A Data Structure for Interface Navigation

by

Linda Marshall
(8639493-3)

Leader
Prof. Derrick Kourie

submitted in partial fulfilment of the requirements for the degree
MAGISTER IN INFORMATION TECHNOLOGY
in the School of Information Technology of the Faculty of Engineering, Built Environment and Information Technology, University of Pretoria

28 January, 2005

Abstract

Computer systems in recent years have become more accessible. This requires the interfaces with which the user interacts with the system to become more accommodating. The Human Computer Interaction discipline looks at how computer systems can be made more accessible to the user by providing a user interface that is sound in design both technically and aesthetically. To address the issues of accessibility Human Computer Interaction includes techniques developed in other disciplines of Computer Science, such as Software Engineering to facilitate the design and development of the interface and Artificial Intelligence to help implement an interface that aids the user when using the system. Such an interface is termed an intelligent user interface. Intelligent user interfaces are classified according to how they behave. The taxonomy focuses *inter alia* on learning agents and learning apprentices.

The development of interface software needs to be done using sound Software Engineering processes. The Navigational Supervisor - which is a generic, system independent learning apprentice – is therefore designed by following such a process. The requirements of the interface are determined before various architectures are evaluated. The design phase incorporates the requirements and architecture. It proposes an algorithm that ranks the possible navigational paths and presents them to the user who is navigating through the interface. Implementation issues that are relevant to the design are presented and possible systems on which the Navigational Supervisor can be used are identified.

Considering the scope of this dissertation, it should be noted that the work presented does not focus on designing the *best* interface for a system, but proposes a design that focuses on *improving* existing interfaces.

Table of contents

1 Introduction

"There are two things which I am confident I can do very well: one is an introduction to any literary work, stating what it is to contain, and how it should be executed in the most perfect manner…."

Samuel Johnson, of Lord Chesterfield's Letters, 1755

Any computer program requires an interface for the human to understand what is happening. This interface is termed the *User Interface*. Many interfaces are static, meaning that they do not change. Often, this may mean that they do not take the needs of the human user into account. The discipline of Human Computer Interaction (HCI) looks at how an interface should be designed so that it is intuitive – i.e. easy for the user to understand and use. Because of advances in modern technology, interfaces can be more dynamic than their predecessors. It is therefore necessary to consider how an interface may be made more adaptive (intelligent) to support user intuition. This idea is captured by Norman (1995), who states that:

> "…people are required to conform to technology. It is time to reverse this trend, time to make technology conform to people."

Before determining what an adaptive interface is, it is important to ask the question, "Why were interfaces traditionally static?" The answer lies in the types of systems written in the past.

According to Langley (1997), early computer software was aimed at solving specific business and scientific problems by following a given algorithm. This meant that the interface required limited user input – a user would merely enter arguments to the program at run-time. As the computer technology improved and the user base increased, the need for a more comprehensive interface became apparent, resulting in menu systems and, eventually, in graphics-based interfaces.

At the simplest level, to approximate an adaptive interface, many earlier systems allowed users to customise the interface according to their needs. This included changing colours, fonts, backgrounds, menus and the like. This customisation does not however classify the interface as intelligent (Jobst, 2002).

The next step was to develop interfaces that did not require the user to do the customisation of the interface. Instead, the system itself changes the interface according the user needs and level. This changing of the interface is often referred to as "personalisation" (Jobst, 2002), "self-customising" (Schlimmer & Hermens, 1993), or "adaptive" (Langley, 1997). It also became evident that this was extremely difficult. For many years (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996), efforts to incorporate intelligence into user interfaces have been underway. The results are referred to as "Intelligent User Interfaces". The commercial use of these interfaces has however not lived up to expectations. This situation has been changing with the use of relatively simple AI techniques to make the interface act more intelligently.

This dissertation will focus on a limited dimension of intelligent interfaces – it considers interfaces that require the user to follow a path through the interface. These interfaces will be referred to as navigational interfaces. The dissertation will argue that *Interface Design and Development* is an interdisciplinary field, encompassing not only computer science but extending out to disciplines dealing with education and the humanities. In addition, within the computer science discipline, interface design and development calls for a symbiosis between sub-disciplines such as Software Engineering, Artificial Intelligence (specifically machine learning techniques), and the Human Computer Interaction fields of study.

In the sections that follow, user interfaces (section 2), and more importantly the types of interfaces will be discussed and a taxonomy will be proposed. In section 3, intelligent interfaces will be studied and the taxonomy will be adapted to make provision for "Intelligence" within an interface. Section 4 will categorise intelligent interface systems that are currently being researched, as well as those currently running on production systems. The main focus of the work is presented in

section 5 in which the development of a technique to navigate existing interfaces will be discussed and suggestions as to how the technique may be implemented will be given. This section effectively covers the requirements, architecture and design phases of a software engineering project Section 6 concludes with an evaluation of the technique, discusses its viability and points to future work that should be done to further advance this development.

2 User Interfaces

"Keep it simple: as simple as possible, but no simpler"
- A Einstein

A user interface (UI) provides a means by which a human can communicate easily and effectively with a computer. This communication is called a dialogue. There are two types of dialogues: sequential, as found in the conversational world; and asynchronous, as found in the model world.

A *sequential dialogue* user interface has a set path to move from one part of the dialogue to the next. This type of dialogue helps the developer and user to visualise the logic sequence that any dialogue path is going to take. Examples of sequential dialogues are: request-response interactions, typed command strings, and navigation through networks of menus.

The user of an *asynchronous dialogue* system manipulates the user interface directly, that is there is no specific path defined for the user to take. The dialogue has multiple threads (which will mean that it is inherently event-based) and the user is free to choose whatever task is desired. The threads themselves can either be sequential or asynchronous. An asynchronous dialogue is usually a graphics based point-and-click environment.

User interface (or more generally speaking – Human-Computer Interaction) researchers are divided into two distinct groups: those who are concerned with the people that use the interface and how the interface looks to them (aesthetics); and those who are interested in the technological aspects of the interface. Even though there is this research distinction, there must be symbiosis between the two for the development of what is termed “good” user interfaces. Benyon (1998) identifies four components in HCI that need to work in harmony: the users; the work that needs to be done; the environment in which the work needs to be done;

and the computer system required to do the work. The first three components may be categorised as human factors while the last one is a technological aspect.

The human factors (also referred to as ergonomics of the interface) will be discussed briefly in sections 2.1 and 2.2 by looking at the aesthetics of a good interface and what properties such an interface may have. Section 2.3 will introduce a classification for interfaces according to technological criteria.

2.1 Aesthetics of a User Interface

When designing an interface the human side should be taken into account. There is a distinction between the user's model of the interface and the interface as defined by the program designer and therefore as "understood" by the computer. The user has a mental model of the system (refer to figure 1). This model is the one that the user expects the computer to have. The user bases this model on past experience. As a result, the model can change as the user becomes acquainted with the system, and consequently the level of the user changes (User levels are discussed under properties of interfaces).

When a program is engineered and a user interface is defined, that which is defined by a designer provides a model of what the user is to see. This is called the conceptual model. The better the conceptual model, the more the mental model of the user corresponds to it. This means that the closer the user's mental model is to the conceptual model, the "better" the interface.

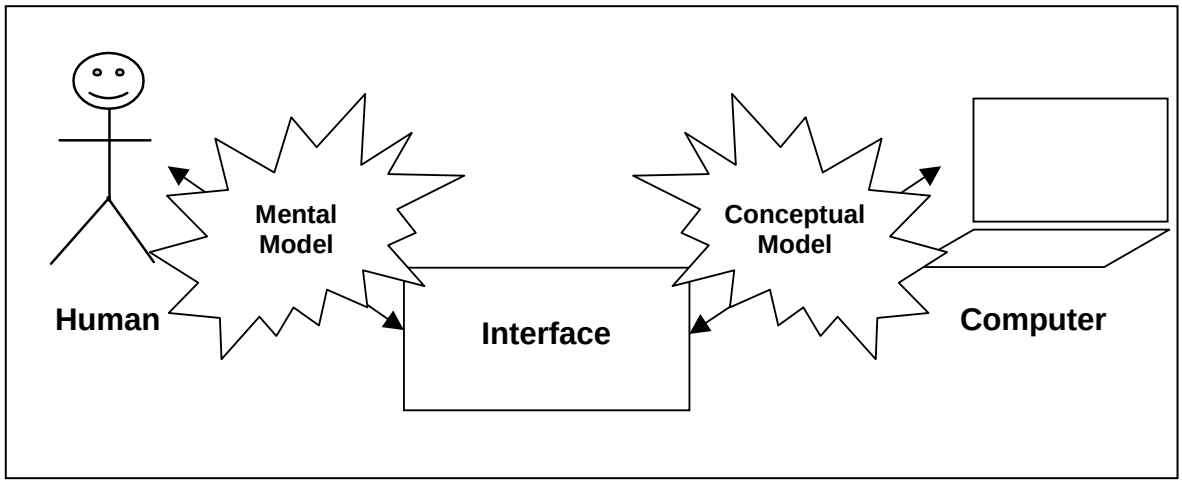


Figure 1 - Human-Computer Interaction (Geyser & Van Brackel,1991)

Notwithstanding the modelling of the interface, attention must be given to how the user perceives the interface. The interface needs to be made more intuitive and therefore the design must be geared more towards how the user is going to use and perceive the data presented by the interface and not how the developer of the interface wishes the data to be perceived. Perception plays on the senses of the user, so aspects of vision, hearing, touch, taste and smell play a role (Downton, 1993). A computer is currently only able to successfully address the senses of vision and hearing, which would mean that an interface must be designed accordingly. Visual aspects of an interface include contrast, brightness, visual angle and field, colour use. Auditory aspects include alerts, background sounds, relevant speech, etc. The properties of an interface that follow, address the aspects of perception that will enable an interface to be defined as aesthetically pleasing.

2.2 Properties of a User Interface

Properties of a user interface have an impact on how “user-friendly” the interface of the system is perceived to be by the user. Pressman (1992) mentions three levels of human factors that must be taken into account to ensure that the system is usable. The first level covers the “look-and-feel” of the interface and how

intuitive it is. Level two encompasses user behaviour, while the third level has to do with the tasks the system performs and the tasks that the user expects the system to perform. Each of the levels will be discussed in more detail in the paragraphs that follow.

2.2.1 Human Perception

The first level has to do with how the user perceives the interface. This means that both human factors and system-based factors need to be taken into consideration (Pressman, 1992; Downton, 1993). These factors are summarised in the following table.

Human factors	**System factors**
Cognition	Visual consistency
Ability to reason	Interactive consistency
Prior experience	User level
Personality	Ease and effectiveness of use

The human factors influence the mental model of the user and it is therefore important that the system factors must enhance the conceptual model so that it moves closer towards the mental model. The intention of the system factors will now be summarised.

Visual consistency

This factor has to do with what is generally termed “screen design”. Screen design is not only a HCI focus, but has in the past and will in the future be a topic debated and researched by all in the arts, education as well as the computer industry.

Aspects that need to be considered when visually designing the interface are (Olson & Wilson, 1985; Lucas, 1991):

- How cluttered is the screen?

- Is there a balance between the items on the screen and are subjects that are related grouped together with areas designated for certain activities?
- Does the screen design follow the natural eye movement of the user, from top left to bottom right?
- If information encompasses more than one screen, is there continuity between the screens?
- Has the screen on which colour has meaning been designed so that the colours are also distinct for the user that is colour blind?
- Is text displayed in both upper and lower case letters?
- Are items that are frequently used on a menu prominently placed, for example first?
- Are screens that follow each other consistent in their layout, use of colour, display of text and menu layout?

Downton (1993) suggests that an interface should also appeal to the human senses of sight, hearing, touch, taste and smell. Currently interfaces predominantly appeal to the first two senses.

Interactive consistency

To help a user become comfortable with an interface it is important to ensure that how the interface reacts and presents options is consistent throughout the system.

Olson and Wilson (1985) suggest that the following must be taken into consideration when developing the interaction aspect of the interface:

- Does the interface help the user to understand what is expected of them?
- Has the mental model of the user and how the user reacts been taken into account during development, rather than the conceptual model?
- Are the keystrokes (if any) that the user needs to make consistent with other software (and within the system itself) and have they been kept to a minimum?
- Have the controls on the interface been used conventionally?
- Have checks and balances been put in place that is a user follows an unintended path of interaction that the system does not terminate abruptly?

User level

There are three distinct user levels; novice, intermediate and expert (Constantine, 1993). When a user uses the system for the first time, the user level is novice. These are users with little experience in computer use and should not be penalised when using an application, but should be able to begin working immediately. As the user becomes more familiar with the system the level progresses from novice through intermediate and finally reaches expert. Expert users on the other hand must not be penalised by having a simple interface for complex problems.

An interface should be able to adjust to the user's skill level (Geyser, 1992) as well as the user's needs. This would mean that a user should be content with the interface and can use it without feeling restricted. The ability of the user to therefore personalise and customise (Jobst, 2002) the interface to their liking, ability and personality (Pressman, 1992) becomes of utmost importance.

It is interesting to note that the more advanced the user, the less the visual and interactive consistency needs to be to make the interface easy to use and effective.

Ease and effectiveness of use

Increasing the ease and effectiveness of use of an interface is achieved by ensuring that the visual consistency and interactive consistency are achieved for the particular user level. In addition to this the attention of the user can be focussed on important aspects of the interface by making use highlighting techniques. Highlighting techniques include the use of colour, icons, animation, sound, font-size etc.

Olson and Wilson (1985) suggest that before an interface is released, it should be tested by both expert and novices users to determine what potential problems are and to be able to sort them out. This will in the long run ensure that the interface is easier and more effective to use.

2.2.2 Behavioural Properties

A user of a system has a particular behaviour and it is necessary for the conceptual model of the interface take into account, and to understand the user and their behaviour (Pressman, 1992). This behaviour is dependent on personality, background and the user level. The consistency of the interface also has an influence on the behaviour of the user when using the system.

2.2.3 Task Analysis

Pressman (1992) emphasises the importance of understanding what tasks the system is to perform for the user as well what tasks the system expects the user to perform. Downton (1993) states that task analysis comprises of setting up a task taxonomy for the system and the development of a mental model that stipulates which tasks are to be performed and when.

Callahan (1994) warns that interfaces and the design thereof must not go too far. The interface must be kept simple yet functional. Complex interfaces discourage the user to reason about how the system works and consequently personalise and customise the interface to their liking and therefore make the work environment more pleasant.

2.3 Classification of User Interfaces

All interfaces must take the human-aspect into account for the interface to be classified as a “good interface”. In this section, however the emphasis is on the technological aspects of interfaces that relate to its internal workings, and not the audience (users) for which the interface is intended. Nor is the emphasis on the question: Is the interface intelligent or not? Intelligence of the interface is not dependent on the type of interface and vice-versa, but is dependent on the technique used to create the perception that the interface is intelligent.

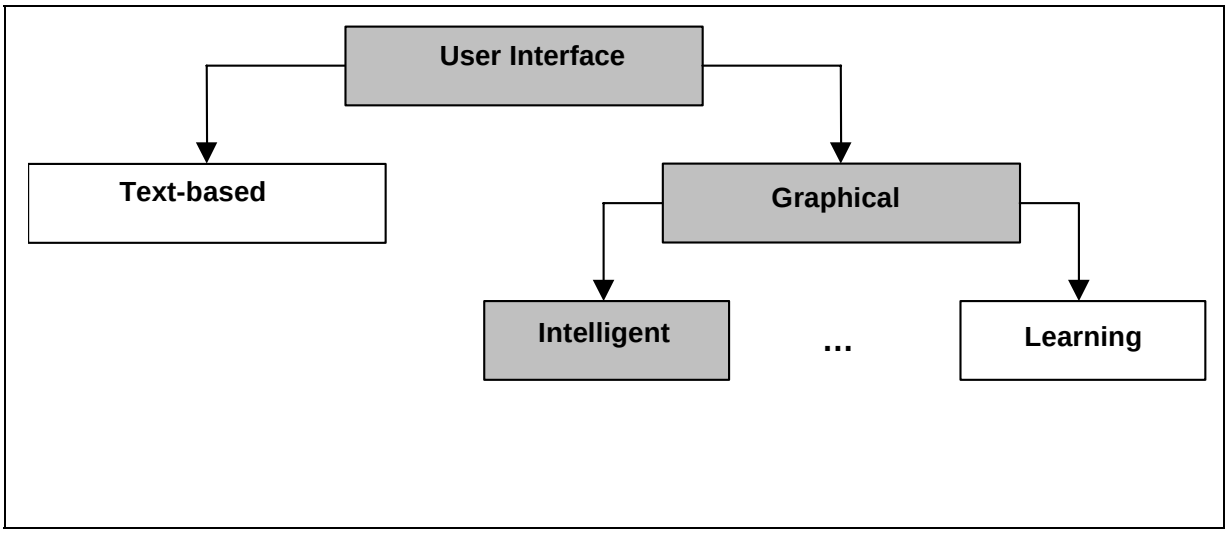


Figure 2 – Taxonomy of a User Interface

The sections that have been greyed, in figure 2, are the interfaces that are of interest in this dissertation. A brief overview of text-based, graphical and learning interfaces will be given. The notion of an Intelligent Interface will be investigated in detail in Section 3.

Learning and Intelligent interfaces can also be found in the text-based systems, but are not as common as their graphical counter-parts. The reason is that, as interfaces became more complex, so did the technology needed to interact with them. Learning interfaces may also be classified as "intelligent" in some circumstances.

2.3.1 Text-based Interfaces

Text-based interfaces were prominent in the days when hardware was more important than the software and processing power was required to run the application rather than the interface. At that stage, the basic method of input was from the keyboard.

With the introduction of the mouse commercially, the text-based interface moved away from being sequential (or command-line) to being asynchronous (menu-driven “point-and-click”) in nature.

2.3.2 Graphical Interfaces

With the advent of graphical interfaces, the asynchronous nature of interfaces became an every day occurrence. The reason is that graphical interfaces are predominantly event-driven. The text-based “point-and-click” interface could now be displayed in a graphics environment. The graphical interfaces became more complex and supposedly “user-friendly”.

2.3.3 Learning Interfaces

Learning interfaces existed to a limited extent in the days of text-based interfaces. They became more prominent with the popularity of the graphical interface and will therefore be briefly discussed as graphical interfaces.

Learning graphical interfaces make it easier for the user to understand the information being displayed. The user is enabled and is able to learn from the interface, which means it provides information in an ergonomic way that is conducive to good learning.

Learning interfaces are in contrast with intelligent interfaces in that they deal with the ergonomics of the interface, while intelligent interfaces deal with the technological adaptability of the interface. The focus of this dissertation is on the technological aspects of interfaces. It is however not disputed that the ergonomics of an interface is of vital importance during the design of the interface and should be taken into account during the software development phase. Often the interface is an afterthought.

An example of a learning interface is a tutoring system – i.e. a system that is intended to instruct the user on some or other domain. Some work on tutoring systems also draws on machine learning with the aim of personalising the instruction process. This means that such systems could also be classified as intelligent interfaces.

Clearly, the most interesting and advanced user interfaces are in some or other sense, “intelligent”. The mechanisms used to make such interfaces intelligent are considered in the section to follow.

3 Intelligent User Interfaces

"I hesitate to say what the functions of the modern journalist may be; but I imagine that they do not exclude the intelligent anticipation of facts even before they occur"
Lord Curzon of Kedleston, House of Commons, 29 March 1898

According to Benyon (1993), an Intelligent User Interface is an interface that automatically alters aspects of its functionality in order to accommodate the differing preferences and requirements of the user. This means that tradeoffs may be made between intelligent user interfaces and the traditional (in this case graphical) user interfaces, as referred to by both Kurlander from Microsoft Research and Lieberman from MIT during a panel discussion (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996). It is important that the "intelligence" of an interface does not cloud the way in which the interface works, does not disturb the user, nor slow the interface down. To gain "intelligence" about the user, the system needs to watch the user's actions and assist the user in making wise choices when using the interface. The manner in which an intelligent interface acts characterises the interface's type.

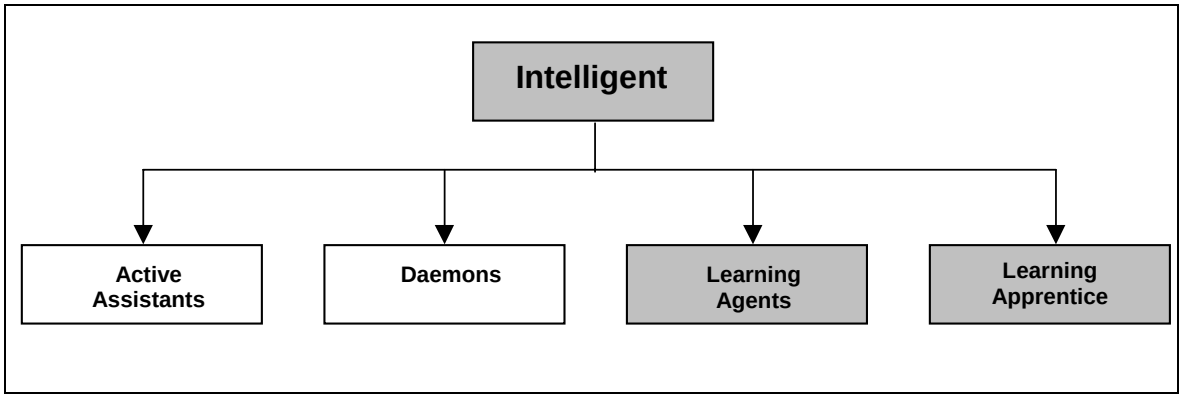


Figure 3 – Taxonomy continued – Intelligent User Interfaces

Holte and Yan (1996) categorise an intelligent user interface as either an active assistant, a daemon, a learning agent (also referred to as an adaptive interface) or a learning apprentice (Holte & Drummond, 1994) (see figure 3).

Active assistants are processes that run in the background. These processes monitor the user's actions and interrupt the user to offer advice. In many instances, the user has not asked the system for the advice.

Daemons are described as programs that will recognise particular patterns of user behaviour and respond accordingly. These patterns have been pre-programmed along with a relevant response. This means that there is a pre-programmed set of situations and action rules.

Langley (1997) refines the classification of Holte, Yan and Drummond by placing adaptive interfaces (learning agents) on the same level as active assistants and daemons, moving learning agents and learning apprentices down a level (figure 4) so that they are children to the adaptive interface level. He referred to learning agents and learning apprentices as informative and generative adaptive interfaces respectively.

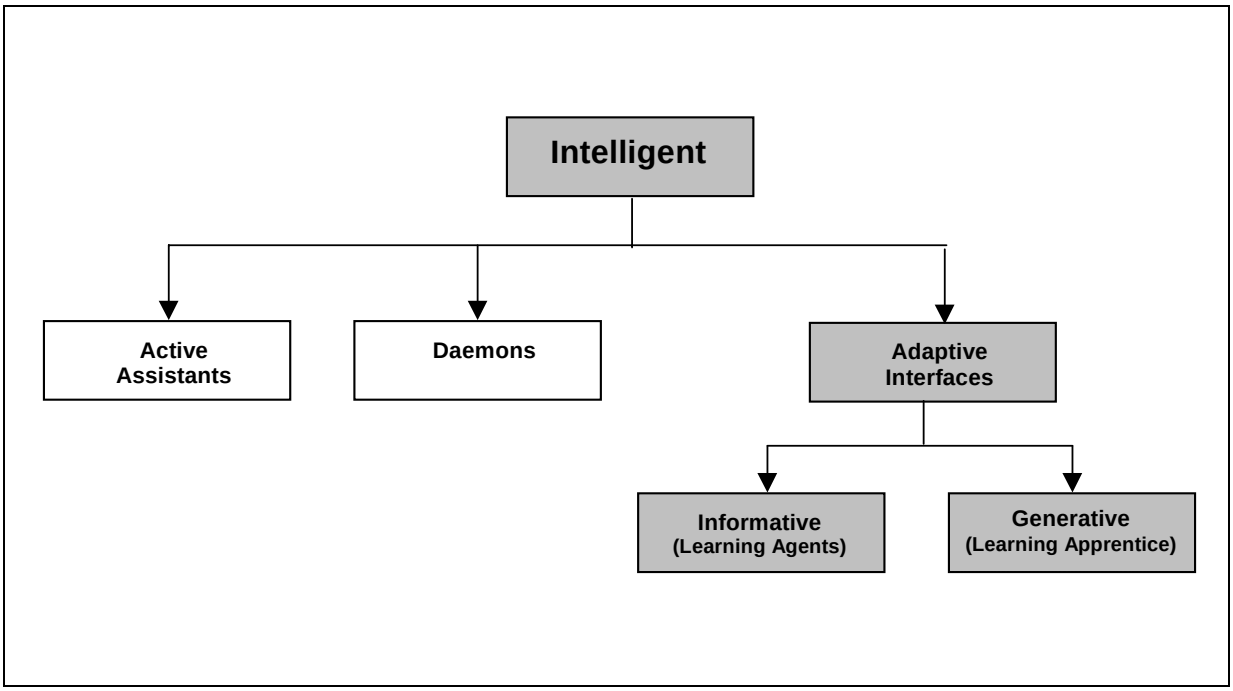


Figure 4 – Taxonomy of Intelligent User Interfaces according to (Langley, 1997)

Irrespective of the model followed, the greyed interface types are of interest in this dissertation, which focuses on learning agents (informative interfaces) and apprentices (generative interfaces).

There is a fine distinction between learning agents and learning apprentices. A learning agent predicts, according to rules and probability, what the *next* move is. A learning apprentice is concerned with predicting the final *goal* of the search rather than merely the next move. Both models make use of machine learning techniques to acquire knowledge and each can be regarded as a type of expert system or advisory system.

Learning agents and apprentices will be discussed in more detail in sections 3.1 and 3.2 respectively. Section 3.3 lists the techniques used to make an interface act in an intelligent way. What must not be lost sight of is that irrespective of what technique is used, the mental model of the user must be taken into account.

3.1 Learning Agents (Informative Adaptive Interfaces)

Learning agents evaluate the user's behaviour over a period time to learn from the user's actions. The learning process then allows the system to "adapt" accordingly and at the appropriate time. Systems of this nature should not interrupt the user at every possible chance, nor should the user necessarily have to take cognisance of the advice given by the system.

A learning agent begins without any knowledge of the user, but builds it up over a period of time giving the user a personalised service. As time progresses the knowledge-base is expanded and the system is more accurately able to suggest the next action to the user. It is very important that the system gives the user the chance to ignore the suggestion. The desirability measure of each suggested action is downgraded if the user does not take notice of it and the action(s) that the user chooses are boosted.

These systems typically form a type of filter. The user is given graded choices according to what the system learnt from the previous actions of the user.

3.2 Learning Apprentices (Generative Adaptive Interfaces)

Learning apprentices are goal-driven. This means that the system tries to predict the user's final destination rather than the next action the user is going to make. The system however does not ignore the user's next possible action, and can in fact suggest it upon request of the user. The user's next action also influences the final outcome of his/her actions. This means that the goal-state of the system could change after each of the user's actions.

This type of system initially requires a lot of the user's time and effort to setup a knowledge base that accurately predicts the user's goals during future encounters with the system. A possible technique, suggested by Holte and Yan (1996), to speed up the prediction process is to infer what the user is not interested in rather than what he is interested in. Inferring what the user is not interested in cuts out a lot of possibilities before the goal is reached.

3.3 Techniques Used to Build Intelligent User Interfaces

Before a user starts, there is no prior source of information from which the system can learn. This means that the Intelligent User Interface system must accumulate the data while the user is working. A user who often makes use of the system will have a wealth of data that has been accumulated and from which decisions may be made. A user who, on the other hand, doesn't use the system as often will have less data accumulated and therefore could perceive the interface as not being as intelligent. This means that the available data from which decisions can be made is a factor of the users' time. To "collect" the data and manipulate the data, the user interface fraternity looked at what was being done in the discipline of Artificial Intelligence. Consequently user interfaces began implementing more

machine-learning techniques to make the user interface appear as intelligent and personalised for a specific user.

Machine learning is based on what is termed learning algorithms. Learning algorithms are software systems that do a task in a domain and improve in performance, based on partial experience within the domain. Two important features of learning algorithms are:

- the goal of learning is to improve performance on a task, possibly involving the creation of new knowledge structures; and
- the algorithm must be able to apply induction beyond the limitations of the training data.

The techniques used may be categorised into two areas: system and ergonomic. System techniques address the way in which the user interface program behaves and the underlying algorithms and disciplines used. Ergonomic techniques have to do with how the user interface behaves to the user.

3.3.1 System Techniques

System techniques focus on how the data is stored, and which algorithms are used to retrieve and manipulate the data and subsequently infer what the user plans to do next or achieve in the long term. The techniques that form the foundation for an intelligent interface system are given in the following table along with a short description of each.

Technique	Description
Data "storage" structures	The data must be stored in some structure. These structures need not be manipulated in memory, but may reside on a database resource to give persistency to the knowledge being built up for a specific user over time. The most prominent data structures are graphs and their specialisation trees or more specifically decision trees using finite state machines (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996). A number use raw data structures to represent their knowledge base. Launching a breadth-first search on a graph would give a good example of learning agents, while a depth-first search is a learning apprentice trait. Learning apprentices also do breadth-first searches so that the level to which the search is done is limited. This is in case the user redirects the search and mainstream time is not wasted going off on a tangent (Standish, 1998). The inference algorithm applied to the structure defines the specific machine-learning technique used.
Filtering	Two types of filtering exist: *Content-based* (also referred to as feature-based or indexing) filtering is one of the older techniques in which objects are classified (Fink, 2003; Holte & Drummond, 1994). *Collaborative* filtering differs from content-based filtering in that users are classified rather than objects. This type of filtering works well in subjective domains in which the user has no particular reason for making a choice.
Concept hierarchies	A concept hierarchy is a structure in which objects are categorised according to concepts. Each concept in turn may be divided into sub-concepts. Concept hierarchies are typically represented using graphs and networks (Tan & Soon, 1996).
Neural networks	A neural network is an artificial simulation of a learning process modelled on the human brain. It accepts inputs which are processed and delivers outputs which depict the results.
Reinforcement learning	Is based on a reward-based learning system. Certain actions will produce greater rewards than others and by trying multiple actions the most rewarding action will come to the fore.
Agents	According to Lieberman (1997) a software agent operates in parallel with the application and the user's interaction with it. The agent has a task to complete and must report the results back to the application that launched the agent.
Constraint propagation	In many cases makes use of graphs and filtering rules to determine if a node complies with the constraints for the particular domain (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996).
Bayesian models and Bayesian (Belief) Networks	These models are also referred to as probabilistic models and require that there is prior knowledge of the domain (Hedberg, 1998). Once more in many instances the networks are modelled using graph structures where the nodes represent data and the edges relationships between the data.
Context-Free Grammars	Constrained CFG's are used to represent role playing (Langley, 1997) in Intelligent User Interfaces. Birnbaum (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996) warns that a system is only as good as the method being used and up till now CFG's haven't made a major impact in intelligent interface research.
Case-based Reasoning	Case-based reasoning makes use of previous solutions to solve current problems.

Figure 5 – Techniques used in Intelligent Interfaces

By no means does a system have to use a single technique to achieve its goal, but multiple techniques may be used to form an algorithm that makes the interface act with intelligence. For example Billsus *et al.* (2002) found that a combination of techniques enhanced the ability of the interface to adapt in time.

Birnbaum (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996) cautions that it is the contents of the intelligent user interface that is important and not the technique or -- by the same token -- combination of techniques used to make the interface intelligent. It is also important to note that not all techniques suite the data-driven characteristics of agents or the goal-driven characteristics of apprentices. Even though apprentices are goal-driven, this does not distract from the fact that they are also able to predict the next action the user makes. It can therefore safely be said that learning apprentices are specialised learning agents. This will be shown in section 4.

3.3.2 Ergonomic Techniques

When discussing ergonomics, the way in which the interface is perceived is of importance. Because of ergonomic techniques, specifically multi-model dialogue, the interface may be perceived as intelligent. However, these are not machine learning techniques *per se*. A multi-modal dialogue gives access to the computer in a number of ways, for example, by: pointing-and-clicking, making use of voice, using gestures etc.

This overview of intelligent interfaces introduced types of interfaces and proposed an interface taxonomy. The sections that follow will discuss systems that have been developed (section 4) and propose an interface supervisor (section 5) for implementation.

4 A Comparison of Existing Intelligent Interfaces

"Would you tell me, please, which way to go from here?"
"That depends a good deal on where you want to go to," said the Cat
Lewis Carroll, Alice in Wonderland

In the previous sections the theory of user interfaces has been highlighted. A taxonomy showing how the different types of interfaces are related has been developed and discussed, and the techniques used to implement the "intelligence" displayed by these interfaces have been mentioned.

Before describing a system that enables existing interfaces to act intelligently, it is informative to consider a number of intelligent user interface systems that have already been developed and to compare these systems with respect to their type (either a learning agent or learning apprentice) and the technique used to implement them. Detailed specifications of the systems discussed are unfortunately not available and therefore the information given is by no means complete. The idea was not to make a comprehensive study of all the systems, either on the market or used for research, but to get a feel for what has been done and how it has been done.

The systems that are to be discussed are classified either as learning agents (figure 6a) or as learning apprentices (figure 6b). The following information regarding each of the systems is given:

- *The name of the system.* This is the name by which the system can be found if it is searched for using a search engine such as Google.
- *The affiliation.* The company or institution that developed the system. Whether the system has changed affiliation. In some cases the system is the brainchild of a specific person and this will also be mentioned.
- *Whether the system is a research or production system.* A research system is usually one that is used to show proof of concept. Many of the successful research systems become production systems. A production

system is one that has been placed in the market and is therefore being used commercially.

- *The techniques used to develop the system*. Billsus *et al.* (2002) stated that using a variety of techniques gives the best results. The techniques used in the system are listed and for comparison purposes are placed in the techniques matrix (figure 7).
- *Any remarks that may be relevant with regards to comparing the system are given*.

System Name	Affiliation	Research/ Production	Description	Technique	Remarks
Syskill and Webert	University of California, Irvine (Pazzani, Muramatsu & Billsus, 1996)	Research	Rates web pages by using the content of a page to build a user profile	Content-based filtering	Biased towards documents that are similar to ones that the user previously ranked high
NewsWeeder	Carnegie Mellon	Research	Filters news on the internet	Collaborative filtering	
Wisewire	www.wisewire.com Commercialised by Ken Lang	Production	Filters news on the internet. Joined Lycos in 1998 to incorporate it into their products	Content-based and Collaborative filtering	Derived from NewsWeeder
Adaptive Place Advisor	Daimler-Benz Research and Technology Center. Initiated by Pat Langley and continued by Cynthia Thompson from Stanford University (Thompson & Göker, 2000)	Research that became Production	Conversational system that helps the user arrive at their destination	Concept Hierarchies	

Figure 6a – Classification of systems – Learning Agents

System Name	Affiliation	Research/ Production	Description	Technique	Remarks
WebWatcher	CMU Project (Armstrong, Freitag, Joachims & Mitchell, 1995)	Research	Helps the user navigate the web	Web-based agent based on probabilities	The user stays in control
Clavier	Hinkle and Toomey (1990)			Case-based reasoning	
Eager	Apple Computer Inc Cypher in 1991	Research	Searched through previous events issued by the user to find a pattern and then follows the pattern	Case-based reasoning	Programming by example system that is written in Lisp
Lumiere	Microsoft Research Decision Theory and Adaptive Systems Group (Horvitz, 1998)	Research that became Production	Reasons about the goals and needs of the user as they work with the system	Intelligent agent that makes use of Bayesian models and language to handle the events	Shipped as part of the Microsoft Office '97 Suite – Office Assistant
Adaptive Route Advisor	Stanford University sponsored by Daimler-Benz Research and Technology Center (Langley, 1997)	Research that became Production	Provides the user with navigational information, with regard to travel, based on previous	Data structure with weights for the routes	
Letizia	MIT Henry Lieberman (Leiberman, 1995)	Research	Incremental search	Software agent that scouts the links on a page that might be of interest to the user depending on their profile	Makes use of the zero-input principle. Does an incremental search by following links on a page
Let's Browse	MIT Henry Lieberman (Lieberman, Van Dyke & Vivacqua, 1999)	Research	Browsing agent for a search engine	Collaborative agent	
PowerScout	MIT Henry Lieberman (Lieberman, Fry & Weitzman, 2001)	Research	Search engine	Reconnaissance agent that searches further than pages in close proximity by using searches based on heuristics	Makes use of the zero input principle. Makes use of complex queries to determine related web pages
Goose	MIT	Research	Search engine	Agent that makes use of Natural Language processing to determine the query	
Alexa	Netscape	Production		Reconnaissance agent that makes use of collaborative filtering	Determines what must be displayed under the "What's related" option

Figure 6b – Classification of systems – Learning Apprentices

From the tables it can be seen that the techniques used to give the user what is perceived as intelligence in the interfaces, defines whether the system is a

learning agent or a learning apprentice. Learning agents make use of techniques based on filtering and hierarchies, while learning apprentices use agents and reasoning techniques.

It is also interesting to note that filtering techniques tend to lend themselves towards systems in which the content of the information viewed needs to be analysed to arrive at the relevant information, in contrast to systems where the method followed to arrive at the information is important. It can therefore be said that learning agents look at the "what?" (Drummond, Holte & Ionescu, 1993) of the information, while apprentices concentrate on the "how?" (Holte & Yan, 1996) the information is retrieved.

Typical questions asked in the learning agent scenario are: "What does the information mean?" and "What information is similar?". Learning apprentices concentrate on questions such as: "How do I get to the information?", "How important is the information?" and "How often is this kind of information retrieved?".

Drummond *et al.* (1993) goes on to suggest that to get to the content (the "what?"), the "how?" is also necessary and therefore the question "How do I get what I require?" is posed. This suggests that a mixed technique model should be used to allow the system to determine the next step as well as predict the final goal. The suggestion that mixed techniques perform better is therefore valid. When implementing an intelligent interface system the techniques should be chosen in such a way that they compliment (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996) each other and are able to both determine the next move (as with a learning agent) and the goal and therefore fall in the category of learning apprentice. It is therefore plausible to conclude that learning apprentices are more specialised learning agents.

	Data "storage" structures	Content-based Filtering	Collaborative Filtering	Concept Hierarchies	Neural Networks	Reinforcement Learning	Agents	Constraint Propagation	Bayesian (Belief) Models	Context-Free Grammars	Case-based reasoning	Total
Adaptive Place Finder				✓								1
Adaptive Route Advisor	✓								✓			2
Alexa			✓				✓					2
Clavier											✓	1
Eager											✓	1
Goose							✓			✓		2
Letizia		✓					✓					2
Let's Browse			✓				✓					2
Lumiere							✓		✓			2
NewsWeeder			✓									1
PowerScout							✓		✓	✓		3
Syskill and Webert		✓										1
WebWatcher	✓						✓		✓			3
Wisewire		✓	✓									2
Total	2	3	4	1	0	0	7	0	4	2	2	

Figure 7 – Implementation techniques matrix

From figure 7 it is easy to see that both filtering techniques and agents are popular when developing intelligent user interfaces. In all the cases when an agent is used, another technique is also included to compliment the agent. Just over 60% of the systems are what is termed multi-modal, that is, they make use of more than one technique.

Many of the systems have been developed for a niche market. Section 5 will look at how a generic system can be developed to make existing interfaces perform in an intelligent manner. This interface will then also be placed in the techniques matrix to see how it matches with the rest.

5 Engineering the Navigational Supervisor

"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult."

CAR Hoare

A navigational interface is considered to be any user interface that allows the user to follow various paths of interaction through the interface of a system. This allows the user the freedom to select a desired path (generally in a stepwise fashion). Basically any Graphical User Interface system that is of an asynchronous nature will fall into this category. Typical examples of asynchronous interfaces are: Web browsers, Help systems with drill down capabilities, SAP R/3, AutoCAD and Microsoft products to name but a few.

The Navigational Supervisor proposed here is a piece of software that will aid in the navigation of the interface in an intelligent way. This software may be added to an existing system or incorporated into a new software product. The focus in this dissertation is as an add-on to an existing system.

The Navigational Supervisor needs to be a hybrid between a learning agent and a learning apprentice. It needs to be able to predict the next move of the user as well as predict what the final destination (goal) will be. Interfaces that track user behaviour and use this tracking information to predict what the user will be interested in next, are called "zero-input" interfaces (Lieberman, Fry & Weitzman, 2001). "Zero-input" interfaces therefore require some prior knowledge about the user. This is in contrast to a "one-input" interface where a user is confronted with a myriad of options and must navigate their own way through the interface. The majority of systems on the market today supply a number of ways for the user to navigate through them: menus, button bars, transactions, short-cuts etc. All these methods form the static (predefined) method of navigating through the system.

To ensure that the interface to which this Navigational Supervisor is attached acts in an intelligent way, it is necessary to predict (or pre-empt for that matter), after a time of learning, what the user's next move is going to be, thereby transforming the interface into a "zero-input" interface. The Navigational Supervisor needs to watch over the users' every move and suggest possible paths for the user to take based on previous experience of the specific user as well as of other users of the system. According to Lieberman *et al.* (2001), supervisors that share navigational information between users are rare. This sharing of information would mean that new users will not be confronted with a "one-input" interface, but may be given a "zero-input" interface, based on other users navigational behaviour, from the start. It should be noted that the Navigational Supervisor may not constrain the user in the choices to be made, but should merely suggest the most common choices made either by the current user or by a number of users.

In the sections that follow, bridging the gap between the mental model and conceptual models (section 5.1) introduced in section 2.1 will be discussed, a design for the Navigational Supervisor will be given (section 5.2) and the structure of the Navigational Supervisor design will be evaluated (section 5.3).

5.1 Introduction of an Interaction Model

To bring the user's mental model of the Navigational Supervisor and the system (onto which the Navigational Supervisor has been added) closer to the conceptual model as programmed into the computer, it is necessary to have a look at both the topics of Software Engineering and Interface Design. The interface of the Navigational Supervisor therefore needs to be designed along with the Navigational Supervisor application. It is also imperative that the Interface of the Navigational Supervisor should not be more prominent on the desktop than the application to which it is attached.

Benyon (1998) introduces what is termed an interaction model between the mental model and the conceptual model. The interaction model describes the interaction

that takes place between the computer and the human. It is the interaction model, encapsulated in the Navigational Supervisor, which will control the adaptability of the system to the needs of the user.

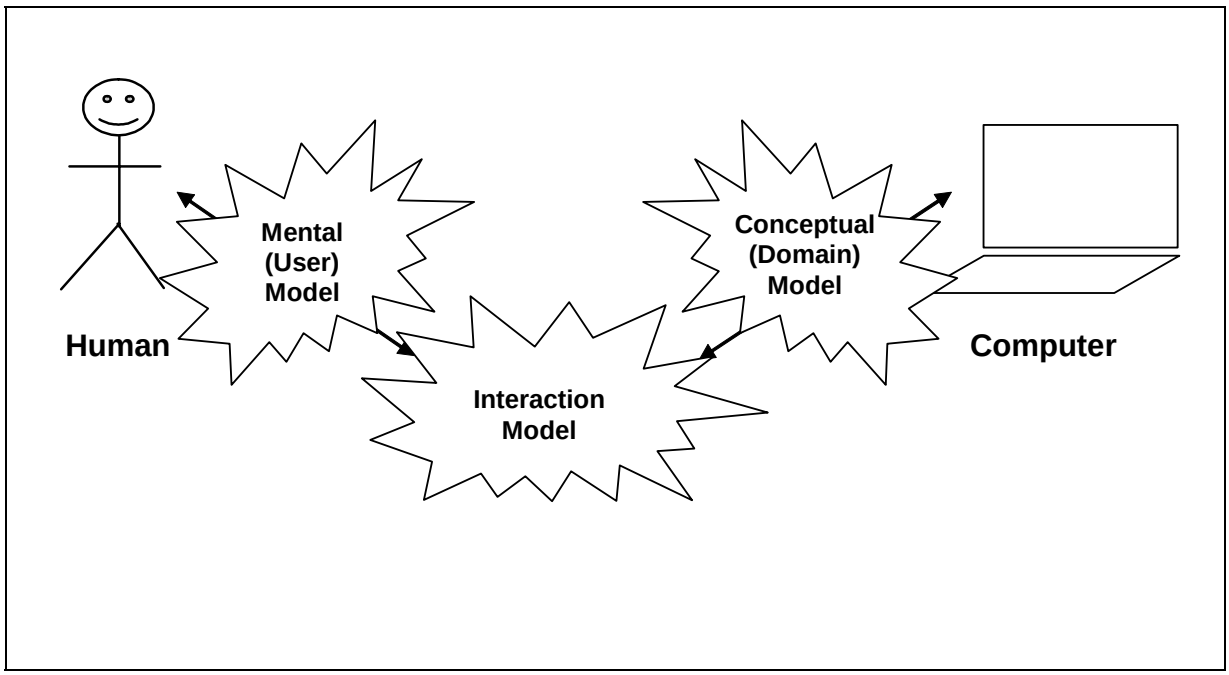


Figure 8 – Human-Computer Interaction architecture for an Adaptive System (Benyon, 1998)

5.2 Designing the Navigational Supervisor

The Navigational Supervisor is a software system that is placed between the human user and the host system. The host system is the system that serves the particular application for which the Navigational Supervisor is being written. Examples of host systems are SAP R/3, Microsoft products, AutoCAD etc.

The process of designing the Navigational Supervisor will follow a basic Software Engineering process in which the requirements of the Navigational Supervisor will be determined (section 5.2.1), followed by the definition of the architecture (section 5.2.2) and then the structure of the design (section 5.2.3). The development of the interface of the Navigational Supervisor will also be taken into consideration (section 5.2.3.4) before discussing implementation issues (section 5.2.4).

5.2.1 Requirements

Each human user (from here on referred to as user) interacts with the Navigational Supervisor (NS), which is placed between the user and the host system. The NS will "instruct" the host system to complete an action. This action may be initiated by the user's direct manipulation of the interface (in which case, the action is merely passed through the NS), or the action may be initiated by the NS to facilitate what may be termed a "jump" in the interface. The user will see the effect of the action in the interface presented to him/her by the host system.

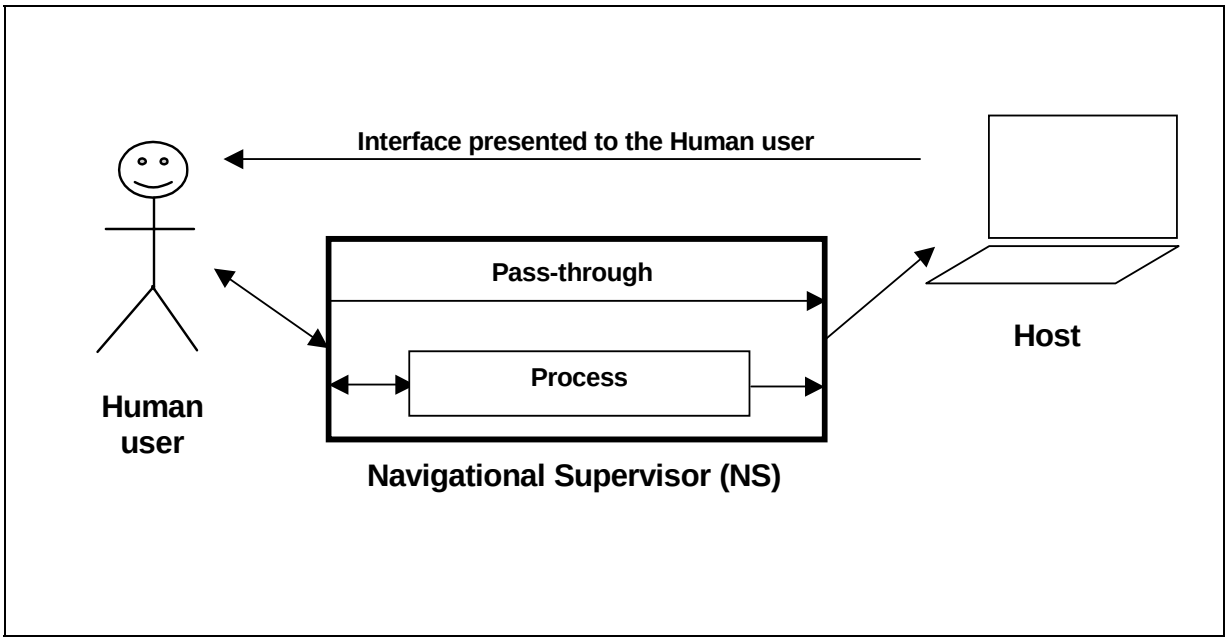


Figure 9 – Requirements of the Navigational Supervisor

The NS provides a personal supervisor (PS) for each user. Each PS communicates with a supervisor of all the personal supervisors (S^2), which communicates either with the host system or with the PS. This means that whenever a user uses the system with the Navigational Supervisor turned on, the system builds or updates a profile for that particular user. According to Lieberman *et al.* (2001), "a profile represents a user's interest in a particular area. The user may create as many profiles as needed to characterise interests." This means

that any one user may have multiple profiles, with the current profile being active. To facilitate the ability to have multiple profiles, the NS will provide and maintain four types of profiles, namely: current, stored, average and system.

- A *current profile* is the profile that was active when the NS was used last or if the NS is being used, the one being used at the moment. The PS manages the current profile.
- The PS manages a *stored profile*, which may be "loaded" and "saved" by the user at will. Once the profile has been "loaded", it becomes the current profile. The stored profile is useful for a user who has multiple devices and can build profiles that are subsets of other profiles depending on the size of the device – for example: Desktop PC vs. Pocket PC. The user may wish to have a subset of a Desktop PC profile available to a Pocket PC.
- The *average profile* is managed by S^2 and is the amalgamation of all current and stored profiles.
- The *system profile* is a structure that records all navigational options currently known to the NS for the system. It however does not give different weights to the options. This profile is also managed by S^2.

The current, stored and average profiles weight the options. The current and stored profiles weight them for the particular user, while each option weight in the average profile comprises of an average of all the corresponding weights over all the users.

The NS is therefore more complex than given in the previous figure (figure 9). A more detailed view of the NS in which the placement of PS and S^2 are given is shown in figure 10.

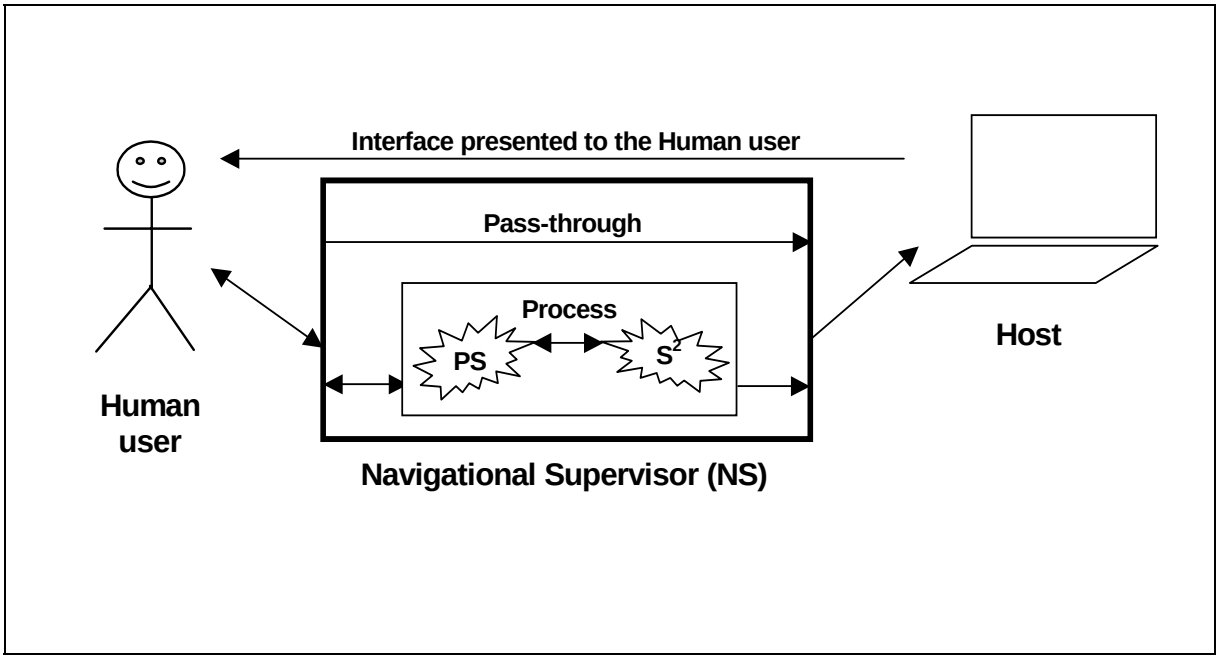


Figure 10 – Navigational Supervisor detail

As the use of the Navigational Supervisor is not compulsory, the NS can be in one of two modes, either inactive (Pass-through) or active (Process). A mode will be in a particular state, as will be explained below. Figure 11 provides a summary of the combinations between the modes, states and which profiles are influenced, a tick(✓) means that the profile is influenced by the state, a cross (**X**) means it is not, and a tick that has been crossed out (✓) means that the profile may be influenced by the state in certain circumstances. For example a profile only becomes a stored profile if the user wishes to save it.

	NS MODES	Inactive NS (Pass-through)		Active NS (Process)		Profile Controlled by
	NS STATES	Dormant	Pass on	Use profile	Pass on	
Profile types	System	✓	✓	✓	✓	S^2
	Average	X	✓	✓	✓	
	Stored	X	X	✓	✓	PS
	Current	X	X	✓	✓	

Figure 11 – Profiles influenced by actions

An inactive NS may be in one of two states:

- *Dormant* – Relevant actions are recorded for the system profile in order for the NS to be able to build up a complete interaction structure for the entire system. The user interacts with the host directly or "as if" directly.
- *Pass on* – All the actions of the user are recorded. These actions are used to update the average and system profiles, respectively. None of the user specific profiles are updated. The NS does not give suggestions either.

An active NS may be in one of the following states:

- *Use profile* – Before interacting with the system, the user is given the option to choose the profile to provide the navigational suggestions. If the user chooses not to exercise this option, the previously stored current profile will be used. In the event that the user makes a choice, the chosen profile will be loaded and it

will become the current profile of the user. By choosing the average profile, the user is making use of what is termed “Mentor Learning” (Fink, 2003). The user also has the option to change the current profile at any time by retrieving a stored profile. Additionally, the user may optionally save the current profile to a stored profile at any time. All the profiles available in the NS are influenced in various ways as described in the table above.

- *Pass on* – All the user actions are recorded and used to update the current, average and system profiles. The user may choose to create or change a stored profile as well. The “Pass on” state is different from the “Use profile” state in that the NS does not make suggestions to the user – it merely monitors user activities and therefore it is conceivable that the current profile may be influenced according to the user actions.

Figure 11 does not show when and how a profile is influenced; this will be discussed in the Implementation section of the NS. It is also important to consider the ethical issues surrounding an inactive NS in a dormant state. To have a facility that monitors the user’s activity without the user knowing about the facility, nor having the ability to switch it off, could potentially be construed as a violation of the user’s right to privacy. It is therefore important that the implementation of the NS should give the user the option to work completely independently of the NS without being monitored.

5.2.2 Architecture

The system architecture shows how the user, NS, host and profile repository are structured and where they are placed in relation to each other. There are two distinct possibilities: close coupling of the host and the NS (effectively embedding the NS in the host); or loose coupling between the host and the NS (in which case, the remote NS provides an API for a host-specific embedded application to call). Each strategy will be discussed in more detail in the sections that follow. How the user will interact with the NS will also be investigated as part of the Architecture.

5.2.2.1 NS Embedded in the Host

Close coupling between the host and the NS means that the NS needs to be written using the language(s), structures and repository (if any) that are native to the host system. In the case of the host system not having its own repository abilities an external repository needs to be used.

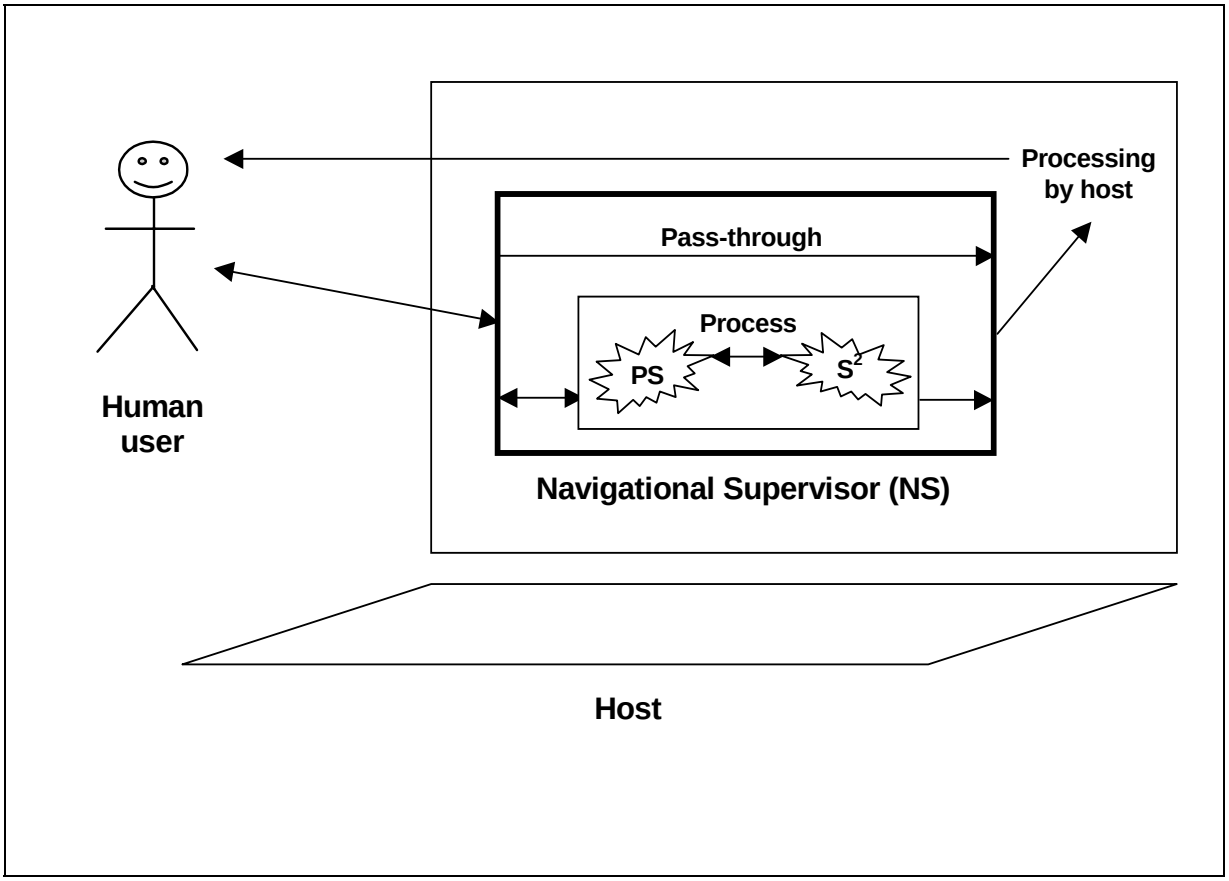


Figure 12 – NS embedded in the host

Embedding the NS into the Host does not necessarily mean that the system runs on only one computer. It is indeed possible that the user could work directly on the host computer. However, the user may also use some kind of front-end to gain access to the host.

5.2.2.2 NS Remote from the Host

The NS may be removed from the host. In this case, the NS needs to provide an Application Programming Interface (API) for the host to link into and a small

application (called NSLink) which will be embedded in the host using languages and structures native to the host.

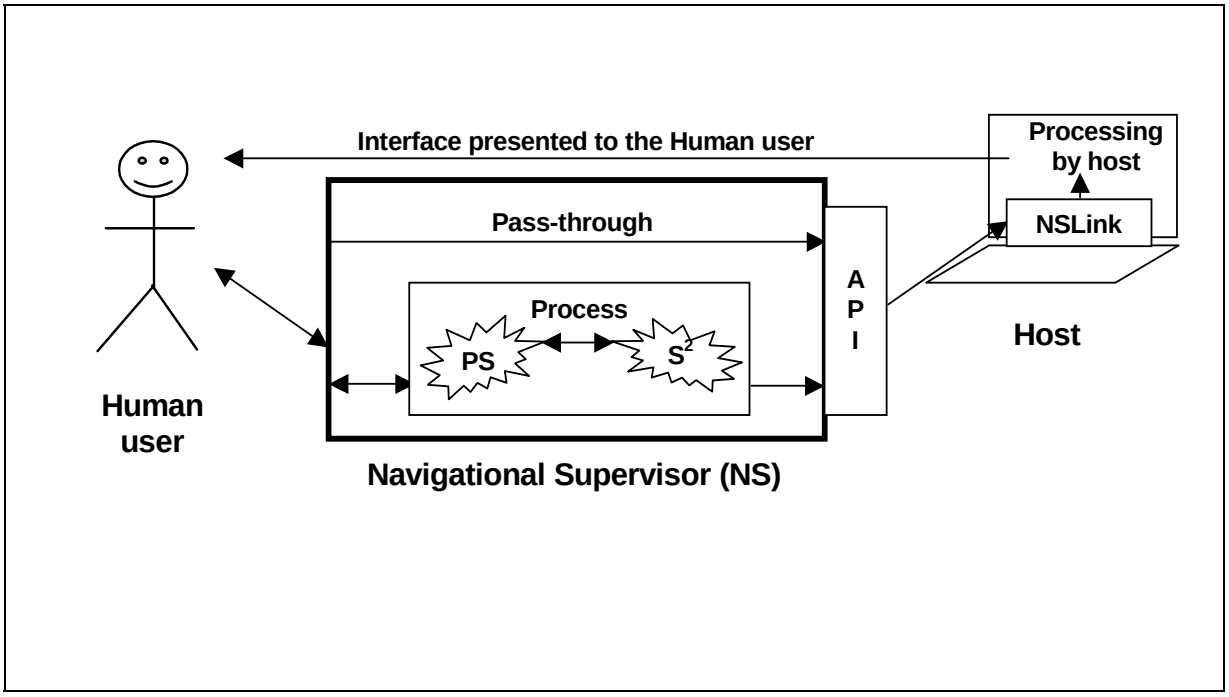


Figure 13 – NS remote from the host

Once more, this architecture can work on one or more computers. The scenario of one computer is where the user is working on the computer that hosts the application and the NS is also available here. It is also possible to place the NS on a computer of its own.

5.2.2.3 Placement of the Profile Repository

The profile repository is used to keep the system, average and the individual current profiles for all the users of the system persistent. It may also need to be able to "store" the stored profile. There will only be one system profile, one average profile, one current profile per user, but there can be multiple stored profiles per user and there may, of course, be numerous users. Two scenarios exist for the placement of the profile repository: either use the repository provided by the host, if one exists; or use an independent repository that is removed from the host to store and manipulate the profiles. The issues regarding the placement

of the repository will be discussed in detail in the Implementation section, as it will have implications on how the NS is implemented.

5.2.2.4 "Embedded" versus "Remote" Architecture

Both an embedded and a remote architecture have inherent characteristics. The table illustrates, according to specified criteria, whether a particular architecture is "favourable" (☺), "unfavourable" (☹) or of "no consequence" (😐). The criteria that are of interest are two-fold:

- Those that are based on developing quality software. Ince (1995) and Lethbridge and Laganière (2001) define various quality attributes. Only those attributes that are relevant to the comparison between the architectures are tabulated in the table (figure 14) below.
- Additional criteria that are considered to be directly of relevance to the Navigational Supervisor. These criteria are viewed in terms of the proposed embedded and remote architectures.

		Embedded	Remote
Software Quality criteria			
Scalability		😐	☺
Reusability		☹	☺
Maintainability		☹	☺
Portability		☹	☺
Extensibility		😐	☺
Navigation Supervisor specific criteria			
Architecture	Implementation	☺	☹
	Visualisation	☹	☺
Communication	User and NS	😐	😐
	Within NS – PS and S^2	☺	☺
	NS and Host	☺	😐
Placement of the profile repository	On the Host	☺	😐
	Remote	☹	😐

Figure 14 – Evaluation of the Architecture

Each of the criteria will be discussed to justify the level of favourability assigned to the various criteria.

- The *scalability* of an embedded system NS is limited to what the host allows. The remote NS on the other hand is not restricted by the host system and therefore it is possible to plug modules in and remove modules at will.
- An NS that is embedded in a host will, with difficulty, be able to serve another host for which it is not written. This decreases the *reusability* of the NS. With a remote system, one NS can easily be used for a number of hosts. The NSLink module would be the only host dependent item in each. This would mean that there will be multiple (one per host being serviced) system and average profiles.
- An embedded NS is dependent on the host and therefore dependent on the structures and functionality provided by the host. This will increase the effort required to maintain the NS. The *maintainability* is linked to the reusability of the NS, particularly for multiple hosts on which the NS may be deployed. Having a remote NS would mean that only one version of the NS code needs to be maintained and displayed.
- A remote NS has no issues regarding *portability*, the embedded version however needs to be customised or rewritten for each host. Also at issue here is the ability of relocating or changing platforms of the remote NS, this is easier than with an embedded NS and will be easier if the repository is also remote.
- The remote NS architecture lends itself to being extended. This does not mean that an embedded system is not *extensible*. It is just more difficult to extend a system that is bounded.
- The architecture of the remote NS is easier to *visualise* and consequently leads to understandability of the system compared to the embedded counterpart. In practice however, the *implementation* of the remote system is more challenging as distance and communication complexities between the NS components become an issue.
- *Communication* takes place bi-directionally between the user and the host. To facilitate the discussion, the communication is broken up into smaller

parts, namely between the *user and the NS*, *within the NS* and between the *NS and the host*. Generally, the communication between the user and the NS, whether the NS is remote or embedded has no consequence on how the user perceives the communication with the system. The user would have also had to communicate with the host and it is therefore important that the communication between the user and the host should be at all times the same irrespective of the presence of the NS or not. Communication within the NS takes place between the PS and S^2. To alleviate a potential communication bottleneck between the NS and the host, the NS also communicates back to the user. In both the embedded and the remote scenarios the communication within the NS does not have a significant impact on the architecture of the NS. What is of significance is the communication between the NS and the host. With reference to the embedded system, there is little chance of a bottleneck between the two while in the remote system there is potential for a bottleneck.

- The *profile repository* can either be placed on the host or remotely from the host. When considering an embedded system, it would be beneficial to place the profile repository on the host to minimise the communication between the host and the repository. In cases where the host does not provide repository facilities it may be necessary to place the repository remotely, in which case a lag in overall response time – as perceived by the user – will be introduced. When considering a remote architecture, placing the repository on the host would cause the host to incur overhead. A remote placement of the repository would mean that it is either placed with the NS or remote from the NS as well. The first instance is more beneficial then remotely. It must be kept in mind that it is the NS that communicates with the repository on a continuous basis to update, store and retrieve the profiles. Therefore the closer the repository is to the NS; the more advantageous it is to the working of the system.

If only the quality criteria defined in figure 14 are taken into account, it would seem that a remote architecture is more favourable than an embedded one. This coincides with the current trends in software development in which re-use and therefore modularisation of design and ultimately code are important. The NS

specific criteria tend towards much of the same: neither architecture shines above the other and therefore, for the purpose of defining the architecture, the NS will be seen as being remote from the host. During implementation a mixed architecture approach may be followed.

Decoupling the NS from the host would imply that another level of decoupling can be achieved, and that is decoupling the NS itself. This would imply decoupling the PS and S^2 within the NS. This once again would raise repository issues and may lead to there being a need for two repositories, one for the PS and one for S^2 each residing with the relevant NS component. If the NS is decoupled, it would be prudent to place the NS on the user's computer. To ensure portability of the profile, the personal profile of the user will need to be replicated on the server to ensure multiple access points.

5.2.2.5 Interaction Process

As part of the architecture it is necessary to determine how the user is going to interact with the NS, how the user registers herself, logs onto the NS, communicates with the NS etc. When a user logs onto the NS, she effectively logs onto the S^2. S^2 will determine whether the user is either a *new user* or an *existing user*.

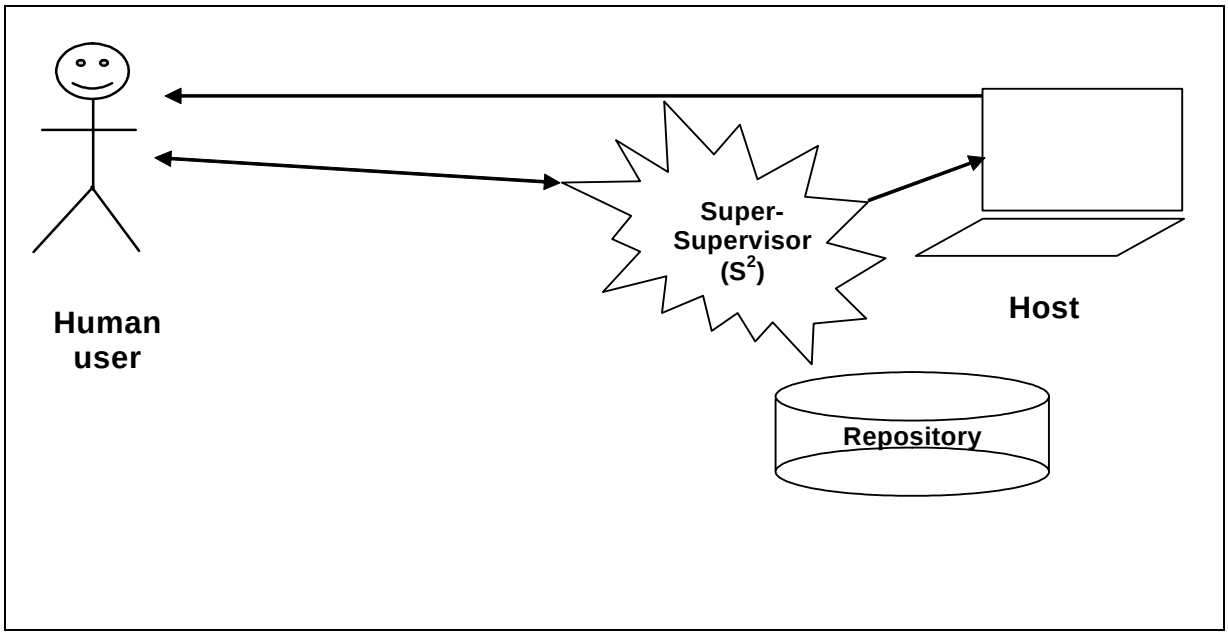


Figure 15 – User interaction with S^2

Depending on the type of user and the choices that the user makes, she will be allowed to influence and/or use specific profiles. The following flow-chart illustrates how the modes, states, profiles and users are interlinked.

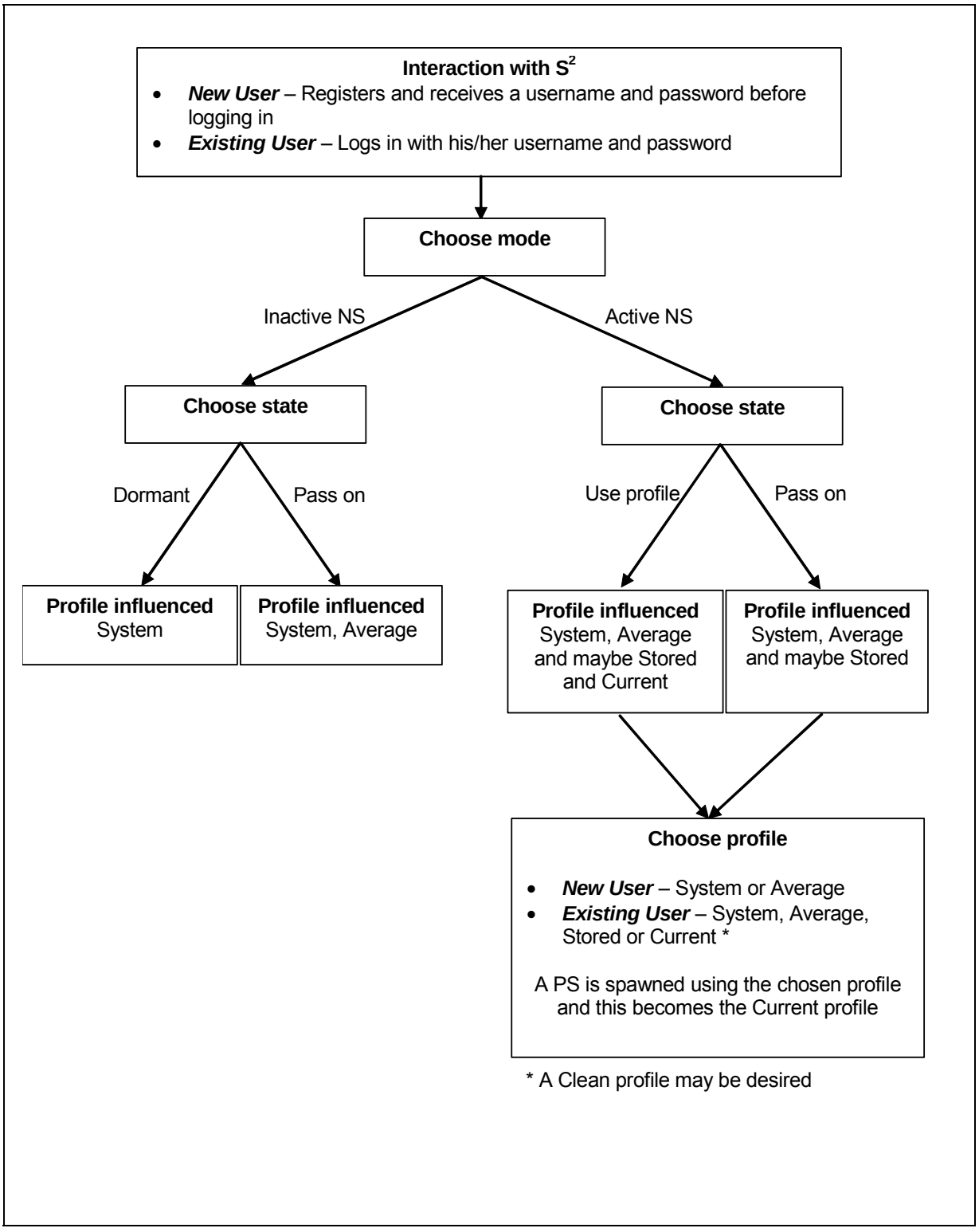


Figure 16 – Interaction between the NS and the type of user

A first-time user, or *new user*, needs to register with the NS before using the functionality provided by it. All users require a username and password and this user will need to have their details captured. Once a user, either *new* or *existing*, has successfully logged into the NS, she may decide whether to make use of (choosing the mode) the NS to help with navigation (Active NS) or not (Inactive NS). The mode chosen will allow the user to choose between two states in which the NS is and these will influence specific profiles. Only the Active NS choice of mode will give the user a choice of profile. Once a profile has been chosen, a PS will be spawned for the user.

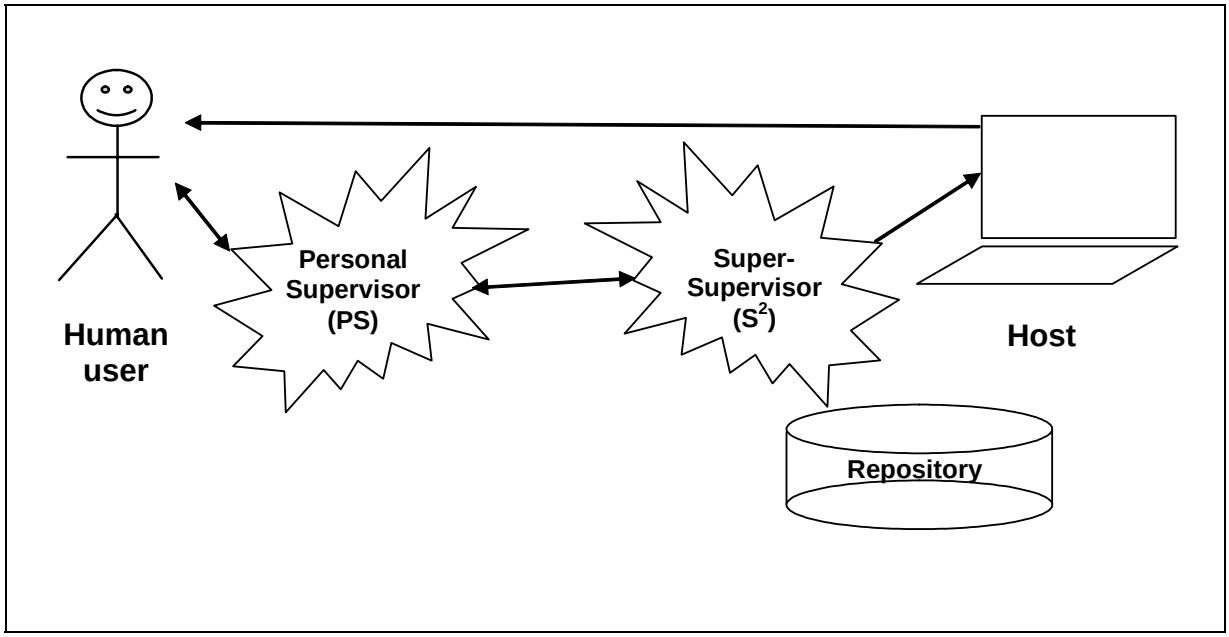


Figure 17 – User interaction with her Personal Supervisor

This means that an Active NS comprises of one S^2 and multiple PS's (one PS per user using an active NS). A user communicates with her PS, which in turn will communicate with S^2, if applicable, or it will react directly with the user. S^2 could react to the PS without any communication to the host or it could communicate with the host. Once S^2 has communicated with the host, the NS is unable to influence the outcome of the communication since the result of this communication is sent directly to the user by the host.

All users (either new or existing) should be given an option to make use of a default when logging onto the NS. The default will place them using an NS in an inactive mode in the pass on state.

5.2.3 Design

In designing the NS, it is important to consider a typical application with which the NS will interact and how these applications react. The typical application is one in which a user is given a number of options via a menu or even sequential buttons to arrive at the particular functionality required. For example, consider an application such as Microsoft Word, where the user chooses the Tools menu-option, followed by the Language option and then the Thesaurus option. An advanced user might remember the short cut if she uses the application and the specific option regularly.

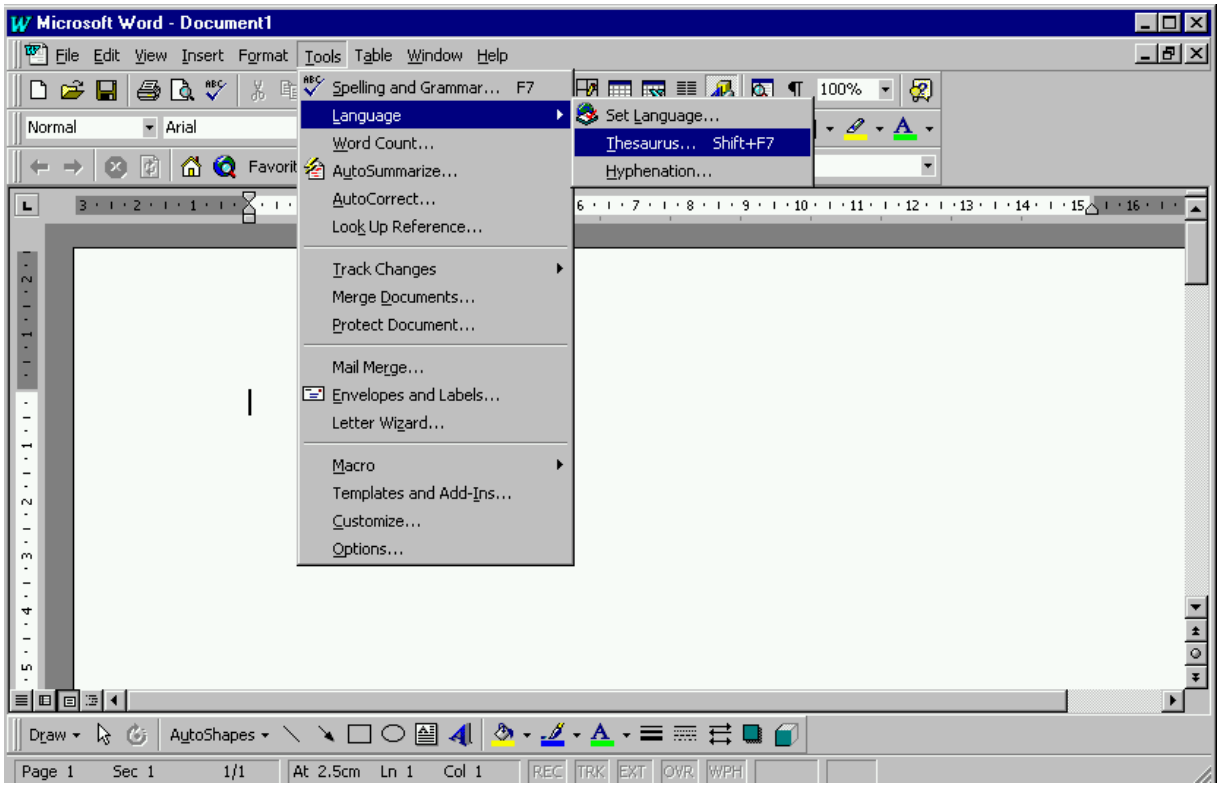
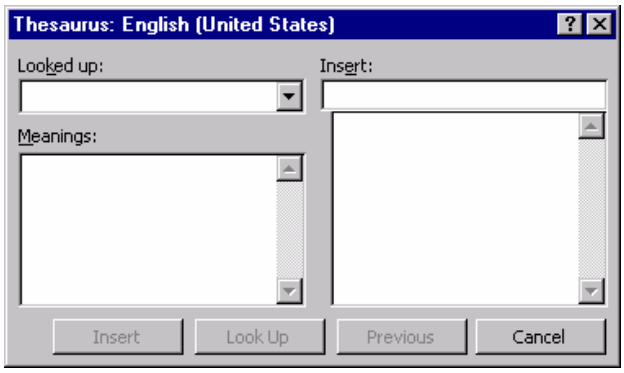


Figure 18 – Screen-capture of MS-Word

Once this option has been chosen, the user is effectively taken to another screen where she is able to exercise additional choices. In this case it is a dialog box.



This example in itself is not overly complex and there is little possibility of the user getting lost in a maze of navigation (drill-down) options. Systems like AutoCAD, SAP R/3, an Integrated Development Environment (IDE) and even a web-browser, to name just a few, may become complex. It is for these cases that the NS will help the user "jump" to the required section of the application and continue working from there.

In the sections that follow, the algorithm designed to achieve and control the "jump" (section 5.2.3.1) and the ergonomics (section 5.2.3.2) of the design and therefore the presentation of it to the user will be discussed.

5.2.3.1 The Algorithm for the Supervisor

Section 3.3.1 listed techniques used for the development of intelligent user interfaces. These techniques ranged from the simple use of a data (storage) structure to the use of languages and complex artificial intelligence techniques.

In this section a generic algorithm, which will give the NS what is perceived as intelligence by the user, will be developed. The algorithm will be developed independently of the architecture used for the NS, and how the algorithm is to be implemented with regard to the architecture will be discussed in 5.2.2. This places a restriction on the technique that will be used for the development of the algorithm

as not all host systems have the capacity to enable the development of an algorithm using the more complex techniques. For this reason, a less complex technique has been chosen. How this algorithm can be developed for multiple hosts that provide multiple structures for persistency will be dealt with in the implementation section (section 5.2.4.1 – Repository issues). First it is necessary to define and discuss the algorithm. This will be done by defining the type of communication that needs to take place and then looking at a structure that can be used to capture the essentials of this communication.

5.2.3.1.1 Sequential and Asynchronous Communication

In section 2, sequential and asynchronous dialogue was defined and discussed. It is necessary that the NS can handle both these types of dialogue.

Sequential dialogue comprises of actions that take place one after the other. These actions within the typical interface for which the NS is being proposed are: click on a menu-item, follow a number of drop-down menu items, choose a link, and follow multiple links. These actions all begin at some “start” point and follow a path (not always the same path) to a destination.

A typical asynchronous action for which the NS needs to make provision is the entering of a code and the host takes the user to the relevant location. Examples are entering transaction codes, using short-cuts and typing in a URL to name but a few.

The user need not follow one type of dialogue, but may mix (interleave) the actions resulting in a combination of an asynchronous and sequential interface. The consequence of this is that the structure controlling the navigation must cater for both.

5.2.3.1.2 The Data Structure - a Graph

The less complex technique chosen is the directed graph (digraph) data structure because it is possible to implement the data structure on systems with different

underlying structures and with less expressive languages. The graph comprises of a number of nodes that are linked by edges that indicate direction.

Each node of the graph represents a state of the dialogue, while an edge represents an action that can take place when in a particular state. The graph requires a "starting node" which corresponds to the "starting point" in the dialogue. Each edge between nodes will have a direction and a weight associated with it. The weight represents the probability of following the edge in a particular direction from a particular node. For illustration purposes, the system structure in figure 19 will be considered. The letters, [A] to [E], next to the screen diagrams represent dialogue states.

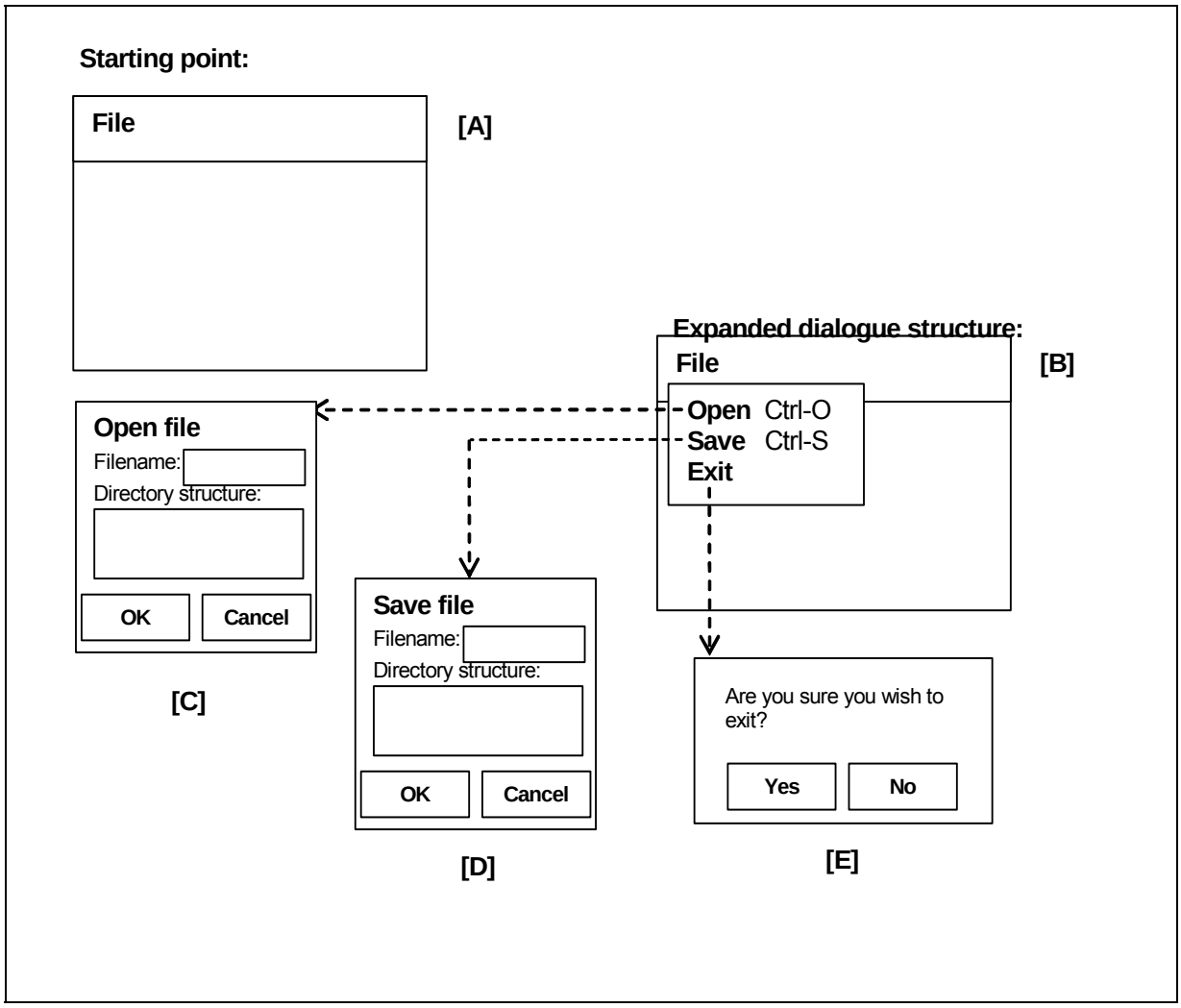
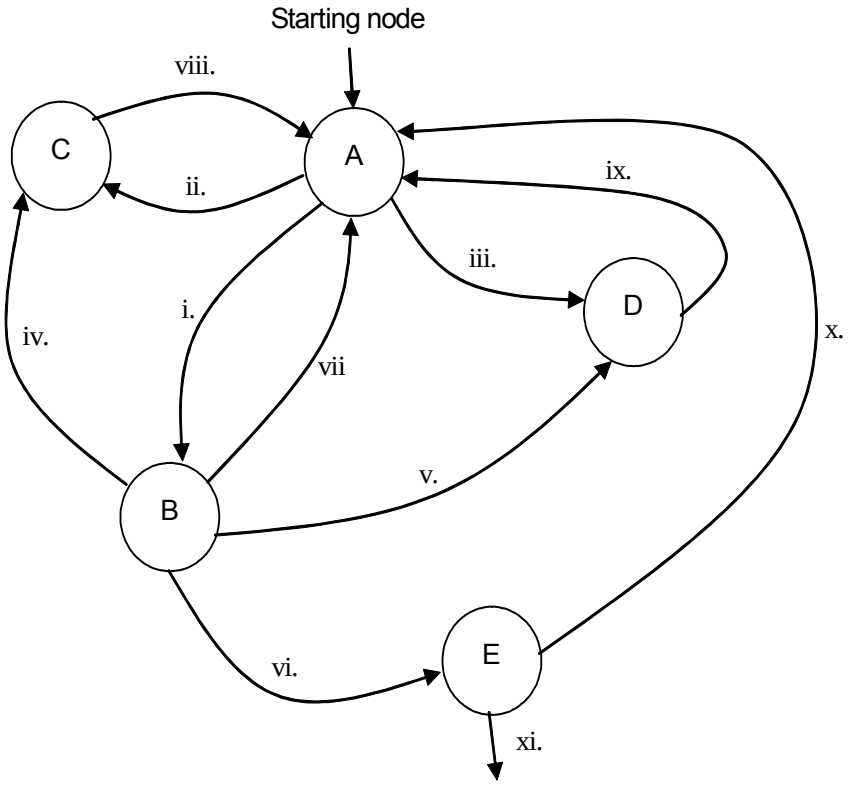


Figure 19 – Illustrative system

The corresponding complete graph that captures all navigational possibilities of the illustrative system given in figure 19 is defined in figure 20. Note that the graph includes all the short-cuts.



Legend

i.	File menu option
ii.	Ctrl-O shortcut
iii.	Ctrl-S shortcut
iv.	Open menu option
v.	Save menu option
vi.	Exit menu option
vii.	Escape key
viii.	OK/Cancel button
ix.	OK/Cancel button
x.	No button
xi.	Yes button

Figure 20 – Digraph of the illustrative system

Nodes with edges that point away from a node indicate options that may be taken by the user for that specific dialogue. Edges that point towards a node indicate paths that lead from other dialogues to the dialogue represented by the current

node. It is important to note that graph traversal may result in cycles and therefore associated with each graph is a current node. The current node of the graph corresponds to the current point of the dialogue which represents the current state in which the dialogue is and therefore the dialogue that the user is currently viewing. For prediction purposes, the graph is effectively picked up by the current node and shaken out. The shake out process places the current node at the root of the tree and nodes with edges moving away from the current node on the next level. For each node on each level, the nodes that can be reached from the node are placed on a level below. If a node is already in the tree on a previous level, the edge is dropped. This will identify possible dialogue destinations that will be presented, in prioritised order, to the user so that the user can decide where to go. Assume that the current node is A, disregard the short-cuts (that is edges ii. and iii.) for the purposes of this example, then the "shake out" of the graph will result in the tree structure depicted in figure 21. Notice that edges vii., viii., ix. and x. are not shown as they point back to a node already in the tree that are on a higher level.

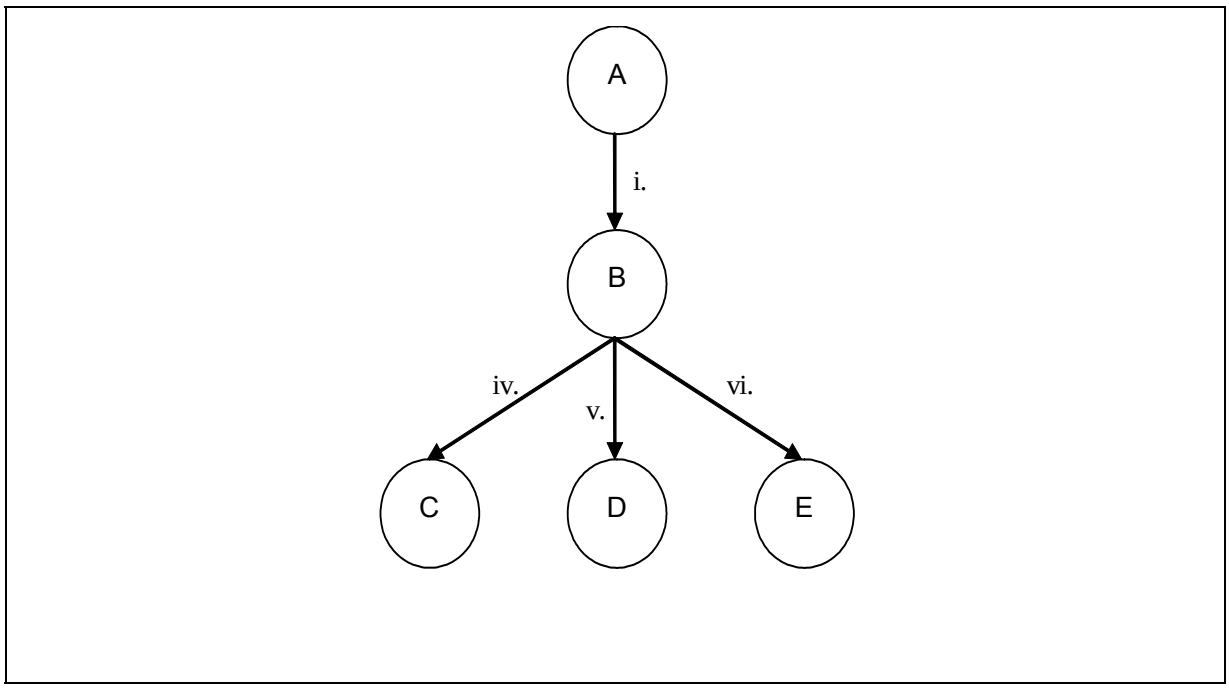


Figure 21 – Digraph shake-out – node A current node

The leaves of the tree represent possible destination dialogues. This means that from dialogue A, it is possible to reach dialogues C, D and E without using the

shortcuts. Effectively dialogue E has also been allocated a shortcut that has not been coded into the host system.

To ensure that dialogues that require user input are not passed over when the shake-out takes places, it is necessary to add business intelligence to the corresponding node to state if input is required or not. This means that a node must be able to reflect on its type at runtime. A node is either a “stop” node or a “pass-through” node. A “stop” node represents a dialogue that requires input from the user. Nodes C, D and E are all stop nodes. If node B is considered the current node, the shake-out (figure 22) will suggest node A as the dialogue destination.

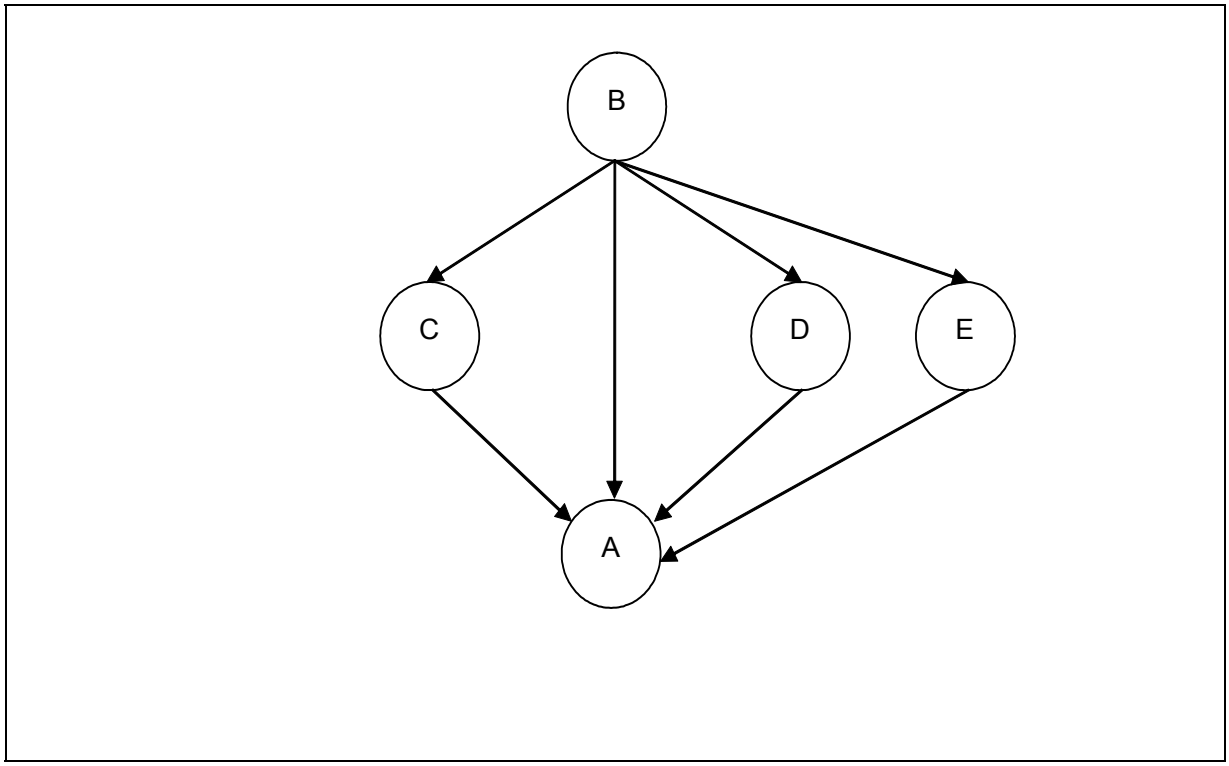


Figure 22 – Digraph shake-out with node B as current node

This however will not work as decisions need to be made at C, D or E before arriving at A. Nodes C, D and E need to be defined as “stop” nodes resulting in a shake-out tree given in figure 23.

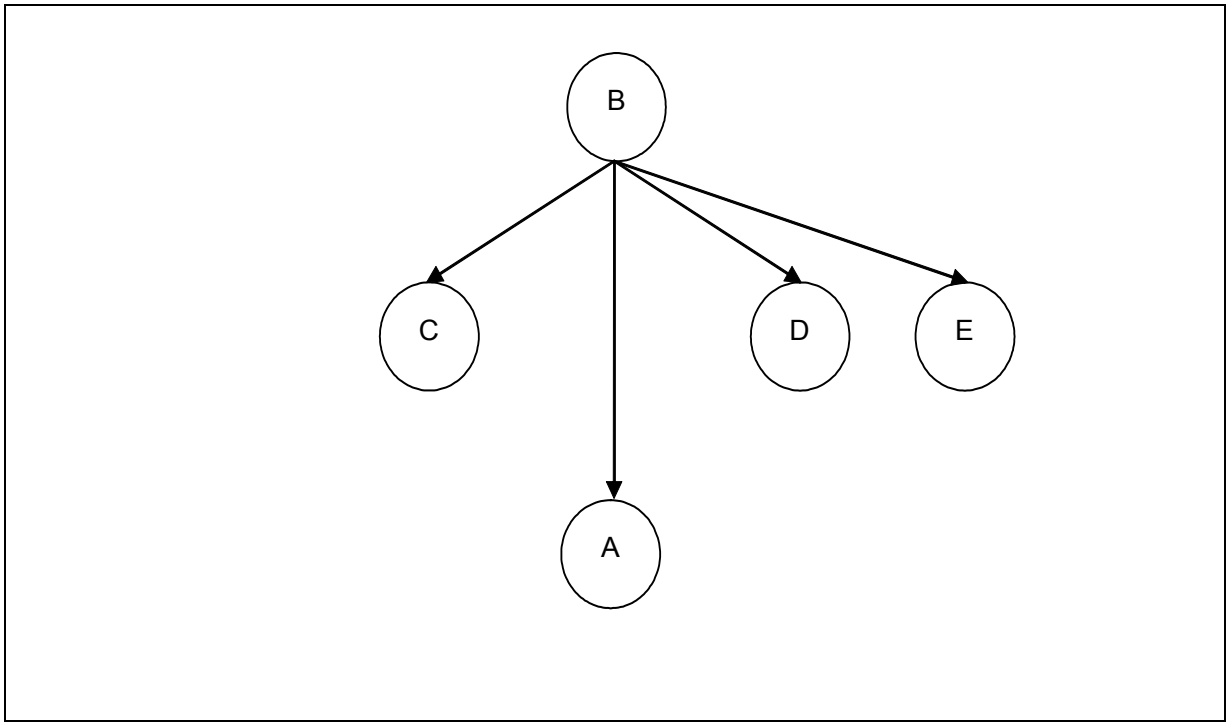


Figure 23 – Tree taking 'stop' nodes into account

One final example to illustrate the shake-out and the removal of cycles assumes that node C is the current node, this would result in a tree with the edges between nodes C and A, A and B, B and D, and B and E.

5.2.3.1.3 Ranking the Navigational Paths

The discussion in the previous section shows how all possible navigational paths that exist in the graph from the current node can be determined. All the paths however are not of interest to the user, and therefore it is necessary to give the user an offering of the most likely paths. To facilitate this, it is necessary to be able to rank the paths for which the edges need to be weighted.

Weighting of edges is a function of how often the edge is followed, the higher the value the higher the likelihood that the edge will be followed again. To keep the function simple, each time an edge is followed the edge weight on the graph is increased by 1. When the shake out takes place and the tree structure is

determined (taking the cycles and stop nodes into account), the weights are transferred from the graph using a one-to-one mapping to the tree.

The score of a path is determined by a function applied to the sum of the weights of the path. The function must take the length of the path into account. It is conceivable that the longest path will have the highest sum of weights, but it may not be the most probable path. This means that for each path (p) the average weight of an edge in the path (v) is determined by using:

$$v(p_i) = \left(\sum_{j=1}^{n} weight\left(p_{e_j}\right) \right) \Big/ n$$

where

n is the number of edges in the path p
p_i is the i^{th} path in the tree
e_j is the j^{th} edge in the path p

Figure 24 – Equation to determine path weights

These values are placed in an ordered list V, ordered by path. The highest value in V is given a score of 100% and the other values are scaled accordingly. This means that the score (s) of each path represented in V is given by:

$$s_i = V(p_i) / \max\%$$

where

i is the ith path in the shaken-out tree
max is the maximum path in V

Figure 25 – Equation to determine the path scores

The values of s are ordered in descending order and are presented to the user as possible navigation paths. The user representation will be discussed in section 5.2.3.4.

To illustrate the use of the formulae, consider the graph in figure 20 and assume that the edges have the following weights (once again to simplify the example, the short-cuts, that is edges ii. and iii., are ignored): i. is 5, iv. is 3, v. is 1, vi. is 1, vii. is 0, viii. is 4, ix. is 0 and x. is 1. The weighted tree (figure 21 with weights) is given in figure 26.

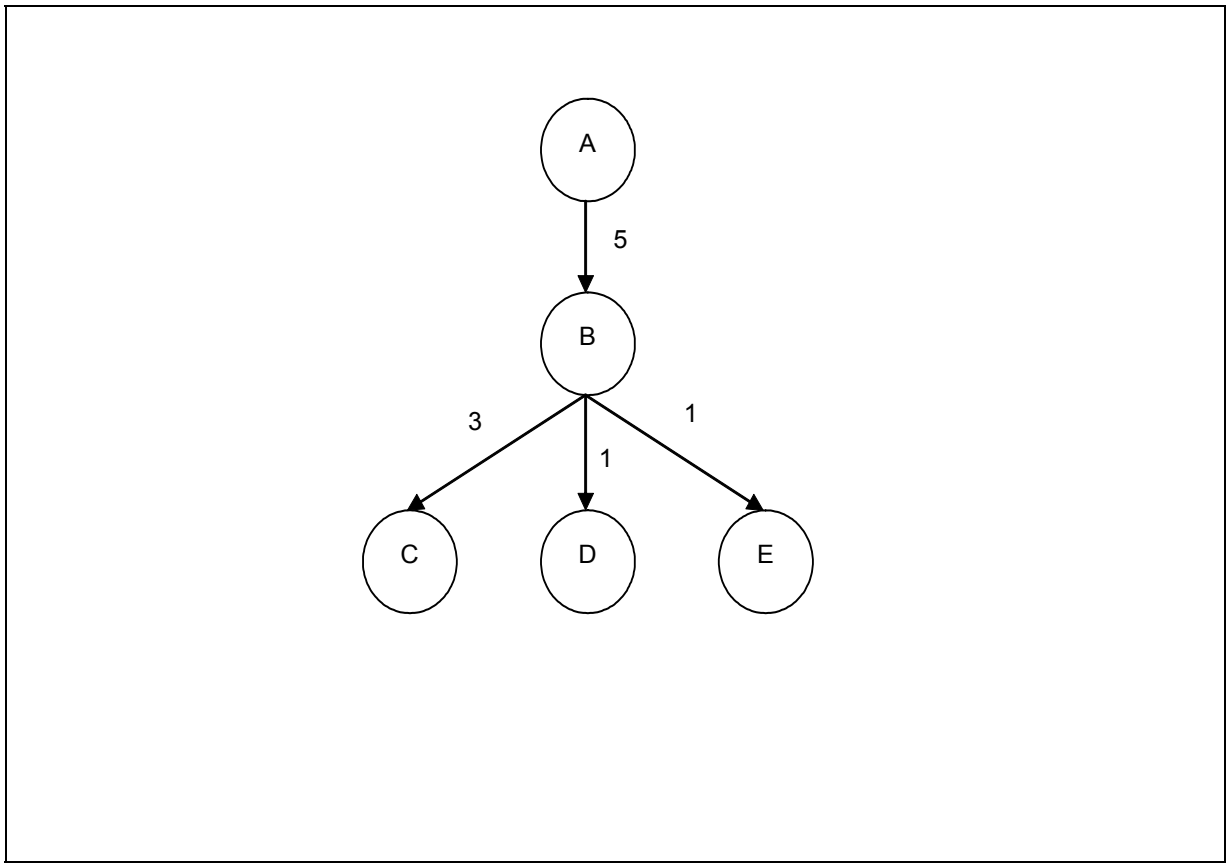


Figure 26 – Tree in figure 21 with weights

There are three possible paths, A to C, A to D and A to E. The table that follows summarises the results of the formulas when they are applied to the tree in figure 26.

i	p_i	$v(p_i)$	s_i
1	A → B → C	8/2 = 4	100%
2	A → B → D	6/2 = 3	75%
3	A → B → E	6/2 = 3	75%

Figure 27 – Table of weights and rankings for the example

This means that it would be most likely for the user to follow the path from A to C. In the case when a user makes a choice, the weight of *each* edge in the path is increased by 1. If the user decides to ignore the suggestions and continues navigating without using the NS, the weights of each edge followed are also increased by one. To accommodate the use of short-cuts that may exist in the host system, it is necessary for the NS to increase weights of the equivalent navigational path. This means that the digraph does not need edges between nodes that signify the short-cuts, but will need to keep a data base up to date that maps the short-cuts between two nodes in terms of the equivalent navigational paths. In the event of a short-cut being followed, the correct edge weights for the particular navigational paths need to be increased.

It is important that the user be given the option to inform the system that a suggestion is to be ignored and the weights of each of the edges of the path that are being explicitly ignored can be downgraded by 1. This will ensure that the suggestion is not necessarily given again for a while (Birnbaum, Horvitz, Kurlander, Lieberman, Marks & Roth, 1996).

5.2.3.1.4 A Graph for Each Profile

As stated as one of the requirements of the NS in section 5.2.1, the NS maintains 4 types of profiles. For each host system there is one system and one average profile and therefore a graph for each of these two profiles. There are multiple stored and current profiles per host and a graph with all the weights needs to be maintained for each of these profiles as well.

To simplify matters, the weights of the edges of the system profile will all be kept at 1. This means that the system profile in effect will archive all possible navigational paths that exist on the host system as they are discovered by the user.

The average profile on the other hand, needs to reflect the system profile, but with the weights included of all the navigational paths that have been taken by the users and what their respective weights are. By using the average profile, it will be

possible to determine what the most popular paths used within the host system are.

Both the stored and current profiles reflect the actions taken by particular users and can therefore be used to predict a particular users habits in the long term.

Effectively, the average, stored and current profiles are overlays of the main template graph – the system profile.

5.2.3.1.5 The Algorithm Defined

The algorithm traverses the graph that represents the current profile of the user in breadth-first fashion and builds a tree from which the possible navigational paths can be deduced. The algorithm comprises of three basic steps (refer to figure 28) and assumes that a reference to the current node (g) of the graph is passed as a parameter to it.

```
Step 1: Initialisation
Step 2: Shake the graph out to produce a tree
Step 3: Rank the navigational paths from the root to each
        leaf of the tree according to path scores
```

Figure 28 – High level structure of the algorithm

```
NS_Algorithm(g) {
// g is the current node in graph G
// s is the current node in the ordered set S
// rt is the root node of the shake-out tree, T, of G
// ct is a reference to the current node in the tree T

//Step 1: Initialisation
     create S and add g to S
     create T and add g to T
     s, rt, ct, Tcompleted := g, g, g, false

//Step 2: Shake the graph out to produce a tree (T)
     while not Tcompleted do
       for each child, n, of s in G do
         if n ∈ S then
           mark ct as a leaf node in T
         elseif n is a "stop" node then
           add n to T as a child of ct and
           mark n as a leaf node
         else
           add n to S
           add n to T as a child of ct
         endif
       endfor
       if S has more unvisited nodes then
          s := next unvisited node in S
          ct points to the equivalent node in T as s is
       else
         Tcompleted := true
       endif
     endwhile
     make ct point to rt

//Step 3: Rank the navigational paths from the root to
        each leaf of the tree according to their path
        scores
     for each path p_i do
       calculate v(p_i)
     endfor
     determine the max v(p_i)
     for each v(p_i) do
       calculate s_i
     endfor
     sort the s_i values in descending order
}
```

Figure 29 – Detailed algorithm

The algorithm makes use of an ordered set data structure (S) to build the tree (T) and to realise the shake-out. Each node in the graph that is visited is also inserted into the set. The reason for using a set is to ensure that no graph node may appear more than once in the set. The set may be implemented using any sequential data structure, for example a list. This is necessary because neighbouring nodes need to be placed at the end of the set if they are not already in the set when the current node in the set is being investigated.

The graph traversal algorithm described is an example of a learning apprentice, referred to section 3.2. It is able to determine the final goal of the user as well as being able to predict the next move.

5.2.3.1.6 Enhancing the Graph

As the algorithm stands now, the graph is constructed by the NS and maintained by the NS. This graph represents the current profile of the user and yet the user is unable to manipulate the graph. The Letizia and PowerScout systems (Lieberman, Fry & Weitzman, 2001) both allow the user to add notes to the profile, and therefore personalising the system even more. To accomplish the same result within the NS, it is possible to allow the user to add notes to the edges of the graph and therefore adding an additional dimension to the algorithm.

The annotations to the edges may be in the form of a text note and a rating. The algorithm will then be modified to rank the possible navigational paths both according to the rankings that are determined using the edge weights as well as according to the users' own rating system.

5.2.3.2 Ergonomic Design of the Supervisor

The interface of the NS needs to adhere to HCI principles that enable the user to interact with the NS in a "user-friendly" manner. To achieve this, the design of the interface needs to take the needs of the user into account. The design, in terms of what the user sees, should make use of the properties mentioned in section 2.2 including layout, controls and interface functionality (Birnbaum, Horvitz, Kurlander,

Lieberman, Marks & Roth, 1996). More importantly the design needs to ensure that the user does not perceive the NS as an add-on to the host system, which it is, but the user should perceive the NS as if it is part of the host system.

The screens designed for the NS should be consistent in their structure and should be customisable and personalisable. These include all the screens ranging from the logging in process as discussed in section 5.2.2.5 (and specifically as depicted in figure 16), through to screens that convey navigational information to the user. The latter screens will be focussed on the next few of paragraphs.

How the results of the rankings of the navigational paths determined by the algorithm are represented to the user must be decided on. The user should be able to customise and personalise the rankings. The following two ways suggest themselves:

- By allowing the user to decide how many of the rankings should be shown. This can be done by either defining a fixed number or by specifying a percentage threshold above which the ranking should be displayed.
- By allowing the user to determine the granularity of the rankings shown.

To manipulate the granularity, the possible navigation paths could be depicted using a collapsible tree structure in which the current point and the possible destination dialogues are given. The following figure shows the layout for the example give in figure 26.

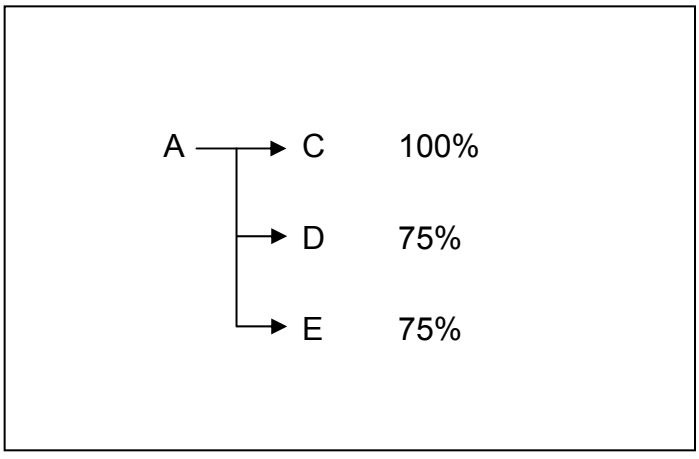


Figure 30 – Possible destinations

The user can choose to either “jump” to C (or D or E) by selecting C (or D or E) or can request that the granularity of the path can be refined at C. The refinement of the granularity could then result in the following:

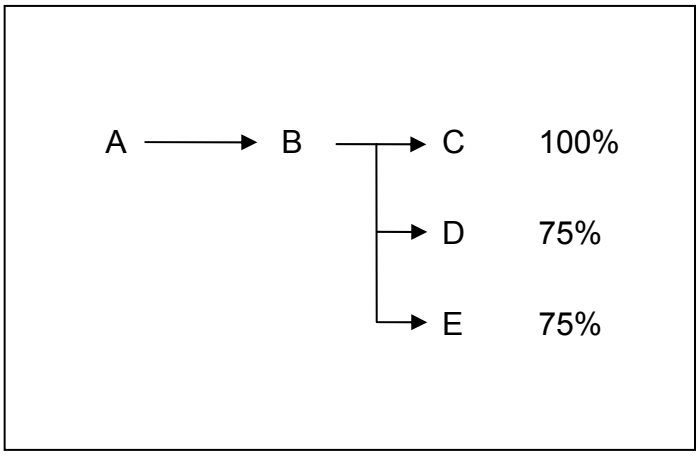


Figure 31 – Possible destinations - refined

In effect, the navigational tree that has been rotated 90˚ anti-clockwise and is displayed. The reason for advocating this structure is that most computer users are familiar with this structure specifically for directory, mail etc. traversal and will be able to find their way around more easily than if confronted with a completely new representation.

5.2.4 Implementation

The NS is intended to aid navigation intelligently. It is designed to make suggestions with regard to existing host systems. This implies that the NS should not interfere with the users’ mental model of the system, for if it did, it would tend to confuse and/or distract the user’s normal functioning. In as much as the host system’s functioning is incorporated into the user’s mental model, it is important that the user of the host system should not see the NS as an add-on or additional tool to the host, but should feel that it is part of the host. For this reason, it is important that the NS be integrated with the host as seamlessly as possible.

Before delving into the possible hosts that the NS can be linked to (section 5.2.4.3), some points regarding how the repository should be implemented (section 5.2.4.1) and which architecture should be developed (section 5.2.4.2) will be discussed.

5.2.4.1 Repository Issues

The placement of the profile repository was mentioned in the discussion of the Architecture (section 5.2.2.3) where two fundamental locations were identified:

- as part of the host, provided the host provides a means to store information;
- remote from the host, which can be subdivided into a number of possibilities.

Each of these possibilities has advantages and disadvantages associated with it. What is of interest for the implementation of the NS, is how the choice of the architecture, the algorithm and the data structure influences the placement of the repository. The following table summarises the possibilities for repository placement and discusses the viability of each taking the placement of the NS into account in each case.

Scenario	Repository Placement	Discussion	
1	As part of the host, embedded in the host.	**General**	The repository is dependent on the infrastructure provided by the host. The information required to make decisions is close to where the decision is to be executed.
		NS as part of the host	Ensures that where the data to make the decisions is stored is close to where the decision is made.
		NS remote from the host	Placement of the NS away from the host would mean that before a decision can be made, the information must be retrieved from the repository and transferred to the NS, where the decision is made and the results of the decision are transferred back to the host where it is executed. This could result in a time lag that the user perceives.
2	As part of the host, but as a separate entity on the host	**General**	The repository is a separate entity from the host, but resides on the same "server" as the host. The information required to make the decision is close to where the decision is to be executed.
		NS as part of the host	The NS and repository all reside on the host "server" but are autonomous entities. Communication between each of the entities needs to take place and this could slow the normal processing that the host does down.

		NS remote from the host	This has the same problems as the scenario sketched in 1 above. There is a lot of communication that will occur between the remote NS and the repository on the host and therefore the possibility to slow the host down.
3	Remote from the host, with the NS	**General**	Both the NS and the repository are placed remote from the host, but are on the same "server".
		NS as part of the host	Not applicable.
		NS remote from the host	The NS and repository are on the same "server". Only final decisions are sent to the host, the rest of the communication takes place between the user and the NS. This alleviates a possible bottle neck with the host.
4	Remote from the host, linked individually with the PS and S^2	**General**	The NS is split between its two components, the PS and the S^2. Each of the components will update and control their particular repositories. The PS will control current profiles and stored profiles, while S^2 controls the average and system profiles.
		NS as part of the host	Not applicable.
		NS remote from the host	This scenario will manage users who have established a work methodology, and therefore have built up a stable current profile. For users who experiment and are still establishing a current profile, this placement will incur a large communication overhead between the repository on the PS and the repository on the S^2 where the average and system profiles are stored.
5	Completely remote from the NS, i.e. as an entity on its own	**General**	The NS, and its components, are remote from the repository. This represents the most general architecture available.
		NS as part of the host	Not applicable.
		NS remote from the host	A similar problem to that in scenarios 1 and 2 will occur. There is a communication overhead between the repository and the NS that could slow the system down.

Figure 32 – Repository placements

Embedding the repository on the host, scenario 1, means that the repository is dependent on the structures and languages provided by the host. In some cases, for example with regard to SAP R/3, the host does not provide direct implementation techniques for data structures such as graphs and trees because it does not provide a language that is expressive enough. Of course, it is theoretically possible to embed the repository on the host by re-writing parts of the application itself. However, such a radical measure is not within the scope of the present discussion, which rather seeks to seamlessly "hook into" an existing

application. It is in these cases that a work around needs to be identified and implemented taking the architecture into account.
In some cases, (for example in the SAP R/3 system) it might be possible to hook into an existing application via a relational database. In such cases, one could consider implementing the required graphs and trees using two dimensional (matrices) arrays (Standish, 1998) where these matrices map onto the relational database tables that interact with the application. As with a matrix, the introduction of another node will introduce a row (tuple for the table) and a column into the table.

To illustrate how matrices (and tables) can be used to represent a digraph, consider the example that follows. The digraph on the right of the figure is represented in the matrix on the left. Edges on the digraph are represented in the matrix by cells that have non-zero values. To determine the direction of the edge it is necessary to consider the ordered pair of (row,column). In this example it will result in 4 pairs, namely: (A,B), (B,A), (B,C) and (C,A) with the weights 7, 2, 5 and 5 respectively.

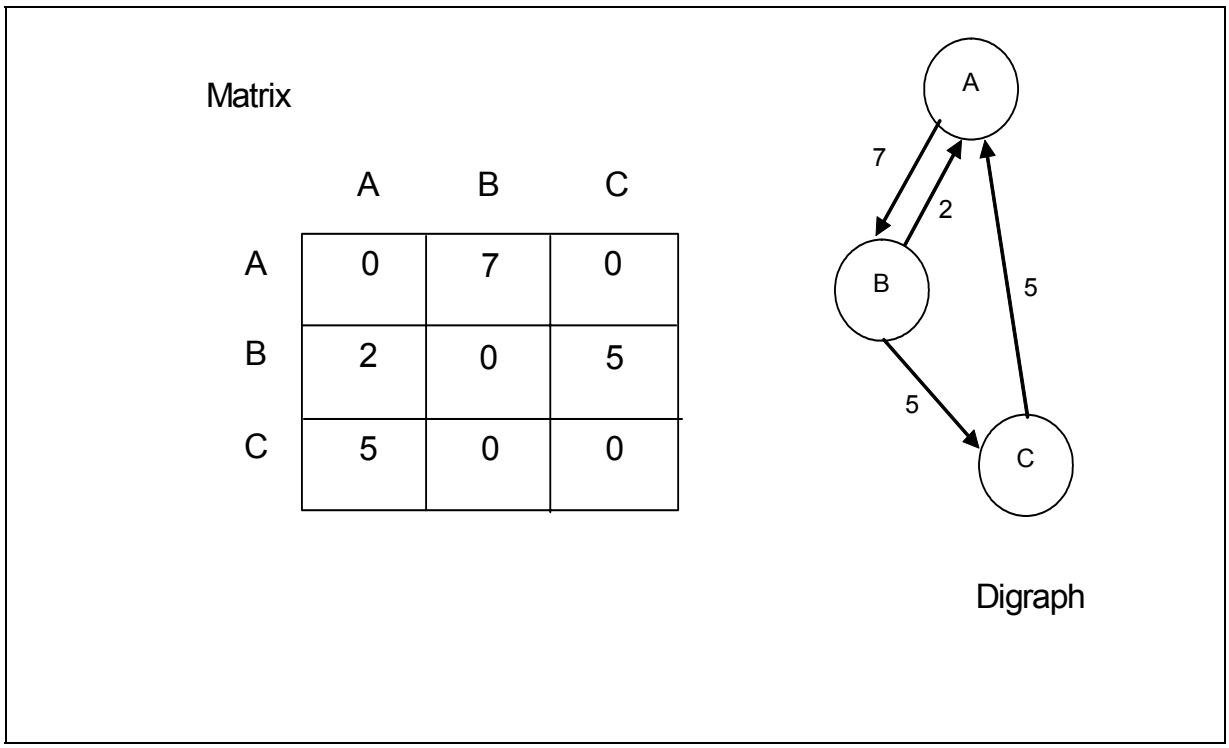


Figure 33 – Matrix representation of a digraph

It is therefore possible to store a profile directly into a relational database table by defining a table per profile to be stored. This method however does not differentiate between an arc that has zero weight and an arc that does not exist and therefore a better solution would be to store the ordered pairs that represent the edges in a table along with their respective weights, user annotations and user ratings.

Note that the NS relies heavily on the information supplied by the repository to determine navigational path possibilities. Therefore when the repository is remote from the NS (as in scenario 5) or architecturally separated from the NS (as in scenario 2) the communication between the two may become problematic. Scenario 4 will work well for established users, where what the user does is limited to a particular set of navigational paths and only the weights of the current profile are updated. These weights can be transferred to the average profile whenever it is most convenient. Scenario 3 differs from scenario 4 in that it caters for cases where both the current and average profiles are being influenced in real-time. A hybrid between scenarios 3 and 4 would seem to provide an optimal solution: when a user first uses the NS, scenario 3 is used; once the user becomes established, the PS of the user splits off with its own repository and continues autonomously as described in scenario 4.

To conclude, the placement of the repository is dependent on the user and on the profiles that are influenced. The four profiles were described in section 5.2.1, along with which section of the architecture they link to. The NS therefore has two levels of repository, one central repository to store and maintain the overall structure of the paths (in the system and average profiles managed by S^2) and a repository per user that maintains the current and stored profiles (managed by the individual PS's). The user repository is an image of the central repository, in which the edges that are not relevant to the user have weights of zero. It is also not necessary to store the system profile in the central repository. The system profile can be represented by the average profile without the weights.

5.2.4.2 The Architecture Used

Architectural issues were discussed in section 5.2.2 where two possibilities were suggested. The first possibility was that the NS is embedded in the host (section 5.2.2.1) and the second was that the NS is remote from the host (section 5.2.2.2). In the previous section, the placement of the repository was rationalised and the conclusion drawn that the repository should not form part of the host or be placed on the host "server".

The repository placement suggests that the NS should be remote from the host. The architecture chosen for the NS should reflect this and therefore a system architecture that is loosely coupled should be followed for implementation. This architecture ensures that the NS based on a modular design making the NS scalable, extensible, maintainable (section 5.2.2.4) etc. which are all desirable properties of a well designed system.

5.2.4.3 Possible Hosts

The loosely coupled architecture model chosen, supports a single NS that can be used for multiple hosts. To achieve this functionality, an NSLink module needs to be written specifically for the host system. NSLink is a middleware application that enables the NS via an API to communicate with the particular host system.

The types of hosts that are targeted by the NS are those that allow the user to navigate the system both sequentially and asynchronously. Examples of host systems are:

- MS Office type products
- Integrated Development Environments (IDE's), such as IBM's eclipse framework
- AutoCAD
- World-wide Web (WWW)
- SAP R/3

Also on the technology side, the NS may help to enhance the navigational process on mobile devices which have limited space and where the time taken to reach online information is crucial.

The two sections that follow will briefly discuss the deployment of the NS in terms of software (section 5.2.4.3.1) and hardware (section 5.2.4.3.2) applications.

5.2.4.3.1 Software Applications

Software application fall into two broad categories, either desktop applications or client-server based applications.

With desktop applications, it would be better if the NS is as close to the desktop as possible. This would require the NS being split into its components and the PS being placed on the desktop (scenario 4 in figure 32) along with the NSLink to facilitate the communication between the desktop application and PS. For desktop applications, it is not crucial that the PS synchronises with the S^2 module on a regular basis. The MS-Office type products and IDE's are straight forward applications in the sense that the graph data structure generated by an NS as proposed here, does not seem to be overly complex. In many cases the user only uses a sub-set of the functionality that is available in the application. A user who uses the additional features such as macros, or for example AutoCAD, and similar systems, where a language is included, may want to annotate the edges of the graph with programs that provide a specific functionality.

The addition of the NS to a client-server application means that the NS becomes an intermediary between the client and the server. The combination of scenarios 3 and 4, as described in figure 32, is feasible and conceptually easy to implement. WWW applications are the most common type of applications for which learning agents and learning apprentices have been developed. The NS will simplify the use of some large systems such as SAP R/3, where a user must either go through a maze of menus to come to the section on the program that is required, or else has to remember a transaction code to get to the desired point.

5.2.4.3.2 Hardware Applications

The use of mobile devices and more specifically, the use of Pocket PC's is becoming more prevalent. Billsus *et al.* (2002) states that the interfaces that are available on this type of hardware are not sophisticated enough to compete with the desktop type device. This is a result of the limitations in memory and storage space that exist on these devices. To increase the usage of the devices, the interface needs to be able to adapt to the needs of the user and the user should be able to personalise it.

The NS can be adapted to make the interface of the mobile device seem intelligent so that only what the user deems relevant will be displayed. To achieve this, the architecture of the NS can be enhanced to include a server that is placed between the host system and the mobile device. This server will do all the processing according to the users' profile and then relay only the relevant information to the mobile device.

5.3 Placing the Supervisor in the Techniques Matrix

To locate the NS in the techniques matrix referred to in section 3.3.1, the relevant techniques used by the NS must be considered.

- It is obvious that the NS makes use of data structures: a graph is used to store the navigational paths; and a tree is used to determine possible destinations depending on node characteristics in the graph.
- The NS can be termed a software agent as it works in parallel with the host application and has the task of find the most probable navigational path to complete.
- The NS also complies with the notion of a probabilistic model as the edges of graph are weighted using the number of times the node has been visited. It would not be difficult to normalise these weights to represent probabilities, placing the NS in the Bayesian model technique as well.

Figure 34 includes the NS in the techniques matrix and updates the techniques totals.

	Data "storage" structures	Content-based Filtering	Collaborative Filtering	Concept Hierarchies	Neural Networks	Reinforcement Learning	Agents	Constraint Propagation	Bayesian (Belief) Models	Context-Free Grammars	Case-based reasoning	Total
Navigational Supervisor	✓						✓		✓			**3**
Previous TOTALS	**2**	3	4	1	0	0	**7**	0	**4**	2	2	
UPDATED TOTALS	**3**	3	4	1	0	0	**8**	0	**5**	2	2	

.

Figure 34 – Placing the NS in the techniques matrix

6 Conclusion

"… the other is a conclusion, shewing from various causes why the execution has not been equal to what the author promised to himself and the public."

Samuel Johnson, Of Lord Chesterfield's Letters, 1755

This dissertation proceeds from the view that Human Computer Interaction (HCI) is a multidisciplinary field of study. It has its roots firmly placed in the Computer Science discipline, but as computer technology progresses, so do the requirements placed on the interfaces of systems. It has been argued that it is no longer adequate to provide a purely functional interface for two reasons:

- Systems have become more sophisticated and therefore require more sophisticated interfaces
- Users of systems are no longer limited to technical people who understand and know what the system does, but are often people with minimal computer skills and therefore need to be guided by the system.

HCI therefore needs to include the disciplines of humanities, such as psychology, to understand why a user reacts in the way they do, and education, specifically to enhance the aesthetics of the interface. These aspects of HCI are extremely important in the development of user interfaces for computer systems, but are not the focus of the work presented.

Within the computer science discipline, interface design and development was initially part of the software engineering (SE) field. Users were given the opportunity to customise the interface to their needs – mostly configuring the aesthetics of the interface. As interfaces got larger and more graphical, the human perception and work methodology became an issue and users wanted to be able to personalise the interfaces. Interfaces also began to show an ability to reason, introducing the field of artificial intelligence (AI) into interface design and development. Currently, interface design and development spans the three disciplines and makes use of both software engineering techniques (for the interface design) and artificial intelligence techniques (for the interface

development) so that the user can perceive the interface as intelligent. The greyed areas forming the shaft within HCI in figure 35, shows how the disciplines and the fields are dependent on each other.

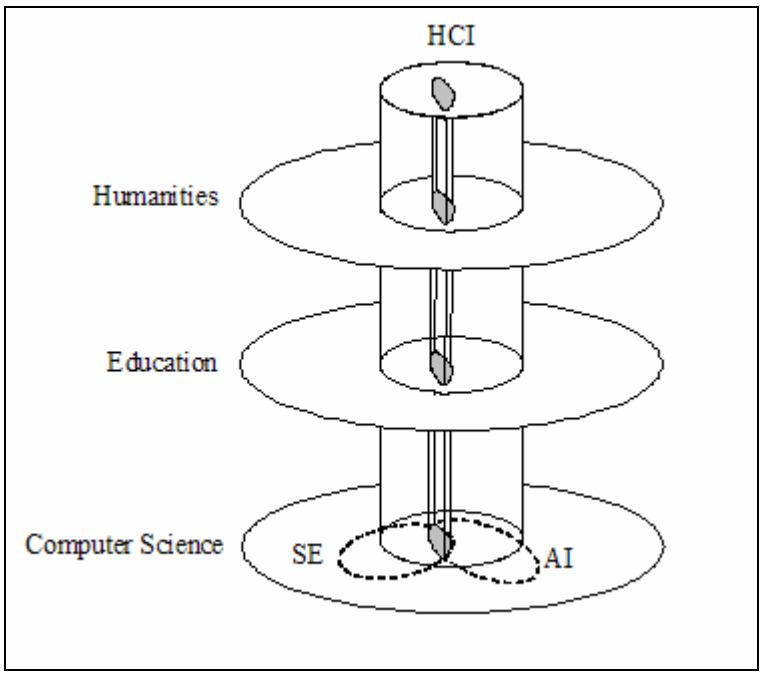


Figure 35 – Interface design and development

To help with the placement of intelligent interfaces in HCI, a taxonomy of interfaces was developed. The taxonomy helps to structure the field of user interfaces. The two aspects of intelligent interfaces that were investigated further were, learning agents and learning apprentices. Learning agents are able to determine the next move the user is to make, while learning apprentices predict the final move the user is to make and consequently the final goal. To predict what the goal is, the next move is known and therefore learning apprentices are also learning agents.

In the development of learning apprentices, AI techniques are used. These techniques range from simple and easy to implement, to complex and difficult to implement as well as impractical to use for large systems. Also, it was shown that a single technique is in many cases inadequate and multiple techniques are used. The techniques that used together are mostly simple, complement each other and

do not require a lot of processing overhead. The NS is implemented using a simple graph structure that goes through a shake-out process resulting in a navigational path tree that comprises of possible ultimate destinations. The graph and tree structures make use of a model based on weighted scores to determine the possible goals. The entire system is encapsulated into software agents. The techniques used for the NS make it scalable, maintainable, etc. and of most importance, it makes the NS portable and therefore generic.

The architecture of the NS is loosely coupled and modular in design. The advantage of the structure is that minimal system dependent code needs to be implemented to link the NS to the host application.

The NS takes learning apprentices one step further by having a generic algorithm that can be deployed on multiple applications. The majority of the learning apprentices that exist today are specifically written for a single application.

What has been presented here is an idea for the implementation of a learning apprentice. The implementation is generic and can be deployed for any application that will allow an application-specific NSLink to be plugged into it. NSLink forms an interface between the application and the NS. NSLink manipulates the application according to what the NS suggests, which is effectively what the user wants. The user however is still given the choice not to heed the NS's suggestions. The user consequently perceives the interface as intelligent.

Finally, a basic design for the NS has been suggested. The design proposes the graph and tree data structures to represent the navigational paths. It carefully considers various architectural alternatives for the placement of the sub-systems of the NS. A high-level specification of an algorithm is given that performs the shake-out of the graph to produce a tree giving the alternative navigational paths. Implementation issues with regards to the architecture are discussed for which suggestions are made, and finally the design of the NS is compared to existing systems by placing it in the techniques matrix.

There is a lot that can still be done to enhance and stream line the design. These include the following.

- An immediate future objective is to identify an application that can serve as a host system, and then to implement, test and possibly refine one or more of the design alternatives in this context. A likely candidate for such a test-bed application is IBM's Eclipse framework. Once the design has been tested, another application with different characteristics (for example AutoCAD) can be used to determine how generic and scalable the design is.
- The algorithm could be enhanced to include options from other profiles (specifically the average) that other users may have chosen. This is possible as the algorithm has been developed generically and has been parameterised. This means the algorithm can be called using the current position in the average profile. The results can then be merged with the results of the current profile.
- Functionality to navigate using natural language phrases could be added. For example in SAP R/3, if the user is interested in "Material Master Data", then the phrase can be used for the search and all relevant navigational paths can be exposed to the user. This is similar to the concepts used in PowerScout (Lieberman, Fry & Weitzman, 2001) which is a combination of a reconnaissance agent and a search engine.

7 Bibliography

Armstrong, R., Freitag, D., Joachims, T. & Mitchell, T. (March 1995). WebWatcher: A Learning Apprentice for the World Wide Web. 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments. Stanford. Available from: http://www.cs.cmu.edu/afs/cs/project/theo-6/web-agent/www/project-home.html.

Benyon, D. (1993). Accommodating Individual Differences through an Adaptive User Interface, *Adaptive User Interfaces – Results and Prospects, M Schneider-Hufschmidt, T Kűhme and U Malinowski (editors).* Elsevier Science Publications, North-Holland, Amsterdam. Available from: www.dcs.napier.ac.uk/~dbenyon. (Accessed 5 May 1999).

Benyon, D. (1998). Employing Intelligence at the Interface. Unpublished first chapter of a Handbook of UI. Available from: *www.dcs.napier.ac.uk/~dbenyon.* (Accessed 5 May 1999).

Billsus, D., Brunk, C.A., Evans, C., Gladish, B. & Pazzani, M. (May 2002). Adaptive Interfaces for Ubiquitous Web Access. *Communications of the ACM,* 45(5):34-38.

Birnbaum, L., Horvitz, E., Kurlander, D., Lieberman, H., Marks, J. & Roth, S. (1996). *Panel: Compelling Intelligent User Interfaces – How much AI?* Appears in the proceedings of the 1997 International Conference on Intelligent Interfaces. Available from: www.merl.com.

Callahan, G. (September 1994). Excessive Realism in GUI Design: Helpful or Harmful? *Computer Language,* 2(9):37-44.
Constantine, L. (February 1993). Improving intermediates, *Computer Language,* 110-112.

Downton, A. (Editor). (1993). *Engineering the Human-Computer Interface*. McGraw-Hill International (UK) Limited

Drummond, C., Holte, R. & Ionescu, D. (1993). *Accelerating Browsing by Automatically Inferring a User's Search Goal.* Proceedings of the Eighth Knowledge-Based Software Engineering Conference. Available from: http://www.site.uottawa.ca/~holte/Publications/index.html.

Fink, J. (2003). *User Modeling Servers – Requirements, Design, and Evaluation*. Unpublished doctoral dissertation. Standort Essen: Universität Duisburg-Essen, Germany. Available from: www.ics.uci.edu/~kobsa/phds/fink.pdf. (Accessed on 20 July 2004).

Geyser, E.P. & Van Brackel, P.A. (1991). Man-machine interaction as a factor in the design of computerized information retrieval systems. *South African Journal of Library and Information Science*. 59(4):256-260.

Geyser, E.P. (1992). Human factors in the interaction process between man and the user friendly information retrieval system. *South African Journal of Library and Information Science.* 60(3):167-173.

Hedberg, S.R., (March/April 1998). Is AI going Mainstream at last? A look inside Microsoft Research. *IEEE Intelligent Systems*. 21-25. Available from: *www.research.microsoft.com/users/horwitz/lum.html.*

Holte, R.C. & Drummond, C. (1994). *A Learning Apprentice for browsing.* AAAI Spring Symposium on Software Agents. Available from: *http://www.site.uottawa.ca/~holte/Publications/index.html*

Holte, R.C. & Yan, J.N.Y (1996). *Inferring What a User is Not Interested In*. Advances in Artificial Intelligence (proceedings of AI'96, the Canadian AI conference), Springer Lecture Notes in AI, LNAI 1081. 159-171. Available from: *http://www.site.uottawa.ca/~holte/Publications/index.html.*

Horvitz, E. (1998). *Lumiere Project: Bayesian Reasoning for Automated Assistance*. Presented at the Decision Theory and Adaptive Systems Group of Microsoft. Available from: *http://research.microsoft.com/~horvitz/*.

Ince, D. (1995). *Software Quality Assurance – A Student Introduction*. McGraw-Hill International (UK) Limited.

Jobst, J.E. (2002). *Exploring Internet Personalization.* Unpublished manuscript submitted for publication. Available from: www.gslis.utexas.edu/~jenj/personalization_paper.html.

Langley, P. (1997). *Machine Learning for Adaptive User Interfaces.* Proceedings of the 21st German Annual Conference on Artificial Intelligence held in Freiburg, Germany. Springer. 53-62. Available from: http://www.isle.org/~langley/adapt.html.

Lethbridge, T.C. & Laganière, R. (2001). *Object-Oriented Software Engineering – Practical software development using UML and Java.* McGraw-Hill International (UK) Limited

Lieberman, H. (August 1995). *Letizia: An Agent That Assists Web Browsing.* Appears in the Proceedings of the International Joint Conference on Artificial Intelligence [IJCAI-95], Montreal. Available from: http://web.media.mit.edu/~lieber/Lieberary/Letizia/Letizia-Intro.html.

Lieberman, H. (May 1997). *Autonomous Interface Agents.* Appears in the Proceedings of the ACM Conference on Computers and Human Interaction [CHI-97], Atlanta, Georgia. Available from: http://web.media.mit.edu/~lieber/Lieberary/Letizia/Letizia-Intro.html

Lieberman, H., Van Dyke, N. & Vivacqua, A. (1999). *Let's Browse: A Collaborative Web Browsing Agent*. Proceedings of the 1999 International Conference on Intelligent User Interfaces,

Collaborative Filtering and Collaborative Interfaces. 65-68. Available from: http://web.media.mit.edu/~lieber/Lieberary/Lets-Browse/Lets-Browse-Intro.html

Lieberman, H., Fry, C. & Weitzman, L. (August 2001). Exploring the Web with Reconnaissance Agents. *Communications of the ACM.* 44(8):69-75. Available from: http://web.media.mit.edu/~lieber/Lieberary/Letizia/Why-Surf/Why-Surf.pdf. (Accessed on 3 July 2003)

Lucas, L. (1991). Visually designing the computer-learner interface. *Educational Technology.* 31(7):56-58.

Norman, D.A. (1995). Designing the Future. *Scientific American*

Olson, S. & Wilson, D. (1985). Designing Computer Screen Displays. *Performance & Instruction Journal.* 16-17

Pazzani, M., Muramatsu, J. & Billsus, D. (1996). *Syskill & Webert: Indentifying interesting web sites.* AAAI Spring Symposium, Stanford, CA. Available from: www.ics.uci.edu/~pazzini/.

Pressman, R.S. (1992). *Software Engineering: A Practitioner's Approach – Third Edition*. McGraw-Hill Inc

Schlimmer, J.C. & Hermens, L.A. (1993). Software Agents: Completing Patterns and Constructing User Interfaces. *Journal of Artificial Intelligence Research*. 61-89

Standish, T.A. (1998). *Data Structures in Java™.* Addision-Wesley Longman, Inc.

Tan, A. & Soon, H. (July 1996). Concept Hierarchy Memory Model: A Neural Architecture for Conceptual Knowledge Representation, Learning, and

Commonsense Reasoning. *International Journal on Neural Systems.* 7(3):305-319. Available from: *http://www.ntu.edu.sg/home/asahtan/papers/CHMM.pdf*

Thompson, C.A. & Göker, M.H. (March 2000). *Learning to Suggest: The Adaptive Place Advisor.* AAAI 2000 Spring Symposium, Adaptive User Interfaces, Stanford.

At the time of downloading the papers from the WWW, the links were active. There is no guarantee that the links are still active after final publication of this dissertation.

8 List of Figures