Reasoning about users’ actions in a Graphical User Interface

Maria Virvou

Department of Informatics, University of Piraeus

Katerina Kabassi

Department of Informatics, University of Piraeus

Abstract

This paper is about a graphical user interface that provides intelligent help to users. The graphical user interface is called IFM (Intelligent File Manipulator). IFM monitors users while they work; in case a user has made a mistake with respect to his/her hypothesised intentions then IFM intervenes automatically and offers advice. IFM has two underlying reasoning mechanisms: one is based on an adaptation of a cognitive theory called Human Plausible Reasoning and the other one performs goal recognition based on the effects of users’ commands. The requirements analysis of the system has been based on an empirical study that was conducted involving real users of a standard file manipulation program like the Windows Explorer; this analysis revealed a need for intelligent help. Finally, IFM has been evaluated in comparison with a standard file manipulation GUI and in comparison with human experts acting as consultants. The results of the evaluation showed that IFM can produce successfully advice helpful to users.
CONTENTS

1. INTRODUCTION

2. RELATED WORK
   2.1. Intelligent Help
   2.2. Human Plausible Reasoning

3. EMPIRICAL STUDY

4. DESCRIPTION OF IFM
   4.1. Underlying Reasoning Mechanisms
   4.2. Component Architecture

5. Hierarchy of User’s Actions

6. Goal Recognition
   6.1. Using Instabilities to Confirm Goals and Plans
   6.2. Categorisation of User’s Actions and IFM’s Hypotheses

7. HPR for User Modelling
   7.1. The Basic Principle
   7.2. Certainty parameters

8. Evaluation of the System

9. Conclusions and Future Work
1. INTRODUCTION

The expansion of the computer users’ community to include users of different backgrounds and levels of expertise triggers the need for user friendlier interfaces that support task accomplishment in a more flexible way than those currently available. There are users with very little computing experience that may need to have access to Internet facilities or software packages addressed to a wide range of users such as word processors. However, even users with limited goals in the use of computers still need to issue commands in order to accomplish these goals. Such users are prone to errors due to their little experience. On the other hand, more experienced users may also make mistakes due to carelessness and/or tiredness.

Traditional approaches to on-line and off-line help to users include user manuals, hypertext facilities for searching information, tutorials, demonstrations etc. However, such approaches may be inefficient in many respects. For example, Matthews et al. (2000) point out that on-line manuals must explain everything and novices find them confusing while more experienced users find it quite annoying to have to browse through a lot of irrelevant material to find the information they desire. Moreover, empirical studies (e.g. Virvou, Jones, & Millington, 2000) have shown that users, who have been involved in a problematic situation, may not realise that they have made an error so that they may look for help.

User interfaces may become more flexible and helpful if they incorporate intelligence. An intelligent interface at its developmental extreme is an intelligent agent that embodies some of the key capabilities of a human assistant: observing and forming models of the world and the user; inferring user intentions based upon those observations; formulating plans and taking actions to help the user achieve those intentions (Tyler, Schlossberg, Gargan, Cook, & Sullivan, 1991). It is generally agreed that an intelligent interface should be both knowledge-based (Hollan et al., 1991), modular (Norman, 1986) and it should be able to infer user plans and intentions (Young, 1991). Users interacting with a program always have a goal. In order to reach their goal they must execute the right sequence of commands. As Polson and Lewis (1990) point out, users determine which action appears to lead most quickly to the goal by comparing the description of the available actions with the description of the goal. In this attempt, they may make mistakes that will complicate the achievement of their goal.

In this paper, we describe the research work involved in the development of an intelligent Graphical User Interface for a program that manipulates files, such as Windows Explorer (Microsoft Corporation, 1998). The GUI developed is called IFM (Intelligent File Manipulator). IFM’s aim is to add intelligence for the benefit of the user of a GUI. Graphical User Interfaces are generally considered user friendly. However, as McGraw (1994) points out, without proper design even graphical user interfaces may prove difficult to traverse and use. Indeed, an empirical analysis involving users of a
standard explorer revealed that both novice and expert users encountered problems in their interaction with the standard explorer.

In order to facilitate users, especially novice ones, IFM generates hypotheses about users’ intentions and in case it suspects that a user deviates from his/her goal, it provides assistance. Therefore, IFM can also be used as a protected learning environment for novice users of GUIs that manipulate files (Virvou & Kabassi, 2000b). At any time of the interaction IFM constructs a user model, which is based on two underlying reasoning mechanisms. One reasoning mechanism performs a limited goal recognition following the users’ actions and the other one generates hypotheses about possible errors based on a domain-independent cognitive theory called Human Plausible Reasoning (Collins & Michalski, 1989).

Both reasoning mechanisms have been used previously in a different intelligent help system called RESCUER (Virvou 1998; Virvou 1999; Virvou & Du Boulay, 1999). RESCUER provided automatic assistance to users of the UNIX operating system. The user interface of UNIX is a command language interface, which is different from a graphical user interface that involves mouse events. However, the successful adaptation to a certain extent of RESCUER’s reasoning mechanisms into IFM reveals that there is a potential for a more general framework for the development and incorporation of intelligent help into user interfaces.

The iteration of several phases of the development was considered crucial while developing IFM. In addition, an object-oriented development was considered most suitable because it would be compatible with the objects needed for the GUI. Therefore, we selected the Rational Objectory Process to form the basis for the system’s life cycle (Virvou & Kabassi 2000c). The Rational Objectory Process is an object-oriented model of software life-cycle that has been suggested to complement the development of software using the Unified Modeling Language (UML) (Booch, Rumbaugh & Jacobson, 1999); UML is a representation language for object-oriented analysis and design. The Rational Objectory Process supports an iterative development, in which the architecture is actually prototyped, tested, measured, and analysed, and then refined in subsequent iterations (Quantrani 1998; Kruchten 1999). The process divides one development cycle in four consecutive phases: the inception, the elaboration, the construction and the transition phase.

IFM’s life cycle was divided into cycles, each cycle working on a new generation of the system. In the inception phase, a primary executable release of IFM was developed (Virvou & Stavrianou 1999). The primary executable release was considered insufficient, so another iteration of the software life cycle took place. In order to capture the requirements for the expansion of IFM an empirical study was conducted during the elaboration phase (Virvou & Kabassi 2000a). The empirical study involved a number of expert and novice users, who were asked to work with the regular Windows 98/NT Explorer and with the primary executable release of IFM. The results of the empirical study formed the basis for the requirements analysis of IFM’s extension. In the construction phase, the second executable release of IFM was developed based on the requirements analysis that was conducted during the previous phase. Finally, a formative
evaluation of the second executable release of IFM was conducted in the phase of elaboration.

The remaining of this document is organised as follows: In Section 2 we present and discuss the related work. In Section 3 we describe the empirical study that was conducted at the early stages of the development of IFM. In Sections 4, 5 and 6 we give a full description of the system itself, its operation, its domain representation and its underlying reasoning mechanisms for user modelling. Finally, we describe the evaluation of the system and we give the conclusions drawn from this work.

2. RELATED WORK

2.1. Intelligent Help

Recently there have been several approaches to intelligent help which all aim at improving the quality of help to the user. Two different kinds of approach in terms of who initiates the response to a user (the computer or the user) may be used to classify Intelligent Help Systems into two categories, namely passive and active. Passive systems respond to users’ queries; therefore the user takes the initiative for the interaction. On the other hand, active systems intervene when they judge that there is a problem without the user having initiated this interaction. Passive systems may be very helpful when a user realises that s/he needs help and knows how to ask for it. However, active systems address the cases when a user becomes involved in problematic situations without his/her realising it.

A prototypical passive intelligent help system is UC (Mayfield, 1992; Chin, 1989; Wilensky et al., 2000). UC acts as a consultant for UNIX users. It responds to user queries in natural language and aims at helping users figure out how to perform a task or receive more information on something they are interested in. The advantage of UC is that users may use their own words to ask questions. However, they still need to know what to ask. Other passive systems include the OSCON consultant (Mc Kevitt, 2000) and the SINIX consultant (Hecking, 2000; Kemke, 2000). In these systems too, users may be given answers to their questions in a flexible way. However, again these systems do not aim at diagnosing problematic situations.

Unlike the above passive systems, IFM is an active system that monitors users’ actions and offers spontaneous help when it thinks that there is a need for it. This is similar with other active systems such as RESCUER (Virvou 1998; Virvou 1999; Virvou & Du Boulay 1999), CHORIS (Tyler, Schlossberg, Gargan, Cook, & Sullivan, 1991), USCSH (Matthews, Pharr, Biswas, & Neelakandan, 2000) and Office Assistant (Horvitz, Breese, Heckerman, Hovcl, & Rommelse, 1998). However, different active systems may focus on varying aspects of help. For example, some systems like USCSH, which is a help system for UNIX users, focus on indicating to the user better and more efficient ways of getting a task done. In contrast to USCSH, IFM pays more attention on user modelling and error diagnosis in order to help users recover from their errors and accomplish their high level goals and plans.
In terms of errors, IFM is quite similar to CHORIS because they both deal with mouse sensitive ‘menu’ commands that when selected, activate a prescribed action; CHORIS is an intelligent interface for manipulating emergency crisis management systems such as earthquakes. In the domain representation, CHORIS keeps all the objects relations and commands in a similar way as IFM keeps hierarchies of commands and objects. However, the two systems differ in the way their user modelling components acquire information about users. CHORIS maintains explicit user models, whereas IFM constructs implicit user models. Explicit user models are based on information that users have provided explicitly about themselves, whereas implicit models infer information, by observing and interpreting the users’ behaviour (Rich 1999).

IFM’s underlying reasoning mechanisms are very similar to RESCUER, which is an intelligent help system for UNIX users. Both systems use Human Plausible Reasoning in hypotheses generation of users’ misconceptions. Therefore, both systems use analogies, specialisations and generalisations in the creation of hypotheses. IFM also uses a limited goal recognition mechanism, which is similar to that of RESCUER’s. This mechanism keeps track of users’ goals and intentions. However, the domain of RESCUER, which is the UNIX command language and the domain of IFM, which is a graphical user interface are quite different. Therefore analogies, specialisations and generalisations are defined on a very different basis. There are also many differences on the limited goal recognition mechanism, mainly because of the different nature of user plans. Nevertheless, the successful adaptation to a certain extent of RESCUER’s reasoning mechanisms into IFM reveals that there is a potential for a more general framework in the construction of intelligent help systems.

Finally, there is an active help system that Microsoft has introduced for helping users. This system is called Tip Wizard and is very similar to the Office Assistant (Horvitz, Breese, Heckerman, Hovel, & Rommelse 1998). Tip Wizard’s main objective is to recommend new commands to users based on alternative commands’ equivalence to the less efficient command sequence that a user may be using in order to perform a task. IFM’s domain is similar to Tip Wizard’s domain. However, IFM’s rationale is quite different from that of Tip Wizard. IFM’s objective is to intervene only when this is considered really essential for helping the user accomplish his/her plans without errors and not comment on the actual way a user may select to accomplish his/her goals. Therefore, in case IFM suspects that an action would not have the desired results for the user, it generates alternative actions that would achieve these hypothesised goals.

### 2.2. Human Plausible Reasoning

Human Plausible Reasoning (henceforth referred to as HPR) is a theory, which is based on the analysis of people’s answers to everyday questions about the world (Collins & Michalski, 1989; Burstein & Collins, 1988; Burstein et al., 1991). The theory consists of three parts:
1. a formal representation of plausible inference patterns; such as deductions, inductions, and analogies, that are frequently employed in answering everyday questions;

2. a set of parameters, such as conditional likelihood, typicality and similarity, that affect the certainty of people’s answers to such questions; and

3. a system relating the different plausible inference patterns and the different certainty parameters.

HPR is a descriptive theory of human plausible inference. The theory is used to formalise the plausible inferences that frequently occur in people’s responses to everyday questions for which they do not have ready answers. HPR detects the similarity/dissimilarity relationship between a question and the knowledge retrieved from memory and drives the line (type) of inference. For example, if the question asked was whether coffee was grown in the Llanos region in Colombia, the answer depended on the knowledge retrieved from memory. If the subject knew that Llanos was in a savanna region similar to that where coffee grows, this would trigger an inductive, analogical inference, and would generate the answer yes (Carbonell & Collins, 1973). HPR models the reasoning of people who have a patchy knowledge of certain domains such as geography. By patchy knowledge we mean partial knowledge of the facts and relations in the domain.

Human knowledge about a domain is represented as a collection of statements. An example of a statement is: precipitation(Egypt) = very-light, which means that the precipitation of Egypt is very light. Precipitation is called a descriptor, Egypt is called an argument and very-light is called a referent. A descriptor is said to apply to an argument and together they form a term. The simplest class of inference patterns are called statement transforms. Statement transforms exploit the 4 possible relations among arguments and among referents to yield 8 types of statement transform. For example, from the statement flower-type(England)=roses, we can make the following statement transforms, given the type hierarchy for flowers shown in Figure 1 and a similar type hierarchy for geographic regions (not illustrated).

Argument transforms

GEN flower-type(Europe)=roses
SPEC flower-type(Surrey)=roses
SIM flower-type(Holland)=roses
DIS flower-type(Brazil)≠roses

Referent transforms
GEN flower-type(England)=temperate flowers
SPEC flower-type(England)=yellow roses
SIM flower-type(England)=peonies
DIS flower-type(England)≠bougainvillea

Figure 1. A type of hierarchy of flowers

The core theory also introduces certainty parameters, which are approximate numbers ranging between 0 and 1. Certainty parameters affect the certainty of different plausible inferences. For example the degree of similarity ($\sigma$) affects the certainty of any SIM or DIS inference. In particular, if the degree of similarity is almost 1 there is great confidence in the transformation, otherwise, the confidence decreases. Some of the certainty parameters described are represented in Table 1.

Table 1. Certainty parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\gamma$</td>
<td>Degree of certainty or belief that an expression is true.</td>
<td>Applies to any expression.</td>
</tr>
<tr>
<td>$\tau$</td>
<td>Degree of typicality of a subset within a set.</td>
<td>Applies to GEN and SPEC statements.</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Degree of similarity of one set to another set.</td>
<td>Applies to SIM and DIS statements.</td>
</tr>
<tr>
<td>$\phi$</td>
<td>Frequency of the referent in the domain of the descriptor.</td>
<td>Applies to any non</td>
</tr>
</tbody>
</table>
HPR formalises the reasoning that people use in order to draw inferences and form responses to questions for which they do not have ready answers. The answers they form may be correct or incorrect. In any case they are based on plausible reasoning. In IFM we use this reasoning to simulate users’ plausible reasoning that may have led them to plausible errors. Therefore, we use HPR to generate hypotheses about users’ errors.

3. EMPIRICAL STUDY

After the construction of a primary executable release of IFM (Virvou & Stavrianou, 1999) an empirical study was conducted. The aim of the empirical study was to identify users’ problems with a standard explorer and also highlight limitations of the primary executable release of IFM in order to specify its extension. The empirical study involved 15 users of different levels of expertise. 7 novice and 8 expert users were asked to interact with a standard file manipulation program such as Windows 98/NT Explorer, as they would normally do. We used computer logging to capture all users’ actions in videos. These videos were then given to 10 human experts to comment on all 15 users’ actions. Human experts were asked to analyse the users’ actions and suggest some advice in case they thought it would be needed. The 10 human experts who were involved in the empirical study as commentators were different from the expert users of the standard file manipulation program. All of them possessed a first and/or higher degree in Computer Science and had extensively used programs like a standard file manipulation program. Most of them had also teaching experience related to the use of such programs in computer labs.

The method of computer logging was selected so that the experts were able to watch the exact users’ actions, e.g. clicking on a certain object, dragging a selected object, and so on. Our main aim of conducting this kind of analysis was to find usability problems of a standard Explorer, through heuristic evaluation (Nielsen 1994). In particular, human experts were asked to focus on two usability heuristics specified by Nielsen:

1. Error Prevention.
2. Help users recognise, diagnose and recover from errors.

The human experts’ sets of comments were then collected and compared with one another. In many cases there was a diversity of human experts’ opinions. However, there were also many cases where there was a high degree of agreement (ranging between 80% to 100%) of experts’ comments and advice. Such comments served as the basis for the conclusions drawn from the empirical study concerning users’ behaviour while interacting with file manipulation programs. The empirical study also revealed the most frequent errors that expert and novice users may make while completing their tasks in a standard Explorer.
In particular, evaluation of the results of the empirical study showed that experts’ errors were different from those of novices. This was mainly due to the fact that experts made errors because of their carelessness whereas novices made errors because of their lack of knowledge. For example, novices did not know how to execute certain commands such as copying an object whereas experts tangled up neighbouring object with similar names. One of the errors, mainly made by novices, was that users neglected to rename a new object (a folder or a file) when they created it and this resulted in a state where many objects in the file store had exactly the same name. Another problem was that novice users did not know how to complete certain operations, such as the copy or move operation. Novice users also seemed to have problems with the structure of the standard Explorer. They could not understand that on the left side the user sees the structure of folders in the file store and on the right side the contents of the selected folder. However, the most significant error concerned the accidental deletion of objects belonging to folders that users deleted without having been aware of their content. Both novice and expert users committed such errors and their results were catastrophic. Another problem that both novice and experts often faced concerned objects with exactly the same name or multiple copies of the same file. Both categories of user often showed evidence that they had confused such objects.

The protocols given to human experts were also given as input to the primary executable release of IFM in order to compare the human expert comments to IFM’s reactions. Indeed, the limitations of IFM were identified and its extension was designed based on the results of the empirical study. In addition, the empirical study showed that users tended to reproduce the same kind of mistake many times. In terms of the design of IFM this meant that a long-term user model (Rich 1999) would be very useful in providing more individualised help.

4. DESCRIPTION OF IFM

IFM is a file manipulation program that provides intelligent help to its users. The system monitors users silently and in case it suspects that a user’s action contradicts the user’s hypothesised goals, it provides spontaneous advice. In particular, every time a user issues an action, IFM reasons about it in terms of the system’s expectations about the user’s recognised goals. In case this action contradicts the system’s expectations, it suggests alternative actions to the user. Otherwise, the action is executed normally.

Alternative actions are generated by applying the HPR theory. IFM reasons about every alternative action generated; in case an alternative action contradicts the system’s expectations about the user’s goals, it is ignored. If at least one, generated alternative action is compatible with the system’s expectations about the user’s goals then this is considered a good alternative action and is shown to the user. However, the user is not obliged to follow IFM’s advice. S/he can execute his/her initial action or issue a completely different action. Every time the user issues an action, the system updates the user model.

In table 2, we present a simple example of the system’s operation. In the first column we give a description of the user’s actions. We have not given a full account of
what the user did but a summary of what the actions meant to an Explorer. For example, the user’s action ‘delete(A:\Projects\Program1\)’ implies that the user had first selected the folder ‘A:\Projects\Program1\’ and then issued a command delete. However, for the sake of simplicity we consider an action the meaning of a certain sequence of selections such as the selection of a folder and then the clicking on delete. In the second column we give a description in natural language of the particular user’s step of the first column. Finally, in the third column we give a description of IFM’s reasoning and advice to the user (in case IFM generated some advice). The initial state of the file store is presented in Figure 2. Folders are represented as boxes and files in plain text.

Table 2. An example of a user’s interaction with IFM

<table>
<thead>
<tr>
<th>No</th>
<th>User’s actions</th>
<th>Description of the user’s actions</th>
<th>IFM’s reasoning and reaction</th>
</tr>
</thead>
</table>
| 1  | delete(A:\Projects\Program1\) | The user tries to delete the folder ‘Program1’. This action will result in the loss of the content of this directory. | Unexpected action and generation of alternative commands. IFM’s advice: delete(A:\Projects\Program2\)
Explanation of IFM’s advice: ‘Program2’ is an empty folder whereas ‘Program1’ is not, so ‘Program2’ may have been what the user intended to select since it has a similar name to ‘Program1’. |
| 2  | delete(A:\Projects\Program2\) | The user takes IFM’s advice and deletes ‘Program2’. | Action confirms the system’s expectations. |
| 3  | mkdir(A:\Courses\)     | The user creates a new folder in directory ‘Courses’ | Unexpected action but no alternative found. No reaction. |
| 4  | rename(A:\Courses\New Folder\, A:\Courses\Course2\) | The user renames the new folder. | Expected action. |
| 5  | cut(A:\Courses\Math.txt) | The user initiates a plan for moving ‘Math.txt’ from the directory ‘Courses’. | Expected action. |
The user completes his/her plan by executing the paste command. However, in folder ‘Course1’ already exists a file with exactly the same name. This will result in the replacement of the particular file and the loss of possibly useful information.

Unexpected action and generation of alternative commands. IFM’s advice: Change Folder to ‘Course2’.

Explanation of IFM’s advice: ‘Course2’ is an empty folder. ‘Course2’ may have been what the user intended to select instead of ‘Course1’ since these folders have very similar names and could be confused.

| 6 | paste(A:\Courses\Course1) | The user completes his/her plan by executing the paste command. However, in folder ‘Course1’ already exists a file with exactly the same name. This will result in the replacement of the particular file and the loss of possibly useful information. | Unexpected action and generation of alternative commands. IFM’s advice: Change Folder to ‘Course2’. Explanation of IFM’s advice: ‘Course2’ is an empty folder. ‘Course2’ may have been what the user intended to select instead of ‘Course1’ since these folders have very similar names and could be confused. |
| 7 | paste(A:\Courses\Course2) | The user takes IFM’s advice and moves the file into the recommended folder. | Action confirms the system’s expectations. |

Figure 2. The user’s initial file store state

4.1. Underlying Reasoning Mechanisms

IFM is a system that monitors users and constantly reasons about their actions in order to diagnose possible errors and give advice concerning error recovery. As Cerri and Loia (1997) point out, if error diagnosis is to be performed, a user modelling component should be incorporated into the system. IFM’s user modelling component generates hypotheses about users’ goals and possible errors or misconceptions involved in users’ actions. An example of an erroneous action is clicking on the wrong object (either file/folder or command). The main aspects examined when a user’s action is issued are:
1. The file store state,

2. The intentions of the user. For example, did the user choose the right object?

3. The semantics of the command chosen. For example, did the user know the preconditions and effects of the command?

However, there may be a variety of explanations of observed errors. This may be a problem since there could be a redundancy of generated hypotheses. In view of this problem, IFM uses two underlying reasoning mechanisms for user modelling. The two reasoning mechanisms are independent of each other in the way they function. However, the compatibility of the hypotheses generated from these two mechanisms increases the certainty degree of these hypotheses. One reasoning mechanism performs limited goal recognition and is explained in more detail in Section 6. The other reasoning mechanism is a simulator of human error generations based on HPR and is explained in more detail in Section 7. HPR uses the hierarchies constructed in the domain representation to draw inferences. Therefore, the domain representation is important for the application of HPR in IFM; the domain representation is described in more detail in Section 5.

4.2. Component Architecture

As mentioned in the introduction we have adopted an object-oriented approach for implementing IFM (Virvou & Kabassi 2000c). In order to illustrate the software modules and their dependencies, we have chosen Unified Modelling Language (UML) (Booch Gr., Rumbaugh J. & Jacobson 1999). UML proposes a component diagram for showing the structure of the code for the implementation. Each component in a component diagram can be a source code component, a binary component, or an executable component (Eriksson & Penker 1998). The component diagram that illustrates how the knowledge-based and the object-oriented part of IFM are connected is presented in Figure 3.

The component ‘reasoning.ql’ corresponds to the knowledge-based part of IFM. This part provides the reasoning for user modelling and is implemented in Prolog. The rest of the components are implemented in an object-oriented programming language. The file manipulation system is implemented by the component ‘main.pas’. The same component is used for the evaluation of users’ actions in terms of whether they have been intended by the user or not. In case an action contradicts the system’s expectations, ‘main.pas’ communicates with ‘reasoning.ql’ to generate advice. The generation of advice is done by searching for alternative commands, similar to the one issued. IFM constantly reasons about every alternative command generated and in case the alternative action contradicts the system’s expectations it is ignored. If at least one alternative command has been generated that is compatible with the system’s expectations then the component ‘alternatives.pas’ is activated in order to present the advice to the user. The component ‘login.pas’ corresponds to the verification of the user to the system.
The component ‘interface.dfm’ implements the interface of IFM. Commands are available to the user in a set of menus and buttons. In IFM’s interface, as in every file manipulation program with graphical user interface, IFM uses a tree-structured representation of the file store’s state like Windows 98/NT Explorer. The presentation of alternative commands to the users is done in a way that does not interfere with the user’s work. The user must feel free either to follow IFM’s advice, or proceed with the execution of his/her initial action, or even issue a completely new one.

5. HIERARCHY OF USER’S ACTIONS

The domain representation contains knowledge about commands and the file store state. All concepts represented in the domain knowledge have associated properties and belong to some isa or ispart hierarchy. Concepts concerning the GUI are classified in isa hierarchies in order to be compatible with the main underlying assumptions of HPR. In this paper we need to refer to many concepts concerning GUI commands; therefore we will use the following terminology:

Commands will mean the actual words that denote commands (e.g. *copy*).

Selections will mean the objects selected (e.g. book.txt).

User actions will mean the complete actions of the user (e.g. copy(book.txt)).

An important hierarchy is that of users’ actions (Figure 4). The hierarchy represents the syntactic structure of actions. Moreover, it is constructed in such a way that every descendant node of a parent node inherits all the properties of the parent node.

In this hierarchy, actions are first distinguished into six categories depending on their purpose:
a) Selector

In this category there is only the action select(T), which corresponds to clicking on an object in order to select it.

b) Clipboard actions.

All actions that use the clipboard at an intermediate stage are called clipboard actions. For example, the command \textit{copy}, which may be issued by the user in three different ways:

- Selection of \textit{copy} from a menu.
- Selection of the icon assigned to \textit{copy} from the toolbar.
- Combination of keys (Ctrl + C)

c) Information providers

All the actions that may be used for providing information to the user. For example, open shows contents of files or folders and \textit{explore} shows contents of folders.

d) Creators.

All the actions that create a new object are considered as creators. For example, the command \textit{mktxt} creates a new text document in the file store.

e) Destroyers.

All the actions that destroy an object are considered as destroyers. For example, the command \textit{delDir} deletes a directory from the file store.

f) Modifiers.

These operators modify the properties of an object. For example, the action Rename(T) changes the name of the object T, where T can either be a file or a folder.

The third level of the hierarchy in Figure 4 represents the actual GUI actions that correspond to the parent nodes specified. The actions are distinguished by their names and arguments. For example, \textit{copy(T)} copies the object T and places it to the clipboard. Some of the actions of the third level of the hierarchy can be analysed further.

For example, the commands in the fourth level of the hierarchy analyse the command \textit{mkfile} and are used for creating different kinds of document, such as text files, word document etc. These commands can be found in menus; if the user selects file, then New, then s/he is presented with options such as Text Document, wav file, Bitmap image etc.

Commands with respect to their syntactic structure can also be divided into two main categories. The first category of command is called ‘with-argument’ and consists of the commands that take at least one argument. This means that the user must have selected at
least one object before executing a command belonging to this category. Examples of such commands include the *delete*, *cut* or *copy* commands because the user must have selected at least one item before executing them. The second category of command is called ‘without-arguments’ and consists of commands that do not take any argument. This means that the user does not have to select any argument before executing a command belonging to this category. For example, the commands *paste*, *mkdir* or *mkfile* belong to this category because the user does not have to select any object before executing them.

In Figure 4, which presents the hierarchy of user actions, under each command we can see whether the commands belonging to this category are with-argument or without-argument. Selector, Information Provider, Delete, Modifier are some categories in which all commands are with-argument. On the other hand, all commands belonging to the category Creator are without-argument. Finally, the commands belonging to the category clipboard can either be with-argument or without-argument commands. In the first subcategory we classify *cut* and *copy* commands and in the second subcategory we classify only the *paste* command.
Figure 4. The hierarchy of user actions
6. GOAL RECOGNITION

When a command is considered as not intended, the problem of identifying the command, among the ones generated, arises. In order to reduce the final number of alternatives, we can use different clues. An important clue is the similarity between objects or commands. Another clue derives from the user’s goals, as these are identified by the system.

This section describes a limited goal recognition mechanism that is used to improve the system’s control. Each action of a user may be categorised in one of four categories, namely ‘expected’, ‘neutral’, ‘suspect’ or ‘erroneous’, depending on how compatible the action is with the system’s hypotheses about the user’s intentions. The categorisation is done based on the notion of ‘instability’, which is a property associated with the user’s file store. IFM uses instabilities to assign meaning to users’ sequences of actions.

The existence of an instability in the file store implies that the user may proceed to a subsequent action that will remove the instability. Instabilities are added and/or deleted from a list as a result of users’ actions. The addition of an instability implies the initiation of a user’s plan whereas the deletion of an instability implies the completion of a user’s plan. For example, if a user places an object into the clipboard through a *copy* command, then an instability is added into the list of instabilities of the file store. This instability is removed if the user pastes the content somewhere in which case the clipboard is emptied.

Instabilities are added when a user issues actions that do not mean a lot on their own. Examples of cases where instabilities are added include the following:

- *When a user creates an empty folder.* In this case one would expect the user to assign contents to the folder or delete the folder; the existence of an empty folder is pointless.

- *When a user creates a directory with only one child.* In this case one would expect the folder to have more contents, otherwise its existence does not have a lot of meaning.

- *When a user creates a folder or file with the name ‘New Folder’ or ‘New File’ respectively.* In this case again one would expect the user to assign meaningful names to the newly created files or folders rather than keep the default names ‘New Folder’ or ‘New File’.

6.1. Using Instabilities to Confirm Goals and Plans

As already discussed above, the existence of instabilities implies that the user may intend to issue subsequent actions that would remove them (although this is not obligatory). Therefore, user actions that delete one or more instabilities are considered as expected.
An instability is removed if a user’s action renders the existence of this instability meaningless. For example, an instability for an empty folder is deleted if the directory obtains some content or is removed. However, there are cases where a user’s action may result in both the deletion of an instability and the addition of some other. These cases correspond to situations where a user may be in the middle of an existing plan.

In particular, there are the following cases depending on the addition and/or deletion of instabilities.

The action does not add but only deletes instabilities.
This implies the completion of some plan. This action is considered as expected. For example, when a file is added to a directory with only one child, it deletes the instability of the directory having only one child.

The action adds and deletes instabilities.
This kind of action results in the continuation of some plan. In this case, the action is considered by IFM as expected. As a matter of fact, the action is considered as expected as long as it deletes at least one instability. We are not concerned with the number of instabilities added. For example, when a file is added to an empty directory, the instability of the directory being empty is deleted but the instability of the file having the default name ‘New File’ is added.

The action adds but does not delete any instabilities.
This kind of action declares the start of a new plan. If there are other previously declared plans that have not been completed then this action is considered as non-expected; the user is supposed to aim at the completion of the already declared plans before declaring a new one. For example, the creation of a new sub-directory in a directory, which already contains two or more children, only adds the new instability of an empty directory and does not delete any old ones. Such actions could be well-intended by the user but they also show that perhaps the user has accidentally neglected to issue some action that would complete previous plans. Therefore, they are used to attract IFM’s attention for further investigation.

The action neither adds nor deletes any instabilities.
This kind of action does not provide any information about the user’s plans. For example, explore(directory), which results in exploring the files and folders that belong to a certain folder.

Instabilities are introduced in order to declare whether a plan is started, continued or completed. In case a user issues a non-expected action the system is alerted in order to search for alternative actions. An alternative action that would be ‘expected’ is believed to be a good replacement for the one issued by the user.

6.2. Categorisation of User’s Actions and IFM’s Hypotheses

One of the most difficult tasks in user modelling is the recognition of a problematic user’s action. As Hoppe (1994) points out, checking the correctness of a given solution,
although being essential for any kind of error diagnosis, is not easy particularly in domains where there is a variety of correct solutions. A user’s action may be considered problematic in terms of his/her hypothesised intentions. As soon as the action is issued, IFM categorises it into one of four categories. This categorisation takes place with respect to the instabilities deleted or added, namely IFM’s expectations. The four categories are the following:

- **Expected actions.**

  A user’s action is considered as expected if it deletes at least one instability. If an action results only in the deletion of instabilities it means that this command completes a plan. Otherwise if the action both deletes and adds instabilities it means the continuation of a plan. When an action is considered as expected the system confirms its predictions about the user’s goals.

- **Neutral actions.**

  These actions have no effect in user’s plans or in the list of instabilities.

- **Suspect actions.**

  These actions contradict IFM’s expectations. In some cases, these actions may result in the destruction of users’ plans, or even useful data.

- **Erroneous actions.**

  Erroneous actions fail to do anything at all. Therefore, they are definitely considered as unintended.

Actions that are categorised as expected are executed at once, because they are believed to lead to the completion of one of the already declared plans. Actions that have no influence on the system’s hypotheses about the user’s intentions are considered as neutral. Such actions are also executed immediately as they are not considered problematic. A problem arises when an action is categorised as suspect. This kind of action is believed to be leading to the creation of a new plan, while other unfinished plans are still pending. These actions are considered as problematic because it has been observed that users tend to complete their already declared plans before starting new ones. However, such actions may also be well intended by the user and correct. Therefore, the category suspect is only used to alert IFM so that it may examine further whether the user intended to issue a different action from the one s/he did; it is not used to mean that the user has definitely made an error. In this respect, IFM searches for alternative actions, which are similar to the one issued and could replace it.

The similarity of the actions is calculated based on HPR. However, an alternative action may be considered as a good replacement if it is not categorised as suspect but rather as expected or at least neutral. Indeed, if an alternative action is found, which is similar to the one issued and is also categorised as expected then the system shows this action to the user in case the user meant to issue this action instead of the one s/he did. In
such cases there are two different reasons for the system to believe that the user may have intended the replacement action. First, the replacement action is very similar to the one issued, which means that the user may have made a mistake. Second, the replacement is more compatible to the system’s hypotheses about the user’s intentions. However, if no alternative command is found then the action becomes neutral and is executed without informing the user that the system went through the previous procedure. In case there are more than one alternative actions found, the system selects the action to be suggested based on a degree of certainty that is computed for every action generated. The degree of certainty is computed using the certainty parameters of HPR. The generated actions are presented to the user in ascending order of their associated certainty degrees.

An example of the categorisation of actions is illustrated in the following example: The state of the user’s file store is presented in Figure 5.

![Figure 5. The user’s file store state](image)

The user issues the action `deldir(A:\vacation\hotel2\)`. The system classifies the particular action as `suspect`, because if the user deletes `A:\vacation\hotel2\`, the files `room1.txt` and `room2.txt` will also be deleted. This action may have been unintended therefore IFM generates alternative actions. However, the system finds no other command similar to the one issued but it finds another object to be deleted. So the alternative action generated is `deldir(A:\vacation\hotel1\)`. This action is considered `expected`, because it deletes the instability for the empty directory `A:\vacation\hotel2\` and adds the instability for the directory with one child `A:\vacation\`. The degree of certainty for this command is high as `hotel1` and `hotel2` have very similar names and are displayed in a neighbouring order.

7. HPR FOR USER MODELLING

IFM constructs a long term and short term user model in order to keep track of the user’s correct and incorrect beliefs about the domain. This user model represents the system’s hypotheses about the user’s beliefs and intentions; henceforth when we refer to the user’s beliefs and goals we will mean the system’s hypotheses about the user’s beliefs and goals.

The user model consists of the user’s beliefs about the file store and his/her goals. Each time a user completes an action the user modelling component examines whether the user intended the particular action, whether s/he was aware of its semantics and finally whether the effects of the particular action were compatible with the user’s goals. IFM uses HPR transforms for generating alternative actions in case an action contradicts the system’s expectations about the user’s beliefs.
7.1. The Basic Principle

One problem with the application of HPR into the system is that HPR presumes that explicit questions are asked to the user. Therefore, the system makes the assumption that users ask questions to themselves. These questions are made to themselves in their effort to choose the right command and the objects that the command is referred to.

Every time the user issues a command IFM assumes that the user believes that the particular action is acceptable by the system and results in the accomplishment of his/her goal. The system’s assumption about the user’s beliefs is called the basic principle. The basic principle assumes that the user asks himself/herself the following questions every time s/he issues a command:

What is the syntactic structure of the command?

Is the execution of the command acceptable to Windows?

The HPR statements that correspond to the above questions are the following:

\[
\text{internal-pattern(action)} = \text{selected-pattern}
\]

\[
\text{Windows-acceptable(selected-pattern)} = \text{yes}
\]

Commands may be classified into two main categories with respect to their syntactic structure as already explained in Section 5 and illustrated in Figure 4; the first category is called ‘with-argument’ and the second category is called ‘without-argument’. This categorisation of commands is used to represent what the user believes about the semantics of a particular command that s/he issues. The user’s beliefs about the semantics of the command are represented in the form of the first HPR statement of the basic principle. For example, if the user has selected the file ‘room1’, as an argument and then executed the command \textit{copy}, the selected-pattern would be \textit{copy selected_item}. So the first statement is formed to be \textit{internal-pattern(copy room1)} = \textit{copy selected_item}. In case the user had made a mistake and had not selected any item before executing the command then the first statement would be \textit{internal-pattern(copy)} = \textit{copy no_selected_item}, which may be what the user erroneously believed about the semantics of the command.

The connection between the first and the second statement is made by the selected-pattern, which refers to the syntactic structure of the command executed. The second statement would be formed by replacing the selected pattern with the referent of the first statement. In the case of the example of the command \textit{copy}, the second statement would be: \textit{Windows-acceptable(copy selected_item)} = \textit{yes}. This means that the user believes that the action s/he has issued is acceptable by Windows.

In case there has been an error, the second statement would not be correct. For example, in the case where the user has not selected any object the second statement would be \textit{Windows-acceptable (copy no_selected_item)} = \textit{yes}, which is not correct. In such cases the system generates HPR transforms that result in different actions that the
user may have meant but would be correct. For example, an argument transform in the first statement could change the action `copy no_selected_item` to `paste`. If this was done then the two statements would represent correct beliefs.

7.2. Certainty Parameters

The main problem with the generation of alternative actions is the production of many alternatives. A solution to this problem is ordering the alternative actions in a way that the ones, which are most likely to have been intended by the user come first. Certainty parameters give a good tool for ordering the alternatives. Certainty parameters are used in order to calculate a degree of certainty for every alternative action.

The certainty parameters that have been used are those of HPR adapted to our problem. The five certainty parameters are: the degree of certainty ($\gamma$), the degree of typicality ($\tau$) of an error set in the set of all errors, the degree of similarity ($\sigma$) of a set to another set, the frequency ($\phi$) of an error set in the set of all errors and the dominance ($\delta$) of a subset in a set.

The degree of similarity is applied in SIM statement transforms. This parameter is used to calculate the similarity between two commands or two objects; and consequently the similarity of two actions. The similarity between two commands is static and is pre – calculated. Its value is based on the relative position of the command in the hierarchy of users’ actions in Figure 4. Two actions that are neighbouring in the hierarchy of users’ actions have a high degree of similarity; for example the commands `mktxt` and `mkdoc`. In addition to their relative distance the effects of the commands have also been taken into account. For example, `cut` and `copy` commands have a similar effect; they place one or more objects into clipboard, so they have a great similarity. Moreover, it has been observed that novice users tend to entangle two commands when these are neighbouring in the screen. So the similarity of two commands also depends on their relative distance in the screen. For example, two commands such as `copy` and `cut` have the greatest similarity since their relative distance in the hierarchy of users’ actions and on the screen is limited to a minimum and their execution has similar effects.

A degree of similarity is also calculated when SIM statement transforms are applied to objects of the file store. This similarity cannot be static since the items of the file store are constantly changing. As a result, the similarity between two objects is dynamically calculated. The value of similarity between two objects depends on their relative position in the file store, as this is displayed in the screen. Moreover, the similarity of their names is also taken into account. For example, files `document1.doc` and `document2.doc` have a high degree of similarity.

Another certainty parameter used is the degree of typicality. The value of typicality is calculated dynamically. A degree of typicality is associated with every command. The calculation of the value of this certainty parameter is partly based on the estimated frequency of the use of the particular command by the set of all users; moreover it is also based on the frequency of use of the command by a particular user, as this frequency has been recorded on his/her individual user model. For example, some users never create
new files using the explorer’s command `mkfile` but rather they create files through wordprocessors or other application packages. It would be a bit ordinary to suppose that this user intended to issue `mkfile` in case of an error.

The degree of frequency of an error represents the frequency of occurrence of the particular error by a particular user. In this way we can easily spot the errors that a user is prone to. Hence past errors may be used to predict new ones. Indeed, it has been observed that users tend to repeat the same errors. In order to find the most frequent error of a user we use the dominance of an error in the set of all errors of the particular user. The value of this parameter shows the percentage of a category of error in the set of all errors. For example, if the dominance of the deletion errors is 0.8, we can conclude that the particular user is mainly prone to deletion errors. This, of course, does not mean that s/he does not make other kinds of mistake as well.

All the above parameters are combined in order to calculate a degree of certainty for every alternative action generated by the system. The degree of certainty represents the likelihood that a user may have intended to issue one of the alternative actions generated. The degree of certainty is an approximate number ranging between 0 and 1 and determines if an action is to be proposed to the user and in what priority. Moreover, the degree of certainty is calculated as a sum of all certainty parameters, with each parameter being multiplied to a weight, which is determined with respect to how important the particular certainty parameter is. The formula of the degree of certainty is shown in equation (1).

\[
\gamma = 0.4 \cdot \sigma + 0.2 \cdot \tau + 0.3 \cdot \varphi + 0.1 \cdot \delta .
\]  

(1)

The weight of each certainty parameter was estimated based on the results of the empirical study that was described in Section 3. The results of the empirical study revealed all the important aspects that human experts are taking into account when they reason about users’ actions in order to give advice.

8. EVALUATION OF THE SYSTEM

IFM was evaluated in order to find out how successful the system was at generating advice. The protocols collected during the empirical study were given to human experts in order to be analysed. They were also given as input to IFM’s second executable release in order to compare IFM’s reactions to the human expert’s comments. This comparison revealed that IFM was quite successful at achieving a high degree of compatibility with the experts’ advice.

In most cases there was a diversity of opinions among the human experts. Therefore, in each case, IFM’s reaction was compared to the opinion expressed by the majority of experts, if there was such opinion. In cases where there was no opinion expressed by a majority of experts it was taken that all kinds of advice would be controversial.
Therefore, if IFM generated advice for such cases we did not count it neither as agreeing nor as disagreeing with the human experts.

There were cases where IFM generated only one alternative action with a high degree of certainty. In such cases IFM was quite successful at achieving a high degree of compatibility with the experts’ advice. Indeed, in 80% of these cases IFM’s advice was the same as that of the majority of the experts. For example, a case where IFM’s advice was exactly the same as the advice suggested by 100% of the human experts was the following. The user’s file store state is illustrated in Figure 6.

The user issues the action `copy(A:\Pascal\prog1.exe)`. This action states the initiation of a new plan of the user, who wishes to copy ‘prog1.exe’ somewhere in the file store. The user’s next action is `copy(A:\Programs\Executable)`, which states the initiation of another new plan although the previous one has not been completed yet. If the user issues the second action, then s/he will not be able to paste the first file selected and in this way s/he will not be able to complete his/her first plan. Both IFM and the human experts considered this action suspect and both advised the user to issue a `paste` command. This means that they advised him/her to issue the action: `paste(A:\Programs\Executable)`.

![Figure 6. Example of agreement](image)

**Figure 6. Example of agreement**

An example of a case where IFM’s advice disagreed with that of the majority of the human experts is the following. The user’s file store state at the beginning of this example is illustrated in Fig. 7.

![Figure 7. Example of disagreement](image)

**Figure 7. Example of disagreement**
The user issues the action `delete(A:\Courses)`. However, this action will result in the loss of two files, which may be rather important for the user, so IFM is alerted to generate alternative actions. The advice produced by IFM is the following: `delete(A:\Projects\Courses)`.

The folder ‘Courses’ is empty and its deletion will not result in the loss of useful information. On the other hand, 70% of the human experts suggested that the user should delete the folder ‘Projects’, which is neighbouring to the folder selected by the user. The folder ‘Projects’ contains two empty folders and the action suggested by the human experts would also result in discarding information, which is probably useless.

However, there were also cases where IFM produced more than one alternative actions with close certainty degrees. In 75% of these cases one of the alternative actions generated by IFM was compatible with the suggestion of the majority of the experts.

In addition to the comparison of IFM to human experts’ comments there has also been another evaluation of the system. This evaluation was based on what users of IFM thought about the system and how they behaved when using IFM.

In this evaluation, there were 8 novice and 8 expert users that took part. All 16 users were asked to interact with IFM, as they would normally do with a standard file manipulation program. During their interaction with the system, we used the computer logging method to register all users’ actions. The protocols collected were studied in order to check where IFM had intervened, the kind of advice that it produced and finally the effectiveness of this advice. After completing their interaction with IFM, users also had a small interview about their impressions and comments on IFM.

The results of this procedure were the following:

- In 85% of the cases where IFM intervened, the user had actually followed IFM’s advice.
- In 10% of the cases that IFM intervened the user had realised that s/he had made an error but did not find any of the proposed alternative commands suitable and issued a completely new action.
- In 5% of the cases that the user was alerted by the system, the user totally ignored IFM’s advice.
- IFM did not recognise all users’ errors, these correspond to 10% of the users’ actions that resulted in a state that was undesirable by the user.

Finally, users were interviewed after their interaction with the system:

- 56,25% of the users found the interaction with the system good, 25% found it mediocre and only 18,75% thought that it needed a lot of improvement.
- Only 12,5% of the users found the method of intervention annoying whereas 56,25% found it good.
Finally, 62.5% of the users thought that the advice produced by IFM really helped them in the interaction. Only 12.5% of the users found the advice unnecessary.

9. CONCLUSIONS AND FUTURE WORK

In this paper we have described IFM, a GUI that incorporates intelligence, so that it may help users accomplish their goals more effectively. The motivation for this research has been reinforced by the results of an empirical study, which showed that users of a standard GUI often encounter problems and frustration. Moreover, the empirical study also showed that users may not realise that they need help. Therefore, an active help system that generates advice on its own initiative would be useful to users.

IFM’s advice is generated spontaneously when the system thinks that the user needs it. There are two main underlying reasoning mechanisms. One mechanism gives more meaning to sequences of users’ actions; it performs goal recognition based on the effects of actions. The other mechanism is based on HPR and generates alternative actions that a user may have intended instead of the ones s/he issued. One very important aspect of these mechanisms is that they are based on domain independent principles. Especially HPR is a domain independent theory that represents human reasoning for producing plausible guesses. The reasoning that may lead to plausible guesses may also lead to human errors. The generality of the theory shows that there is a potential for its use in other help systems as well. Indeed, IFM is the second case study for the adaptation of the underlying reasoning mechanisms; the first one was for the domain of the command language of UNIX rather than a GUI.

The two underlying reasoning mechanisms work independently of each other and they both contribute to the user modelling of the system. One mechanism confirms the results of the other. In this way, the system achieves better control. Therefore, the alternative actions that are generated using HPR have to be confirmed by the goal recognition scheme in order to be considered as good enough advice to be suggested to the user.

HPR has been successfully adapted for the purposes of the intelligent help for the GUI. We have used HPR’s statement transforms and certainty parameters. The statements that are used are formed in the basic principle and the certainty parameters have been adapted to the use and misuse of commands by computer users. The values of certainty parameters are calculated based on observations made by the system about groups of users and individual users. Certainty parameters provide a good tool for ordering alternative actions in terms of their likelihood to have been intended by the user. However, HPR has not been fully implemented and used in IFM. Further implementation of more HPR concepts may render the system’s advice even more useful for users.

The evaluation of IFM showed that it was quite successful at producing reasonable and helpful advice that was compatible to advice suggested by human experts. This supports the idea that adding intelligence to an interface may result in less frustrated and more productive software users. However, the evaluation also showed that there is scope for improvement of the system so that it may achieve better standards of advice. Indeed
in subsequent releases of IFM, we intend to enhance the user modelling component to include more sources of information about the individual user. Such sources may be the user’s level of expertise, the degree of his/her carefulness and specific preferences.
REFERENCES


