indicating that the text should be read aloud). It is now up to each service agent to select a number of

Figure 3. Interaction schemas for adapting user triggered domain help.

appropriate help options adapted to this very situation. Each service agent that is still ready to provide help sends one or several help topics to the adaptation agent, informing it about its decision. Since more than one service agent can find help options for the same situation (probably with different help texts), the adaptation agent is left with the opportunity to sort and thin out among all service agent's contributions, before the final set of help topics are sent to the presentation system. It is now up to the user to select a help topic. The selection triggers a message sent from the presentation system directly to the service agent responsible for the adaptation. The service agent compiles the help text, taking into consideration the adaptation to be made, and sends it back to the presentation system for displaying.
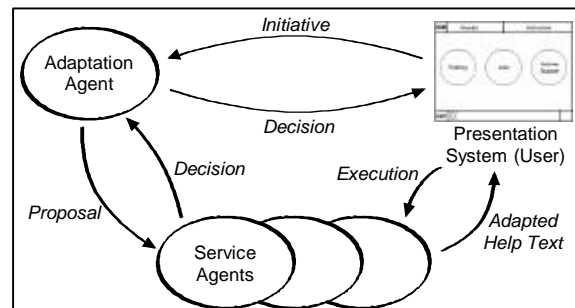
The third example of interaction schemas is illustrated in *Figure 4*, which shows how active adaptations of explanations of results can be made. Results of some kind (e.g. search results), compiled within a service agent, may need further explanation to suit users that are unfamiliar with

---

[2] *Active* in this context refers to active in the system's point of view; meaning that the system is actively providing the user with help, without requiring any actions from the user out of the normal (like for example pressing a help button). In the same manner, passive help would require the user to perform a special action.

the compilation process that leads to the result. The initiative in this example comes from the service agent that is responsible for the result; but it could also come from the user, as a request to clarify the result. Just as in the previous example, the adaptation agent proposes an adaptation by sending user model information back to the service agent. The service agent uses this information when deciding upon an adaptation and compiles a suitable explanation (i.e. the execution) that is sent to the presentation system for displaying.

Adaptations

The adaptation agent must be able to discuss which system services are to be adapted. The set of possible adaptations is design specific, and determines the functionality of a specific adaptation agent.

In the special case of an adaptive help agent, it would be involved in tasks that deal with providing help to the user. We have identified four categories of help, which are useful to adapt to a user model: system help, hints on what to do next, terminology explanations and explanations of results.

- *System help.* This category deals with help on how to use the actual system: what buttons do, how to navigate between services etc. This kind of help can be useful to adapt to the current context, user expertise, and user task. Help messages could either be supplied by services, or by the adaptivity agent. It is useful to let the adaptivity agent supply help for such system tasks that have a common design within all services, we will see an example of this in the KIMSAC design.

- *What to do next.* The system can provide hints on useful next actions, within the system or outside it (e.g. "contact an officer on phone number..."). To provide useful hints, the adaptivity agent must model the user's current task. The actual hints on what to do next are domain specific and must be supplied by service agents.

- *Explanations of difficult terminology.* Terminology explanations can be useful to adapt both to the user's current task, the current service domain, and the user's knowledge of other terms. Both the terms themselves, and the explanations of terms, are domain-specific, and must be supplied by service agents.

- *Explanation of interaction data, such as a search result.* In particular, the system should provide explanations of negative results or failed actions. This requires that the system can see that a result is negative from the user's point of view, and this will sometimes require a model of the user's task. The explanations themselves are service- or domain-specific.

3 The KIMSAC System

KIMSAC (Kiosk-based Integrated Multimedia Service Access for Citizens) [CHA96] is a project supported by EU's ACTS program. It aims to provide a flexible way for citizens to access intricate and complex information provided on public information kiosks. An open agent architecture is used to coordinate, enhance and extend service provision and the necessary information flow. The metaphor of a personal service assistant (PSA) builds on this architecture and is used to provide citizens with flexible and efficient access to services and information.

The application in focus for the KIMSAC system is to present information to unemployed people. The KIMSAC project is trial based, and during trials, kiosks are put up in social welfare and unemployment agency offices. The first trial was performed in February to April 1997, and focussed on social welfare services. The second trial is intended to start in January 1998, and includes services from social welfare and the unemployment offices in a uniform way.
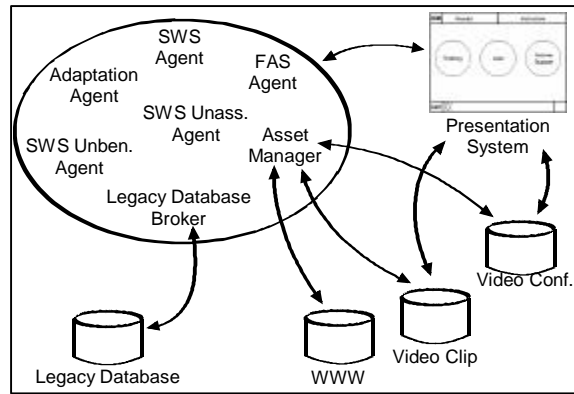

Figure 5. A generalized graph of the KIMSAC system architecture.

Figure 5 shows a somewhat simplified overview of the KIMSAC architecture. This architecture uses the triplet agent structure proposed in the previous section. Several service agents (FAS, SWS, SWS un.ben. and SWS un.ass.) all provide services to the user, and the adaptation agent is used to provide a common model of adaptations for these.

In this section, we describe the service contract between service agents and the adaptivity agent that is used within the KIMSAC system.

3.1 Service Initialization

The trial two system imposes a very strict interaction structure on all KIMSAC services, as discussed below in 3.4 System Context and Domain. This means that the service contract is to a large extent built into the design of the KIMSAC system, and need not be set up during initialization. So are for example the questions of what needs to be adapted, how agents interact, and what information should be shared between which entities already decided upon when the system is about to initialize.

The central part of the design is a search engine, in which the user can supply a set of domain specific search parameters and obtain a search result. The adaptation agent in KIMSAC assumes that all novel services adhere to this model, which is why setup is very simple in this application. The drawback is of course that the interaction structure must be known and adhered to by service designers, and that it may prove awkward for novel services.

During setup, only very few matters have to be considered, the main question being which entity can provide what type of help in which situations. This is done during initialization by having every agent that is capable of generating help in a specific situation, sending messages to the adaptivity agent and registering the type of help together with a situation specification (in this case a context or context × domain value).

3.2 Adaptations and interaction schemas

In the KIMSAC system, we have concentrated on adapting help. Four types of help are handled and each require different types of adaptations. Since each of the adaptations use a specific interaction schema, interaction schemas need not be handled explicitly in the KIMSAC ontologies. Below, we describe each adaptation together with which interaction schema it uses.

User Interface

In order to understand the purpose of these adaptations as well as how they are performed, a brief introduction to the UI is needed.

Figure 6 shows a screen shot taken from a prototype of the KIMSAC system UI. It is divided into several fixed fields, which remain the same during the whole course of execution:

- KIM – this is the area for visualization of the helper metaphor. If the user presses this icon a help menu appears.

- *H* – Contains a situation header text.

- *I* – An area that the adaptation agent uses for presenting instruction texts and system messages to the user.

- *The big rectangular area in the middle* – This is the main area for interacting with the user. The service agent currently in charge controls this part of the UI.

- *Exit button*

- *Bar with three icons* – The so-called navigation bar. This area presents a navigation history to the user, who can use it to access previous screens.

- *Marker* – This is a generic OK button. Depending on context its semantics differ (e.g. perform search, approve parameters etc.).

Figure 6-9 show four screen shots taken form a KIMSAC prototype that exemplifies the usage of the help menu (marked as KIM)[3].

Instruction Text

Instruction texts are in the first place meant to provide the user with hints about how to use the interface and details about what is currently facing them. This help is actively updated on a dedicated area in the GUI as the user navigates through the system. In addition to system help, the instruction text area is used for displaying system messages (i.e. error messages and warnings).

Instruction texts are adapted to current context and the user's expertise with current context. They use the interaction schema for active system help, as illustrated in *Figure 2*.

Where to Next?

Where to Next help gives the user hints about suitable actions to take, adapted to the
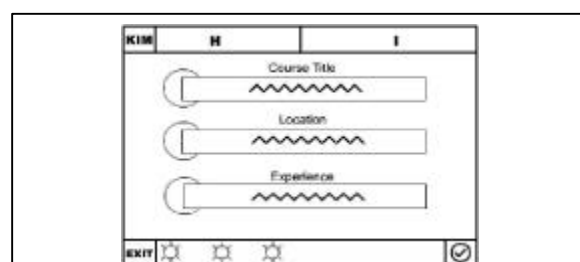

Figure 6. The KIMSAC GUI layout.

The user is about to set search parameters (the main interaction area). Instructions on how to handle the situation, adapted to the current user, can be seen in the instruction text area. The user has not yet activated the help menu (KIM).
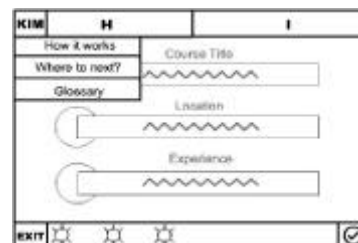

Figure 7. The KIM menu.

The user has now requested help and the help menu is showing. Three entries are showing:

- *How it works* – triggers a brief demo of the system.

- *Where to Next* – gives hints and tips about logical actions to take.

- *Glossary* – presents a small dictionary.

---

[3] To maintain readability in the small format, the actual design and layout has been somewhat simplified.

current user, context, and domain. Suitable actions includes both system actions (such as tips on where to navigate in the system) as well as concrete doings (e.g. visit a FAS officer). *Figure 6-9* illustrates how the user takes the initiative to this kind of help.

Where to Next texts are adapted using the schema for passive domain help illustrated in *Figure 3*.

Explanations

Explanations are used to explain results that the system has created for the user, e.g. the result from a job search or the outcome of a benefit calculation. In a situation like this, the user is likely to be interested in not only the result itself, but also in what caused the result. These kinds of help texts are very dependent on the domain, which is also reflected in the interaction schema that is used when adapting the explanation (see *Figure 4*). This kind of help is only adapted to the current domain and user expertise, whereas the other types of help also adapt to the current context.

Figure 8. The Where to Next sub menu.

The user selected Where to Next help and a sub menu containing three entries appeared. Choosing the glossary would have had a similar effect.

Figure 9. Execution of Where to Next help.

The user selected entry number 1 and the corresponding help text appears in a text box.

Glossary

A glossary is available to the user through the KIM menu, which contains a small set of terms; adapted to the current user expertise, context and domain. As for Where to Next help, glossary help is adapted using the schema for passive, domain oriented help (see *Figure 3*).

The glossary is built on a hierarchical structure of terms, where the explanation of a term must not contain terms that are higher up in the structure. This arrangement is used to avoid forcing users to learn all terminology at once.

3.3 The User Model

The user model in the KIMSAC system is fairly simple, only the user's expertise and a few preferences are modeled. This is mainly due to the nature of the system, which greatly limits the possibilities for keeping models of the user's goal and cognitive abilities. The user is likely to use the system on a very irregular basis, probably with a diverse set of goals in mind and over an extended period of time. Further more, the navigation structure is such, that it is nearly impossible to predict the user's task based solely on the user's navigation patterns.

These difficulties, in combination with the large set of user goals that are possible due to the openness in system domain, results in a quite limited user model.

Expertise

The expertise model is based on the user's familiarity with the system in two dimensions: system context and system domain. A function from both dimensions gives the user's expertise for every possible situation.
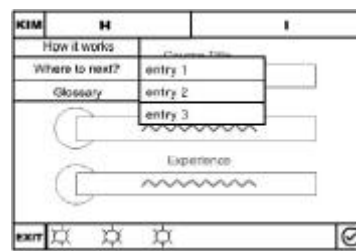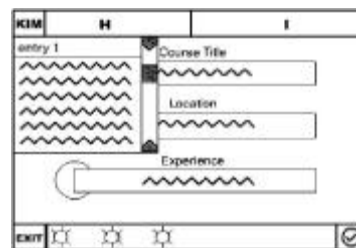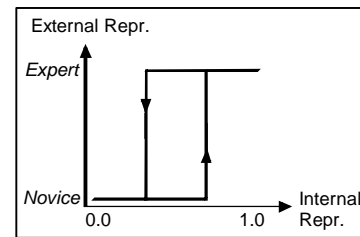
```
context_expertise(c) → ]0.0, 1.0[
 domain_expertise(d) → ]0.0, 1.0[
     expertise(c, d) → ]0.0, 1.0[
```


Figure 10. The expertise flip/flop function.

Internally to the adaptation agent, the user expertise is represented as a real number between 0 and 1. Externally however, the user expertise is represented as a function from system context and domain to a binary value from the set {novice, expert}. To avoid situations where the external expertise flips back and forth between its two values, causing the behavior of the system to change constantly, there is no direct mapping between the internal representation and the external one. Instead a flip/flop function as shown in *Figure 10* is used.

User expertise in the KIMSAC system is based on a limited set of parameters:

- *Been before* – refers to keeping track of where in the system (both domain and context wise) the user has been. At places where the user has been a number of times, the user's level of expertise is upgraded.

- *Usage of Where to Next help* – affects both the domain and the context part of the user's expertise, multiple usage of this help functionality degrades it.

- *Usage of glossary* – affects only the domain part of the user's expertise, multiple lookups in the glossary degrades the user's expertise in the current domain.

- *Optimal usage of the UI* – in situations where the user keeps navigating back and forth in the navigation structure, without ever finishing a system goal, the user is degraded in context expertise. This is also true in situations where the user keeps touching the screen outside buttons or tries to press non-button icon.

Preferences

The user can set its level of expertise in different domains to one of three different values (low, medium or high). The two extreme values low and high degrades and upgrades the user's expertise in the domain respectively. Choosing the medium value does not affect the model at all.

The user also has the possibility to turn on and off a few modalities, such as voice output of instruction texts and graphical layovers for Where to Next help.

3.4 System Context and Domain

The KIMSAC system is designed for performing searches of personalized information in different domains, domains which in one way or another may share ontologies (complete or in part). These searches are performed in the same way, using the same UI, no matter which domain the information is taken from. Furthermore, by letting the user choose whether he/she is to be identified or unidentified (using a fake profile), or just search for general information by topic, the requirements on the UI becomes somewhat different. Taken together, all these factors makes for a quite complex UI navigation structure. This structure is rendered in the KIMSAC system context graph (see *Figure 11*), which lists every possible value of the system context, as well as which paths between them are allowed.

While some nodes in the context graph are independent of domain, most are not. The *Topic* node (where the user first selects domain), as well as the *Route* and *Pin Process* nodes, are examples of domain independent nodes. They face the user with an identical behavior, no matter which domain he/she has chosen. The rest of the nodes share the surface of their behavior by providing a UI with a common syntax. This can be exemplified with the *Perform Search* button in the *Profile* node, which always performs a search that leads to some kind of result. However, how the search is done is completely dependent of the current domain, and so are the databases that are used, the type of result that can be expected etc.
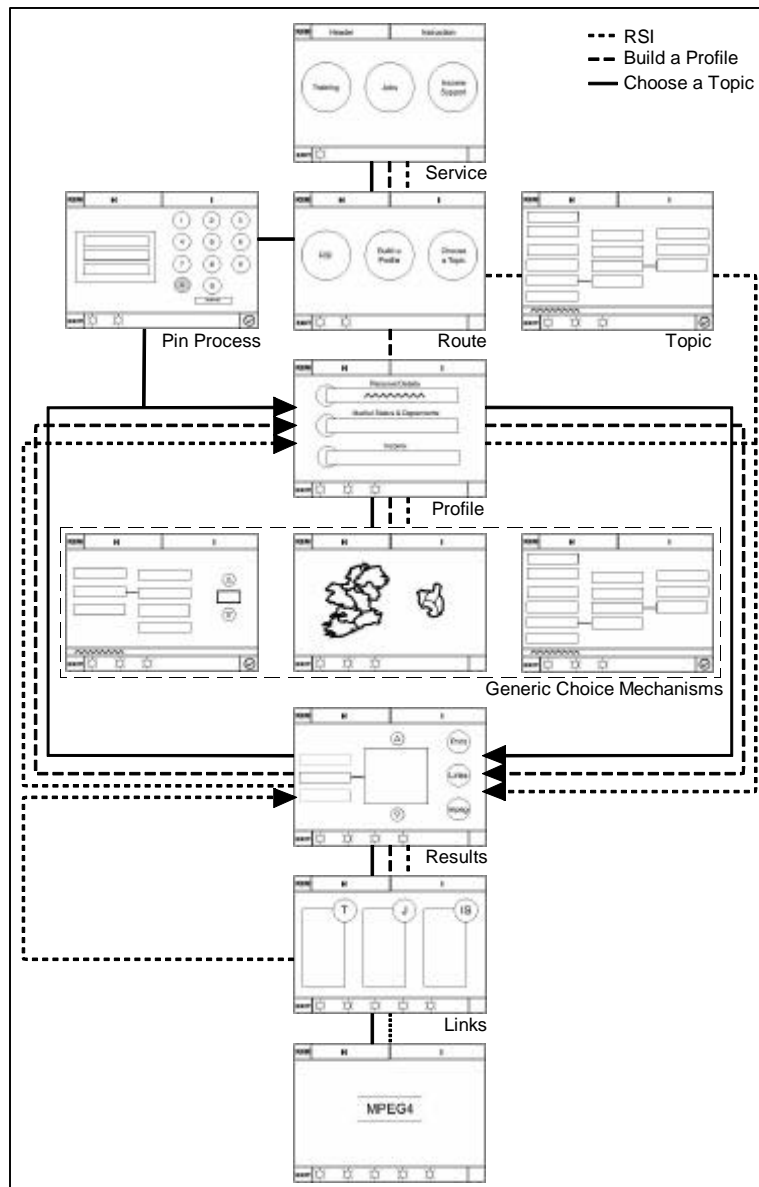
Figure 11. The Kimsac navigation structure. Each node corresponds to a unique context value.

Three domains (each partitioned into additional subdomains) have currently been implemented in the KIMSAC system, for performing searches in the fields of: Training, Jobs, and Income Support.

These three domains overlap to some extent, which makes for interesting dynamic effects. It is for example suitable to let unemployed users, that just found a job to apply for, know how their current income support is affected if they gets the position.

4 Conclusions

We have shown how to define adaptivity agents, agents that can be used to adapt an open range of services to a common user model using a common adaptation scheme. We illustrated the idea with a specific adaptation agent developed within the KIMSAC system, an agent based system for an open range of public information services to citizens.

The usage of adaptation agents has several advantages to existing work on user adaptive systems.

- The user model is shared between services, allowing services to make use of each other's inferences about the user. This could also be obtained by simply using a common user model as a system resource. But the adaptation agent is intended to be owned by the user, who is free to inspect and control its content and decide how information can be distributed to services. This way, the important privacy issue for open service architectures is addressed.

- The adaptation agent serves service agents with a common model of user adaptations. This way, the user needs not learn a new interaction model each time a new service is added. The drawback of this is that services must either be designed so that they adhere to the common model of interaction, or be able to adapt their behavior to it. Some previous user adaptive shells also achieve this; our chief examples have been Firefly and InterBook. These shells all implement one particular adaptation design, they are applicable to a restricted range of services, and could be re-implemented as specific types of adaptation agents.

- The functionality of the adaptation agent is determined by its *service contract* with service agents. The service contract specifies which adaptations will be made, and what information is needed to pass between agents to achieve these adaptations. In previous work on adaptive shells, the service contract has been implicit in the design of the system. More flexible systems require much of the service contract to be negotiated explicitly, as a new service is registered with the adaptation agent. We provided a first step towards a truly open service environment, by providing an analysis of the ontologies needed for communication between service agents and adaptation agents.

Future Work

We expect to extend the notion of adaptivity agents in several ways. To truly achieve open integration of new services, much work is needed on the runtime negotiation of service contracts. We expect a solution where none of the ontologies need to be shared to initiate negotiation, but that a contract can only be established if some, but not all, of the ontologies discussed in this paper are shared. Domain ontologies should not need to be shared at all.

Another interesting line of research is to extend the adaptivity agent to become a true user agent; the user's representative in the agent architecture. To achieve this, we must find a way for user agents to negotiate with service agents about which services the user might want, and at what price. Within a sibling project, SICS has developed the AgentBase platform [ERI95], which allows agents to perform this kind of negotiation. For our purposes, the crucial issue is to find a suitable payment scheme, information about the user may actually be a useful currency, and to find a way in which users can inspect and control the negotiation procedure without having to run it themselves.

References

[BEN93] David R. Benyon, and Diane Murray. "Developing Adaptive Systems to Fit Individual Aptitudes", *Proceedings of the 1993 International Workshop on Intelligent User Interfaces*, ACM Press, 1993

[BRE90] Joost Breuker. *EUROHELP: Developing Intelligent Help Systems*. Lawrence-Erlbaum Assoc., Hove, England, 1990.

[BRU97] Peter Brusilovsky, and Elmar Schwarz, "User as Student: Towards an Adaptive Interface for advanced Web-Based Applications", User Modeling: Proceedings of the Sixth International Conference, UM97 (pp.177-188). Vienna, New York: Springer Wien New York, 1997.

[ERI95] Joakim Eriksson, Fredrik Espinoza, Niclas Finne, Fredrik Holmgren, Sverker Janson, Niklas Kaltea, and Olle Olsson, "An Internet software platform based on SICStus Prolog", *Position Paper*, Swedish Institute of Computer Science, Kista, Sweden, 1995

[CHA97] Patricia Charlton, Yan Chen, Fredrik Espinoza, Abe Mamdani, Olle Olsson, Jeremy Pitt, Fergal Somers, and Annika Waern. "An open agent architecture supporting multimedia services on public information kiosks", *Proceedings of PAAM'97*, London, U.K, 1997.

[CHE97] Adam J. Cheyer, Luc E. Julia, David L. Martin, Douglas B. Moran, and Sangkyu Park, "Multimodal User Interfaces in the Open Agent Architecture*", 1997 International Conference on Intelligent User Interfaces*, IUI97 (pp. 61-68), Association for Computing Machinery, Inc.(ACM), 1997

[GRO86] Barbara J. Grosz and Candace L. Sidner, "Plans for Discourse", *Intentions in Communication*, 1990. MIT Press, Cambridge, Massachusets.

[HOO96b] Kristina Hook. *A Glass-Box Approach to Adaptive Hypermedia*. Ph.D. thesis, Stockholm University, Dept. of Computer and System Sciences, 1996.

[HOO96a] Kristina Hook, Jussi Karlgren, Annika Waern, Nils Dahlback, Carl-Gustav Jansson, Klas Karlgren, and Benoit Lemaire. , "A Glass-Box Approach to Adaptive Hypermedia", *User Modeling and User-Adapted Interaction*, 1996.

[KAU86] Henry A. Kautz and James F. Allen, "Generalized Plan Recognition", *Proceedings of the 5th National Conference on Artificial Intelligence*, Philadelphia, Pennsylvania 1986. AAAI, Morgan Kaufmann.

[KIF92] Richard E. Fikes, and Michael R. Genesereth, "Knowledge Interchange Format: Version 3.0, Reference Manual", Logic Group Technical Report Logic-92-1, Stanford University

[KOB94] Alfred Kobsa, Dietmar Muller, and Andreas Nill, "KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS", *Proceedings of the 4th International Conference on User Modeling*, Hyannis, Mass.*, 1994. Mitre Corp.

[KQML92] Tim Finin, Rich Fritzson and Don McKay, "A Language and Protocol to Support Intelligent Agent Interoperability" *Proceedings of the CE \& CALS Washington `92 Conference* , June 1992.

[KUH92] T. Kuhme, H. Dietrich, U. Malinowski, and M. Schneider-Hufschmidt, "Approaches to Adaptivity in User Interface Technology: Survey and Taxonomy", *Proceedings of the IFIP TC2/WG2.7 working conference on Engineering for Human-Computer Interaction*, 1992. Elsevier, North-Holland.

[LAS94] Y. Lashkari, M. Metral, and P. Maes, "Collaborative Interface Agents," *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Vol. 1, AAAI Press, Seattle, WA, August 1994

[OPP94] Reinhard Oppermann, "Adaptively Supported Adaptability", *International Journal of Human-Computer Studies* **40**:455-472, 1994

[POL86] Martha E. Pollack, "A model of Plan Inference that Distinguishes between the Beliefs of the Actors and Observers", *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics* , Columbia University, New York, 1986. ACL.

[RIC79] Elaine Rich. "User Modeling via Stereotypes", *Cognitive Science 3*, pp. 329-354, 1979.

[WAE96] Annika Waern. *Recognizing Human Plans: Issues for Plan Recognition in Human-Computer Interaction*. Ph.D. thesis, Royal Institute of Technology, Dept. of Computer and System Sciences, 1996.

[WAE97] Annika W? rn, "Local Plan Recognition in Direct Manipulation Interfaces*", Proceedings of the International Conference on Intelligent Interfaces*, Orlando, Florida, 1997.