A Glass Box Approach to Adaptive Hypermedia

Kristina Höök* Jussi Karlgren* Annika Wærn *
Nils Dahlbäck[†] Carl Gustaf Jansson Klas Karlgren[‡]
Benoît Lemaire[§]

October 4, 1995

Abstract

Utilising adaptive interface techniques in the design of systems introduces certain risks. An adaptive interface is not static, but will actively adapt to the perceived needs of the user. Unless carefully designed, these changes may lead to an unpredictable, obscure and uncontrollable interface. Therefore the design of adaptive interfaces must ensure that users can inspect the adaptivity mechanisms, and control their results. One way to do this is to rely on the user's understanding of the application and the domain, and relate the adaptivity mechanisms to domain-specific concepts. We present an example of an adaptive hypertext help system POP, which is being built according to these principles, and discuss the design considerations and empirical findings that lead to this design.

Keywords: adaptive hypermedia, plan inference, multimodality, user modelling

1 Introduction

Utilising adaptive interface techniques in interactive systems introduces certain risks. An adaptive interface is not static, but will actively adapt to the perceived needs of the user. Unless carefully designed, adaptation and the changes it occasions may lead to an unpredictable, obscure and uncontrollable interface.

As is frequently pointed out it is important that users feel to be in control of the systems they work with. This becomes increasingly important when systems act autonomously: e.g. read and sort our mail, choose which news items to read, book our meetings. Systems that act too independently, e.g. knowledge-based systems

*Swedish Institute of Computer Science. The authors can be reached by electronic mail to `push@sics.se`. http://www.sics.se/humle/

[†]Linköping University

[‡]Stockholm University

[§]Now University of Grenoble

or systems with adaptive interfaces have not always been acceptable to users (Berry and Broadbent, 1986; Meyer, 1994: Vassileva 1994). One reason for this is that complex problem solving should not be implemented as a system task alone, but rather be approached as a joint task of the system and the user, to be solved in interaction (Pollack et al 1982).

Giving users a sense of control can be achieved only if the systems' internal workings are *transparent* to users or if the systems' actions are *predictable* to users. Adaptive interfaces sometimes make very bold assumptions of user characteristics, and adapt accordingly. It is not to be expected that such adaptation always will be correct. We believe that virtually all adaptive interfaces will at times make mistakes resulting in erroneous adaptations (see e.g. Kay, 1994). This is a strong argument for ensuring that an adaptive interface provides mechanisms for control, again on an appropriate level of abstraction, in order not to confuse the user with technical details of the adaptivity mechanism.

Transparency gives the user a view of the internal workings of the system. Ideally, users should see a system as a glass box, within which the lower level components act as black boxes (du Boulay et al., 1981; Karlgren et al., 1994). The view given can be very abstract: Maes (1994), e.g., represents the internal state of a personal meeting booking agent as a facial expression – the form of visualisation and the level of abstraction must be chosen carefully in order not to lead the users' expectations astray.

Predictability can be more difficult to achieve in an adaptive interface. Meyer (1994) describes this requirement as a requirement of a stable relation between stimuli and response, that is, the same input in the same context (or what the user perceives as being the same input in the same context) should always give the same output. There is an inherent contradiction between this requirement and the general idea of adaptive interfaces, that of changing presentations according to the perceived needs of the user. The design of adaptive interfaces must thus aim at achieving predictability in some alternative way than a strict adherence to the stable stimuli – response requirement. One solution is to split the interface into a stable unchangeable component, which is carefully designed to be predictable, and one which does change, as, e.g. an interface agent.

In this paper, we present an example of an information seeking application, for which we have developed an user-adaptive hypertext solution. This solution does not rely on an agent metaphor to achieve transparency, predictability and control in an adaptive interface, but instead focuses on finding a *domain specific* design of the interaction. The (generic) adaptive functionalitites are made visible to users by means of language and functionalitites inherent in the particular domain of information. The example application, an adaptive information seeking assistant, enhances interaction and control through multimodal interaction including unconstrained text input, and utilises task adaptation for information selections. Task adaptation is made controllable both through explicit task selection and user-verified plan recognition. We briefly describe each of these mechanisms, and describe how they are designed to deal with transparency, predictability and control.

In general, domain-dependent solutions allow users to communicate with the system in the domain language. Most other approaches to transparent and controllable user modelling are generic. An example are the adaptive prompts by Kuhme et al. (1993). This design allows a user, or a program analyst, to tailor the mechanisms for adaptivity, but in order to do this, the user must learn a new, complex vocabulary which distinguishes between sets of terms such as e.g. "goal", "action", and "interaction". Furthermore, it is unlikely that users will be able to predict the *effects* of this tailoring, as the different parameters interact in a complex way to achieve adaptiveness. Still, the architecture of our solution aims to separate the domain specific information from the general inference mechanisms, and both from the interface design. This should make it possible to transfer our solution to other information seeking applications with little effort.

We first describe our studies which underlie the design (2), followed by a description of the non-adaptive hypertext system(3. We then describe the multimodal interface chosen (4.1) and the adaptivitiy mechanisms (4.3 and 4.4).

Project Background

The Plan- and User Sensitive Help (PUSH) project aims at developing and testing intelligent help solutions to information seeking tasks. The domain is a method for development of large telecommunications software systems, SDP-TA. The method itself is documented in a huge information space, about 500 documents consisting of 5 - 20 pages each SDP-TA is structured in *objects* and *processes*. Objects denote results produced when applying the method, e.g. code, documentation and similar. The work performed in a project using SDP-TA is organised in processes, e.g. defining the requirements on the system, performing design, etc. Processes are related to one another, objects are also related to one another, and finally, objects are produced and refined in certain processes.

The design solution discussed in this paper is currently being implemented, and a prototype named POP (PUSH Operational Prototype) runs on SUN workstations. POP is implemented in an object-oriented extension of Sicstus Prolog; Sicstus Objects (SICStus User's Manual, 1995).

2 Studies which underlie the design

Adaptivity in hypermedia is proposed as a means to meet users with different needs, background knowledge, interaction style, and cognitive characteristics (Kobsa et al 1994). In order to do so, we must have ways of knowing more about the targeted group of users and their needs, related to the domain of the information system that we are developing.

Acquiring knowledge about users in order to build an intelligent help system is similar to, but distinct from the problem of gathering knowledge for knowledge-based systems: when developing knowledge-based systems, the focus is on the *domain expert* and the problem of how to "extract" the expert's knowledge, whereas in the case at hand, where we design an adaptive information system, we must focus on the *users* and their needs. While the area of knowledge-based systems has had a fair amount of research done on system development methods, methodology for domain analysis from the adaptive system perspective remains largely a field in its infancy. Often claims are made about user needs that have very little to do with what will actually be of real use to users in a particular domain. For instance, a common claim is that explanations must avoid concepts unfamiliar to the particular user reading the text, or that they should be provided with less details (Kobsa et al 1994). This may be true in some cases, but not in the general case: for instance, if the user is attempting to *learn* more about a particular issue, the contrary might just as well be true (Höök 1995).

The most challenging goal in the PUSH project has been to find which adaptive techniques in fact do improve the interaction, given the side constraint that the knowledge representation and reasoning strategies must be scalable and easy to update, to accommodate new releases and changes in the very large target system.

Knowledge Acquisition Methods Utilised in PUSH

The PUSH project was fortunate in that SDP-TA already is well documented in a large on-line manual, and that the project was given access to a large number of users for whom the documentation of SDP-TA was a crucial issue in their daily work situation. This has enabled us to set up a series of studies where we could find what users' needs are, what support they have currently, what supports need to be added, when users turn to the manual, etc. In this work, we have followed the principles of Cognitive Task Analysis (Roth and Woods, 1989). We first tried to understand the problem area, focusing on the types of tasks that users typically were involved in when addressing the manual information. This was done to avoid ending up with a solution that would only solve an example scenario, but not scale up to the whole problem area.

During our initial work, we found several problems that could not be solved by a help system on the process itself. The main reason was that the user's specific problems arise from their current *project task* rather than from difficulties in understanding the development method. Essentially, the tasks of different projects using SDP-TA are all unique. We also found some problems with SDP-TA that could not be addressed by a help system alone. Some of these problems originated from the SDP-TA process itself, others from a mismatch between the process and the organisation in which it was applied. This is not surprising; it is a frequent result in the development of help systems, that you end up wanting to make changes to the underlying system or the environment in which it is applied (Breuker 1990). These findings have proved useful in the subsequent development of SDP-TA, but they lie outside the scope of the help system – the help system is supposed to give help on the method as it is, not compensate for it. For these reason, our latter studies have concentrated on extracting the users *information seeking needs*, as these are the primary reason for addressing the help system, as opposed to their project tasks.

Our initial interviews (we interviewed about 20 persons) did point to a number of problems that were directly connected to the method and its documentation. When

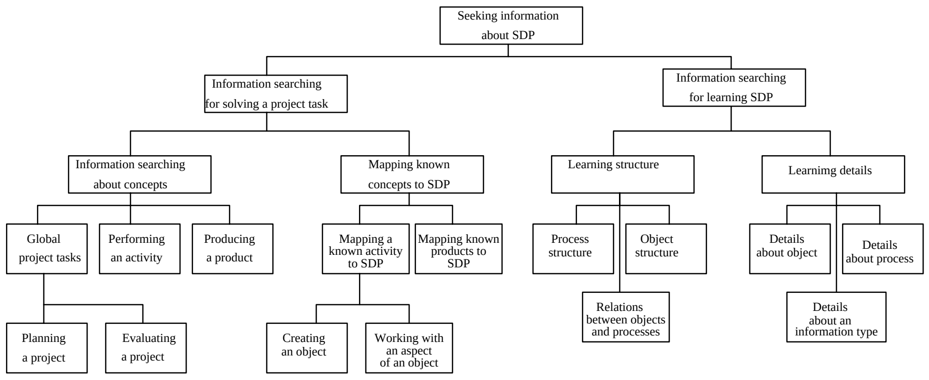

Seeking information about SDP
Information searching for solving a project task
Information searching for learning SDP
Information searching about concepts
Mapping known concepts to SDP
Learning structure
Learnimg details
Global project tasks
Performing an activity
Producing a product
Mapping a known activity to SDP
Mapping known products to SDP
Process structure
Object structure
Details about object
Details about process
Relations between objects and processes
Details about an information type
Planning a project
Evaluating a project
Creating an object
Working with an aspect of an object

Figure 1: Parts of the domain-dependant task hierarchy of information seeking needs of SDP-TA users.

using help, users felt a problem of information overload, they had difficulties in finding information, and once they found information they had problems in interpreting and making use of the text and graphs they had found. User suffered from the classical bind of too much and yet too little information! (McDermid and Ereback, 1994).

We then went on to the more focused study on users information seeking needs in their daily work. This study was based on interactions with 23 users in their daily work during a quite long period of time. The purpose was both to collect a "corpus" of questions from users, to test which answers would have been most satisfying and also to look beyond the surface level and extract a hierarhcy of *information seeking tasks*, depicted in figure 1 (Bladh and Höök, 1995).

In the task hierarchy, we may observe that learning about SDP constitutes quite a large part of the tasks. This caused us to make a study on the novices and experts conceptual understanding of the ideas behind SDP-TA (18 novices and 10 experts paticipated in this study). The purpose was both to see whether we could apply any theories of learning that could influence the design of our help system, but also to get a general grip of how difficult it is to grasp this domain for the users. The theory of learning that we applied was from Micki Chi's work (Chi et al., 1994), and it turned out to be quite hard to apply. Our hypothesis is that her theory works well for learning "natural concepts", while the more abstract concepts in SDP-TA are not as easy to classify according to her theory.

We did find that there were quite some confusion about fundamental concepts in SDP-TA among the novices. The primary explanation for this is that a lot of the concepts are fuzzy and users are expected to get an intuition about the whole concept of object-oriented thinking rather than being able to find precise, formal definitions of the concepts. We cater for some of this concept communication in our POP system.

Finally, we have made a set of evaluative studies on some of our design choices. In particular, we have produced a set of explanations that differ in how they meet the perceived needs of users and tasks they engage in, and tested these explanations on users (Höök, 1995), also described below. Seven persons participated in the first of our controlled evaluative studies. We have also continously shown our prototypes to real users.

3 The basic system

The knowledge acquisition phase has been intertwined with the design and development of a prototype help assistant, the POP system. In order to understand how our adapivity and multimodal interface has been designed, we first need to describe the basic hypertext system that we have developed. First, we describe an example scenario (without adaptivity) and the basic interaction principles of POP. We then describe the underlying knowledge representation with information entities.

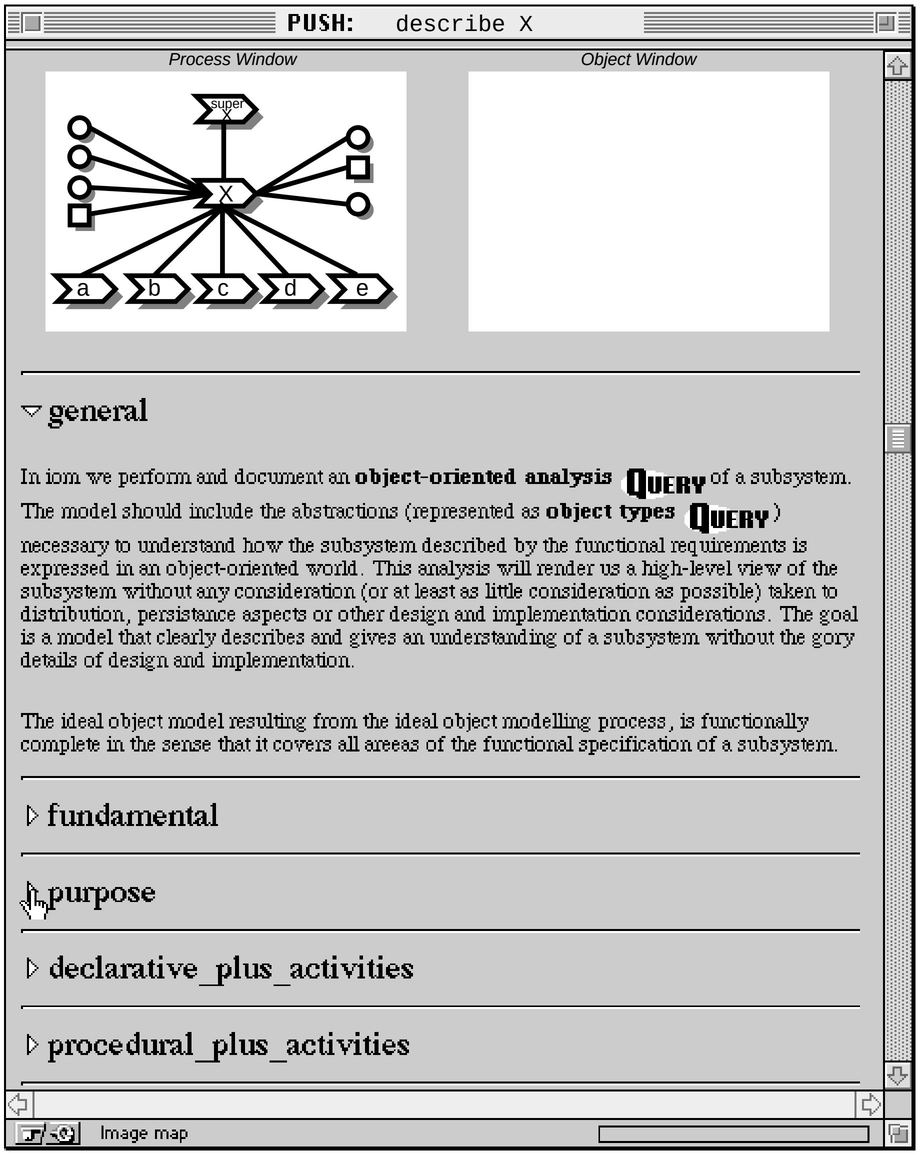
3.1 Example scenario in the POP Prototype

A software developer at Ellemtel that has been assigned the task of producing a specification of a subsystem in the software her project is developing. The software developer has quite extensive knowledge of the application domain, but is fairly unfamiliar with the SDP-TA method. Her project manager has informed the project members briefly about where in the method they should start working, let us name that part of the method "process X"[1]. She enters the on-line help system, POP, through entering Netscape (a World Wide Web (www) viewer, ref) and enters an unspecific question about process X: "describe process X". The system then provides her with some information about process X, see figure 2, in what we shall name an *answer page*. The answer page consists of both some graphics and also some text under different headings.

As shown in the graphs, the process is part of the general process SuperX, and process X has seven activities: a, b, c, d, and e. There is a set of input objects and another set of output objects. This information helps the software developer to see how the method is structured with the focus on process X. The user can click on any of the symbols causing the graphic displays to change, putting the object that was clicked on in the centre. She may, for example, click on one of the input objects. This object is then displayed in the centre of the object graphics window and any objects which are related to it are also shown in that window.

In the page, there is also some textual information. Some fairly basic information about the process X is provided and the underlying purpose behind process X is also shown. There are also some headers without associated texts. These indicate that there is some more information available. Our user decides to click on the header titled "Purpose". A description about the underlying purpose behind the process X is is inserted into the page below the header she clicked on.

[1]The SDP-TA method is proprietary to Ellemtel. Only minimal information about the method itself is given in this paper.



▽ general

In iom we perform and document an **object-oriented analysis** QUERY of a subsystem. The model should include the abstractions (represented as **object types** QUERY) necessary to understand how the subsystem described by the functional requirements is expressed in an object-oriented world. This analysis will render us a high-level view of the subsystem without any consideration (or at least as little consideration as possible) taken to distribution, persistance aspects or other design and implementation considerations. The goal is a model that clearly describes and gives an understanding of a subsystem without the gory details of design and implementation.

The ideal object model resulting from the ideal object modelling process, is functionally complete in the sense that it covers all areeas of the functional specification of a subsystem.

▷ fundamental

▷ purpose

▷ declarative_plus_activities

▷ procedural_plus_activities

Figure 2: The interface of the current version of POP.

When reading through the text some terms are quite unfamiliar to our software developer, she gets confused by: "perform and document an object-oriented analysis?". This text is displayed in bold style, indicating that it is possible to ask follow-up questions on those concepts, which she does, and the information is also inserted into the page, see figure 3. The follow-up questions are currently available trough first "opening" the bold word (i.e. clicking on the Query-symbol next to the hotword (Kobsa et al., 1994)), which makes a set of alternative follow-up questions available marked as links. (Once www allows it, we aim to turn these into pop-up menues.)

When our user feels satisfied with the information about process X, she can turn to other processes in SDP-TA, or gain information about the objects in SDP-TA. The new object or process is then presented in it's own, dynamically generated, answer page in www.

In summary we can observe that:

- the user may enter questions and follow-up questions and move on to other processes or objects.
- the user may also choose to navigate through the graphs.
- the information was provided in both text and graphics - some of it in both modes simultaneously, whereas other information was shown in only one mode.
- the user can manipulate an answer through opening and closing subsections, manipulating the graphics and asking follow-up questions which are placed in a context.

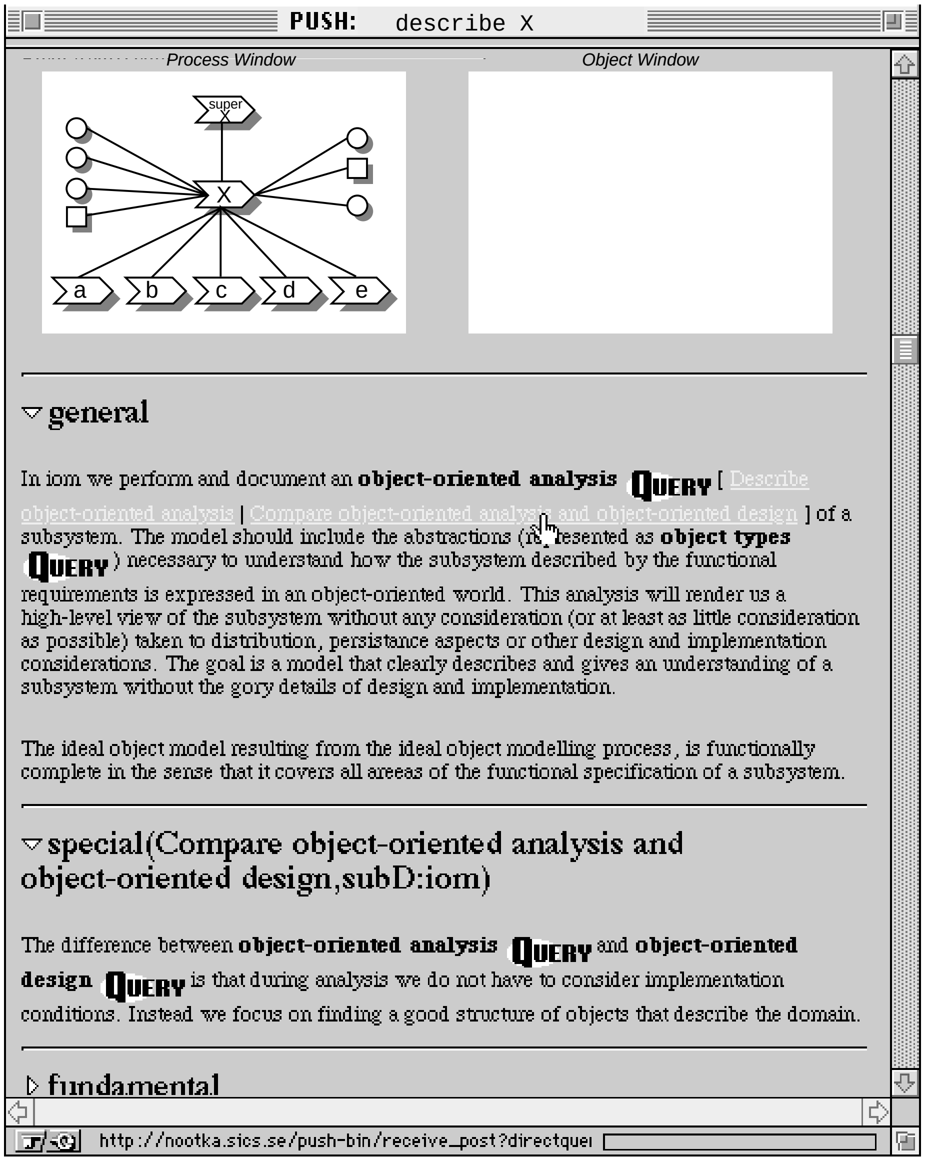
In summary, the basic interaction mechanism of POP combines navigation and search (Lemaire et al., 1994) through a hypertext space.

3.2 Hypertext as Information Entities

Let is now discuss how the information is organised in the underlying knowledge representation.

As we could see from the example, the processes and objects and their relationships in SDP-TA do not by themselves convey much information. In addition to this structure, a lot of mainly textual information is available. This information may concern why a certain object is produced, how a process is carried out, hints on how to produce good quality output, entry and exit criteria and similar issues. There will also be information that is not tightly related to any single object, such as comparisons between similar process steps with slightly different purposes, or explanations of commonly used terms (as "object- oriented analysis" in our example above).

In the POP prototype, we divide all available information about processes and objects into typed *information entities*. Each information entity is a piece of hypertext (possibly including a static picture), that in turn can contain semantic links to related information. Links are formulated as follow-up questions to the text. For example,



▽ general

In iom we perform and document an **object-oriented analysis** QUERY [Describe object-oriented analysis | Compare object-oriented analysis and object-oriented design] of a subsystem. The model should include the abstractions (represented as **object types** QUERY) necessary to understand how the subsystem described by the functional requirements is expressed in an object-oriented world. This analysis will render us a high-level view of the subsystem without any consideration (or at least as little consideration as possible) taken to distribution, persistance aspects or other design and implementation considerations. The goal is a model that clearly describes and gives an understanding of a subsystem without the gory details of design and implementation.

The ideal object model resulting from the ideal object modelling process, is functionally complete in the sense that it covers all areeas of the functional specification of a subsystem.

▽ special(Compare object-oriented analysis and object-oriented design,subD:iom)

The difference between **object-oriented analysis** QUERY and **object-oriented design** QUERY is that during analysis we do not have to consider implementation conditions. Instead we focus on finding a good structure of objects that describe the domain.

▷ fundamental

http://nootka.sics.se/push-bin/receive_post?directquer

Figure 3: The interface after “opening” the purpose-description and posing a follow-up query on “object-oriented analysis”.

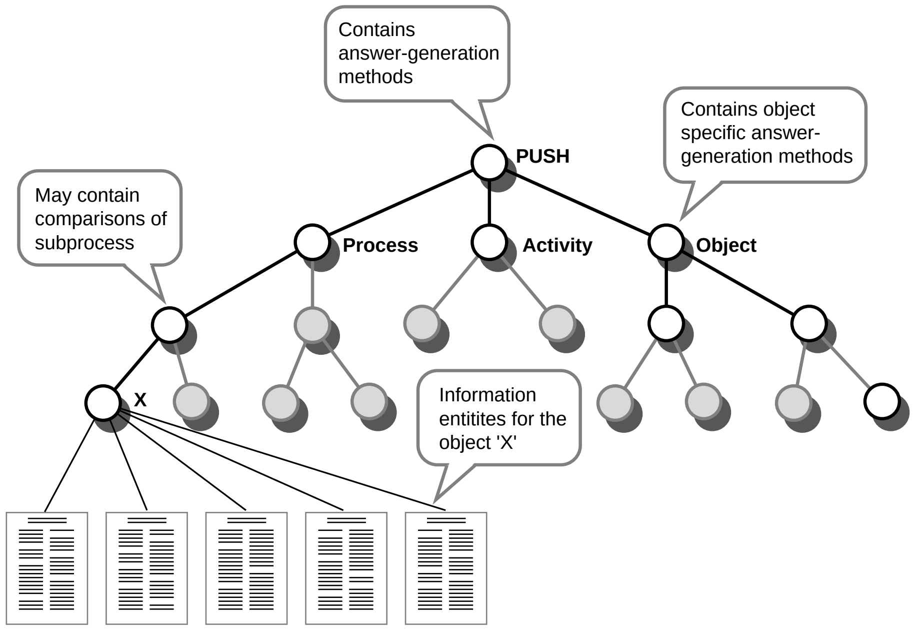

Contains answer-generation methods
Contains object specific answer-generation methods
PUSH
May contain comparisons of subprocess
Process
Activity
Object
X
Information entitites for the object 'X'

Figure 4: The knowledge representation and explanation operators in POP.

a short description of a process step may contain a link to a comparison between this process step and another. The object relationships are also represented as information entities, where the type denotes the type of relationship: for example, an information entity such as “input objects” is a list of the names of the input objects, associated with a set of possible queries for each object.

Since SDP-TA is structured in an object-oriented fashion, it was a natural choice for us to structure the database as a hierarchy of software objects where the information entities constitute attributes of the software objects, see figure 4. In this hierarchy, we can associate some information entities as comparisons to generic classes (or superobjects) to be inherited by all objects sharing the same information. Similarily, the explanation operators consistute inherited methods of each software object. Each software object therefor has knowledge about what it “knows” about itself and can explain itself through using an inherited explanation method (Lemaire 1995).

When a question is posed about one of the objects in our knowledge base, the object itself will know which information entities it should return as an answer, and it will also know how to provide the necessary information that is used for constructing the graphs in the interface, i.e. the graphs are not predefined pictures, but generated on the spot. In fact, the whole answer page is generated dynamically as a response to the question.

4 Multimodal interaction and adaptivity

Our hypertext system is carefully designed to enhance the understanding of the SDP-TA domain and the structure of it. However, the empirical investigations point towards difficulties that are not addressed by this solution alone. Some users immediately get lost and do not know where to search or what to ask, other users will need a specific explanation style to understand the information, and finally users in general suffer from "getting more than they asked for".

In order to support users with only a vague understanding of the content and organisation of the information system, we provide the possibility to pose vague questions using text input. Users with very specific information needs are also helped by the free text input possibilities. This solution is further described in 4.1.

In order to deal with the need for different types of explanations in different situations, we made a careful analysis of which information entities should exist in the database. For example, we introduced three different ways to describe activities in a process: either they can just be listed by name, or we can briefly describe each in a declarative, factual style, or, finally we can provide quite long, procedural, description that are full of hints on how to think and what to do (Höök, 1995). Each description style is represented by a hypertext information entity. We describe our work in this area in 4.2.

In order to deal with the information overflow, answers are tailored using knowledge about the user's current task. Each information entity has an associated list of tasks which is provides to fulfilling. Our system can then choose which information entities are best fitted given that we know which task the user is performing. How to know about the users task and how we provide this kind of adaptivity is further described in 4.3 and 4.4.

In all of these solutions, it is very important to allow the user to inspect and control the functionality. In the following sections, we shall describe in detail how this is achieved in POP.

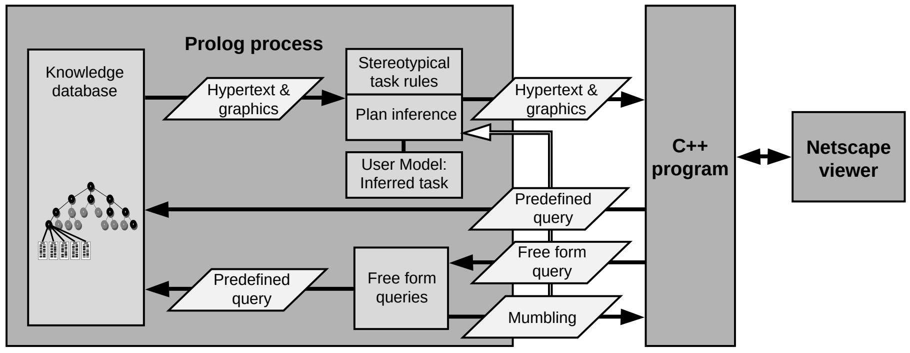


Figure 5: The architecture of the POP system.

In figure 5 we see a picture of the basic architecture of POP. The knowledge database consists of the objects of the domain and their information entities. It also has all the methods for explanations inherited down into each domain object. The free form queries are transformed into the set of standard queries that the system is able to understand. This sometimes require a dialogue - this is further dicussed below. The answer in turn, is altered by the inferred task of the user. This process might affect the explanation so that certain information entities are hidden and other are opened - the task inference and adaptation of explanation is further described below.

The whole result of the query is finally sent to a C++ program which turns our extended html-format (with information about what is "hidden" in the page and which follow-up menues to display if requested) into a proper www-page which is shown to the user in, e.g. Netscape.

4.1 Issues in Multimodal Interaction

The interface is multimodal – meaning that it produces both graphics and text, and accepts both text, menu choice, and pointing and selection as input. Text and graphics (or, indeed, any combination of abstract and direct access representations) complement each other, in the sense that tasks of different types require different modalities (Cohen, 1992), and that users have varying preferences for modalities or differing capacity to make use of them (Bretan, 1995).

Text and language in an interface is often taken in opposition to the direct manipulation paradigm. The three main points noted of direct manipulation interfaces are (Shneiderman, 1983):

1) continuous representation of the object of interest,

2) physical actions or labelled button presses instead of complex syntax, and

3) rapid incremental reversible operations, whose impact on the object of interest is immediately visible.

For points one and three, pictures and graphs show their strongest side: pictures or graphics can achieve the persistence sought for. At the same time, points one and three do not in any way contradict the possibility of using text or other language input – indeed, any interface at all, be it language based or point-and-click based would do well to follow the principles. We will use natural language, in our case typewritten text, as one of the mechanisms of interaction, thus relaxing the constraints posed by Shneiderman's second point, but continuing to observe points one and three.

Multimodal systems with strong cross-modal interactivity – where text input is but a component in an interface which otherwise behaves along Shneiderman's principles – show very encouraging results (Biermann et al. 1983, Bos et al. 1994, Capindale and Crawford 1990).

We find that in SDP-TA, which has structures that can be presented in graph form with little effort, some of the relations between objects and processes are of a type

```
I have no clear picture of which .hh files and .cc
files belong to an II or an OU respectively.
POP interprets your query as:
1 compare II and OU.
2 describe hh.
3 describe cc.
```

Figure 6: A query which cannot be generalised without extensive inferencing. POP simply catches the referents and proposes queries to describes them. In this case, where multiple queries result, the user will be given a chance to choose which ones to process.

that need textual explication for users. Any relation that is semantically complex will need elucidation difficult to achieve in graph form. In addition, the user studies have also indicated that users do not want to rely on pointing and clicking to thread their way through a hypertext space, which is the way the manual is organised at present: they want more direct goal-oriented mechanisms, especially if they are not exploring the information but looking up something concrete. Users in the study expressed frustration at not finding a correct answer a few clicks away: hypertext somehow raises the expectations of users to the point that the information they seek should be immediately available. By not only using ’point and click’ but also allow the user to make use of the abstractions possible in natural language we can at least partly overcome these drawbacks.

Another reason to include a free-form text input is that the standard queries only have been designed to answer a specific set of foreseen questions, and users of a help system almost by definition are at odds with the information structure of the target system – e.g. when the user really is novel to the subject and knows too little of the domain to be able to use the predefined queries, or when the user has a specific information need which has simply not been anticipated in the set of queries. Affording text input provides these users with a way to enter queries and needs that were unexpected.

The system is designed to accept all queries, and map them on to the set of internal query templates. If no exact match is found, the least general query that is consistent with the query representation will be chosen: the entities that have been identified as reasonable references are assumed to be the topic of the query, and a general query which includes the identified references is constructed. This is displayed to the user, who thus will see what assumptions the system is capable of making together with the answer. An example is shown in figure 6. This means that users are always given answers, but that the answers may well be inconsistent with the gist of their queries; this will still give the user an idea of what POP handles, better than error messages would.

A well-known problem for natural language interfaces is reference resolution (Cohen, 1992). Our system allows the user to refer to previously mentioned entities by using pronouns and definite references: “Compare it to IOM”. This is done keeping track of a *focus stack* where referents are stored together with a score indicating how salient for subsequent mention they are. This salience score is composed by a weighted sum of scores from recent actions, linguistic as well as non-linguistic. The system will certainly make inappropriate weightings at times – but as Bos et al. (1994) found, users will accept the occasional error if the error itself is understandable. The top elements of the focus stack are displayed to the user to make focusing explicit (Karlgren et al., 1995; Bretan, 1995, Bretan et al., 1995), which adds to the permanence of referents, and displays the results of focus calculation. Together with the dialogue history this gives the users transparency to the system view of the discourse situation, thus addressing Cohen's point above.

4.2 Adaptation to the User's Information Seeking Task

The main problem we faced in the PUSH application was the vast amount of information available. The information connected to a single domain object may well exceed more than twenty pages. In the initial studies, we found that users were likely to read only the first page of information they found relevant to their purposes (Bladh and Höök, 1995). This made it necessary to find rich mechanisms for information extraction, so that the individual user initially would be presented with the information that was deemed most relevant for him or her.

In line with the work of Kaplan (Kaplan et al 1993), we can identify several possible "parameters" which could be relevant for determining whether a certain piece of information is relevant to a particular user. The object-oriented structure of the domain itself is such a parameter: if the user is reading about a specific object, he or she may be interested in information about objects related to this one. The knowledge base is structured according to the domain, into objects, information entities belonging to objects, and semantic links between information entities.

One potential usage of this knowledge structure would be to construct all queries to be very specific (some of them represented by the navigational manœuvres in the interface), so that only a small fraction of all information would be relevant as an answer to any query. However, since the amount of information is so large, this solution would run the risk of users losing track of the *structure* of the domain in their search for the right information – users may get "lost in hyperspace". Furthermore, this solution will not provide users with any sense of *how much* information they have found and what is still available.

Instead, in answers to precise queries about a specific object, we include all information around the object, although the information that is not directly relevant for the query is "hidden" from the user's immediate view: in the example above, 3.1, the hidden parts are the ones under the headings that are not yet opened, and also the follow-up questions associated with hotwords. This way, users will know that if they decide to open all the hidden information in a page, they will have accessed all information related to this object. Furthermore, we allow users to ask "open" questions, such as "`compare` **object1** `and` **object2**". The answers to these imprecise queries will be very large, but if users read the entire answer, they can be certain of having exhausted the information around a certain subject (represented by the query).

An important concern of ours has been to define what textual information entities each domain object should have. It is important to plan for information entities that cover all available information in the current system, as well as the additional needs of users that we found during our knowledge acquisition work. But another concern is that the domain experts who will eventually write these texts, must be able to understand the purpose of each information entity, to write text answers to the needs of a user addressing this entity. In other words, several different writers must *keep user models in their heads* while writing the text.

Initially we started out with a small set of identified information entities, and a small set of stereotypical user models, intended to capture a variation of user backgrounds and user roles. Our idea was that each information entity was to be written in several versions, one for each user stereotype. However, this approach is unfeasible in practice, since it grows intractable in this large domain. Furthermore, it proved very hard to grasp what a stereotype really meant, and how it would affect the formulations of a certain information entity. The stereotypes also modelled aspects of user background that turned out not to affect the discourse. Much the same critique was put forward by John Self (Self, 1988) in the area of student modelling in tutoring systems. Self expresses this as "don't diagnose what you cannot treat". We can paraphrase this in the area of user modelling as "don't model user characteristics that do not affect interaction": in our case this meant that we should not try and write different information entities when in fact, they did not turn out to be sufficiently different.

Instead, we have chosen to further refine our division of information, into information entities that describe some aspect of an object or a concept, where the description is geared to a specific purpose. It turns out that the classification by task subsumes both the level of expertise and the role: for example, a person who is trying to "learn structure" is a novice (at least in the area of SDP-TA which he or she is working with), and a person who is "planning a project" at least nominally is a project leader.

This structure has proven very useful. Each information entity is large enough to be self-contained and internally consistent, which makes it possible to combine several of them into one answer. Coherence is further enhanced by only selecting such texts for presentation, that have been written with either the same or closely related tasks in mind. The structure also provides flexibility for design and maintenance of the help assistant: if a novel information need is detected, this can be handled either by defining a novel combination of information entities relevant to this need, or through adding entirely new information entities geared specifically to the novel need.

The different typical task descriptions will in a specific situation determine the choice of information entities to display. For instance, an explanation of a process for a user engaged in a "Follow activity" task will display a "How To Do It" text (a procedural description replete with practical tips); whereas an explanation for a user engaged in "Product Planning" will include an "Activities" text which is a more declarative description of a process, similar to the difference proposed by Paris (1988).

The mapping between tasks, queries and information entities is rule-based, see figure 7.

Learning structure $\longrightarrow$

Fundamental, Purpose, Activities, Input objects, Output objects,
Information model, Relations to other processes, Simple example

Projectplanning $\longrightarrow$

Project planning, Activities, Information model, Simple example

Producing a product $\longrightarrow$

General, How to do it, Release, Input objects, Output objects, Entry,
Exit, Information model, Advanced example, Frequently asked questions

Reverse engineering $\longrightarrow$

Information model, Activities, Release

Figure 7: Four rules for the query “`describe` **process**”, describing which information entities should be displayed if the user is pursuing a certain task.

The information presented to users is affected in two ways by the selection of a task:

- The information entities that are deemed relevant for the current task are opened at the time an answer is generated.
- Follow-up questions are organised into two-level menus, where the first level contains only a few questions, relevant to the current task, and the second level contains all follow-up questions. This solution is inspired by the “adaptive prompts” in (Kuhme et al., 1993). A similar approach to dynamic menus is described by Mittal and Moore (1995).
- The acronyms in SDP-TA are expanded into full terms for some tasks; for example, SDP-TA can be spelled as "System Development Process for Telecommunications Applications".

In accordance with our fundamental glass-box approach, the system must display its task knowledge to the users in order to allow them to inspect and control the adaptation. In the POP prototype, we have decided to investigate two parallel approaches to do this, namely *plan recognition* and *explicit task selection*. The interface design allows us to use both mechanisms in combination or separately from each other. The motivations for and detailed form for these two approaches are described further below.

4.3 Transparent Task Stereotypes

One approach to making the task knowledge of POP transparent to the users, is simply to allow the users to choose which task they themselves think that they are performing. As indicated in the introduction, dynamic adaptation has some drawbacks. In particular, user modelling cannot be anything but a guess if it attempts to model the user's knowledge (Kay, 1994). In a hypertext application the input from the user may be quite limited; we know which texts the user chooses to see and whether any links are followed from those texts. These manœuvres provide very little information about the user.

As discussed in the introduction, the requirements on transparency, control, and predictability make it less desirable to keep a separate and generic user model, since users will have great difficulties in inspecting and controlling such a model. Even if users are allowed to alter the user model, it might be very difficult for them to foresee what the effects of a modification would be. Furthermore, in our domain, and indeed in most applications, users are likely to spend very little time modifying the adaptive components, since this does not immediately provide mileage for their main task. This leads to a behaviour where users are forced to use unnatural and lengthy sequences of interactions geared to "circumventing" the adaptivity when it goes wrong (Woods 1993).

So, with our solution where the users are allowed to choose which task they think they are performing, we avoid some of the problems inherent in user modelling, but we also introduce some new ones. Firstly, if there are too many tasks to choose form it will become very difficult for the user to predict what an alternation of the task will result in. Second, even with a small set of tasks, the users might not be willing to always restate which tasks they are performing as their goals and needs change during a session with our system. Users might start out trying to do some reverse engineering task, and then discover that they need to learn more about some aspect, i.e. a move to the learning details task. As shown by (Oppermann 1994), the best solution seems to be somewhere inbetween system-controlled and user-controlled adaptation, which is why we think that allowing the user to choose among a small set of predefined, stereotypical tasks, should be combined with the plan inference approach described below.

Our users are therefore required to state which out of a small set of tasks that the think they are adhering to. Currently, we are experimenting with four tasks: "Learning the structure of SDP-TA", "Project Planning", "Reverse engineering", and "Following an activity". The task stereotypes were selected to allow users to predict which explanation will satisfy their need in a particular situation, and are based on the empirically established task hierarchy in figure 1. These tasks are expressed in domain concepts, and are understood by users of SDP-TA. They know the meaning of "reverse engineering" and they also know whether they are working this way or not. So, as said in the introduction, we are not forcing the user to learn and understand an abstract language with words as "goal", "action", etc., instead they express their needs for adaptation in the domain language.

In order to establish that users would be able to choose among these four tasks and then be able to predict or somehow understand what their choice would result in, we performed a small study where we asked 7 subjects (with varying familiarity with SDP-TA) to couple four tasks with four explanations and to provide motivations for their choices. 5 subjects did a correct coupling and 2 did a partially correct coupling. More important were their comments on our explanations, which we have used to improve the mapping rules from task to explanation. That made us add, for example, one information entity specifically geared towards the needs of project planners explaining the purpose of a certain step in the SDP-TA process. This new information entity was needed in order to help the planners to determine what in SDP-TA their projects absolutely have to do, and what they can skip in certain circumstances. Other similar alternations where made.

A critique of our user-controlled task adaptation, could be that the explanations we provide will be too static to be a good approximation of what kind of explanations our users need. But, since we allow users to change the explanation through opening and closing parts of it and through clicking on hotwords and receiving explanations to concepts that are unfamiliar to them, we have in fact introduced what Moore and Swartout have raised as a requirement on explanations (1989):

> "Explanation is an interactive process, requiring a dialogue between advice-giver and advice-seeker. Yet current expert systems cannot participate in a dialogue with users. In particular these systems cannot clarify misunderstood explanations, elaborate on previous explanations or respond to follow-up questions in the context of the on-going dialogue."

Instead of drawing the conclusion that Moore and Swartout does, namely that explanation systems must engage the user in a text dialogue, we have a direct-manipulation, multimodal, hypertext solution to the same problem. We see many advantages with this approach: it does not require text dialogue, it builds on direct-manipulation techniques and simple queries in a multimodal setting. Finally, the appearance of the system as that of a familiar direct-manipulation interface (with the adition of our free form queries) will not cause users to false expectations of its conversational competence.

In a way, the direct-manipulation and the follow-up queries provide the user with ways in which they can alter the explanations that the task stereotypes provide.

4.4 User Verified Task Recognition

The proposed solution with static task stereotypes still runs into some problems, that instead may be addressed by *dynamic* adaptation to the user's task. As said above, the first problem is that the set of task stereotypes must be rather small, in order for the user to select among them and learn to understand their effects on explanations. As seen in figure 1, the complete set of empirically identified tasks is large, and this analysis is not exhaustive. The second problem is that in our prestudies, we found that users very often *move between* tasks. Typically, a user may start using the system in order to get help to develop a particular object. But while reading the detailed description of this object, he or she comes across a term that is not understood, and starts looking for information explaining this term. This represents a shift in task to the task of "learning a concept". It is unlikely that users will go through the effort to always mark these new tasks explicitly – especially since users are unlikely to actively reflect on the focus shift. Rather, they will open new information entities or pose new queries to cover their needs. This is of course perfectly possible, but this means that the user must go through the extra effort of searching for the right information entities when posing new questions or moving around in the information domain. We discussed earlier that a dynamic interface may require unnecessary manœuvres to circumvent erroneous adaptations. Here, we have the opposite situation: users may also perform unnecessary manœuvres to circumvent an unnecessarily *static* nature of the interface.

To deal with these problems, we have decided to utilise plan recognition techniques to allow the system to actively adapt to the users' tasks. The plan recognition mechanism is the only intelligent technique we have included that actively adapts the presentation. In order to keep the interface transparent and controllable, we have selected to let plan recognition affect only the answers to queries, from menues or the free format input, while the navigation in the information structure still is done through direct manipulation.

The answer to a user question will be affected one additional way by the plan recognition. As for explicit task selection, the information entities that are deemed relevant for the current task are opened at the time an answer is generated, and the menus with follow-up questions are organised to show the queries first, that are most relevant for the current task. But in addition to this, the plan recognition method generates a textual phrase in the beginning of the answer, that reads "This answer is assuming that your current information seeking task is X". X is a mouse sensitive hotword, with an associated menu containing a list of alternative tasks. Both the statement itself and the menu of alternatives are generated dynamically. The menu is tailored to include only such tasks that actually would alter the presented information.

Plan Recognition Characteristics in the PUSH domain

Plan recognition can be interpreted as the task of recognising, or guessing at, an agent's *intention* underlying its actions and utterances. The reason this is usually called plan recognition is that the most widespread approaches to the problem try to follow an agent's actions or utterances, and match these to some more or less procedural description of a plan, corresponding to a particular intention (Kass and Finin 1988, Kautz 1987).

The plan recognition problem appears in three different forms: plan recognition when the actor is aware and actively co-operating to the recognition, for example by choosing actions that make the task easier (intended plan recognition), plan recognition when the actor is unaware of or indifferent to the plan recognition process (keyhole plan recognition[2]), or plan recognition when the actor is aware of and actively obstructs

[2]Keyhole plan recognition has got its name from the analogy to "looking through a keyhole" – the system

the plan recognition process (obstructed plan recognition).

Applications of plan recognition in human - machine interfaces are to some extent almost always keyhole. There are several reasons for this: firstly, users are not always provided with means to communicate their intentions, they do not find any reason to communicate their intentions, but foremost, they are not enough *aware of the plan recognition abilities* of the system to aid the plan recognition process. It is highly desirable that whenever plan recognition is utilised in an interface, users should be provided with means to inspect, understand and control the plan recognition process, to achieve a situation where users actually intend the system to perform plan recognition. Pure keyhole plan recognition in direct manipulation interfaces can be extremely difficult, since the user's goal usually is at a very high level compared to the individual interactions with the system, and since users may frequently change task, or adopt a novel strategy for the same task, without in any way signalling this to the system.

One way to do better the odds for intended plan recognition is to provide the user with *co-operative task enrichment* (Wærn 1994). An interface provides co-operative task enrichment if

- it adapts responses to individual user interactions on the user's task, that is, plan recognition is an *integral part of* the dialogue,
- it communicates its assumptions to the user, and
- it allows the user to explicitly interact with the plan recognition mechanism.

The POP interface is a characteristic example of a point-and-click interface that combines features both of intended and plan recognition by providing co-operative task enrichment. There are several keys available for inferring a user's task: his or her moves between different information pages, the way of selecting the page (navigation or direct query), the opening and closing of information entities within a page, and finally the explicit selections of tasks. The plan inference also is an integral part of the dialogue, as it selects what information to present and what to hide, at each time a new page is generated.

Still, this system shares a lot of properties with other direct-manipulation applications of plan recognition: the individual actions are at a low level compared to the user's task, and users may frequently move between different strategies for a task, and even between tasks. For example, we cannot draw any definite conclusions about a user's task from his or her way of selecting a page of information – some users will always navigate to the sought information, whereas other will prefer search commands. In our prestudies on an existing information system in the domain, we also found that users that started out with one particular task, frequently would move over to other tasks.

The most interesting aspect of this application is that since plan recognition is an integral part of the human - system interaction, we cannot determine the plan patterns that users will exhibit prior to the implementation of the plan recognition

is observing the user's actions, but the user is not aware of this happening, or indifferent to it.

component! Clearly, a kind of bootstrapping procedure is needed, where machine learning techniques are used to gradually build up the system's plan knowledge.

In the second version of POP currently under development, we utilise an extremely simplistic plan recognition algorithm called "intention guessing", that can be described as a probabilistic plan parsing approach with a limited "attention span" (Wærn and Stenborg, 1995). This algorithm is useful in this domain since it requires little or no declarative knowledge about user plans, and instead it can be trained from examples. It is also inherently forgetful, making it suited to deal with users changing intentions. More advanced approaches to plan recognition such as the plan parsing approach by Kautz (1987) have problems with recognising when users move between tasks. Furthermore, the intention guessing algorithm only keeps track of such tasks that are likely to be performed at the time of observation, and no more is really needed to produce answers to a specific query.

5 Example extended with adaptivity

Turning back to the example introduced above, 3.1, in which ways would the adaptivity have changed the interaction with the user? Assuming that we combine the sterotypical tasks with the plan inference mechanism, the following scenario is possible:

- When the user first entered the system she is encouraged to choose among a set of stereotypical tasks.
- The choice of task then affects the first answer page, making certain informations entities open and others closed, and affecting which follow-up questions (on hotwords) are immideately available.
- As the user starts manipulating the answer page, and asking follow-up questions, the plan inference component might decide to alter the assumed task, and that in turn then affects the answer page.
- The user may at this point, disagree with what the system has inferred and decide to change the task back to what it was orignially or to some other task. This can be done through clicking on the hotword in the answer page which displays which task the system has inferred that the user is trying to complete. Only those alternative tasks that actually will alter the presentation will then be shown to the user.
- In addition to this, the free form questions will allow the user to search for specific information, or to pose a vague question.

In other words, the adaptivity will try and make the answer page as short as possible given the demands inherit from the information seeking task, and it will furthermore affect the navigation from this answer page to other pages through affecting the follow-up queries available from the page. The free form questions will also affect the navigation and search aspects of the system, allowing both experienced and inexperiences users to express their questions.

6 Conclusions

The focus of this paper has been to discuss how user adaptive techniques can be integrated with direct manipulation in an interface design to achieve an inspectable, controllable and predictable interface. In particular, we have pointed out the need for several integrated modes of interaction, and the need for means of inspection and control of adaptive functionalities. In order to make these means understandable by users, we advocate that they are presented in *domain-specific* terms, and not by system-internal terms that may be difficult for the user to understand, and predict the effects of alterations.

This consideration has affected in particular the design of linguistic interaction in the system, and the means for inspection and control of adaptivity. The linguistic interaction may lead to obscure results for users: we remedy this by giving users *query paraphrases* and a visible dialogue history which will add to the transparency of the interface behaviour. To inspect and control task adaptation, users are provided either with *task stereotypes* that can be actively selected, or with *user verifiable task presumptions* and a menu of other candidate tasks for the user to choose from.

Deliberately, we have introduced *two* means for users to control the system's adaptation to the user task, both task stereotypes and user verified plan recognition. We have voiced arguments for and against both approaches, and we plan to perform a comparative study of these two techniques, aiming to determine which of them that provides the best user performance and acceptance. However, one should note that it is perfectly feasible to include them both in the same interface, and let the user chose the level of self-adaptiveness from the system. In our prototype system, users are provided with a separate window dealing with "task information", in which they can select from a small set of task stereotypes. If one of these are selected, the generated answers are tailored to this task. If none is selected, plan recognition is used to determine the user's task. Again, this provides the user with a high-level and domain-specific control of the adaptivity of the system, in accordance with our general glass-box approach to adaptiveness.

The hypertext solution selected for the POP help system is an example of the balance between finding intelligent and flexible techniques for aiding users, and the requirements on maintainability and tractability in a large real-world domain such as SDP-TA. The singular hypertext items are written for users seeking information for specific purposes. Each item is a self-contained coherent piece of text, that still can be combined with other texts to provide answers to more general queries. This gives a modular and extendible solution, which still is understandable for the text writer who has to write, and update, the hypertext.

Since the interface design of POP entirely relies on domain-specific terms, an obvious question that arises is to what extent our solution carries over to other domains.

However, even though the particular realisation is domain specific, the fundamental concepts used are generic. In particular, this concerns the object-oriented structure of the domain information, and the definition of a task hierarchy. This generality is mirrored in a clear division of generic and domain-specific information in the implementation of POP. It is our current belief that the POP system currently under development will be fairly easy to transfer to other information seeking applications, and we aim to attempt this within a close future.

Acknowledgements

The authors wish to thank all the anonymous subjects who took part in our studies.

PUSH is funded jointly by Ellemtel Utvecklings AB, SICS and NUTEK. Ellemtel, SICS, Stockholm University and Linköping University participate in the project.

References

van Beek, P. and R. Cohen. 1991. “Resolving plan ambiguity for cooperative response generation” , proc. IJCAI-91, Sydney, Australia, Morgan Kaufmann.

Berry, D.C. and D.E. Broadbent. 1986. “Expert systems and the man machine interface: part 2: The user interface”, *Expert systems: The International Journal of Knowledge Engineering* 4.

Biermann, Alan W. , Bruce W. Ballard, and Anne H. Sigmon. 1983. “An experimental study of natural language programming”, *International journal of man-machine studies* 18, pages 71–87.

Bladh, Malin and Kristina Höök. 1995. “Satisfying User Needs Through a Combination of Interface Design Techniques”, proceedings of INTERACT’95, Lillehammer, Norway, 1995.

Bos, Edwin, Carla Huls, and Wim Claassen. 1994. “EDWARD: full integration of language and action in a multimodal user interface” *International Journal of Human-Computer Studies* 40, pages 473–495.

du Boulay, Ben, Tim O’Shea, and J. Monk. 1981. “The Black Box inside the Glass Box: Presenting Computing Concepts to Novices”, *International Journal of Man-Machine Studies*, 14.

Bretan, Ivan. 1995. “Natural Language in Model World Interfaces”. Licentiate Thesis, Department of Computer and Systems Sciences. Stockholm: The Royal Institute of Technology and Stockholm University.

Bretan, Ivan, Niklas Frost and Jussi Karlgren. 1995. “Using Surface Syntax in Interactive Interfaces”, *Paper presented to The 10th Nordic Conference of Computational Linguistics*. Helsinki: University of Helsinki.

Breuker, Joost. 1990. *EUROHELP: Developing Intelligent Help Systems*, Lawrence erlbaum associates, publishers”, EC, Report on the P280 ESPRIT Project EUROHELP.

Capindale, Ruth A. and Robert G. Crawford. 1990. “Using a natural language interface with casual users.” *International Journal of Man-Machine Studies* 32, pages 341-362.

Chi, Michelene T. H., Slotta, James, D., de Leeuw, Nicholas. 1994. “From Things to Processes: A Theory of Conceptual Change for Learning Science Concepts”. *Learning and Instruction - The Journal of the European Association for Research on Learning and Instruction* 4(1), pages 27-43.

Cohen, P. 1992. “The Role of Natural Language in a Multimodal Interface”, Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), Monterey, 1992, pages 143–150.

Höök, Kristina. 1995. “Providing Explanations Fitted to the User’s Task”, available from SICS.

Kaplan, Craig, Justine Fenwick and James Chen. 1993. “Adaptive Hypertext Navigation Based On User Goals and Context”, *User Modeling and User-Adapted Interaction* 3, pages 193–220.

Karlgren, Jussi, Ivan Bretan, Niklas Frost and Lars Jonsson. 1995. “Interaction Models, Reference, and Interactivity for Speech Interfaces to Virtual Environments”, Proceedings of Second Eurographics Workshop on Virtual Environments – Realism and Real Time, Monte Carlo. Darmstadt:Fraunhofer IGD.

Karlgren, Jussi, Kristina Höök, Ann Lantz, Jacob Palme, and Daniel Pargman. 1994. “The Glass Box User Model for Filtering”, 4th International Conference on User Modeling, Hyannis, ACM.

Kass, Robert and Tim Finin. 1988. “Modeling the User in Natural Language Systems”, in *Computational Linguistics* 14(3), Special issue on user modeling, ed. A. Kobsa and W. Wahlster.

Kautz, Henry A. 1990. “A Circumscriptive Theory of Plan Recognition”, in ed. P. R. Cohen, J. Morgan and M. E. Pollack, *Intentions in communication*, MIT Press , Cambridge, Mass., pages 105 – 133.

Kay, Judy. 1994. “Lies, Damned Lies and Stereotypes”, 4th International Conference on User Modeling, Hyannis: ACM.

Kobsa, A., D. Müller and A. Nill. 1994. “KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS”, Fourth International Conference on UM, Hyannis, MA, ACM.

Kühme, T., U. Malinowski, and J. D. Foley. 1993. “Adaptive Prompting”, Technical Report GIT-GVU-93-05, Georgia Institute of Technology.

Lemaire, Benoit. February 1995. "Object-oriented explanation planning", in "Principles of Natural Language Generation - Papers from a Dagstuhl Seminar", W. Hoeppner W, Horacek H (eds), Report SI-12 of the University of Duisburg, Germany.

Lemaire, Benoit, Catriona McDermid and Annika Wærn. 1994. “Adaptive Help by Navigation and Explanation”. SICS technical report T94:05.

Maes, Pattie. 1994. “Agents that Reduce Work And Information Overload”. *Communications of the ACM* 37:7.

McDermid, Catriona and Anna-Lena Ereback. 1994. Initial application evaluation and help requirements for SDP. PUSH working paper WP94:01/01.

Meyer, B. 1994. “Adaptive Performance Support: User Acceptance of a Self-Adapting System”, 4th International Conference on User Modeling, Hyannis, ACM.

Mittal, V. O. and J. D. Moore. 1995. “Dynamic Generation of Follow Up Question Menus: Facilitating Natural Language Dialogues”, SIGCHI ’95 (Denver Colorado, May 7-11, 1995) Human Factors in Computing System Proceedings 1995. New York: ACM SIGCHI, pp. 90-97

Moore, J D and W. R. Swartout. 1989. “A Reactive Approach to Explanation”, 11th Int. Conf. on AI, pages 1504 – 1510.

Oppermann, Reinhard 1994. “Adaptively supported adaptability”, Int. J. Human-Computer Studies 40:455-472.

Paris, Cecile. (1988). Tailoring Object Descriptions to a User’s Level of Expertise, Computational Linguistics, 14(3):64-78.

Pollack, M. E., J. Hirschberg, and B. L. Webber. 1982. “User Participation in the reasoning processes of expert systems.” in proceedings of the second national conference on Artificial Intelligence, Pittsburgh, Penn., 1982.

Raskutti, B. and I. Zukerman. 1994. “Query and response generation during information-seeking interactions”, 4th International Conference on User Modeling, Hyannis, ACM.

Roth, E. M., and Woods, D. D. 1989. “Cognitive Task Analysis: An Approach to Knowledge Acquisition for Intelligent System Design”. In G.Guida and C. Tasso (eds.) *Topics in Expert System Design*, Elsevier Science Publ. B.V. (North-Holland).

Self, J. 1988. “Bypassing the Intractable Problem of Student Modelling”, Proc. of the Conference on Intelligent Tutoring Systems ITS-88, pages 107 – 123.

Shneiderman, Ben. 1983. “Direct Manipulation: A Step Beyond Programming Languages.” *IEEE Computer* 16(8), pages 57–69.

SICStus Prolog User’s Manual (Release 3#0). Swedish Institute of Computer Science, Box 1263, S-164 28 Kista, Sweden. ISBN 91-630-3648-7.

Julita Vassileva. 1994.“A Practical Architecture for User Modeling in a Hypermedia-Based Information System”. Fourth International Conference on UM, Hyannis, MA, ACM.

Wærn, Annika. 1994. “Cooperative Enrichment and Reactive Plan Inference - applying plan inference outside Natural Language Dialogue”. Presented at the UM-94 workshop on applied planning and plan recognition, available from SICS.

Wærn, Annika and Ola Stenborg. 1995. “Recognizing the Plan of a Replanning User”. To be presented at the IJCAI workshop on Plan Recognition, Montreal, Canada.

Woods, David D. 1993. “The price of flexibility”, in ed. W.D. Gray, W. H. Hefley and D. Murray, proc. of the 1993 workshop on Intelligent User Interfaces, acm press, pages 19 – 25.