An Adaptive Environment for Personal Information Management

A thesis submitted for the degree of Doctor of Philosophy

by

Richard John Keeble B.Sc. (Hons)

Department of Information Systems and Computing

Brunel University

May 1999

Abstract

This dissertation reports the results of research into the provision of adaptive user interfaces to support individuals in the management of their personal information. Many individuals find that they have increased responsibility for managing aspects of their own lives, including the information associated with their jobs. In contrast with traditional approaches to information management, which are generally driven by organisational or business requirements, the requirements of personal information management systems tend to be less rigidly defined. This dissertation employs research from the areas of personal information management and adaptive user interfaces – systems which can monitor how they are used, and adapt on a personal level to their user – to address some of the particular requirements of personal information management systems. An adaptive user interface can be implemented using a variety of techniques, and this dissertation draws on research from the area of software agents to suggest that reactive software agents can be fruitfully applied to realise the required adaptivity. The reactive approach is then used in the specification and development of an adaptive interface which supports simple elements of personal information management tasks. The resulting application is evaluated by means of user trials and a usability inspection, and the theoretical architectures and techniques used in the specification and development of the software are critically appraised. The dissertation demonstrates an application of reactive software agents in adaptive systems design and shows how the behaviour of the system can be specified based on the analysis of some representative personal information management tasks.

Table of Contents

List of Publications

This section gives details of publications arising directly and indirectly from the research reported in this dissertation.

Keeble, R. and Macredie, R. D. (1999). Experience with Adaptive Interface Agents. *Cognition, Technology and Work*, forthcoming.

Keeble, R. and Macredie, R. D. (1999). Assistant Agents for the World-Wide Web: Intelligent Interface Design Challenges. *Interacting with Computers*, in press.

Keeble, R. and Macredie, R. D. (1999). Software Agents and Issues in Personalisation: Technology to Accommodate the Individual. *Personal Technologies* **2**(3): 131-140.

Macredie, R. D. and Keeble, R. (1997). Software Agents and Agency: A Personal Information Management Perspective. *Personal Technologies* **1**(2): 44-56.

Macredie, R. D., Taylor, S. J. E., Yu, X. and Keeble, R. J. (1997). Virtual Reality and Simulation: An Overview. *Informatica* **21**(4): 621-626.

Macredie, R. D., Taylor, S. J. E., Yu X. and Keeble, R. J. (1996). Virtual Reality and Simulation: An Overview. In: *Proceedings of the Winter Simulation Conference*, December 8-11, Coronado, CA, USA.

List of Figures

List of Tables

Acknowledgements

I would firstly like to thank Dr Rob Macredie for his infuriatingly accurate supervision during my research, for introducing me to the idea of writing something more than once, and for making obvious to me all those obvious things that people still need to be told.

I would also like to thank Professor Ray Paul for his calm and yet firmly directed comments, and for finally 'nailing me down' (his words).

I also owe a debt of thanks to my family and my friends, without whose support I would most definitely not be in the position of finishing. Special thanks are due to Simon K (thanks again for the trip to the land vowels forgot) and the rest of the Attic for conspiring to get me to submit before the Millennium. Rotters.

I am also grateful to my funding body, the EPSRC (Award Reference Number 95700906) and my CASE industrial sponsors, Nuclear Electric Plc., for their support.

And one last thing. A'da tzuika, mi pu*a. (Mine's a pint, if you're buying.)

Chapter 1

Introduction

1.1. Introduction

An awareness of information technology (IT) is rapidly becoming a prerequisite for many people, as information, in its many forms, is becoming more central to our everyday existence. The availability of cheap computing resources and public telecommunications networks is resulting in increasing amounts of electronic information in many forms, accessible via different media, in many places (Stamper, 1994). Articles in the popular press serve to illustrate how the awareness of this electronic information, and the IT-based tools used to access and manipulate it is now finding its way into the lives of many people, at both personal (Sunday Times, 1997a) and more importantly, professional levels (Sunday Times, 1997b). Whereas IT was once the sole preserve of technical staff, the majority of office workers could not now perform their jobs effectively without knowledge of it (Kling, 1996).

Since much office work tends to revolve around managing information which refers to a company's resources, the management of information is therefore central

to the function of many businesses (Seddon, 1988). The majority of businesses rely on information about customers, suppliers and orders in order to be able to operate. For the section of the business community whose products embody information of some kind, the reliance upon information is far more acute. The effective management of information can be an important contributing factor to the success or failure of such enterprises. Organisations which manage their information effectively ought to enjoy an advantage over those competitors with poorer information management facilities (Earl, 1988).

In order to extract maximum benefit from the wide variety of information resources available, a range of factors needs to be considered: what information is required; will the raw information need to be 'filtered' or selectively discarded according to 'usefulness'; where can it be obtained from; by what means can it be obtained; how may it best be stored and/or organised, and how is it to be used.

To address these points in turn, the task to be accomplished will partially determine which information is required and within a given body of information which might be useful and which is irrelevant. This 'useful' information must then be obtained, raising the questions of location and method of access. This encompasses technical details such as the physical access method and network types involved and the protocols and addressing (Stamper, 1994) necessary to locate and transfer the desired information.

Once obtained, the information must be stored in a system ready for subsequent use. At the appropriate time it will be retrieved from storage and applied to the 'problem' – used as a basis for decision-making or perhaps used simply as input to other systems – often in conjunction with information gathered from other sources.

Systems which aim to support functions such as those described are broadly termed 'information systems' (Senn, 1989). Information systems development is

informed by principles derived from, amongst others, the field of information management.

1.2. Information Management

There are many different techniques and approaches to information management, each of which rely on particular assumptions. Depending upon the approach in question, there will be assumptions about the nature of the information to be managed, the organisation within which the information will be managed and the purpose for which the information will be used. In this section, several approaches are briefly summarised and the trend towards the personal management of information – with which this study will be concerned – is introduced.

Certain approaches to information management treat the problem from a top-down, organisational viewpoint: knowledge of the processes involved in a business is used to derive a plan for an information system to support that business' operation (Robson, 1997). This may be at a corporate level, providing a company-wide strategy or 'vision' for the way information is to be managed. This approach has the advantage that a standard policy exists across the organisation, giving a sense of security and uniformity. This can also be a disadvantage, since such an all-encompassing policy will be difficult to agree on, particularly if a company's activities are diverse.

If the company does have a diverse range of activities it is likely – simply due to practical considerations – that the company's internal structure will reflect the different activities carried out. The company may be structured into components which each perform a significant business function. These components are often known as 'business units' or 'strategic business units' and can be characterised as cohesive, identifiable fragments of a company with uniform business goals (Galliers and Baker, 1994).

Instead of enforcing a universal information management policy across the whole company, each business unit or strategic business unit could have its own information management strategy. This should ensure that the activities of the business unit are not hampered by irrelevant management procedures designed for other parts of the company.

Both of the above approaches (focused on corporate policy or business unit policy, respectively) indicate a organisation-centric approach to the management of information. Employees or groups of employees have access to repositories of information usually stored centrally at some sort of 'Information Centre' facility. This reflects the origins of many corporate information management policies: a central mainframe processing system, with terminals or other access points for the workers (Senn, 1989).

Whilst still valid as approaches to information management today, resulting systems do not usually take great account of the way in which individuals work, focusing instead on how the business or business unit works as a whole. Increasingly we see offices with a personal computer on every desk, connected to a corporate local-area network or 'LAN' (Stamper, 1994). Project groups within business units are given more responsibility for managing their own information on workgroup servers and other local machines. The trend is generally that of decentralisation (Lacity and Hirschheim, 1993) and the eventual endpoint of such a process may be that the responsibility for the effective management of information would reside firmly with the individuals involved.

The existence of this trend is echoed by the current moves toward decentralisation and outsourcing prevalent in the employment market (Reilly and Tamkin, 1996). Individuals will have more and more responsibility for obtaining, managing and applying information in their jobs and lives in general.

This shift in emphasis from the organisation to the individual requires a different way of conceptualising computer systems meant to manage information. The field of Personal Information Management aims to support this change in emphasis.

1.3. Personal Information Management

Personal Information Management (PIM) is concerned with how individuals manage their personal information: how they obtain; organise; and subsequently use information in their tasks. The aim is to support these activities to enhance an individual's effectiveness (Etzel and Thomas, 1996). PIM has its research roots in the cognitive and psychological study of individual work, as a thread of research which aims to develop techniques and artefacts to support the individual at work. PIM emphasises the 'personal' aspect of the process: the term implies a contrast to corporate, collaborative, shared and collective information management.

Systems designed to support such 'personal' information management need to be adaptable to their users' characteristics – that is, they should be 'tailorable' or 'personalisable' in terms of both their functionality and user interface(s) (Thomas *et al.*, 1994). The aim is to allow the user to work in their own personal manner, unhindered by the technology that is supposed to ease their tasks.

This is a difficult requirement to satisfy, since the exact processes carried out by an individual to perform a particular task will be entirely dependent upon the individual in question, as no two people work in exactly the same way. A range of approaches exist to provide solutions satisfying the problems resulting from these individual differences.

1.3.1. Approaches to Personal Information Management

One approach to the task might be for the user to carry out the whole information management task, as is often the case. A classic example of this approach in action

is the personal organiser – for example, the 'Filo*fax*™' – which saw a huge surge in popularity towards the end of the 1980's. This artefact brings together a diary, a contact list or address book, daily schedules and reminder notes, forming a valuable repository of personal information. This is in itself useful as a device to complement the human memory, but goes further than that to illustrate some of the key challenges faced by the providers of systems meant to manage personal information.

The physical integration of different types of information is attractive to begin with – everything is in one place, so time spent searching for a specific item is reduced to a minimum. This in turn promotes a functional integration of the information 'stores', in that tasks which require different types of information are supported. At the simplest level, calling someone to confirm an appointment later in the day 'integrates' the information from the diary and the contact list.

The personal organiser is also readily accessible at a cognitive level – people know *how* to use diaries and the other components of the artefact (Lees *et al.*, 1996; Jones and Thomas, 1996). In addition, personal organisers are tailorable, in two key ways. Firstly, components may be removed or added as required – new diary sheets, and indeed new sections that may not have been available before, can be added to satisfy changing requirements. Secondly, being paper-based in nature means that the human user is not rigidly bound by the form of the system, and is free to use it in ways not originally envisaged.

An approach such as this does guarantee that the user is able to work in their own particular fashion but is ultimately limited by the passive nature of the device. A large proportion of time spent managing any information is spent carrying out mechanistic tasks such as sorting and filing since almost any information requires some kind of organisation before useful work can be carried out using it. A man-

ual system requires the human user to spend time and effort on these routine tasks.

The mechanistic nature of these tasks implies that an automatic system could take control of some of the routine detail involved in managing information, leaving more time for doing work, rather than preparing to do it. Examples of the use of technology to provide support for the lowest levels of detail can be found in devices such as 'Personal Digital Assistants' or PDAs (Davids, 1996a; 1996b).

These usually provide facilities such as diary, contact list and to-do list management, thereby automating some of the most mundane parts of the PIM task. However, even using PDAs does not relieve the user of much of the routine activities of managing their information, since with much current technology the user is required to initiate all actions, to enter much of the information themselves, and to indicate what is to be done with it.

Such user interfaces rely upon 'direct management' since users interact directly with the computer (or whatever device they perceive it to be) to perform their tasks. This style of interaction comprises the majority of the user interfaces of today's systems.

In contrast to the 'direct management' of user interfaces, an emerging trend in computing is the use of 'indirect management' interfaces. Instead of the user directly managing their tasks, they could indirectly manage some autonomous process. Kay (1990) argues that a paradigm based on 'indirect management' can provide a more effective use of time when certain types of tasks are to be performed. Human users could instruct their systems to perform tasks to their particular requirements and then leave the system to carry out the tasks autonomously, only stopping to ask advice when needed.

The realisations of the 'indirect management' systems that Kay (1990) talks about are usually known as software 'agents'. The term 'agent' refers to some entity which can accept tasks delegated by a human operator and then accomplish them with the minimum of interference by (or dependence upon) the human. Although tasks are still 'initiated' by the user, the agent 'accepts' responsibility for carrying them out, and agents can (depending upon the situation and the agent) initiate subsequent tasks themselves.

The concept of agency can make a substantial contribution to the area of PIM, as systems implemented using agents can satisfy the key requirements of PIM systems. Perhaps most importantly from a user's point of view, they provide facilities for the delegation of routine tasks to autonomous systems. They can act as simple 'personal assistants' which work to complement a human user's intelligence, taking on the routine tasks which form part of the activity of managing personal information. This suggests that agents may form a worthwhile area of study within PIM.

1.4. Software Agents: An Introduction

Software agents can be thought of as virtual 'personal helpers' (Maes, 1994). They can be given simple 'goals' to accomplish and hints on what to do and how to do it in the form of user preferences. They can then be left to carry out their tasks on the user's behalf, reporting back when the tasks are complete.

As examples, Jennings and Wooldridge (1996) discuss software agents in the context of PIM, and the difficulties in obtaining and organising information from the World-Wide Web. Nwana (1996) gives a typology of software agents including a category he terms 'Information Agents' which perform similar information retrieval and categorisation tasks. Maes (1994) presents some software agents meant

to relieve users of some of the repetitive tasks involved in basic information management such as scheduling and e-mail sorting.

These routine activities give an idea of the kind of tasks agents can be used for, and fit in well with some of the underlying activities involved in PIM that were mentioned in section 1.3.

Software agents may be implemented using several different underlying technologies or architectures. Two of the most commonly encountered agent architectures are 'deliberative' and 'reactive' (Wooldridge and Jennings, 1995). Deliberative architectures utilise logic-based theorem-proving and reasoning to infer knowledge and plan actions according to observed events, whereas reactive architectures use a much simpler stimulus-response paradigm.

Although agents based on deliberative architectures are theoretically capable of far more advanced reasoning and planning than reactive agents, they are far more complex to design and implement and have potential problems which stem from their logic-based roots (Wooldridge and Jennings, 1995). It is difficult to produce formal models of the system's environment and to maintain them in a timely manner, and it is also difficult to design logics which can be manipulated so that conclusions may be reached in a finite time.

These considerations must be taken into account when a choice of agent architecture is to be made for a specific application area.

1.5. Software Agents for Personal Information Management

This dissertation will argue that reactive agents, although less 'powerful' in learning and reasoning terms, need not be any less useful than their deliberative counterparts. Mundane information management tasks do not usually require much intelligence to carry out, and so effectively represent wasted time on the

part of any human doing them. These simpler agents are also easier for developers to create and for users to relate to – an important issue for any system for which usability is important.

As mentioned in section 1.3, any PIM system must be tailored to (or tailorable by) its user(s). To be most effective at work, a user will not want to constantly adjust the way they work in order to accommodate some idiosyncratic feature of the system. If this were to be the case, a possible outcome might be that they could lose interest with it and in the worst case, cease to use it – even if it did offer some support for routine tasks. To combat this, a means of supplying the agent(s) with information about their user(s) will be required.

1.5.1. User and Task Modelling

The 'Personal' in 'Personal Information Management' is the driving force in this research. To make any agents (particularly those for PIM) useful, they must be designed around the user, as must any system meant to participate in human interaction (Norman and Draper, 1986; Shneiderman, 1998). This will encompass knowledge of a user's habits and working practices, the overall job they must eventually complete, and the sub-tasks they undertake in their progress toward their final goal.

To be able to design an agent or a set of agents to support a user in their work, an adequate modelling system is required. Two approaches might be considered. A specific user/task domain could be modelled: someone searching for information on the World-Wide Web for example. Another approach might be to attempt to construct a 'meta-model': to assemble components necessary for workers to define their tasks.

The latter approach would be more widely applicable and extensible. It may also reduce the probability of developing irrelevant agents: agents which perform tasks

as envisaged by the developers, which consequently prove to be unhelpful to their end users. In the majority of cases the final users of the agents will know far more about the way they work – that is, the way the *users* work – than the agent developers can hope to learn. The user's own knowledge could therefore be usefully applied to the design of software agents to support them if such a mechanism existed.

1.6. Statement of Thesis

This chapter has identified the broad area within which this thesis seeks to make its contribution – that of information management. Within this area the field of PIM was shown to be of increasing importance. This dissertation aims to provide a framework for the design of software agents which can complement a user's intelligence by supporting them in the more routine aspects of their PIM work. The framework will be based around the user's PIM tasks and working habits, in order to ensure that (as far as possible) the user can work in their own way without having to make adjustments for arbitrary technical reasons due to the system. This will be accomplished by providing the system with a model of its user which will enhance the 'fit' between user and system.

1.7. Breakdown of Thesis

This section gives a high-level outline of the rest of this dissertation.

Chapter 2 provides an examination of the concepts involved in PIM. It will be argued that user interfaces which adapt to their users offer potential advantages in the support of PIM. The use of software agents as an approach to providing the active elements in these adaptive user interface systems is expanded upon. Examples of current agent technology are evaluated to provide insights into both their particular strengths and weaknesses and those of the concept of agency in general.

This leads on to the choice of a suitable agent technology for the provision of agent-based adaptive user interfaces to support PIM.

Chapter 3 considers the needs of prospective users of adaptive PIM systems. The tasks and underlying activities involved are analysed and specific potential areas are identified where machine support is particularly desirable, using some self-contained PIM scenarios. The task elements noted are used to develop a specification for a user interface environment which provides adaptive support for the user. The specification is then used in a design process which concludes with a high-level abstract design for a class of user interface systems which can implement an adaptive PIM environment.

Chapter 4 documents the concrete design and implementation of a particular instance of the class of system specified in Chapter 3. The behaviour argued for in Chapter 3 is used as a starting point for a process of decomposition, leading to a set of simple reactive behaviours which are implementable as software agents. Contemporary techniques of object-oriented design and component-based development are applied to the problem, in order to yield a system which has a limited dependency on the target implementation platform. The prototype system is then integrated into an existing operating system's user-interface.

Chapter 5 evaluates the usefulness both of the resulting adaptive PIM environment, and the theory and techniques used in its design, development and implementation. A range of techniques are used to evaluate the different elements of this study: an empirical evaluation gains information about real individuals using the system, and their opinions of it; a heuristic evaluation uses high-level guidelines from the literature to examine elements of the system; the theoretical architecture used to specify the system is appraised; and the design approach and implementation techniques used are also examined.

Chapter 6 addresses problems uncovered in Chapters 4 and 5 and suggests how they may be addressed. A technical re-engineering of elements of the system is undertaken to illustrate how some of the target platform's technological features can be used to improve the prototype system's capabilities. A theoretical re-working addresses some of the limitations of the techniques used in the specification of the system.

Chapter 7 provides a summary and review of the dissertation and makes a statement of the contributions made by this study. A critique of the study as a whole is provided, addressing the limitations and the constraints of the work. Finally, suggestions for future research and other work in the area are made.

Chapter 2

Managing Personal Information: Taking an Agent-Based View

2.1. Introduction

Chapter 1 introduced the background to this dissertation, asserting that information and IT play an important role in the lives of individuals and many businesses (Martin, 1995). A side-effect of the increasing use of computers is that the volume of information created and managed by such systems is also growing. Ensuring that this information can be accessed efficiently is therefore an important issue. A current trend, particularly in directly IT-related employment but shared across many jobs generally, is the growth of the number of individuals who have more responsibility for themselves as well as their jobs. The increase in the numbers of workers classed as 'knowledge workers' (Collin, 1995; Rifkin, 1996; Roos, 1997; Eden and Spender, 1998) provides evidence for this. These individuals, in addition to the responsibilities of their jobs, are also responsible for managing their own affairs and their personal information (Etzel and Thomas, 1996).

This chapter will begin by examining how such individuals manage their personal information; looking at the activities they carry out and the devices or systems they use to aid themselves. PIM (Thomas *et al.*, 1994; Etzel and Thomas, 1996), as a sub-field of information management, can be supported using IT. To do this, some kind of user interface is required between the user and their information. Since the constituent activities involved in PIM are highly personalised, user interfaces for PIM should be designed around their users, to make the interaction involving the user and the interface more natural.

The act of 'designing' something implies some kind of fixed model on the part of the designer. Since users are different individuals to begin with *and* can change over time as they learn, this fixed model means that the 'fit' between the user and the interface may deteriorate over time (Benyon, 1993). Having an interface that is not fixed would help, but if the interface can change, something must bring about this change. Either the user has to customise, 'tailor' or 'adapt' it to their preferences (Henderson and Kyng, 1991) – which takes time to learn about and time to do – or the interface can adapt *itself* (Benyon, 1993).

An adaptive interface which monitors the user's interactions with it and responds by changing itself in an obvious and predictable manner can alleviate this additional load. The scope both for, and of, adaptivity should be limited in these systems (Höök, 1997) – much scepticism exists towards these systems, as a result in the main of past work on systems termed 'intelligent', which placed emphasis on the technological part of these systems at the cost of maintaining usability (Shneiderman, 1997).

This study does not aim to duplicate or replace the user's intelligence by 'doing their job for them'. The goal instead is to *complement* the user's intelligence by providing systems that can perform the simple, routine aspects of more complex

tasks without being explicitly instructed to do so. The approach taken is to provide simple adaptive elements to the interfaces used by individuals as they work.

Adaptive interfaces can be constructed in a variety of ways. One way to conceptualise them is as consisting of a changeable interface and a set of software 'agents' (Maes, 1994; Nwana, 1996; Kay, 1990) which monitor how the user interacts with the interface and changes it in response to observed behaviour. This is the line that this study will follow. Work in the field of adaptive interfaces is reviewed (Benyon, 1993; Schneider-Hufschmidt *et al.*, 1993; Browne *et al.*, 1990), providing a framework which will then used to identify how these 'agents' can be embedded in the interface and what the requirements for them are in terms of adaptive behaviour.

This discussion will lead on to a study of how the agents themselves may actually be thought about, designed and implemented. The term 'software agent' is used to mean many things. This is a consequence of the fact that the fields of research in which it is used are highly diverse, although some key agent characteristics do seem to be shared amongst the research areas which have adopted the term. A brief review of some of the concepts and terminology used will be provided, and will then be used to inform the choice of a suitable agent technology for adaptive interface systems. One particular category of software agents, labelled 'reactive agents' (Brooks, 1991a; Brooks, 1991b), matches the kind of sensing and reacting behaviour which this chapter will argue is important to the design of agent-based systems for PIM.

The chapter concludes by summarising the important issues raised in the chapter and the decisions arising from them. These are then used to restate the aim of the dissertation more precisely, in preparation for the more detailed design work in the next chapter.

2.2. Personal Information Management

As was discussed earlier, in Chapter 1 and the introduction to this chapter, information is fundamental to today's society, since much of it is centred around IT (Seddon, 1988; Senn 1989). As computers have become cheaper and more powerful, they have been adopted into a wide range of settings, resulting in a greatly increased ability to create, distribute, and accumulate information. Systems that support the management of this information – such as databases and spreadsheets – have become extremely popular and are an essential part of many organisations' work (Stamper, 1994).

However, one trend in computing is to develop systems and applications that are personal rather than organisational, aimed particularly at knowledge workers (Collin, 1995), suggesting the decentralisation of aspects of organisational computing. Today's offices and workplaces demonstrate this, for the most part being characterised by a personal computer on almost every desk (Kling, 1996; Zuboff, 1995). Decentralisation tends to result in individuals being given more responsibility for managing themselves and their information (Etzel, 1995) – tasks that historically might have been undertaken by secretaries, such as document preparation and filing.

To support and enable this move to information management at the 'personal' level it is important that we consider the development of computing systems which emphasise the personal aspects of work and that complement the individual user's capabilities and working practices.

This raises a central challenge – it is extremely difficult (if not impossible) to characterise the exact activities involved in the process of PIM (Etzel and Thomas, 1996). Each person works in a different way, influenced by their background and previous experience, the particular task in which they are currently engaged, and so on. A system meant to support users in managing their personal information

needs to take account of the activities undertaken as part of the process. A non-exhaustive set of representative key activities is given in Thomas *et al.* (1994), and is briefly reviewed here, in preparation for a more thorough exploration of approaches to the process as a whole. Knowledge of these activities, in conjunction with the nature of the information involved in them, will then lead to a discussion of the more complex practices adopted by individuals as they manage their personal information.

2.2.1. Activities Involved in Personal Information Management

At the most basic level, PIM systems obviously require information storage and retrieval facilities. Local information, created or manipulated by users, must be stored for later reference – information such as personal files, data, diaries and contact lists. Both local and non-local information required by users for their tasks will often need to be retrieved – this might be anything from flight booking information, or perhaps the contents of a financial newspaper.

Support for communication and integration is also necessary. Users will need to communicate with other users and other computer systems, to pass messages to them or to link to other information sources. More complex tasks will require integration – information of different types and from different sources might need to be brought together. Examples might include arranging meetings, scheduling events more generally and preparing supporting documentation or securing other prerequisite resources (Thomas *et al.*, 1994).

Although it is outside the scope of this study, managerial or executive level users may also require decision-making systems.[1] Stored, retrieved and integrated information might be linked together in various ways and some sort of 'intelligent

[1] For further information on this topic, consult Fidler (1996); Dhar (1997).

decision aid' or 'decision support system' used (Dhar, 1997), to highlight factors relevant to a particular problem and to help suggest possible solutions.

While PIM as a whole is a complex and varied activity, the component tasks mentioned above all have one quality in common – a significant amount of the time devoted to them will be spent carrying out actions which are repetitive in nature and exhibit scope for selective automation. This is particularly true of storage, retrieval and communication. This suggests that an aim of any PIM system should at least be to support the automation of such activities, where possible.

Having given a brief overview of some of the common, basic activities undertaken in managing personal information, the next section explores the nature of the information involved, as this has a large impact on the requirements placed upon any system designed to help manage it.

2.2.2. 'Personal' Information

The use of the term 'personal' to refer to the information which forms the focus of PIM systems can be misleading. The term 'personal' does have connotations associated with 'private', 'secret' or 'sensitive', which may or may not be true in any given situation, but is not necessarily the case. The use of the term 'personal' stems from the fact that information may only have meaning to a particular individual.

The information we are concerned with here is highly 'situated' in nature (Suchman, 1987), and (as an extreme example) may take forms similar to an annotation on a Post-It™ note. In this case, it is probable that the individual who wrote it meant it as a self-reminder and the information contained in the annotation is therefore highly context-dependent – without knowing what was being thought about or done at the time, it may be impossible to comprehend the information fully (if at all).

This has an important implication for systems meant to manage personal information – sometimes a piece of information may be 'incomplete' if examined *in vacuo,* and other items of information about the user may be required in order to interpret any meaning present. Two assumptions made here are that: the system will attempt to 'work alongside' the user by at least partially 'interpreting' the information they work with; and therefore the system will possess some 'knowledge' of the user in whatever form.

The basic activities mentioned earlier in section 2.2.1 (such as storage, retrieval and communication), in combination with the personal nature of the information involved, leads to the often subconscious development of a set of 'practices' adopted by individuals.

2.2.3. Personal Information Management Practices

The processes undertaken by people in order to manage their personal information are difficult to describe precisely. This section discusses some factors which cause this to be the case, and examines how this results in an impact on the design requirements for systems meant to support PIM.

The 'management' processes carried out are often highly implicit in nature, to such an extent that they are not consciously reasoned about, occurring almost automatically. They form part of the 'common sense' segment of a person's working practices and may therefore be overlooked as just part of 'being organised'. These processes are also highly individualised, being a function of a person's past experience and their current tasks and habits (Etzel and Thomas, 1996).

A spectrum of activities is involved, ranging from the sophisticated to the simple. The aim in this study is to complement the simpler activities, removing some of the tedium associated with the routine management of personal information. Any system meant to support PIM must at least acknowledge the existence of the more

sophisticated (and possibly unconscious) practices which develop over time (Jones and Thomas, 1997), as it is important to restrict the scope of PIM support systems in an appropriate manner – trying to 'second-guess' a user who is working according to some subconscious habit would be very difficult to do, and of rather dubious benefit.

At a basic level, the example concerning the Post-It™ note illustrates a means of temporarily recording an item of information for later reference. It is likely that this simple 'unit' of activity forms a component of a more complex process, simply because of the contextual nature of the information. For example, Post-It™ notes can be used for information which is meant to be recorded for a short time – a message or a reminder, for example – or for information which acts as an annotation to a larger document.

The processes undertaken are often based around physical artefacts such as diaries or personal organisers. This may be for several reasons, although it is most likely to be due to the fact that these items predate any technological support and are therefore highly 'accessible' from the user's point of view (Norman and Draper, 1986). Put simply, people already 'know how to use them', at least at a basic level, even if they are not used as part of an all-encompassing strategy for improving personal effectiveness. The field of PIM encompasses these ideas as well as newer technological solutions. For example, Etzel and Thomas (1996) give a detailed description of how an individual can adopt a strategy for managing their personal information which need not involve any computing devices (although their use can make the job easier).

2.2.4. The Development of Personal Information Management

The field of PIM also adopts some of the principles behind time management. Classical 'time management' involves just freeing up time to do more 'work'. More recent time management literature (Allen, 1995; Bliss, 1995; Croft, 1997) con-

centrates on combining these ideas with prioritisation and goals – as the most effective individuals do as a matter of course (Drucker, 1967; Drucker, 1968; Covey, 1992) – to ensure both that sufficient time is kept free for important and/or urgent work and that long-term goals can be met gradually, rather than just making sure there is enough time to do whatever happens to be on a to-do list. As a corollary of this, PIM's premise is that management of information impacts an individual's effectiveness – depending upon whether it is done well or badly, it can help or hinder the individual's performance. The ideas given in Etzel and Thomas (1996) show how a person's information may be made more manageable and maintained in the same way, thereby saving time spent searching for information and reducing the volume of information retained.

As the field of PIM has evolved, ideas for devices have been developed based around some of the activities noted earlier. Initial products were little more than electronic emulations of diary and personal organisers, taking the existing artefacts directly, as inspiration for metaphors for their user interfaces. These devices were termed Personal Digital Assistants (PDAs) (Davies, 1996a; 1996b), and were not as immediately successful as was originally expected (Laberis, 1995; Dieckmann, 1996). Later work attempts to go further than traditional PDAs in supporting PIM practices more comprehensively. One particular set of devices are termed 'Personal Information Appliances' and are examined in the next section as they illustrate some key requirements of systems meant to help in the management of personal information.

2.2.5. Personal Information Appliances

Thomas *et al.* (1994) suggest that a central concept should be that of *integration* – the ability of PIM systems or devices to use greatly differing sources and types of information and to combine or 'integrate' them in a manner which is transparent to the user (as mentioned earlier in section 2.2.1). They introduce the concept of

integrated 'personal information appliances' – hardware and/or software systems which manage personal information and are particularly designed around the concept of integration. These devices embody many of the principles mentioned so far in this chapter, and as such illustrate some of the qualities that this dissertation will argue are essential for successful PIM systems.

Two key characteristics of personal information appliances relevant to the discussion concerning user interfaces from Thomas *et al.* (1994) are:

(i) *appropriate information provision*: users do not want to waste time by having to mechanistically sift through information to discard irrelevant items. Information filters which can alleviate routine tasks such as these save unnecessary effort on the part of the user; and

(ii) *radically tailorable end-user interfaces*: users want to be able to work in their own way and will work more effectively if they are allowed to do so. A user interface that is radically tailorable or *personalisable* will support this, resulting in a medium for interaction which does not increase the mental load of the user.

These requirements are essentially consequences of the nature of personal information and PIM practices. The first characteristic results from the use of knowledge about the user to automatically provide information of use and/or interest, and the second characteristic results from the needs of the user to adapt the appliance to their habits and practices, rather than the other way round.

These characteristics place demanding requirements upon systems meant to support the activities that comprise PIM. Differing approaches exist to the problem of providing systems to aid the practice of PIM. These are examined in the next section.

2.3. Approaches to Personal Information Management

An 'approach' to PIM could be defined as the combination of artefacts (physical or virtual; hardware or software) designed to aid PIM, and the techniques used to take advantage of those artefacts. Two well-known approaches will be examined here using this definition in order to highlight some of the central issues currently facing the field. These issues will then inform the recommendation of an approach based on the idea of 'agency'.

A 'traditional' approach to PIM uses paper-based artefacts to aid the organisation of the information to be managed, relying on 'common knowledge' of such items as diaries, contact lists or address books, schedules and reminders (Lees *et al.*, 1996; Jones and Thomas, 1996), which can be brought together in a personal organiser, to form a system with a degree of integration – the differing types of information can be used in concert more easily. Newer PDA systems are generally based around metaphors derived from the traditional techniques of information management.

One limitation that must be borne in mind is that an artefact such as a personal organiser will be most effective if its user adopts a sensible set of practices to complement the stores of information contained in it. The artefact can only be seen as an 'aid', not an automatic cure for any organisational failings on the part of the individual. There is a spectrum of organisational ability, from those people who naturally have good self-organisation, to those who make little effort to organise themselves and their lives. The benefit that a person will derive from any organisational aid will depend upon their position on this spectrum to begin with.

The first generation of electronic personal organisers (Wheelwright, 1995) used embedded computing systems to computerise the information storage and retrieval functions associated with PIM. They were largely designed around the

same ideas as paper-based systems, with the emphasis purely on the automation of entry and searching (Dieckmann, 1996).

A problem with these systems was that although they did automate some of the underlying activities necessary, they did not tend to take account of the 'bigger picture'. Diaries, planners and telephone lists are useful items in their own right, but the success of paper-based personal organisers springs mainly from the ability to use these information resources in an integrated manner. To be effectively organised requires that the user be aware not only of the contents of each repository, but also of the relationships between them – a lot of cross-referencing and maintenance activities are required to use the facilities to their fullest, a burden which is placed on the users of these artefacts.

The main issue here is that traditional paper-based physical systems are passive in nature and have a static design. Their strengths spring from the fact that, whilst they are passive in nature, they can still be used in many ways – the contents and arrangement of one individual's personal organiser will probably be very difficult for another person to understand. However, these systems do require their users to undertake lots of mechanistic cross-referencing and maintenance tasks – for example, copying appointments and other information between the sections of a personal organiser – in order to keep them up to date, and therefore useful.

If the design of an 'active' artefact (such as a PDA) is based too closely on the corresponding passive artefact, there is a risk that the information management facilities provided by the artefact might be artificially limited by the original artefact's inherent design. Given the wide range of processes adopted by users in the management of their personal information, a device which overly constrains how it can be used may consequently be perceived as much less useful than the traditional alternative (Lees *et al.*, 1996).

A paper-based system such as a traditional personal organiser is 'open' – not tied to any mode of operation as such – and can therefore be used in any number of ways that were not originally envisaged by its designers. If a PDA were to support only direct analogues of traditional activities such as creating, altering or deleting entries in several different databases (examples being appointments, contact lists and so on) it would be missing opportunities to take into account a user's personal preferences as regards the way they work, and the information that they need. Thomas *et al.* (1994) highlight three main issues arising from this discussion.

Firstly, there is a lack of design guidance related to PIM processes and systems, which can lead to unverified assumptions and therefore a misunderstanding of the nature of PIM, where and how it occurs and the opportunities it offers for technological support.

Secondly, there is also a natural tendency for vendors to provide systems which are the easiest to implement rather than those which would provide the most benefit to their user – the process can be driven more by available technology, rather than the users' needs.

Finally, appropriate metaphors need to be used in the design phase (Jones, 1989; Erickson, 1990). If a paper system which is replaced by a technological solution is too closely echoed, this can lead to a computer-based system whose usefulness may be artificially limited. A designer cannot hope to anticipate all the ways in which the artefact will be used and has to assume that users will adopt processes which suit the metaphors chosen.

These issues indicate some of the immediate needs of the field. To address these issues, work needs to be done to yield design guidelines or some type of design framework for systems actually based on the analysis of PIM practices rather than just assumptions. This suggests that the design of PIM systems should be approached by considering the needs of the user. A variety of techniques based

upon this underlying idea exist, such as user-centred design (Norman and Draper, 1986), cooperative and participatory design (Greenbaum and Kyng, 1991; Schuler and Namioka, 1993) and usability engineering (Nielsen, 1993). Adopting an approach which focuses on the eventual users of the systems can rectify many of the problems arising from lack of knowledge of the nature of PIM, and can avoid providing a solely technological solution which proves mis-directed.

The choice of an appropriate metaphor is primarily concerned with the nature of the interface to a user's personal information. There must obviously be some medium which sits between the user and their electronic information repository, providing the mechanisms by which an individual accesses their information, searching and updating it for example.

Together, the issues discussed here imply that the design of the interface should be the subject of detailed analysis. As regards the choice of an appropriate metaphor, the dynamic nature of users would tend to indicate that, as well as being carefully designed around their users, PIM systems should not be static in nature – they too need to be dynamic so that they may be adapted to the processes undertaken by their users. Taking this idea further, we could consider user interfaces which can not only be tailored or adapted *by* their users, but can adapt *themselves* based on the way they are used. Such systems are termed adaptive user interfaces (Benyon, 1993; Benyon and Murray, 1993) or intelligent user interfaces (Sullivan and Tyler, 1991; Puerta, 1998), and will form the basis of the user interface design work in this study.

In addition to the idea of a PIM system having a dynamic interface, the low-level activities introduced in section 2.2.1 show that there are also opportunities for introducing *autonomy* into these systems – the idea that delegation of routine tasks can occur, where a user allows the system to carry out (for example) maintenance tasks without the need for every operation to be directly triggered.

2.3.1. Indirect Management of Personal Information

The nature of many of the activities which go to make up PIM naturally lend themselves to automation of some sort. A key part of effective management (of anything) is the delegation of certain tasks. This delegation may involve other people or technologies of some kind. We might think of e-mail systems, for example, as supporting certain functions of their human manager – undertaking 'delegated' tasks such as adding signatures and contact details to all out-going messages, or automatically copying and filing mail to certain respondents. This type of 'delegation' to computer systems is immediate in nature – such systems are used directly as tools where the dialogue is always user-initiated, specific and sequential (and therefore time-consuming) in nature.

In contrast to this 'direct management' of computer systems (and hence information) via user interfaces, a newer approach to interacting with computer systems can be used – that of *indirect management* (Kay, 1990). Instead of viewing the computer simply as a passive, user-driven tool, it becomes a system capable of accepting delegated tasks and autonomously carrying them out. Users can then manage their information *indirectly* by managing autonomous 'helpers' or 'agents' of some kind.

The different component tasks upon which PIM relies all have one common denominator, in that there exists the opportunity to automate significant parts of the tasks involved. A meeting scheduler would need access to information which may be stored in different locations and must be retrieved and integrated to yield an acceptable solution, which could take some time to do by hand. An e-mail organiser could use some set of rules or conditions to alleviate a similarly routine activity – as e-mail arrives it could be prioritised and sorted into different folders, without the user having to *explicitly* do this. These systems for information man-

agement support again perform necessary but basic tasks which should really be *transparent* to the end-users of these systems.

We have now identified two methods by which PIM systems might be improved: the interfaces seen by their users could adapt themselves to improve the interactions between system and user, and there could be the facility for the system to accept tasks delegated by the user. These two areas can actually be seen as part of the same problem – in order to be successful, they both depend upon the design of the user interface. A user interface which changes as it is used must do so in a predictable and obvious manner, otherwise the user may become disorientated and confused. In parallel with this, if a system is to accept 'commands' or 'tasks' as a delegate, there needs to be a way for the user to specify these tasks accurately enough to be able to have confidence that they will be carried out correctly.

The next section discusses user interfaces which fit into this category. The discussion will lead on to the means by which such interfaces can be implemented, which includes the use of entities referred to earlier as 'software agents'.

2.4. Adaptive Interfaces and Personal Information Management

We have identified that the use of a concept termed 'adaptivity' may be beneficial in enhancing the usability of an interface meant to support the activities of PIM. This section considers traditional user interfaces, demonstrating the differences between them and newer interfacing techniques, and examining the justification for their use.

Classical user interfaces (Norman and Draper, 1986; Shneiderman, 1998) are static in nature, embodying a design which (it is hoped) will accommodate the vast majority of the future users of that system. The fact that a system is designed and then fixed when implemented at some point means that it must be adapted *to* by its users, where any mismatch in a user's conceptualisation of the system occurs.

Similarly, since the system's design springs from the designers' perceptions of the system users, certain classes of users may be catered for poorly. Customisable or 'adaptable' systems can provide a partial solution to this problem, by allowing users to express their preferences as to the function or appearance of a user interface, yet this still places the burden of change upon the user. A central tenet of this class of systems is that the interface works purely as a tool, manipulated directly by its users.

In contrast, an adaptive or 'intelligent' user interface (Benyon, 1993; Browne *et al.*, 1990; Sullivan and Tyler, 1991) is one where the appearance, function or content of the interface can be changed by the interface (or the underlying application) itself in response to the user's interactions with it. The system will typically contain some component which receives signals or events from interface as the interaction progresses, and uses some rules to effect adaptation of the interface based on a set of criteria. In order to be able to do this, the system must possess a model of the user, a model of the system (or interface) itself, and a model of the interaction between the two – these themes will be returned to later, in section 2.6.

Having illustrated the differences between these two different approaches to user interfacing, the next section shows how adaptive user interfaces can be fruitfully employed to support PIM.

2.4.1. Adaptive User Interfaces for Personal Information Management

The nature of PIM – the fact that it is composed of a set of activities that are highly specific to the person working – means that any system meant to support PIM must be extremely flexible in nature if it is to be moulded around its user. As was discussed in section 2.4, increasing the flexibility of a system too much can render it much less usable – it is this demand for enhanced flexibility without too great a cost in terms of usability that provides some of the key motivation for the development of adaptive systems (Benyon, 1993).

The justification for having adaptivity in an interface is an important question for the designer, since adaptivity should only be used where it is appropriate. Benyon and Murray (1993) propose three criteria which can act as a guide in this situation. The *changing user* argument is based on the fact that people do not remain the same over time, and change according to their knowledge and experiences. Consequently, an initial 'perfect fit' between user and interface will deteriorate over time, necessitating some change on the part of one or other. The *a priori* argument states that for certain tasks, such as natural language processing, a fixed interface will simply not work, and adaptivity is required from first principles. The *usability* argument applies if it can be shown simply that the interface can be made easier or more effective to use if adaptivity is included in it.

Examining two of the key criteria for the utilisation of adaptivity in the context of PIM illustrates where adaptive systems may be applicable in this domain.

(i) *The changing user argument.* Adaptivity can be appropriate in situations where the user of a system changes over time, according to their tasks and abilities. Since the management of personal information is a means to and end rather than an end in itself – it supports the 'working context' of an individual; how they approach their work – they way the task is carried out will depend upon the context in which it is being carried out. As the individual's tasks change, they therefore may also change in terms of how they work;

(ii) *The a priori argument.* Adaptivity can be appropriate where a system must cater for a wide range of users to begin with, making the design of a sufficiently flexible system too difficult while providing adequate levels of usability. The class of users at whom these systems are aimed – professionals with responsibility for self-management and organisation – while sharing certain characteristics, can be found in a wide range of employment settings. This implies that a wide range of users would indeed exist.

Given the close relationship between the user and their personal information, a user interface meant to support PIM must support a wide spectrum of users with different habits *and* be able to cope with a user whose working practices may evolve over time – these facts satisfy both the *a priori* and the changing user arguments, and imply that appropriately designed adaptive interfaces can be fruitfully applied in this situation.

2.4.2. Justification for Adaptivity in User Interfaces

The use of adaptivity within a user interface should be driven by the needs of the user(s) of the interface, but must also consider the impact that the introduction of adaptivity has on the *usability* of the resulting interface (Höök, 1997). This dissertation uses the term 'adaptive' rather than 'intelligent' as far as they are applied to user interfaces. The reason for this is pragmatic in nature – human users are naturally suited to intelligent behaviour, whilst a great deal of time and effort must be expended in order for machines to echo such behaviour. In any case Petrie (1996) notes that in order to be useful, software agents do not necessarily need intelligence.

This dissertation supports the views of others in the field (such as Lieberman and Maulsby, 1996) that there is a theoretical spectrum of interactive systems which range from the totally fixed 'designed' system to the totally flexible 'adaptive' system, and a parallel spectrum of varying requirements in terms of 'intelligence'. Lieberman and Maulsby (1996) argue that there is a direct trade-off between flexibility and usability, and that the key aim of designing adaptive systems is in essence to pick the correct point on the spectrum between the two. There is no (non-trivial) 'absolutely usable' (and therefore highly specialised in design) system which has much flexibility, and the 'ultimately flexible' system, applicable to a wide range of situations and users, would be difficult to use.

The stance taken on this issue in this dissertation is that the simpler the desired adaptivity of the interface, the easier it should be to quantify and implement. A central principle of usability is that the user of a system should feel 'in control' of it (Shneiderman, 1998). Any adaptation that occurs ought therefore to be obvious from two perspectives: both obvious that it has occurred, and obvious *why* it has occurred. Much of the scepticism attracted by adaptive user interfaces – and agent technology as an approach to supporting adaptation – springs from the fact that this has not always been the case (Shneiderman, 1997). The inclusion of adaptivity or 'intelligence' in an interface has sometimes been seen as either an abdication of the responsibility on the part of the designer(s) to make the interface usable in the first place (Shneiderman, 1997), or as an attempt to 'replace' the user's intelligence in some way (Lanier, 1996a; 1996b).

Having suggested that adaptive user interfaces can offer advantages over traditional user interfaces for systems to support PIM, the discussion now turns in detail to the nature of adaptive user interfaces. This will clarify the requirements placed upon any software agent system meant to form part of an adaptive interface.

2.5. Adaptive User Interfaces

Any *adaptive* system can alter its state, and possibly its behaviour, in response to an interaction with another system. For example, humans can change the way they behave – they can learn new skills, for example – rather than being 'fixed' in terms of behaviour. Similarly, an adaptive user interface can monitor a user's interactions with it, and change itself based upon them. Any system which interacts somehow with another must possess a 'model' of that system in order to interact with it (Benyon, 1993). This is true of all systems and particularly so with user interfaces – the design of the interface embodies the designer's model of how the interface will be perceived and used (Norman and Draper, 1986).

To be able to function in this way, the system needs to be able to receive and process signals from some other system, and automatically change state or behaviour in response – it must be able both to *interact* and *adapt*. To do this, the system needs some way of converting raw 'sensor input' into information about the interaction, and it needs to have some mechanism by which it can change itself. These conversions necessitate three other classes of models: a model of the other systems with which the system interacts, a model of how this interaction can proceed, and a model of the system itself (Benyon, 1993; Benyon and Murray, 1993).

These models may be extremely simple. Benyon (1993) takes the example of a thermostat, a simple adaptive system which senses the ambient temperature and adapts the setting of the heating system based upon it. The model of the world used in this case is a bimetallic strip forming part of the switch. The model is entirely concerned with the temperature of the world, and nothing else – this is all that is required for the thermostat to function.

Adaptive systems can be very much more sophisticated than this, according to the function they hope to perform. An intelligent tutoring system (Mislevy and Gitomer, 1995; Mitrovic *et al.*, 1996) would be an example of an application requiring more complex models. The aim for such a system is to maintain a model of the student's level of knowledge and expertise, in order to adapt the instruction given to the student and to detect and correct misapprehension on the part of the student. In this case, it is extremely important to maintain synchronisation between the models held by both the user and the system – the system's model of the user's knowledge in the instruction domain must accurately reflect the user's actual knowledge, or the dialogue between the two will break down as a result of the user having been led 'up the garden path' (Suchman, 1987).

As was mentioned in section 2.4.2, concerning the justification for adaptivity in a user interface, these different types of system relate back to a spectrum of com-

plexity in both the user interface itself and its adaptive elements. A well-designed user interface which exhibits reasonably simple adaptivity should prove to be easier for individuals to think about and use more effectively than either a rigidly-designed, fixed user interface, or one that attempts to adapt too much by drawing tenous inferences about the user.

Having examined adaptive user interfaces at a reasonably theoretical level, this chapter will now proceed to explore how such interfaces can be decomposed into a set of component objects in order to be realised in software. This will show what technologies are appropriate in order to support adaptive interfaces, which forms the basis for the design and implementation work undertaken in Chapters 3 and 4. In order to do this, the concept of a 'reference architecture' – a set of component objects which are common to almost all adaptive systems – will be introduced. This will be used to inform both the design and implementation, by providing a framework within which the functionality of an adaptive system can be expressed.

2.6. A Reference Architecture for Adaptive User Interfaces

The discussion in the previous section approached adaptive systems by considering them as part of a larger system – a user, interacting with a computer system, in order to accomplish some task. In doing so, it was noted that this implied a need on the part of both these interacting sub-systems (i.e., the human and the computer) for models used to 'reason' about the interaction as it proceeds.

Three types of models were mentioned as being important in this situation: a model of the other party in the interaction; a model of the domain in which the interaction occurs; and model of the interaction process itself. Both parties to any interaction require elements of these models in order to communicate with each other, although the level of communication in terms of syntax, semantics and information will dictate the complexity of the models required.

The models referred to here can be thought of as components of what is termed an 'adaptive interface architecture' – a framework which binds together these components and provides some techniques for specifying and building the models. These may be developed into a 'reference architecture' for adaptive interfaces.

Benyon and Murray (1993) provide such a reference architecture, reproduced as Figure 2.1. This splits the adaptive system up into three major components, each in itself a composite model. The models are respectively termed the User Model, the Domain Model, and the Interaction Model. Each of the three models has a particular purpose, and constraining relationships exist between these models.

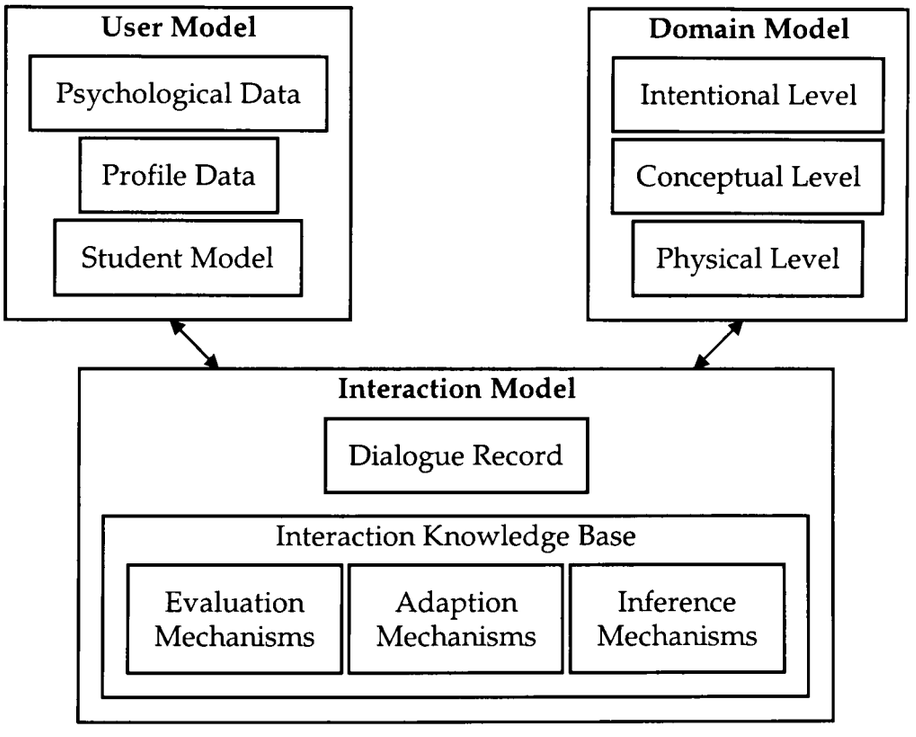

Figure 2.1. A reference architecture for adaptive interface technology (Benyon and Murray, 1993).

2.6.1. The User Model

The user model stores information about the user(s) of an adaptive system. This might seem to imply that there will be one (human) user of the system, but this is not necessarily the case – if the system interacts with a number of other adaptive

systems, a user model may be required for each. Three different sorts of information make up the user model:

(i) *Psychological data*: The psychological data reflect innate aspects of the user – abilities which are difficult or impossible for the user to change, such as spatial ability, character traits or personality;

(ii) *Profile data*: The profile data gives information about the user's likes, dislikes and so on – in a customisable system, the user profile would store information about the user's preferences;

(iii) *The 'student' model*: This contains information about what the user is deemed to know about the system they are using, termed 'domain knowledge'.

2.6.2. The Domain Model

The domain model stores information about the application area of an adaptive system. The domain model stores information at three different levels:

(i) *The intentional level*: Information at this level reflects the goals and objectives of the system's users as they work;

(ii) *The conceptual level*: This level contains information about components of the system that users are expected to reason about;

(iii) *The physical level*: This part of the domain model is concerned with the mechanics of the system – how a particular action can be brought about.

The intentional level's *goal* information is the highest level of data manipulated by any adaptive system – it effectively answers questions about *why* users do things. Conceptual level data is concerned with application-specific information about *what* operations users bring about – clicking on buttons, for example. Physical

level data comprises information about raw syntactic elements of the system, such as programming interfaces.

Information in the domain model forms the basis for the construction of the user model – that is, the user model cannot contain information about any aspect of the system that is not adequately reflected in the domain model.

2.6.3. The Interaction Model

The interaction model contains information about the user's interactions with the system and the mechanisms by which the system can adapt portions of itself. It consists of the 'dialogue record' and the interaction knowledge base. The dialogue record is simply a 'history list' of the interaction, containing raw data about syntactic events such as mouse clicks, menu selections and so on, forming the basic 'sensory input' to the adaptive system. The dialogue record must be transformed in order to be of use to the higher-level elements of the system, and this is part of the function of the interaction knowledge base.

The interaction knowledge base contains information about three different sets of mechanisms:

(i) *Inference mechanisms*: These are used to draw conclusions about users as they interact with the system;

(ii) *Adaptation mechanisms*: These reflect *how* the system can change itself, embodying the dynamic facets of the interface;

(iii) *Evaluation mechanisms*: These allow the system to judge whether its adaptions are making it more usable.

The inference mechanisms referred to in (i) are those used to gather information about the user without the system having to be explicitly instructed – an immediate issue is the quality and quantity of information available about the user. Any

conclusions drawn will only be valid or useful if there is a sufficient amount of reliable information available. The sorts of conclusions which might be drawn concern (for example) a user's expertise with a particular system, or a user's interest in a particular subject.

Adaptation mechanisms (ii) represent the facilities available to the system to effect change in whatever interface is presented to the user. Once the inference mechanisms have arrived at a particular conclusion about the user, the adaptation mechanisms are responsible for putting that conclusion into action, actually producing the change in the system. An example might be the mechanism needed to alter a hierarchy of menu items based on the history of a user's choices.

The evaluation mechanisms referred to in (iii) are used to self-regulate the adaptive system, forming a kind of 'feedback loop'. Some issues raised here are the means by which the system can judge whether its adaptions are making it more usable; what evidence is required to make such judgements, and how sufficiently accurate data can be acquired to make these judgements sound. There are different levels of complexity as regards the evaluation an adaptive system carries out. At the highest level, the system would need to possess a model *of itself*, to allow it to carry out the proposed adaptation on the model and to evaluate the effects of it before choosing to apply it to itself.

2.6.4. Key Relationships Between Models

The user model's contents are partially 'inherited' from the information stored in the domain model. Informally it could be said that the domain model constrains the extent of what may be 'known' by the system – that is, the information stored in the user model will be 'parameters' of elements of the domain model. This has the consequence that the domain model effectively limits the scope of the adaptive system's abilities – as ever, *any* system must have some underlying design which is fixed at any given time. The complexity of the domain model is an important

issue in adaptive systems design – a more flexible domain model which can change or be changed may allow more sophisticated adaptation. Any benefit arising from this has to be considered in parallel with the added complication introduced by adding it to the system.

2.6.5. Explicit and Implicit Models

Although the architecture described here consists of a set of models, each storing information about a particular facet of the adaptive system, it must be borne in mind that some of the details in these models may actually be implemented in an implicit manner. For example, the physical level of the domain model is concerned with the *syntactic* elements of the domain – how the system communicates with the user, for example.

In most software systems, this will be via a programming interface of some kind. Almost all programming languages require that such an interface be fixed (in interface definition files used to compile the software, for example). To make these elements of the domain part of a modifiable model would introduce considerable additional complexity into the design and implementation process of the system.

2.6.6. Realism in Adaptive Systems Design

These three levels may not be reflected in all adaptive systems. It may not be feasible to attempt to derive information about a user's goals and intentions in a given system for a number of reasons – primarily, the quality of information about the interaction must limit the inferences made upon it. If the information is unreliable or not sufficiently conclusive, any inferences drawn upon it will be subject to error. If any adaptions are subsequently carried out as a result of these inferences, the result may well be that the user ends up being confused about why the interface has changed in the way that it has. Limitations such as these must therefore be taken account of when specifying the desired behaviour of adaptive systems.

This tends to be true of all work which relies on what is basically an information-processing view of human-computer interaction (Landauer, 1991).

2.6.7. Software Agents as Components of Adaptive Interfaces

The preceding sections have described the nature and requirements of adaptive user interfaces, using an existing architecture for these systems. Adaptive systems have certain active components – for example, the inferencing and adaption mechanisms discussed in section 2.6.3, which will need to be implemented somehow. An adaptive user interface can be thought of as a *passive* collection of databases each storing information about some facet of the adaptive system, which is then manipulated by some *active* entity or entities. These entities or 'agents' may be specialised to some particular function such as recognising a given user action and generating an appropriate response.

The use of 'agents' as a conceptual tool within adaptive user interfaces can suit the requirements of PIM systems, in two different ways. Firstly, they can be configured to carry out routine tasks (such as the storage, retrieval and communication requirements) on behalf of their users, leaving the user with more time to spend on activities that do require human intelligence. Secondly, they can be incorporated into the interfaces used to access, manipulate and manage personal information, in order to permit the interface to adapt itself to the user, making the interaction more natural.

As yet the discussion has not focused on the theories or technologies which can actually be used to provide agent-based systems. The following sections present a review of the field of software agents to show what the idea encompasses, the different views of agents and the technical detail behind the theory.

2.7. Software Agents: An Overview of the Area

This section will begin with a discussion of general issues in the field of software agents and lead on to an exploration of the varying technologies which are associated with the term 'agent', in order to inform a decision as to which technologies may be appropriate to the development of adaptive user interfaces.

Thus far, the terms 'agent' and 'agency' have been used in this dissertation, without a great amount of clarification as to their meaning or implications – conceptually, technically or otherwise. A variety of software agent technologies exist, and the next section will discuss software agents themselves in more detail. This will lead to an informed decision about the types of software agents that will be appropriate to the task of supporting PIM through user interfaces which contain adaptive components.

2.7.1. Views of Software Agents

Many different definitions of the term 'agent' exist. Consequently, there are many different types of agents with different properties and characteristics. The central idea is that an 'agent' embodies some kind of autonomous process working 'in the background' to help achieve the user's goals.

There are a range of levels at which such agents can be considered to be working. At a high level, they can be viewed as an enabling functionality providing a service of some kind. Maes (1994) gives the example of an agent which will schedule meetings in accordance with a user's preferences. In this case, the agent is working at the information level – within the context of human understanding.

At a more basic level, Negroponte (1990) makes the suggestion that agents will be used for the simpler, basic tasks such as sorting and managing mail and files, handling telecommunications requests, etc. He visualises the majority of agents as

being far simpler in nature than Maes – ubiquitous and small (both in physical form and functionality).

An even lower level view given by Thomas *et al.* (1994) is that of controllable, minimal 'information management support' agents. These inhabit the heterogeneous 'information space' of personal information appliances, computing devices and networks present in any particular organisation. In this case the agents provide much lower-level data-oriented services such as store-and-forwarding, directory lookups or network fault-tolerance.

This illustrates the spectrum of agent types, from the very simple and low-level types of agent, up to agents which seem to be working almost at the same level as the human user. The next section identifies some key characteristics, drawn from a range of software agent research, which form the background to the development of software agents. These will then be used to make some informed choices about the particular agent technology appropriate for the application being considered.

2.7.2. Key Agent Characteristics

All agents share *some* properties or characteristics. A useful characterisation of agents is given by Jennings and Wooldridge (1996) who identify three classes, based upon their capabilities:

(i) simple *gopher* agents, which execute straightforward tasks using pre-specified rules and assumptions;

(ii) *service performing* agents, which act on well-defined requests from users, in a goal-oriented manner;

(iii) complex *predictive* agents, which provide information or services to users when the agent decides that it is appropriate to do so.

These classes of agents all have certain common qualities. In a review of the field of 'intelligent agents', Wooldridge and Jennings (1995) introduce what they term a 'weak' notion of agency. It is used to denote a (usually software-based) system that exhibits the following properties:

(i) *Autonomy:* agents should work alone, without intervention from humans or others, and have control over their internal state;

(ii) *Social ability:* agents should be able to interact with other agents and possibly humans through some medium, interface or language;

(iii) *Reactivity* or *responsiveness:* agents should be able to perceive their environment, whatever it is defined as, and react in a timely fashion to changes occurring in it;

(iv) *Pro-activeness:* agents should not only respond to their environment but also be able to act in an opportunistic, goal-directed manner and *take the initiative* where appropriate.

These 'hallmarks' might be present in differing amounts, depending on the nature and 'behaviour' of the agent. For example, a 'gopher' agent whose task is simply to monitor a web document for changes does not exhibit a large degree of pro-activeness. At the other end of the scale, 'predictive' agents which might recommend articles of news based on a user's interests and reading history do require the ability to take action without the receipt of a direct request from the user.

Agents can easily be visualised as roughly equivalent to a traditional operating system 'process' which exhibits these qualities. A typical example of an agent which corresponds to this style of agency is a *softbot*, or 'software robot' (Etzioni and Weld, 1994), used for goal-directed information retrieval.

As opposed to the 'weak' notion of agents and agency (Wooldridge and Jennings, 1995), for some artificial intelligence researchers the term 'agency' encompasses much more than discussed already. 'Strong' agents (typically referred to as 'intelligent agents') are often discussed using concepts more usually applied to humans, in addition to the weaker agents' capabilities. *Mentalistic* activities such as reasoning and learning (and perhaps emotion) are usually involved. While there are good reasons for ascribing such attributes to agents – see for example Bates (1994) and Minsky (1994) – they are unnecessary for the basic tasks involved in PIM. Introducing more human qualities, particularly those such as emotion, could lead to potential users of the system inferring more sophisticated functionality than that possessed by the system. Based on the needs of the application – that of complementing the human user in the management of their information – emotion would carry very little useful information, and would distract from the real focus of the system. The 'weak' notion of agency will therefore form the focus of this discussion.

The reason for this is primarily because we are concerned with the development of simple but ubiquitous software agents that can be used primarily in the role of assistant, capable of performing the mechanistic aspects of tasks whilst leaving users to perform those (aspects of) tasks that demand human intelligence.[2]

The term 'agent' is very much a 'banner', under which many different types of research are being undertaken. No one definition can possibly cover all work, and in a parallel fashion, agents may be classified in a great number of ways. To illustrate the wide variety of different ways of defining the term agent, an alternate treatment of the concept is given by Nwana (1996). He does not attempt rigorous

[2] The motivation for this approach is explored in more detail in Macredie and Keeble (1997).

classifications and instead provides some 'dimensions' along which to characterise agents:

(i) *Mobility:* agents may either be *static* or *mobile* – located in a fixed place, as traditional computer processes, or with the ability to move around (a network for example) in order to be closer to required resources;

(ii) *'Thought' model:* an agent which possesses some internal symbolic reasoning model and planning and negotiation faculties may be referred to as *deliberative*, after the deliberative thinking paradigm. In contrast a purely *reactive* agent functions according to a stimulus/response principle;

(iii) *Attributes of agenthood:* agents can also be classified according to the possession of certain key attributes (not unlike the hallmarks of agenthood mentioned earlier). Nwana (1996) gives autonomy, learning and collaboration as a basic minimum set, indicating that there are certain qualities that most people agree upon, although even this limited set can be controversial;

(iv) *Roles:* the tasks agents are meant to accomplish can sometimes be used to classify them – examples given of major roles are World-Wide Web search engines and spiders, classified as *information agents*;

(v) *Hybridisation:* a agent can be termed a *hybrid* if two or more separate agent philosophies are embodied within a single agent.

Given the diversity of the field, a precise definition of the term *agent* is not generally possible, and is probably not desirable since such a wide variety of work is ongoing under the 'agent' banner. The central issue is the underlying concept of *agency* itself – that humans could explicitly delegate routine or boring tasks to autonomous processes that can carry them out more efficiently and without suffering the associated boredom.

2.8. Agent Theories, Architectures and Languages

The technologies upon which software agents are based is an important factor – in the context of this dissertation, a choice will need to be made in order to implement an agent-based system as part of this study. There are many different views of how agents should be formally conceptualised and reasoned about, how they should be constructed and the notations used to specify their actions. Although the main focus of this dissertation lies elsewhere (in the ways in which agents will actually be used) the material referred to here will be used to inform the development and implementation of software agents accomplished later in Chapter 4.

An *agent theory* is an underlying formalisation involved in conceptualising some set of agents. Agent theories are primarily useful in describing the properties of systems to be built, while adequately reflecting the different levels of abstraction used by the designers, implementors and users of agent systems (Shoham, 1993). Strong agent theories are often expressed in some sort of modal or temporal logic (Singh, 1994) and include means of specifying agent properties such as *intent, belief, desire,* and so on. Weaker agent theories tend to concentrate more upon the agents as being part of a larger system, rather than being disembodied 'bits of intelligence' in their own right (Brooks 1986; 1990).

An *agent architecture* provides a set of analytical tools and techniques used to reason about how some agent(s) functionality can be realised as either software or hardware (Maes, 1991). Since agent systems originally emerged as a subset of multi-agent systems (Singh, 1994) in distributed AI (Bond and Gasser, 1988; Chaib-draa, 1994; Chaib-draa *et al.*, 1992a; Chaib-draa *et al.*, 1992b), there is a tendency for them to be viewed as a particular type of knowledge-based system, known as *symbolic AI* (Bond and Gasser, 1988), although there are other means of viewing them. Classical AI recommends *deliberative* agent architectures (Huhns and Singh,

1994), where agents possess some logical model of the world, and have the ability to reason about the world using it.

However, Wooldridge and Jennings (1995) indicate central, difficult problems, which must be resolved in order to build deliberative agents: *transduction* – the problem of the timely translation of the real world into an accurate and adequate symbolic description; and *representation* or *reasoning* – how to symbolically represent information about complex real-world entities or processes and how to get agents to reason with it in time for the results to be useful. To combat the problems associated with these deliberative architectures, *reactive* architectures were devised (Brooks, 1986; 1990; 1991a; 1991b; Maes, 1991; Etzioni, 1993). These do not use explicit models of the world in the style of symbolic AI and also do not use any complex symbolic deductive reasoning, but can still exhibit 'intelligent behaviour' Brooks (1991a; 1991b).

An *agent language* is a programming language which permits the implementation of an executable agent process of some kind. The symbolic AI roots of much agent research has led to many agent implementations in LISP, due to the language's ability to explicitly encode programs as data with ease (Wayner, 1995). Sun's 'Java' language (Gosling and McGilton, 1995) provides an intermediate solution, realising applications as consisting of a set of 'applets', although the mobility of Java applets is severely limited (for security purposes). To combat this, a public-domain mobile agent toolkit developed in Java has been developed by the Agents Group at IBM (Chang and Lange, 1996). As a purpose-designed agent language, General Magic's 'Telescript' environment (White, 1994) provides for agent programs which can run at one location, and then issue a 'go' instruction which results in the program moving between hosts as if nothing had occurred.

2.8.1. Existing Agent Systems: An Overview

The previous section described some underlying 'enabling' technologies which fulfil elements of the requirements of agent-based systems. This section presents a brief review of the state of the art in software agent research and development to give some specific examples of how agents and user interfaces have been brought together to yield adaptive or dynamic interfaces that have some of the characteristics of 'assistants', rather than simply being tools.

Nwana (1996) gives an informal typology of agents, which is adopted in this dissertation to act as an analytical framework, using which the different agent systems which exist at the moment may be critically examined. Nwana himself admits that many researchers find fault with his typology, reproduced as Table 2.1 – this is to a great extent inevitable, since so many agent types overlap with others to a greater or lesser degree.

It can be seen that a single agent may be classified as belonging to more than one of these classes – yet the set of classes does cover the vast majority, if not all, current examples of deployed and emergent agent technologies. Some examples of agent system technology are now introduced to show how they relate to each other, and the typology given in Table 2.1.

Chin (1991) describes how 'intelligent' user interfaces themselves can be characterised as containing agents, as entities that cooperate with the user – collaborative agents. Cypher (1991) reports on the development of an assistant agent called 'Eager' which can spot repetitive patterns in the user's behaviour, infer the sequence, and complete it under supervision. Maes (1991) gives the example of a somewhat simpler approach to using agents in user interfaces – rather than inferring sequences of operations, the interface is conceptualised as consisting of a set of reactive 'competence modules', each responsible for a different behaviour.

Agent Type	Agent Description
Collaborative	Agents which emphasise autonomy and cooperation (with other agents) – and therefore social ability – as well as responsiveness and proactiveness in order to perform tasks for their owners.
Interface	Agents which emphasise autonomy and learning to perform tasks as 'personal assistants' for their owners, thereby with less emphasis on inter-agent collaboration.
Mobile	Processes which are capable of roaming across wide-area networks (WANs) or indeed the Internet (and therefore the World-Wide Web), obtaining information from remote host computers on behalf of their owners, then return having accomplished the tasks set.
Information/Internet	Agents which aid the acquisition, integration and interpretation of information from a range of possibly distributed data sources, shielding their owners from the huge amounts of irrelevant information now available.
Reactive	Agents which work without modelling the external world and instead operate on simplistic a stimulus-response principle, situated in some kind of environment. A group of these agents can then exhibit emergent behaviour when viewed as a whole.
Hybrid	Agent systems which bring together components belonging to two or more of the five previous agent 'classes', in order to both maximise the strengths and to combat inherent weaknesses of the component technologies.

Table 2.1. A typology of software agent technologies (Nwana, 1996).

Maes' work (1991) has its original foundations in robotics – its original proponent Brooks (1986) insisted that physical embodiment was a prerequisite, although Etzioni (1993) countered by asserting that software environments could be just as effective. The principles were further used to build interfaces to large bodies of information (in organisations or on the World-Wide Web), which used 'software robots' or *softbots* (Etzioni and Weld, 1994; Etzioni *et al.*, 1994; Etzioni, 1997) with a user interface which could accept searching tasks and carry them out without direct supervision, as information/internet agents.

Other work focuses more closely on the 'assistant' style of user interfaces described in Table 2.1. NewT (Maes, 1994) supports the automatic sorting and organisation of electronic mail messages. Letizia (Lieberman, 1995) and Webmate

(Chen and Sycara, 1998) are two examples of assistant agents embedded in user interfaces to act as a Web browsing assistants, recording a user's browsing actions and making suggestions for Web pages that may be of interest.

The discussion up to this point has illustrated the range of technologies which can be classed as 'agent systems'. The next step in the development of a system to support PIM is to choose an agent technology which fits well with the needs of both of the user, and of the interface within which the agents will be embedded.

2.9. An Appropriate Agent Technology

We see the objective of providing software agents to support PIM as *complementing* the user's abilities, in much the same way as Maes (1994). We do not want to attempt to 'replace' the user's intelligence, as this is simply not feasible given the current state of the art. Instead, we wish to provide systems more in the mould of 'assistants' (Cypher, 1991; Lieberman, 1995), which can spot opportunities to automate routine activities and offer to take over, or to provide suggestions which can make the interaction more effective. At the same time the overriding necessity of user-centred design for PIM systems must be borne in mind, due to the highly personal nature of the processes and information management practices that take place.

Based upon the agent typology introduced in section 2.8.1, we could combine the interface agent and reactive agent categories, to yield an adaptive interface which does not make complex inferences about its user, and therefore requires inputs of a more limited (and therefore more realistic) nature. Out of the differing agent technologies reviewed in this chapter, reactive agents (or at least, systems based on reactive agents embedded in user interfaces) appear to offer a beneficial avenue of research, based upon the evidence provided thus far. A set of reactive agents

embedded in an adaptive user interface would be suited to the task to their characteristics in several different areas:

(i) Reactive agents have good 'cognitive economy' (Ferber, 1994) in use – that is, they are easy to conceptualise and think about for the user. This is in agreement with one of the central tenets of classical user-centred design philosophy (Norman and Draper, 1986), in that users can only control interfaces effectively if they have an accurate mental model of them. As a side-effect, this decreases the 'cognitive load' – the mental effort – on the user so that they are not distracted from their work;

(ii) The 'situatedness' and 'embodiment' (Brooks, 1990) qualities of reactive agent technologies are well-suited to the application area of software agents embedded in user interfaces;

(iii) Reactive agents are computationally simple (Wooldridge and Jennings, 1995) and therefore quick-functioning, which is essential in a hard real-time situation such as a user interface, where sluggish response times can markedly decrease the perceived effectiveness of an interface (Johnson, 1997).

Although reactive agents do offer a profitable direction in research, they do have their own characteristic weaknesses. These are considered in the next section, which then guides the final focus of this dissertation.

2.9.1. Reactive Agents: Issues to be Addressed

Some of the advantages of reactive agents lead to key research issues in their own right. The most important in this situation is that purely reactive agent systems do not possess any sort of model of the world. For a personal assistant system that must adapt to a user's way of working, some kind of model of the user is clearly required to facilitate any degree of 'shared understanding', required where 'partnership' is to take place. Even simple limited 'adaption' to users would also be

useful, as something (as long as it is the 'right' adaption) is better than nothing (Orwant, 1996). Allying the adaptive interface architecture with its models of the user and system to the simple adaption facilitated by reactive agents should go some way to addressing this issue.

On a more pragmatic level, one of the main criticisms of reactive agent systems research is that much of the current work is based on an ad-hoc, trial and error approach where systems of reactive agents are put together anew for each application (Nwana, 1996). There is no guarantee that the resulting emergent behaviour will be quasi-intelligent as required, which indicates that this needs to be addressed by developing some design guidance for a system of this type. In fact, similar problems are suffered by adaptive user interface architectures (Benyon, 1993) – it should be noted that development of these can lead to improved design guidance both for the interfaces themselves and the reactive-style software agents used to construct them, which acts as part of the contribution by this dissertation.

2.10. Restatement of Thesis

The direction of this dissertation can now be restated in more detail. This dissertation will focus on the development of a system based on an adaptive user interface, hybridising reactive software agents with an existing adaptive interface architecture which provides a user modelling resource. A system of this type, where existing research can guide the choice of reactive agent behaviours will provide a better chance of obtaining useful emergent behaviour that does appear 'intelligent' and can be used to assist individuals in managing personal information. This process will be informed by the development of an associated design framework for a suitable interface architecture using reactive software agents as the active element of an existing adaptive interface architecture.

2.11. Conclusion

This chapter has explored the field of PIM in more depth, driven in particular by the trend towards the individual and therefore more 'personal' technologies. Some traditional approaches to PIM were examined, and some key issues arising from them noted. It has been shown that significant opportunities exist to provide automated support in many PIM scenarios.

The chapter highlighted some areas in which knowledge is lacking – design guidance for PIM systems based around the user and able to adapt to her. The use of adaptive interfaces in systems to support the management of personal information was suggested, illustrating how the requirements of PIM systems can be fulfilled using interfaces which can adapt themselves based on their user's working habits and practices. A concept known as 'agency', based on an indirect-management paradigm involving delegation to software agents was introduced, and it was shown that a user interfacing approach based around this idea can provide solutions to some of the central problems concerned with PIM. It was shown that software agents can be used in the realisation of these interfaces, which led to an overview of current work in the development of agent systems.

This chapter has also shown that the class of agents referred to in the literature as 'reactive' can offer behaviour which appears to be 'intelligent', whilst bypassing the problems associated with 'deliberative' architectures, but also demonstrated the need for more concrete design methodologies for reactive systems (this issue forms the basis for much of the design work in Chapter 3). The use of reactive agents as part of adaptive interfaces was put forward as a possible solution to some of these problems. To provide PIM systems using reactive agents together as part of an adaptive interface, it was shown that through a combination of requirements from user-centred design and the design guidance needs of reactive

agent systems, that the provision of a system based around a user model, as part of an existing adaptive interface architecture is needed.

The next chapter will present the development of a design framework for adaptive user interfaces which utilise suitable reactive software agents based on the issues raised in this chapter.

Chapter 3

Developing a Framework for an Agent-Based Adaptive Interface for PIM

3.1. Introduction

Chapter 2 established that information management, and in particular Personal Information Management (PIM) will become increasingly important due to various factors, and illustrated the nature of the activities involved and some of the central issues facing the development of systems to aid the practice of PIM. It demonstrated that these systems can be conceptualised as interfaces which contain adaptive components and suggested that such systems can be implemented as a set of reactive agents as part of an generic architecture for adaptive interfaces.

This chapter will therefore present the design of an adaptive interface which utilises reactive agent technology within a traditional direct-manipulation interface (Shneiderman, 1998). The design will be kept abstract to begin with – the aim of this work is to provide a framework which is language, platform and operating-system neutral (as far as is practical). The resulting abstract design will act as the basis for a concrete design and implementation in Chapter 4. Factors such as the

operating system and user interfacing toolkits in use will then influence the exact form of the design and subsequent implementation undertaken.

3.1.1. Overview of Chapter

One of the main aims of this study is to provide systems to support PIM. This chapter will therefore present the design for an interface between the user and their personal information, which can then be augmented with adaptable elements and adaptive behaviour to support routine activities. This interface is scoped and specified in terms of the basic activities that users perform when working with computer-based information.

For the final system to be implementable, the chapter will then show how user interfaces work in general – that is, how they provide a usable 'buffer' between the user and the underlying complexity of a computer and its operating system. It will also demonstrate the additional requirements of interfacing systems which can change or be changed. The chapter will then show how such a system can be designed, based on the requirements of the interface itself, and the technical requirements involved in user-interfacing software. These requirements include the event-driven paradigm of software development; the implications of platform-independency; the availability and use of user interfacing toolkits; the effect of object-orientation on the design and implementation of interfaces; and the procedures which can be used to integrate such a system into the user's interface environment.

An abstract design will then be derived for a class of system that will address the requirements of the interface and the implementation issues. Specifically, the framework of the architecture for adaptive interface technology (AIT) will be used to partition details of functional requirements into manageable elements. These will then be considered in the light of technical information concerned with user interface implementation to show how they can be realised in software.

3.2. User Interfaces to Support Personal Information Management

In order to provide an adaptive user interface which aims to support PIM, we must understand the requirements of user interfaces in general and in particular the requirements of PIM systems. Considering the stance taken earlier in the dissertation (see section 2.4.2) – that simple, well-defined and obvious adaption in an interface runs much less of a risk of violating the tenets of user-centred design – key requirements of PIM systems need to be adequately reflected in a user interface which *remains* usable.

We need to know what would be useful in an interface that claims to support PIM; we need to understand what we can provide that will ease the task of PIM. It should be noted that this does not necessarily imply that the system will automate the *entire* information management task (whatever that *is*), but will provide some kind of support for a particular sub-task or area of PIM which can sensibly be augmented in a straightforward manner.

Rather than concentrating purely on the automation of basic activities, one approach suggested by Thomas *et al.* (1994) is to 'informate' elements of the interface – that is, to make better use of available information resources to enhance human-computer interactions. Although it could be argued that the end results may be similar, the idea of focusing on applying available information to augment user interactions instead of slavishly automating interfaces appears to offer a more realistic chance of improving the situation.

3.2.1. PIM in Action: A Scenario

A brief scenario may be useful in order to provide some insight into the component tasks undertaken by an individual as they manage their personal information. A user, sitting at their PC, works to satisfy some goal. The goal may be entirely personal, or may be for the benefit of others. In any case, they generally

work with 'applications' in order to fulfil the task in hand. These 'application' programs (such as word processors, spreadsheets and databases) manipulate information, usually in the form of files. Many tasks require that information from different sources be combined to yield a final product – an example would be the preparation of a report, including financial performance figures of a particular project, to be sent to a given group of people. This task might require that information about a project (in a word-processed document) is combined with numerical data (from a spreadsheet or database), and is then sent to a given list of people, whose details are stored in a contact list (a specialised form of database). These various sorts of information need to be accessed, which additionally poses the problem of retrieval – locating a file which contains a particular item of information of interest, and organising files such that they can be found easily at a later date. Such organisational activities are usually based around 'directories', analogous to filing cabinets, which store related items of information in a hierarchical structure.

Following the philosophy of aiming to provide simple systems (which are consequently more usable), an area which would benefit from support is that concerned with the organisation, presentation, storage and retrieval of information. This task is implicit in almost all others which take place in the course of working with information. It also takes place at a reasonably low level of abstraction, and will therefore be more amenable to the analysis and design processes needed to construct software.

The more high-level 'goal-directed' behaviours, which utilise the basic information management activities mentioned, are far more difficult to infer; assuming even that this was feasible, it may not be clear how to assist them. Yet again, the theme of simplicity and predictability leads us to consider simpler activities first, but to remain aware of the more complicated possibilities for the application of software assistance.

3.2.2. User Interfacing for PIM: Simple Information Management Tasks

In the context of PIM, the elements of interest will be at the conceptual level (referring back to the discussion in Chapter 2 and particularly section 2.6.2 of the different levels of abstraction in adaptive interfaces), concerned with reflecting the user's collections of personal information. From the scenario, we see that well-known items such as files, folders and applications are manipulated by users. It would be possible to use refinement types of these object classes – for example, specialised 'database', 'diary' or 'telephone list' files, associated with their respective applications. In addition to simply using file types to know which application to run with a particular file (as is commonplace today), a PIM system with information about file types, groupings and contents could perform checking and cross-referencing between files that might otherwise have to be done by hand.

Another possibility is to aim to aid the organisation of these objects as well as their content. Some kind of 'associative indexing' of objects based on content would be useful – for example, someone working on a particular project will generate a set of files related to that (and possibly other) projects. Ensuring that this repository of associated information is arranged well and is easily accessible is a task which requires fairly simple organisational skills. Automatically organising a set of files (or presenting an associatively-organised view of a set of files) could offer advantages in terms of reducing the time spent actively managing the arrangement of files and time spent searching for a particular file.[3]

---

[3] It is important to bear in mind as this scoping and design process continues that the eventual output of this study will be a tool for an individual. It may well have components which can adapt to some characteristics of its user, but it nevertheless remains a tool. As was noted earlier in section 2.3, an individual with good innate organisational abilities will tend to be able to exploit such a tool to better effect than a disorganised person – however, we do not seek to train individuals in self-organisational practice.

The information management tasks mentioned so far all have one feature in common – as they centre on the manipulation of information, their use requires that the information upon which they work can be obtained easily. As discussed earlier in section 2.2.1, there are some readily identifiable key activities which form much of the basis for PIM: storage, retrieval, communication and integration. These activities are briefly discussed to give an understanding of the basic functional requirements necessary to provide workable PIM systems.

Both storage and retrieval activities require some kind of underlying organisation. A repository of information is of greatly diminished value if it is poorly organised, so we should aim to support individuals in organising their personal information. Communication is more poorly-defined and more wide-ranging (phone, fax, e-mail, LANs and intranets and the Internet), and integration is even harder to characterise precisely.

Storage and retrieval activities have a reasonably well-defined scope, and typically occur in an interactive fashion. Given the stance of this dissertation – that adaption needs to be simple, obvious and predictable – the basic activities of storage and retrieval are suitable candidates for automated, 'informated' support.

In this particular context, storage will be defined as placing information in some repository for later access and manipulation, and retrieval will be defined as gaining access to information previously stored in some repository. Fundamental to both these activities are the methods used to locate information within the repository – information cannot be retrieved (without searching) if its location is not known or easily guessable. Associations between information – such as different information related to the same project – would also ideally be reflected by having related information located together. A means of supporting the location of information within a structured repository would therefore facilitate both storage and retrieval activities.

Means to assist communication and integration of information would be feasible if it were possible to infer a user's intentions from their actions within the user interface. This is quite a problematic area, due to the wide range of tasks which encompass elements of PIM and the fact that the accuracy of inferences tends to be dependent upon the narrowness of the domain involved. The initial focus of this study will therefore be limited to supporting the management of information storage and retrieval activities, and means to aid the organisation of the information involved.

3.3. Analysis of Simple PIM Activities

Based on the discussion in the previous section, this section examines, in detail, some specific and complete examples of activities that fall within the remit of PIM. This approach is inspired by so-called 'task-centred' design approaches (Lewis and Rieman, 1993). A task-centred approach requires the designers of an interactive system to actively consider complete examples of tasks that users need to be able to do, and to use these task specifications as the basis for the system's design.

The chief characteristics of useful task specifications in this context are as follows: They describe the user's aims, but not the mechanisms by which they are accomplished – eliminating any pre-conceived ideas about how tasks should be done. They describe specific details, allowing consideration of the exact procedure to be followed, thereby prohibiting designers from hiding behind 'generality' – i.e., deferring the consideration of details until (too) late in the design process. They describe a whole, self-contained 'job', in its entirety. Some simple activities are now analysed using this approach, to give possible objectives for the final system.

The tasks discussed below are not entirely separate, but overlap somewhat. There may be no entirely 'right' way to support these activities, but a selection of good, safe alternatives ought to be better than none.

3.3.1. Improving Access to Poorly-Placed Files

In retrieving information from some kind of hierarchical file system, the user often needs to navigate through it. If a user repeatedly selects a file using an inefficient route, some attempt should be made to improve the situation:

> *Scenario:* The user might wish to open a file called 'Fax Template', in a folder 'Letters', in a folder 'Documents' that appears on the desktop, and do so by opening the two folders, and selecting the file. They may wish to access the same file within ten minutes, having to repeat the same actions (opening two folders, then opening the file). The file may be poorly placed, and the system should offer a better alternative, allowing the user to access the file without so much intervening navigation.

If the user opens a desktop folder 'Documents', immediately opens a subfolder 'Letters' within it, and then opens a file 'Fax Template' contained in 'Letters' (with that sequence of access operations having a gap no greater than twenty seconds between each), the system should record this. If the user subsequently repeats these folder navigation and file opening actions within ten minutes, the system should offer some means to repeat these actions in a more efficient manner. Many platforms provide 'shortcuts' – logical 'links' to files – and these could be used to help.

Some applications do offer a 'recently-used file' feature – although it is possible to unintentionally defeat these for some tasks, particularly when a boilerplate file is opened, modified and then stored as a new file. In many applications (MS Word and Excel, for example) the recent file list will contain the new file's name, but not the name of the template from which it was derived. (Again, some applications offer the ability to set up template files for often-used documents, but the procedure for doing this varies and is often complex, involving details that have nothing to do with everyday work.)

3.3.2. Improving Access to Regularly-Used Files

Several files in a user's filesystem may be used quite regularly – perhaps several times in a week, or maybe a couple of times a day – but not necessarily be used multiple times in quick succession. Some kind of aide-memoire to help the user locate their file could be useful.

> *Scenario:* The user might wish to open a file that they have used a few times before and that they last accessed yesterday, but does not now appear on their word-processor's default four-item recent file list. They know that they have accessed the file before, but cannot recall where it is (and may not remember exactly what it was called). The system should offer a better alternative, allowing the user to access files historically somehow.

A brute-force exhaustive search might yield the answer, assuming the user can remember enough about the file – a fragment of it's name, for example – but would be inefficient. A list of some kind, with details of previously-opened files (not just saved files, as most applications offer) could provide a solution. Files that have been used approximately daily might be presented in one list, whereas those used (but used less) may appear in another.

3.3.3. Allowing Contextual Annotation of Files

People often want to annotate files with information, without wanting to modify the contents of the files themselves. This is also true of paper documents, as the sales figures of sticky-note suppliers will attest to. The exact objective of a user in annotating a file may be difficult to fathom, but it is still an inherently useful thing to be able to do.

> *Scenario:* The user has a set of document files in a folder, some of which need work to be done on them, requiring some information which is not yet available. Some days later, one of the items of information arrives and the user cannot recall which

document required it, without scanning through each of the documents it may apply to – or all of them if the user has forgotten. The system should offer a better alternative, allowing the user to locate the correct file directly without having to manually search through all the alternatives..

The ability to store information about other information (so-called 'meta-information') could be provided by using a similar mechanism to paper notes, where small text annotations could be stored, associated to particular files. It would be preferable to have this fact reflected visually, allowing the user to see which files have annotations attached without requiring any further operations.

3.3.4. Aiding the Location of Files of Interest

There is another category of files (in terms of usage patterns) that has not yet been considered. A file may not be used regularly, but might be part of an important task that has yet to be completed for some reason. Nevertheless, the user might need to be able to access a file that had been worked on in the past and left with work outstanding.

Scenario: The user has a document file that must be sent to a particular recipient, for whom an item of address information is lacking, and may be some time in arriving. The user writes themselves a reminder note, sticks it on the side of the monitor, and puts the file in a folder somewhere. Several days later, the information arrives, and the user cannot recall where they put the file. The system should offer a better alternative, by allowing the user to locate files quickly based on their contents somehow.

The problem here is that although the user thinks ahead enough to write themselves a reminder, the note is still separated from the object to which it refers, and cannot be used to find it (unless, of course, the location of the file is noted as well). In any case, it is still tiresome to have to scan a lot of notes in the hope that one may contain the nugget of information required.

Instead, the file-annotation support described in section 3.3.3 might be augmented to include a feature that paper note documents do not have – i.e., associative recall. There is no reason why a user might search for a file based not on information contained *in* the file, but on information *about* the file. Although there is no direct analogue in the physical world, it is very likely a user might want this to be possible. In essence, to rule this option out would be to impose the limitation of a parallel paper system upon a computing solution, which would be most unhelpful.

The four task specifications discussed in sections 3.3.1 to 3.3.4 give some grounding to the types of support it may be possible to provide. Using these as concrete examples to work with, a synthesis of PIM support techniques will now be presented, which will then lead to a set of software requirements.

3.4. Opportunities to 'Informate' PIM

This section draws together the preceding discussion to identify specific ways in which PIM could be supported by augmenting the user interface. Support by the interface could either be active or passive – that is, either by adapting to the user or acting directly under her control. These two categories might be characterised as follows:

(i) *Active support*: the system, acting autonomously, responds to the user's actions and attempts to adapt their workspace, making the interaction environment closer to some kind of 'ideal' situation;

(ii) *Passive support*: the system provides tools with which the user can organise their own workspace or augment it so as to better reflect their habits, preferences and current tasks.

Essentially, passive support is the more traditional of the two – many systems allow users to express preferences and tailor their environments, whereas active support (provided in this case using adaptive interfacing) tends to be more novel.

The four scenarios described within section 3.3 each have different requirements in terms of the kind of support which would be appropriate. Using the scenarios and the two different support types, the following mechanisms seem to be appropriate.

Considering passive support first, a strength of paper-based organisation systems is that they can be augmented using notes – free-form pieces of information which can be attached to the items to which they refer. It would be possible to provide a tool which echoes this idea, and would allow people to annotate objects (files, folders, applications) with textual information. Once the ability to allow people to annotate objects with textual information has been provided, an obvious extension is to allow the contents of these notes to be searched at a single point, permitting users to locate files or objects based on their annotations.

With respect to active support mechanisms, a user's filestore is a central element in their work. Certain files in a user's filestore are used more often than others, and it would be useful if these files were more readily available without any searching. By monitoring which files are opened by users, a prioritised list of regularly-used files can be compiled, based on how often and how recently the files have been accessed.

The information about a user's file accesses could also be used to draw inferences about the placement of files. If a file is poorly placed in the filestore, the result will be an increased amount of folder navigation necessary to locate and open it. Where a user navigates through some kind of directory structure to find the same file more than once within a short time, a suggestion could therefore be made that a short-cut to the file might be of use.

Following the call for an integrated approach to the support of PIM, these individual support elements could be provided as part of a holistic PIM support application/environment. The next section examines how this could be accomplished.

3.5. An Adaptive Information Management System

A system could be designed to act as the 'medium' between a user and their personal information. In the scenarios mentioned earlier in sections 3.3.1 to 3.3.4, the user operates a set of applications which work on information. In that sense, although the applications' interfaces shield the user from the precise details of the individual operations on their information, they do not shield the user from the organisation required to maintain their collection of files. An interface which can work at a higher level of abstraction could augment existing file-oriented applications in such as way as to support file organisation as well as being able to monitor and adapt to the user's actions as they work.

Such a system could take the form of some kind of 'Adaptive Information Management System', which can support and integrate a user's interactions with information-based – and hence file-oriented – applications such as databases, spreadsheets or word-processors. One component could be like a 'desktop management' system which monitors how the user works with files and applications – helping users to maintain order in a collection of files (of different types) related to different tasks. (The majority of current personal computer systems employ proven metaphors to aid the representation and manipulation of such items based on graphical displays, so similar techniques should be employed by this system.)

Other components to be provided might be simple assistants to existing applications, in the mould of traditional Web assistant-style software (Lieberman, 1995). In PIM terms, tasks like cross-referencing and information integration (such as providing assisted support for merging document and address information) could

also be supported. The idea of automatically cross-referencing files and documents based on content overlaps is attractive – a problem in retrieving information is often that one remembers roughly what the item referred to, but not the exact content or the exact location.

Autonomous assistance may be possible – perhaps support for reorganising file locations (or adding shortcuts) based on the frequency of use of a particular file. However, the goal must be to maintain a usable system, so excessively complex adaptive behaviour should be discouraged as it may confuse an individual to have elements of the user interface changing very much in a short time.

Now that we have a good idea of the sort of thing that an adaptive interface to support PIM will need to do, a more precise scope for the system to be implemented will be outlined.

3.5.1. Scope

The preceding discussion has suggested that the Adaptive Information Management System (AIMS) could fruitfully be realised as an interactive 'desktop' style application. As such, it would need to be either integrated into the operating system in question, or provided as a separate 'File Manager', 'Explorer' or 'Finder' style application. Each of these applications acts as an intermediary between the user and their files, directories and applications. In operating system terms, they are referred to as 'shells', in that they encapsulate details about the underlying information and operating system commands and data.

In either case, there would be a 'top level' desktop, which would reflect filing systems and applications. Files and directories should be represented in the normal fashion, as icons or detail lists presented in windows. It is normal for these applications to provide support for file-type associations, based on well-known filename extensions and/or contents. For example, any file whose name ends in the

sequence '.TXT' might be deemed by the operating system to be a plain text file, and '.DOC' files might contain word processor documents.

It would be desirable to provide support for 'shortcuts' – essentially, 'links' to files, so that popular files could be 'left out' on the top-level desktop for ease of access, removing the need to constantly navigate through the filing system to reach them. From a basic PIM perspective, this would provide a mechanism which would allow users to tailor their environment. In addition, this facility could be used by the system in proposing automatic shortcuts – if a file was deemed to be poorly placed by the system, a shortcut to it could be suggested.

These ideas will now be used to generate some detailed functional and subsequent interfacing requirements for the adaptive information management application as a whole, which can then be used to develop the abstract design.

3.5.2. AIMS: Functional Requirements

Following on from the informal description of the AIMS application and the scenarios discussed earlier, this section provides a set of functional requirements for the application, bearing in mind the scope set out in the previous section. As the system will augment an existing graphical user interface (GUI), certain operations need to be present as a matter of course. The fact that the system will also offer adaptive support to the user adds more requirements on top of these.

At a basic minimum, the user must be able to gain access to their files and run applications, using the conventions supported by the eventual target platform. This should ensure that any user who has been exposed to the vast majority of desktop-style GUIs should be able to use the application simply as a normal desktop, without difficulty.

In addition, the scenarios presented in sections 3.3.1 to 3.3.4 suggested that a useful feature would be to enable the user to annotate objects with textual 'notes'.

This will allow users to record ad-hoc textual data in context, by associating it with the information to which it refers. In tandem with this feature, the system needs to allow objects to be located using a search based on annotation text, reflecting the advisability for the system to be able to locate files associatively by annotation. These requirements essentially echo the passive support of PIM by the system.

With respect to active support, the system should be able to suggest shortcuts to files which are used frequently but are not readily available – that is, are located in places which need several folder navigations to access. Based on the user's file access behaviour, the system should also provide an adapted view of the user's files, in response to their usage patterns. In addition, the system will need to maintain an index of file annotation contents and information about the referred-to objects. These requirements then echo both the needs of the passive support mechanisms and the opportunities for active support.

3.5.3. AIMS: Interfacing Requirements

Following on from the informal description of the AIMS application and the scenario discussed earlier, this section examines the functional requirements arrived at in the previous section to yield requirements for an interface to support the system. The development of the application embodies several assumptions, the most fundamental of which is that the information to be manipulated by the users is stored in a traditional desktop filestore, using such concepts as files and folders. As one of the predominant styles of interaction, this may mask the fact that regardless of how information is stored and represented, there will be needs and opportunities to support the organisation of any body of useful information.

Within the AIMS application, well-known items such as files, folders and applications will be manipulated by users. These objects can be reflected directly in a free-form desktop-style user interface, in a similar fashion to the majority of direct-

manipulation interfaces (Shneiderman, 1998) in use today. The object's type can be used at a basic level to represent it in the user interface – in the same way as current systems such as the Windows Program Manager, the Windows 95 and NT Explorer, or the Macintosh desktop and Finder – by choosing an appropriate pictorial representation, or 'icon', for the object.

The more advanced features – such as the file-annotation and searching support, the shortcut suggestion mechanism and the file-usage monitor – will need to be presented to the user in such a ways as to be as obvious as possible. The goal is to provide mechanisms which are sufficiently simple to be almost 'instantly' usable – that is, an individual with experience of the target platform should have no trouble using the features without instruction.

The next step in producing a detailed specification for a system to support PIM is to be aware of the techniques and technology available to be used in the provision of user interfaces. This will then inform the design of the system, ensuring that the design arrived at can be implemented as easily as possible.

3.6. User Interfaces Revisited

Chapter 2 discussed the interplay between user interfaces and personal information management, containing adaptive components. Up to this point, user interfaces have been mentioned in the context of other technologies but have not been concentrated upon in their own right. A more comprehensive understanding of how user interfaces can be conceptualised, designed and implemented, and the ways in which interfaces can be presented to their users, now follows.

This discussion begins at a reasonably abstract level, presenting the principles underlying user interfaces and the techniques used to present interfaces in such a way as their function can easily be deduced from their appearance. This discussion then moves on to the technology necessary to provide *dynamic* elements in

user interfaces – whether this happens as a result of explicit tailoring or customisation by the user, or as a result of the system adapting itself to the user. Both these techniques have consequences in terms of additional functionality required as part of dynamic user interfaces, which influence how the final interface is implemented.

3.6.1. Basic Principles

This section provides a brief review of common user interfacing techniques at a reasonably general level, in preparation for the detailed design work to come later.

The design for a user interface is critical in establishing the user's ideas of what the user interface is capable of, and how it can be used in order to accomplish a given task. In essence, the user interface is acting as a sort of communication medium between the designer of a system and the eventual user of the system. Norman and Draper (1986) refer to a set of three related models – termed the Designer's Model, the System Image and the User's Model. The designer of an interactive system has an idea about how the system will work, and constructs the design model to reflect this. The resultant visible implementation of the system is termed the system image, and is the primary representation of the system available to the user.

The system image must therefore convey sufficient information about what the interface is and how it may be used, whilst at the same time hiding irrelevant details from the user. The user, as a result of exposure to the system image, forms their own model of how the system works, and interacts with it based upon this model. If there is some discrepancy between the designer's model of how the system works and the user's model of how to exploit it – as a result of presenting the system inappropriately – the end result will be that the user will have misconceptions about how the system operates and will consequently find the system to be less usable than it would otherwise be.

The system image may take several forms. The vast majority of modern personal computer systems use graphical means to represent objects within the system to the user, and hence, to generate the system image referred to. A user interacts with the computer by manipulating pictorial elements on a display screen, and these actions are 'translated' internally into the underlying operations to which they refer.

An example is the process of moving a file between directories in a filesystem. Folders or directories and files might be represented to the user as rectangular 'windows' or icons which can be 'clicked' or 'dragged' to perform actions. To move a file from one folder to another, the user might have to indicate the file to be moved by pointing at it with the mouse cursor, depressing the mouse button – thereby 'picking up' the icon – moving to the destination folder's window, and releasing the mouse button – a 'drag and drop' manoeuvre. This combination of actions in the GUI results in some set of underlying calls to the operating system, which effect the desired result – the file disappears from one directory, and reappears in another.

The previous section referred to 'icons' and 'windows' as graphical objects that users can manipulate. These are essentially visual metaphors which act as abstractions from underlying complexity. A system's graphical representation – i.e., the system image – should clearly demonstrate what the state of the system is, and perhaps just as importantly, how it can be used.

Almost every user interface employs a subset of well-known elements from the user interface designer's palette. These components can be separated into two sets: those concerned with simply interacting with the interface (*syntactic level* – for example windows, icons, dialogs and pushbuttons) and those concerned with manipulating the user's information (*conceptual level* – files, folders and applications – the items of interest in a PIM system). These components form the underlying

'currency' of the interaction between a human and a computer, and need to be chosen well in order for the interface to have an obvious 'meaning' to its users. Many interface components commonly in use are traditional metaphor-based elements (windows, applications, files and folders, for example), using direct abstractions to represent objects within the system.

3.6.2. Providing Dynamic Functionality in User Interfaces

In addition to the basic elements, a user interface which is meant to be applicable to a wide range of users needs to possess some means by which to accommodate them. Two options immediately suggest themselves: either the interface needs to be *adaptable* or *tailorable* – the user changes the interface to reflect their preferences, or it needs to be *adaptive*, where the interface monitors how it is being used and changes *itself* in response.

Both approaches can be used simultaneously and a balance of the two is often a more preferable situation – radical tailoring (if required) can be done by the user, and the simpler (and hopefully more predictable) adaption can be the responsibility of the system. Both of these approaches require additional functionality in the user interface over and above that needed to implement a traditional fixed interface. This section provides a brief summary (with a small amount of technical detail, leading up to the later design work) of user interfacing with tailoring and adaption.

The provision of tailoring or user preferences generally requires use of some kind of a software toolkit such that the appearance of the user interface can be generated programmatically, rather than being wholly static and predefined in nature. Examples of this are modifiable menus and toolbars, common on many systems today. User Interface Management Systems (Olsen, 1992) are applications which provide both programmatic support for dynamic interfaces, and also provide an

interface for *generating* and maintaining interfaces, thereby reducing development load.

Obviously, such an approach requires additional interfacing support to control the rearrangement of elements of the user interface under user control. Many applications allow user-customisation, e.g., the "Tools | Customise" menu selection in Microsoft Word, which allows toolbars, menus and keyboard shortcuts to be altered to suit a user's preference.

Customisation such as this represents one end of a spectrum from simple to complex or 'radical' tailorability (Malone *et al.*, 1995). As usual in such situations, there is a corresponding trade-off in terms of cost/benefit – the more customisable a system is, the more effort must be expended in producing it, and the harder it can be to use. Some studies have shown that many users choose to exploit preferences and customisation to make new versions of software appear and work exactly like the old ones (Mackay, 1990), and consequently that 'tinkering' with an interface accounts for significant amounts of wasted work time.

The provision of interface adaption brings a slightly different set of requirements. As is the case for tailorability, dynamic aspects in the interface require some kind of GUI 'toolkit' approach which can build and reconfigure user interfaces 'on the fly', so almost all the requirements noted for tailorable systems apply to adaptive systems. However, instead of providing an interface to allow users to directly change how the interface looks or responds, some element of the interface must react to events triggered by the user as they work with the interface. This can be provided by embedding reactive agents (as discussed in section 2.9) within the interface to observe events and respond to them.

A combination of *adaptable* and *adaptive* interfacing techniques seems to be required (as Benyon, 1993 and Fischer, 1993 agree), particularly as an *adaptable* interface possesses the scope for many of the underlying dynamic aspects mentioned as

being necessary for *adaptive* user interfaces. This combination is therefore sensible from a usability viewpoint as well as acknowledging practical issues in the implementation of adaptive user interface systems.

Given the scope and requirements for the AIMS application, the adaptive interface architecture (as discussed at a theoretical level in Chapter 2) is now used to embody these requirements.

3.7. Applying the Adaptive Interface Architecture

This section briefly reviews the architecture for adaptive interface technology introduced and explained in section 2.6. It then provides details on the application of the AIT architecture for this particular project, based on the functional description and requirements derived earlier in this chapter.

The key concept to bear in mind is that the architecture, and consequently, any adaptive system built according to it – is based around agent/device interactions. In this situation, an 'agent' is a system which has internal states and – in any sense – acts to perform some goal. Agents are distinguished from devices in that devices do not contain any internal state information, and agents interact using devices. Since the architecture is based on a 'soft systems' methodological approach (Checkland and Scholes, 1990), the human actually interacting with the computer is seen as forming part of the system, as an agent in their own right. The class of adaptive systems built according to this architecture perform transformations from 'syntactic' sensory-type event information, through the level of semantic events – where meaning in a particular context has been arrived at – to 'goal-based' information. These different levels of abstraction are shown in Figure 3.1.

Syntactic events give raw input to a user interface – they are the basic signals a user can supply, such as mouse-button clicks and keypresses. Semantic events are formed from a composition of syntactic events and context – the same syntactic

event (e.g., a keystroke) could have different meanings depending upon 'where' in the interface that event took place. Such events are related to conceptual level models, as they express a limited, specific intention on the part of the user. Examples might include the user issuing an instruction to save a file, or to exit a program. Generic controls – for example, pushbuttons – triggered by a syntactic event, such as a mouseclick, have specific meaning in a particular setting – such as a 'save or exit' dialog.

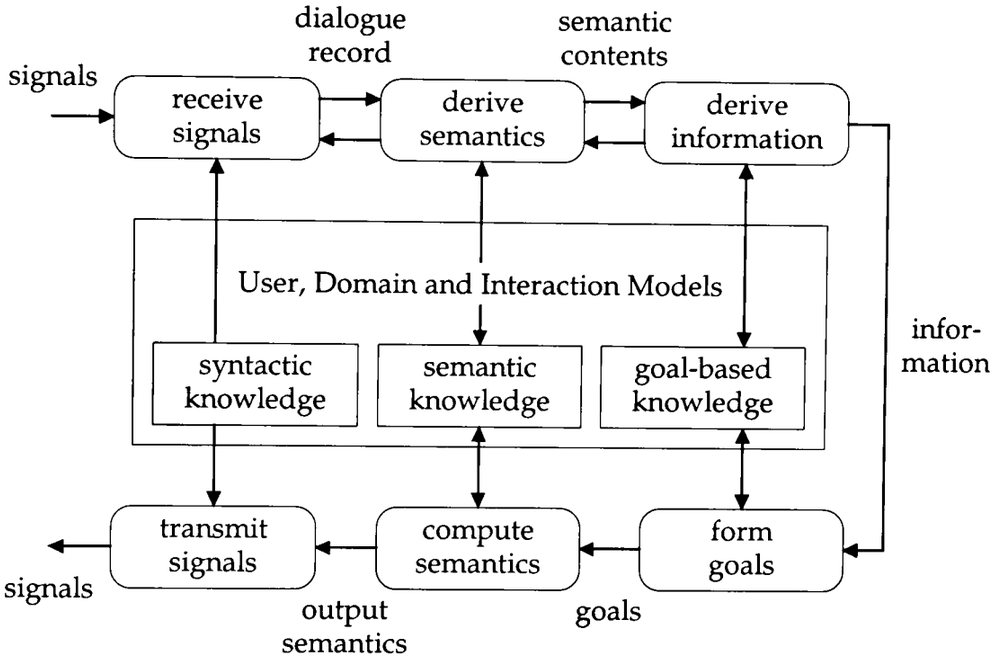

Figure 3.1. Syntactic, semantic and goal-based levels in an adaptive system, from (Benyon and Murray, 1993).

Higher levels of event information are possible. More sophisticated systems may attempt goal-based reasoning, trying to fathom the user's intention in performing an action. It may be possible to do this in some settings, by restricting the methods available for the user to interact with the system and narrowing the domain of the interface. It is therefore not the focus of this study as it is difficult to do and prone to influence by quality and quantity of interaction information available. Consequently, the systems developed in this study will essentially bypass the goal-based level, resulting in a direct flow from the 'derive semantics' process to the 'compute semantics' process of Figure 3.1.

As introduced in Chapter 2, different models are used to perform these transformations. These models reflect different elements of the system as a whole: the user, the domain within which they and the interface are working, and the interaction which proceeds between them. The content of each of these models depends upon the task for which the interface is designed. The different levels of event information are echoed by different levels within the models present in the AIT architecture, used to transform between the syntactic, semantic and goal-directed event levels. This can be illustrated by examining a simple adaptive system as an agent-based solution where different levels are involved in acquiring raw sensor data, transforming it into syntactic event information using contextual information, processing this, and then re-transforming them back to syntactic information used to adapt the underlying interface. (For a more detailed exposition of the content and purpose of the components of the AIT architecture, refer to section 2.6.)

We are now in a position to begin a specification for each of the elements of the AIT architecture, based on the requirements of PIM systems in general, and the interfacing needs of the example system put forward earlier in section 3.3.

3.7.1. User Model Requirements

This model is concerned with which elements of the user should be modelled by this system. A more realistic view of the user model is to think of it as that set of 'facts' about the user which the system believes is true – rather than attempting to construct a detailed model of the user, which has its own set of problems, this study will attempt to take account only of simple information about which we can be reasonably sure, as supported by others in the field (Orwant, 1996). Any items of data to be stored in the user model should therefore be readily available – or at least, able to be obtained without too great a danger of misinterpretation.

As mentioned in section 2.6.1, there are three types of information which may be stored in the user model: psychological data, the student model and profile data.

Each of these types is now examined to arrive at the content of the user model for this application.

Psychological Data

Psychological data are supposed to quantify a user's inherent cognitive characteristics. A simple example of this is spatial ability – an individual's aptitude for navigating within environments, like a windowed desktop. For a system meant to support a user working in this type of environment, information about their ability to find their way around could be used to tune any adaption meant to ease navigation, although it is unclear how the system might infer good or bad spatial ability.

The Student Model

The student model finds most of its applications in situations where the adaptive system is attempting to teach its user. Computer-based training applications are an example of adaptive systems which try to gauge the student's level knowledge, altering the questions or advice accordingly. As the focus of this project is not primarily on training, the student model plays a correspondingly diminished part in this study. However, it could be used to store information about the user's knowledge of the system at simple level – for example, whether the user is a novice or an expert. This could alter the way in which the system adapts in terms of presenting adaptions, since a novice may be surprised and confused by sudden adaptions, whereas a user with more experience of the system may take such occurrences in their stride.

Profile Data

The profile data is concerned with a user's individual preferences and past usage history – essentially, any data about the user which do not fit into the two preceding categories. It is this type of information with which we are primarily con-

cerned in this application. These data should be unambiguous in nature and reasonably straightforward to obtain, thereby minimising the risk of drawing conclusions which subsequently prove to be erroneous. At a basic level, simple user preferences need to be taken account of. Such information would be concerned mainly with presentation criteria – for example, should the system display files as labelled icons, or as lists with full or partial details about the file name, type and association.

3.7.2. Domain Model Requirements

The domain model is concerned with how and where the system fits into the 'bigger picture' – that is, how it is integrated into the environment, how it can represent itself, and how it can interact with its environment – in this case, encompassing both the computer and operating system *and* the user interacting with it. This model actually covers both design and implementation issues – for example, how the system is structured logically (as an interactive graphical window-based application) and how is this actually provided (as an adaptive interface possibly based upon an existing user interface toolkit, implemented as an event-driven graphical application).

As discussed in section 2.6.2, the domain model contains information at three different levels of abstraction: the intentional level, the conceptual level and the physical level. Each of these levels map onto a different portion of the application domain – some may be modelled explicitly, others may be represented implicitly in the final design and implementation of the system.

Information at the intentional level reflects the goals and objectives of the system's users as they work. As has been said before, it is not the goal of this study to attempt complex analysis of the user's behaviour in order to draw tenuous conclusions about what they might be trying to do. Such conclusions, made on the basis of a relatively small amount of information, would amount to little more than

guesses, and would therefore be of limited use in attempting to adapt a user interface.

The two lower levels of the model are the main concern in this study. The conceptual level contains information about components of the system that users are expected to reason about, and the physical level deals with the mechanics of the system – how a particular action can be brought about, and the effects thereof.

The conceptual level, in this case, refers to the items that were discussed earlier in section 3.2.2 – well-known items such as files, folders and applications. The items that must be modelled at the conceptual level dovetail well with the earlier discussion concerned with the user interface design – since the interface will be based on direct-manipulation, augmented by the adaptive systems working 'in the background', the conceptual model must address the mechanics of how the system can provide a visible model of its state to the user, allowing them to interact effectively with it. The use of appropriate metaphors in the design of the interface – i.e., those with which users are familiar, and those which are effective at communicating information to the user – are supported through the conceptual level of the adaptive system's domain model. The eventual application must model these items internally, in order to be able to manipulate information about them, and must reflect these models externally – via the user interface – in order for the user to be able to interact with them.

The physical level defines the 'nuts-and-bolts' mechanisms through which the interaction between system and user actually occurs. Objects at this level include things like windows, menus and buttons as 'clickable things' – graphical items on the display that suggest, through their appearance, that they can act as the targets for commands. These objects form the basic visual components of the interface and occur as part of the user's 'common sense knowledge', gained from past expe-

rience, including the use of computers. The system conforms to models of these at the program's implementation and application programming interface levels.

The physical level, in common with the conceptual level, is coded implicitly in many systems and is represented likewise in this system. To enable these models to be stored, displayed and changed would pose several significant problems in itself: how the model should be represented to the user; how the system could elicit useful information about the user (since users are not always the best source of information about themselves), and whether this would be useful or usable in any case; and at a practical level, how to implement such a radically open and reconfigurable system. Our focus in this study is at a more basic and pragmatic level, so we defer these issues to future work (see section 7.5.2).

3.7.3. Interaction Model Requirements

The interaction model's two primary constituents are the dialogue record and the interaction knowledge base, itself composed of three sets of active mechanisms. The dialogue record holds historical details about the user's interactions with the system which can then be used to make inferences about the user's behaviour. The interaction knowledge base, composed of three distinct sets of mechanisms, forms the active part of the adaptive interface. It is responsible for actually making inferences about the user, carrying out any adaption in the interface deemed necessary as a result, and monitoring the effectiveness of the interface as a whole.

Dialogue Record

The contents of the dialogue record are dependent upon the information that can be acquired concerning the interaction between the user and the system. This in turn is dependent upon two factors: the design of the user interface for this system, and the means by which it is integrated into the user's environment. In order to arrive at a reasonable design for the dialogue record, it is necessary to decide

what elements of the interaction between the user and their information will be either simply useful or absolutely required in order to provide helpful, predictable adaption. This effectively boils down to the question of what information is available for this component of the system, in terms of user interface events.

Using the functional requirements from section 3.5.2 and the resulting interface requirements from section 3.5.3, it is obvious that certain events will need to be available and will be both useful and necessary in accomplishing meaningful adaption in the interface. As the AIMS application has been cast in terms of a 'shell' application – in the same mould as the Windows 'Explorer' or Macintosh 'Finder' – events will be available that are similar to those that occur as these programs are operated.

The majority of these events are based around file management – events such as 'opening' objects by double-clicking on icons to open files with an application, rearranging files and applications within folders, and maintaining the desktop.

Interaction Knowledge Base

There are three groups of mechanisms at work in this component of the architecture: inference mechanisms are used to draw conclusions about users as they interact with the system; adaptation mechanisms reflect how the system can change itself, embodying the dynamic facets of the interface, and evaluation mechanisms allow the system to judge whether its adaptions are making it more 'usable'.

In common with the dialogue record, these components of the interaction model are closely related to the dynamic aspects of the system that will be required. They form a part of the 'feedback loop', responsible for drawing conclusions about the user as the results of observed interactions, testing these conclusions, and then changing the representation presented to the user in order to reflect the results of these conclusions. The issue about testing the conclusions drawn is important, al-

though in this application it is possibly not too much of an issue, as the conclusions will be quite simple in nature.

Inferencing mechanisms

The inferencing mechanisms are the active elements within the system which draw 'conclusions' about the user concerning their use and manipulation of objects within the system. These conclusions need to be as simple as possible for two reasons: they will be easier to draw, and they will then have clearly-defined implications for the interface (Browne, 1990). At the same time, they need to be worthwhile in that they should add value to the interface. Conclusions which would prove useful are:

(i) Files which are used quite often but which are located at a deep level within the hierarchy could be moved higher up for ease of access. The conclusion here would be that the file may be in the wrong place;

(ii) Files which are not used regularly could be marked as suitable for archival, and be moved to backup locations. Again, the conclusion here would be that the file may be in the wrong place.

These conclusions, whilst being quite basic in nature, seem to be similar to the routine management activities performed by users when managing their files, especially when the file management scenarios in sections 3.3.1 to 3.3.4 are considered. The criteria upon which they are based need to be carefully chosen – if a file is referenced a certain number of times over a given time period, this could be taken as evidence for conclusion (i) above, for example. However, care must be taken to ensure that the user retains a sense of control and awareness of the situation – where files are deemed to be in the wrong place, the mechanism used to 'move' them must not be likely to leave the user searching for a file that has mysteriously disappeared. These problems are dealt with in the next section.

Adaption mechanisms

This section deals with the mechanisms used to effect change in the user interface. Once a conclusion has been drawn by the inferencing mechanisms, it needs to be reflected in the user interface in response. These mechanisms can be considered to be working at two levels, analogously to the two lower levels of the domain model discussed in section 3.7.2 – they must take account of the conceptual objects presented to the user as the means of communication, as well as the physical objects used to actually make the necessary changes. The adaption mechanisms need to fulfil the requirements of the inference mechanisms proposed in the previous section.

File moving: Where a file is to be 'moved', the basic mechanism to do this would actually be to move the file between directories or folders (however these may be stored and represented). A file which disappears and reappears will be likely to confuse the user, so an alternative solution is proposed. In conceptual terms, rather than move the file in on operation, a 'link' or shortcut to the file could be placed in the new location, and the old reference to the file only removed when the shortcut has been use a few times – in this way, the user is tacitly acknowledging the adaption at the same time as using it. The physical operations necessary here may vary in precise detail by implementation platform, but files, directories and links are common currency amongst most graphical shells.

File tracking: As files are opened by the user (not necessarily often, but maybe a few times in a session) the system needs to maintain a set of access statistics for each file used. These can then be used by more traditional elements of the user interface to present the user's file-access history in a convenient format – filtered or ranked according to frequency or recentness of use. Again, the underlying operating system will provide the physical operations necessary to compile these statistics.

For each of these adaption mechanisms, we can see that the physical level of the domain model plays a central part in providing the basic facilities to make changes in both the interface and the user's file storage. This part of the domain model is therefore implicit in the system once it is implemented, although it is considered to be part of the model.

Evaluation mechanisms

All adaptive systems need to regulate themselves in some way, to ensure that the adaptions performed by the system are actually benefiting the interface, by making it easier to use, however that may be defined. The evaluation mechanisms of the interaction knowledge base are intended to fulfil this need. Once a conclusion has been drawn, and an adaption based on that conclusion made, it should be monitored to ensure that it remains valid.

For this application, we would like to ensure that the user is spending less time in routine managing activities, and more time in working with their information. In practice, this goal may not be directly attainable and we may need to settle for some indirect measure of increased 'usability' – perhaps based on the number of file-manipulation operations performed by the user. Two criteria that could be used are:

(i) Are files which have been moved to new locations used more than others?

(ii) Does the user appear to be able to select files with as small a set of navigations as possible?

Evidence of much 'hunting' – for example, a large number of navigation operations, opening folders and directories and closing them quickly without selecting any files – would imply that the adaption (in general) is not succeeding, and that the system's confidence in its own predictions should be reduced. In contrast,

when only those navigations required to reach files are performed, the system could have greater confidence in itself.

3.7.4. From Adaptive Elements to Reactive Interface Agents

The previous section has documented a variety of active behaviour required in the adaptive components of this system. Given the dynamics required of the interface, the next step is to arrive at a situation where they can be expressed as a set of 'reactive' processes or 'software agents'.

The mechanisms mentioned as part of the interaction knowledge base can be conceptualised as active rules or 'agents', and as such requiring implementation as active components within the interface. These agents, then, have requirements of their own – as well as implementing the behaviours discussed already, they need to be able to: gain information about the user's actions; exploit the data stored in the user model; and make their responses known to the interface.

Each of the behaviours noted in the set of mechanisms forming the interaction knowledge base could be implemented as a separate simple agent. This set of agents once integrated into the application's user interface, will then exhibit the required behaviour.

Although the simplistic behaviours of these reactive agents may not qualify for 'agenthood' under some of the stronger definitions presented in Chapter 2, the aim of this study is not solely to write agents – it is to bring together some existing tools and techniques in order to provide a tool which can aid an individual in managing their information.

The next step towards the finished application is to demonstrate how a user interface which contains embedded reactive agents may be implemented. An appreciation is required of the features of user interfacing software is required to be able to do this successfully, and this provides the starting point for the next chapter.

3.8. Conclusions

This chapter developed an abstract design for a system which will act as an example implementation of a system which embodies some of the principles noted in Chapter 2 as being important to systems which support PIM. The starting point was the notion that an adaptive interface meant to support the basic activities in PIM could be provided using a set of reactive agents as part of an adaptive interface architecture; both these concepts had been introduced and explored at a theoretical level in Chapter 2, and were made more concrete in this chapter.

The chapter proceeded with the development of a design for an adaptive interface which utilises reactive agent technology within a traditional interface. A sample 'adaptive information manager' application was introduced and used as the scope for a detailed design for a user interface which exhibits simple adaptive characteristics as well as providing support in more traditional ways. At the same time, the design activity was kept abstract as far as possible, resulting in a framework which, to as great an extent as is feasible, is language, platform and operating-system neutral. Ideas covered at a theoretical level in Chapter 2 were examined more closely in the light of the scope laid down, as the requirements became more concrete. The end result was a set of functional requirements – behaviours that the user interface will be required to respond to and exhibit – and in turn, an informal set of requirements to be satisfied by the reactive software agents that must implement them.

Chapter 4

Design and Implementation

4.1. Introduction

Using the informal abstract specification and design developed in Chapter 3, this chapter develops and implements a concrete design which takes account of a particular application, platform, environment and operating system, and implements it as a working software application. The application will demonstrate how the principles argued for in Chapter 2 can be realised in a piece of software that can provide automatic support for some simple activities encountered as part of Personal Information Management (PIM).

The previous chapter arrived at a set of informal agent requirements based upon a subset of user behaviours observed during file management activities (see the 'task specifications' in section 3.3). These are analysed in detail to yield a number of basic 'fragments' of adaptive behaviour required to implement a system which supports the tasks specified. These elements of adaptivity, in turn, allow the information requirements of the adaptive system to be ascertained – that is, the set of events and conditions that must be supplied to, and monitored by, the system

in order to have a basis on which to draw inferences. These also lead to the final requirements in terms of the support for adaption within the interface, and the means of its presentation within the interface itself.

An outline class hierarchy is developed at the start of this chapter, to serve as a template for a class of systems which satisfy the general requirements discussed in Chapter 3. This basic class hierarchy is then augmented according to the precise application required. In this case, the requirement is for a system which can monitor a user's interactions with a system of files. A behavioural specification for the system is developed which contains details about events required to be observable – opening files and folders, for example – and actions required of the interface in adaptive terms – making suggestions to the user and adding shortcuts, and ranking lists of popular files.

These compound behaviours are partitioned and analysed to yield simpler fragments of functionality which can be translated into agent specifications – the file manipulation operations such as opening and moving will trigger successively more sophisticated 'layers' of reactivity – pairs of related actions may be detected by one agent, in turn yielding events at a higher level of abstraction which could then be used as input by the next higher layer. At the top layer, events generated by the system will finally be used to trigger adaptions in the user interface.

Also under consideration will be the information required by the system to make inferences about the user's actions. There will need to be means to represent and process the user-interface events mentioned in the agent specifications. Other events, possibly generated by the agents themselves, may need to be processed. For example, information concerning the user's file manipulation actions gained from the file system user interface will need to be considered in tandem with internal events, such as the notification of inferences made by the agents. In order to

implement the system, the class hierarchy is refined further with the addition of objects to reflect such event information.

Having arrived at an abstract design for the system, the next issue to be settled is the choice of platform to be used in the implementation phase. Although the discussion in this and previous chapters has mentioned windows, desktops, user interfaces and so on, no decisions have so far been discussed about the platform to be used for the final implementation of the system. An informed choice of platform is therefore made based on several pertinent factors – the discussion encompasses technical issues both of infrastructure and implementation as well as issues concerning how the graphical user interface (GUI) to the system may be presented to the user without distracting them from their tasks. A suitable platform, together with a supporting infrastructure and integration techniques are then used to implement the system as designed.

4.2. Design Background and Informal Specification

The aim of this section is to develop the ideas introduced so far in this chapter into an abstract design for a 'class of systems' which fulfil the functional requirements arrived at in the previous chapter, without explicitly binding them to a particular language, platform or operating system.

All designs are based on a number of assumptions, and these are presented and justified in the next section. Bearing these assumptions in mind, the subsequent sections develop an architectural decomposition of the system and then express it in object-oriented (OO) terms. Finally, an OO class hierarchy and an abstract system design implemented using it are presented.

4.2.1. Design Rationale

The design work presented here is based on the OO paradigm (Coad and Yourdon, 1991; Booch, 1994) of software development. In a wider context, good software engineering practice recommends the use of some methodology for developing the specification and design of any software system (Sommerville, 1996; Pressman, 1997), so an OO approach seems justified on those grounds alone.

However, a user interface which has been even informally outlined in OO terms will benefit from it – after all, when finally implemented the user interface will be presented to the user as a set of components utilising aggregation and inheritance characteristics (albeit subconsciously, from the user's point of view). In addition, an interface which contains a set of co-operating interface agents, deemed to share certain characteristics – responding to user events and triggering interface adaption, for example – shows features which are clearly OO in nature.

As well as having benefits for the implementation of the system itself, the realisation of the user interface will be eased by using OO techniques. Using an OO approach can insulate the implementation (as far as possible) from the specific details of the interface environment, and result in a more general system. In fact, the majority of interface implementation toolkits use some measure of object-orientation as a matter of course, and many are entirely OO in nature.

For example, Microsoft Windows uses the Microsoft Foundation Class (MFC) hierarchy to implement GUI applications (Kruglinski, 1997), the X Window System uses the OO Xt/Xlib interfaces (Nye, 1992; Nye and O'Reilly, 1993) and OSF/Motif (Heller and Ferguson, 1994; Ferguson, 1993) user interface toolkits, and Sun's Java language (Gosling and McGilton, 1995) has an Abstract Windowing Toolkit (AWT) based on object-orientation. The use of OO software therefore appears to be a widely-accepted technique in the implementation of user interface toolkits and related user interface support.

A related design and implementation technique, especially relevant for implementing software which has particular integration requirements, is interface-based development (Box, 1998a). This approach relies on the total separation of an object's external appearance – its interface(s) – and the internal details used to actually implement the object. This technique reduces the interdependencies between elements of a software system, reducing the behavioural complexity of interactions within and between objects and is discussed more fully in section 4.4.11.

At a more general level, another of the aims of this work is to arrive at some guidelines for the development of systems such as these. Even if the software itself is not re-used, hopefully these techniques, as by-products of the design and development process, can be.

Having given the background and rationale for the design work presented in this chapter, the discussion now turns to the requirements of the system to be implemented. The following subsections examine the adaptive behaviour required of the system and develop informal specifications for two active support mechanisms: short-cut suggestions and file 'hot-list' maintenance. As discussed in section 3.5.2, the adaptive mechanisms are complemented by some passive support, and informal specifications are developed for two passive support mechanisms: file annotation, and annotation-based retrieval. These informal specifications, in conjunction with issues concerned with the process of integrating the final application with an existing graphical shell, then lead on to a discussion of the interfacing techniques to be used.

4.2.2. Behavioural Requirements and Analysis

This section re-examines the adaptive behaviour required of the system as described in section 3.7, in order to arrive at a set of 'atomic' behaviours, directly implementable as simple reactive software agents. The adaptive behaviour desired of the system, arrived at in section 3.7.3, is as follows:

(i) To monitor navigation behaviour and suggest optimisations for filesystem access based on these observations – for example, multiple similar navigate-selections (that is, repetitive navigating through several levels of folders in order to open a single file) could imply that a shortcut is needed;

(ii) To maintain a 'hot-list' of often-used objects (files or folders), which can be accessed by the user with minimal navigation, and which can act as a reminder of recent tasks;

The active elements of the adaptive system are to be cast in terms of 'agents' as part of the Adaptive Interface Technology (AIT) architecture (Benyon and Murray, 1993, see section 2.6), and will form part of the interaction knowledge base – specifically, they will implement the inference, adaptation and evaluation mechanisms. These behaviours are now examined in more detail to provide a behavioural specification for the set of agents required.

To implement the two behaviours described above, certain features will be required. At a basic level, different types of user event information must be available to the system – files being opened, folders being navigated through. The user's responses to suggestions also need to be acquired.

In addition to these events triggered directly by the user, the system may trigger events which signify the results of inferences made about the user (rather than just direct observation). Self-evaluation of the system's effectiveness – whether the user is taking advantage of the adaptive components of the system or not – might result in the triggering of further events causing re-examination of the system's state, and further adaptation based upon it.

4.2.3. Short-Cut Suggestions

As the user works with their files and folders, the system will be monitoring their actions. The desired goal is to identify which particular files may be poorly

placed, and offer suggestions to make them more easily accessible. In this situation, a file might be considered 'poorly-placed' (as the task specification given in section 3.3.1 illustrates) if:

(i) It requires navigation through two or more folders to locate and open it; and

(ii) It is opened more than once in a short period of time.

Informally, the system needs to be able to spot a compound 'navigation-selection' action. If the user opens a folder, a subfolder within it, and a file within that, the system should make a note of this access. If the same access is repeated within a short while, the particular file might benefit from having a short-cut to it.

There are two main parameters to consider, relating to timing the individual events which make up the compound access. For the purposes of the prototype, opening a folder, followed by opening a subfolder within less than ten seconds (for argument's sake) might be deemed to be a connected pair of navigation events.

4.2.4. Hot-List Maintenance

As with the short-cut suggestion facility, as the user works with their files and folders, the system will be monitoring their actions. The desired goal is to record accesses to files in order that the user may have a dated record of all their file accesses, which can be interrogated at any time.

This is an example of ongoing adaptation and evaluation – as the user works, the logical state of the hotlist will be continuously monitored and updated. Again, this process has several parameters which control how the final result may appear, although the information that needs to be recorded is reasonably straightforward. Each physical file that has been accessed by the user is recorded, along with the last access time and the number of times it has been accessed. These items of in-

formation will allow the user to search for recently-accessed files ranked by name, last access, total number of accesses, and frequency of access.

4.2.5. Passive Support

The task specifications discussed earlier (see sections 3.3.3 and 3.3.4) indicated that some passive support can helpfully complement active support in aiding PIM activities. One facility which recommended itself was to provide a way for users to add annotations to filesystem objects – for example, allowing information about a given document to be stored, associated closely with the document, but not actually as part of the document's contents.

Although no explicit adaptivity is required to provide these facilities, these mechanisms do require that the eventual system is closely integrated with the target platform's existing graphical shell system. The succeeding sections examine how these might be provided and the additional impact they have on the implementation requirements.

4.2.6. File Annotations

Many applications exist which provide note-pad like functionality – for example, Microsoft's commercial 'Outlook' organiser application allows users to stick coloured 'notes' on a 'board' (basically, icons within a window) and search them. A number of shareware 'notes' applications also exist, which allow users to place 'notes' windows, which remain above others, on the desktop. However, this ignores one of the basic qualities of a sticky note. They can not only be used for short-term repositories of small fragments of information, but they are also extremely useful for *contextual* annotations – small pieces of information placed in a particular location. This attribute of sticky-notes seems to be important, so the objective in this part of the study is to provide a parallel ability – notes that can be

attached to objects such as applications or files, providing the ability to annotate them.

The underlying idea of this mechanism is therefore very simple – to allow a user to 'stick' a text note on a file. In providing this ability, however, several issues need to be considered. The purely technical requirement is quite simple – the system will need to store a list of one-to-one associations, holding the names of files that have been annotated, linked to text objects which store their annotations.

The more profound problems revolve around user-interfacing mechanisms which are required to represent this facility to the user and to allow them to use it. Users need to be able to add notes to filesystem objects, to be able to view them, and to be able to know that a particular object has an annotation. Although these issues will have technical ramifications as far as the software implementation is concerned, they will be noted here as requiring special consideration when the user interface is designed.

4.2.7. Annotation-Based Retrieval

This feature builds on the file-annotation support described in the previous section. The objective is to provide the ability for a user to locate and retrieve a file based on the contents of its annotation. The information entered into these notes will probably be stored by some centralised application, even though they will primarily be accessed via the annotated object. This would provide a natural 'associative searching' mechanism, where users could ask to open an object based on the contents of its annotation.

Given that the underlying information store used to implement the mechanism described in the preceding section – that is, a list of files and associated annotations – the purely technical issues involved in providing this facility are relatively straightforward. Given a word or phrase, the list of text notes needs to be

searched for those containing occurrences of it. If it is found in a given note, a means should be provided to allow the associated file to be opened.

The integration and user interfacing requirements for this feature should be slightly less demanding than for the representation and manipulation of the notes themselves – after all, the idea of 'searching' for something using a computer is a concept with which almost all users will have had some contact. An suitable solution will probably be an interface element (such as a dialog) which accepts a word or phrase and responds with a list of matching files, one of which the user can choose to open.

4.2.8. Integration Concerns

The previous two sections show that the final implementation of the system will need to be quite closely integrated with the target platform and its user interfacing environment. Assuming that the target platform will be based around some kind of graphical shell – insulating the users from the underlying representations of files, folders, applications and suchlike – there will be a need to communicate closely with this shell application.

The target platform will therefore need to provide some kind of 'extension' mechanism, to allow external applications to work alongside the existing user interface, augmenting it with the additional features discussed in sections 4.2.6 up to this point. As well as this, suitable mechanisms will need to be identified within that environment to allow notes to be created, added, viewed and searched in a predictable manner, in such a way as to be consistent with the platform's interfacing conventions.

4.3. User Interfacing: Practical Issues

Up to this point, the discussion has centred upon the 'interface' only in a reasonably informal state – in order to be implemented, it must obviously be expressed as software. This section briefly describes general issues in user-interfacing techniques and theory, and then concentrates on the particular additional requirements of adaptive systems.

Interactive software, rather than non-interactive software, has an additional set of requirements, resulting in the need for a slightly different approach to the design and implementation of applications with GUIs. Interactive applications developed obviously need to present their interfaces graphically, and must also be able to respond to user events, rather than explicitly requesting information or choices from their users. GUI toolkits, such as OSF/Motif (Heller and Ferguson, 1994; Ferguson, 1993), and integrated development environments (Kruglinski, 1997) exist which relieve some of the burden of providing graphical interfaces and dealing with user input.

A GUI can be realised within an application using a programmatic toolkit approach or by using a some kind of visual development tool. However, adaptive interfaces have slightly more complex requirements than simple interactive systems, in that they may need to be 'reflective' – that is, they need to be able to manipulate *themselves*. The next section addresses this issue.

4.3.1. Software Environments for Adaptive Interfaces

This section examines the problems faced by adaptive interfaces with respect to self-modification. All GUIs exist as graphical representations of the application's system image, so that the user can interact with the application. However, an adaptive system may need to be able to change the GUI. This therefore requires that the model be made explicit in software *internal to the system*, so that it may be

manipulated either by the user (as interface designer) or by the system itself (to make adaptions, as required by the system under development). Although the physical and conceptual levels of the domain model reflect the design of the application's GUI, these models are often coded implicitly within the program, and cannot be changed.

Different possibilities exist for providing these 'reflective' interfaces, such as high-level user-interface toolkits and user-interface management systems (UIMS). These are both systems which either contain or manipulate explicit representations of user interfaces. This allows programs to enquire about the form and capabilities of their user interfaces, which is be required for programs that need to be able to change aspects of the interface as they run.

A high-level user interface toolkit (Myers, 1993) provides re-usable components which can be refined as necessary and integrated into larger systems, while at the same time remaining modifiable. An approach such as this naturally promotes the use of techniques concerning OO specification and development of software and is useful in terms of abstract design, to encapsulate functionality not of concern within components that can then be used to implement the interface desired.

A UIMS (Olsen, 1992) is essentially a GUI-based application which can be used for building GUI-based applications, and may also provide associated runtime support for programmatic manipulation of a user interface. Most modern software development environments provide some kind of UIMS, enabling developers to specify the appearance and layout of a GUI's elements visually, and then add program code to it later.

Many development environments such as Visual C++ (Kruglinski, 1997) actually provide a mix of the two techniques discussed here – a UIMS is provided to allow visual specification of an interface, and, once running, the software uses a high-level toolkit to access features of the user interface. This is the approach that will

be followed in this study, as a UIMS provides quite a rapid means to develop prototype software, yet the environment does not sacrifice the generality of a toolkit programming interface.

4.4. Detailed Design

This section presents the final design for the prototype AIMS application. This design encompasses two main elements: the design for the system, using OO and interface-based approaches for modularity and extensibility, without considering the target platform as far as possible; and the integration techniques to be used to present the visible parts of the system to the user. These two elements will then be brought together to yield the final system which will then be evaluated in Chapter 5.

Firstly an OO architectural decomposition will be presented, identifying the main components of the system and the object types which will be necessary to reflect the important parts of the system. An interface-based communications architecture will then be developed, which shows how these different objects can 'talk' to each other, invoking methods and so on. These two complementary models are then used to arrive at a detailed design for the AIMS application's objects and their interfaces.

4.4.1. Architectural Decomposition

This section builds on the functional specifications derived in sections 4.2 to 4.2.7, to yield a high-level architecture for the system. This will illustrate how it will be integrated with existing systems and how the major components of the system will be structured.

The first step is to consider the environment into which the AIMS application must be integrated, and the means necessary to isolate the application (as far as

possible) from that environment. Figure 4.1 shows the context within which the AIMS 'application' will have to function.

The overall architecture is presented at three levels: the application level; the portability level; and the environment level. This will enable the application's internals to be designed without needing detailed knowledge of the final implementation platform. The portability components will encapsulate these details, exploiting features of the environment in order to present a platform-independent interface to the application.

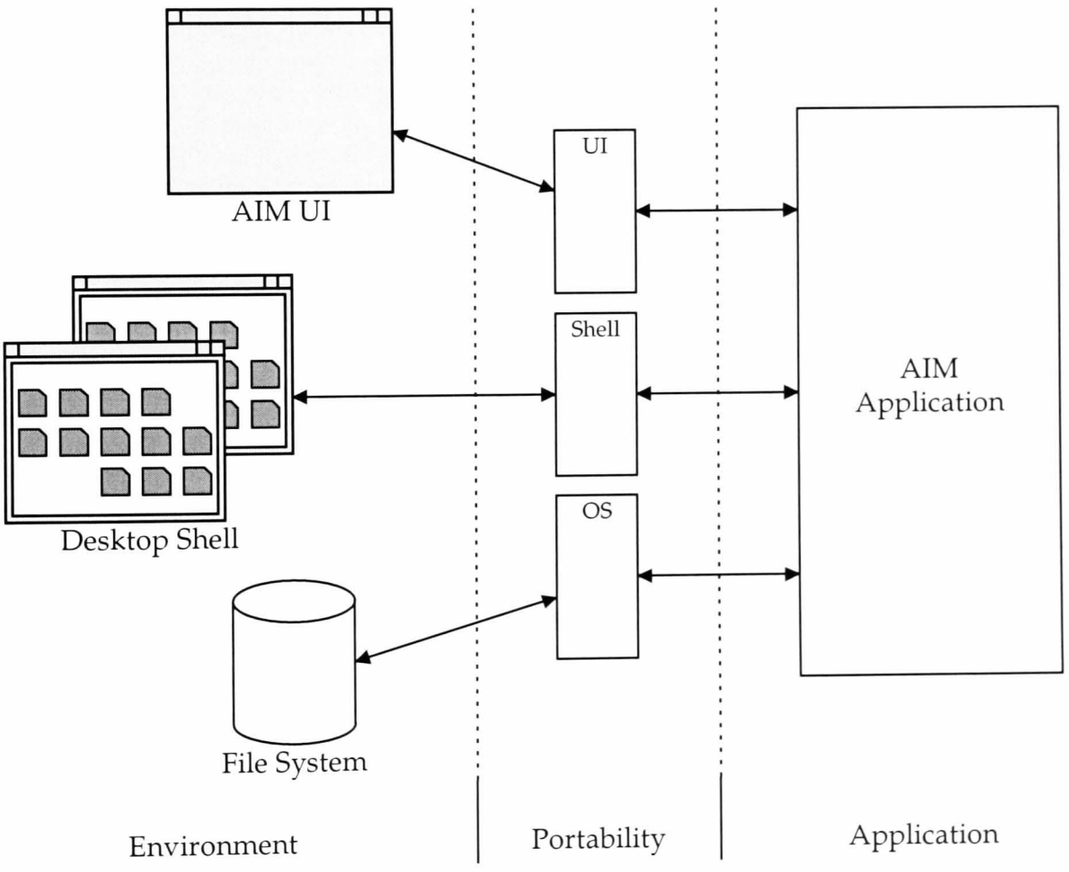

Figure 4.1. Architectural overview for the AIMS application.

An important feature to notice is that the final application will need to respond to events in its own user interface – so the user may tailor the operation of the AIMS application if desired and respond to questions posed by it – while at the same time it must respond to events in the 'shell' application of the operating system. This would appear to imply that the final system may need to exhibit a degree of

parallelism, since the system is real-time in nature. Excessive delays in responding to the user's actions are highly undesirable, so implementation techniques which can alleviate this problem should be taken into account.

The next step is to consider how the various elements of the Adaptive Interface Technology (AIT) architecture will be reflected in the design of this system. To do this, the requirements arrived at in this chapter will be re-examined to yield a concrete set of agents, events and modelling resources which satisfy PIM support system design criteria, as discussed in the previous chapters.

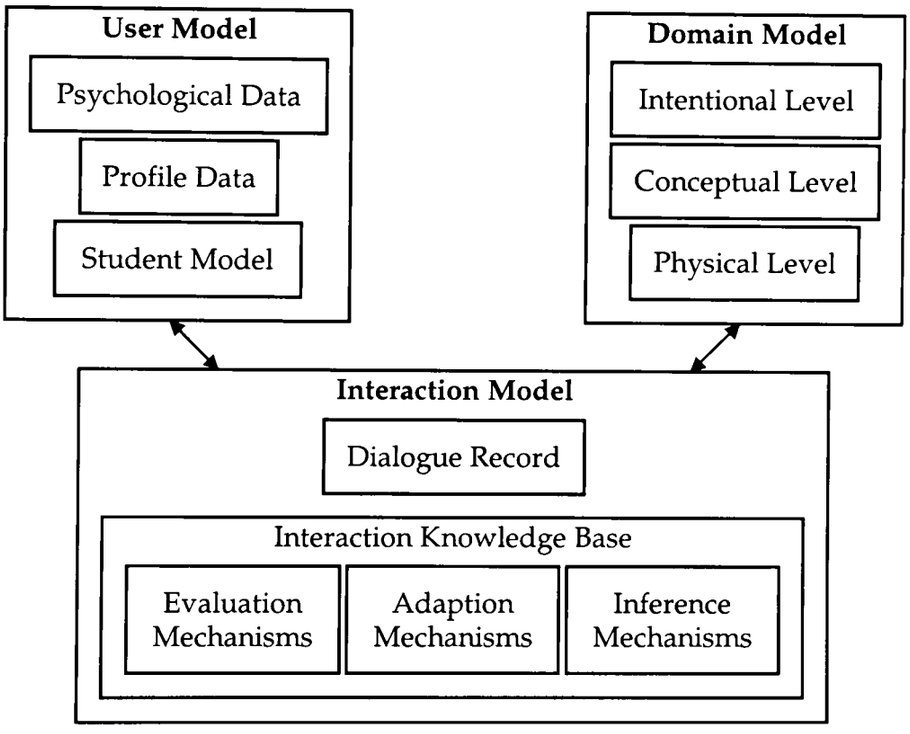

Figure 4.2. A reference architecture for adaptive interface technology (Benyon and Murray, 1993).

This is done by using the AIT architecture as a template for the system design, implementing it as an aggregate component. To reprise some of the earlier description of the architecture (shown as Figure 4.2, originally as Figure 2.1 in section 2.6), the main elements of interest are: the Domain Model, describing the system's application; the User Model, containing data concerned with user preferences and the user's profile; the Dialogue Record, giving a historical log of events that have

occurred; and the Interaction Knowledge Base, comprising mechanisms for inference, adaption and evaluation.

As mentioned in section 3.7.2, some of these elements are to be implemented in an implicit manner – in particular, the Domain Model will be expressed mainly in terms of the definitions of internal data structures. The Interaction Knowledge Base is actually implemented as a set of agents – each of which fall into one of the categories of inference, adaptation and evaluation – which respond to events and, using information stored in the dialogue record and the user model, can generate internal events reflecting inferences drawn or adaptions to be made. Each of these elements will now be examined in detail, with the accompanying design documentation necessary for the implementation of each.

4.4.2. Domain Model

The domain modelling necessary for this study occurs at the conceptual and physical levels (the intentional level being too abstract for consideration in this rather general application).

Entities which must be reflected at the conceptual level are those manipulated by the user – files, directories, applications, and so on. These will be implicitly modelled in the data structures used by the system to store and process information about the user's actions.

Also at the conceptual level, we encounter those 'measurements' about the user that the system must store – effectively, the template for the design of the user model. If something is *not* incorporated within the domain model, the system cannot 'know' anything about it. Although the 'modelling', such as it is, is generally limited to acknowledging that something exists and we want to store information about it, this forms the basis for the following specific attributes which appear in the user model:

(i) *'Hot' files*: files which the user tends to use more often than others;

(ii) *'Shortcut' files*: files which have shortcuts to them;

(iii) *Notes*: textual annotations applied to objects;

Physical level entities are largely concerned with the interface to the external operating system. These are reflected implicitly, in the program's run-time library bindings and the binary layout of the program itself.

4.4.3. User Model

The user model holds information about the individual using the system. Three sorts of data may be considered: psychological data; the student model; and profile data (as described in sections 2.6.1 to 2.6.3). Information associated with psychological data and the student model do not play a role in this study, as we seek to support the lower levels of user behaviour, rather than modelling the user's abilities or knowledge.

4.4.4. Profile Data

The profile data stored in the user model reflects personal information about an individual user. It stores details of accessed files – a list of files which the user has opened, containing the files' names, frequency and time of last use. The profile also stores information about the list of shortcut files, recording the names of referring and referenced files – this is the list of files which the system has proposed shortcuts for, and the names of the shortcuts which refer to them. Finally, the profile contains data regarding the objects (files, directories or applications) which have had annotations applied to them, stored as a list of associations between an annotated item and the textual annotation applied to it.

4.4.5. Interaction Knowledge Base

The interaction knowledge base provides the 'active' elements of the adaptive system, responding to the event information as it is entered in the dialogue record, producing corresponding changes in the interface and monitoring any changes made. Each of the following areas (inferencing, adapting and evaluating) requires sensing and reactive behaviour of a sort, and the final set of agents which will be needed by the implementation is governed by the adaptive behaviour required.

Inferencing mechanisms respond to combinations of events observed, usually making reference to 'historical' data in the dialogue record, to draw conclusions about the user based on their behaviour. These are then acted upon by the adaption mechanisms, bringing about some change in the user interface. Evaluation mechanisms could monitor shortcut and hotlist usage in relation to direct use of the underlying objects, and if a significant ratio, of references disregard the shortcut/hotlist entry, suggest that the shortcut/hotlist entry be removed.

4.4.6. Detailed Agent Design: Short-Cut Suggestions

The subsequent sections present descriptions of the processes behind each of the behavioural elements described above. They show how simple events are used in combination to draw simple general conclusions, which are in turn used to form more complex, specific conclusions.

One way of realising the adaptive short-cut suggestion behaviour is to partition the task into three levels of reactive sensing, allied with the information contained in the dialogue record. This partitioning is shown in Figure 4.3.

At the lowest level, two agents are responsible for detecting relationships between simple events. One examines the stream of events as they arrive for a *folder-opened* event followed by a *file-opened* event, generating a *navigate-open(1)* event – where the parameter 1 indicates the number of directories referred to in the event – for

any pair where the file is contained within the folder. The other searches for connected pairs of *folder-opened* events, where one folder is directly contained within another, resulting in a *navigate-folders* event. The parent-child relationships are detected by comparing the file and folder names, to show whether a given file or folder object is contained within another given folder.

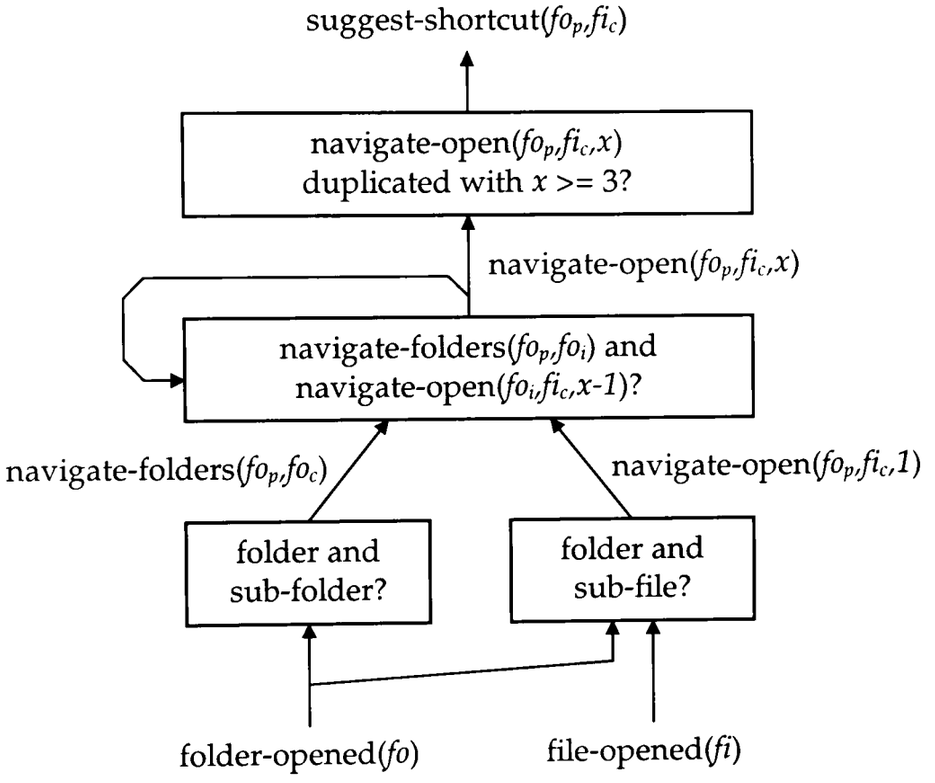

Figure 4.3. Three-level architecture for shortcut detection.

At the next level up, a single agent examines incoming *navigate-folders* and *navigate-open(x-1)* events for any pair connected by a parent-child relationship – specifically, where the subfolder of the *navigate-folders* event is the parent folder of the *navigate-open(x-1)* event. If a connected parent-child pair is spotted, a compound *navigate-open(x)* event is generated, with an incremented directory level count to denote the additional connection.

At the top level, a single agent checks any *navigate-open(x)* event with a level count of three or more (denoting two folders opened, followed by a subfile) to find if the

same event has occurred previously. If so, a *suggest-shortcut* event is generated which will be handled by the OS-specific part of the application.

The short-cut suggestion mechanisms are subject to several parameters governing the detection of duplicate *navigate-open* sequences. The two most obvious are the time periods permissible between connected events. Firstly, elapsed time must be considered when looking at a *folder-opened* event and a *file-opened* event referring to an object in that folder – if the time between the two is too great, they should not be accepted as a connected pair. Similarly, the time elapsed between two duplicate *navigate-open(x>3)* sequences must be considered when deciding whether to suggest a short-cut.

For the prototype system to be developed, these timing periods are as follows. For a *folder-opened* event and a *file-opened* event to be cause the generation of a *navigate-open(1)* event, the time between the folder being opened and the sub-file being opened must be 10 seconds or less. For two duplicate *navigate-open(x>3)* sequences to cause the generation of a *suggest-shortcut* event, the time between the two duplicates must be 10 minutes or less.

It should be restated that the selection of the parameter values in these specifications are in fact somewhat arbitrary in nature. The exact values of the parameters are not really the prime consideration – the goal is to provide a system which exhibits good face validity as the basis for the later practical testing in Chapter 5. As long as the system supports the elements of the PIM tasks identified earlier in sections 3.3.1 to 3.3.4, this goal should be satisfied.

4.4.7. Dialogue Record

The system's dialogue record stores a journal-type log of events which may occur due to the user's actions, or may be produced internally in response to certain combinations of actions. The event log is therefore a timed list of input events of

interest. These input events are used by the system to observe the user's behaviour and form the basis for inferences drawn about adaptions to be made. Events are listed at two levels, syntactic and semantic, as defined in section 3.6, concerning abstraction levels within the AIT architecture.

Syntactic events are at a low level of abstraction, and reflect basic occurrences within the interface. These events are independent, and do not take into account any context other than the objects to which they refer. Semantic events, to a certain extent, take context into account by reflecting relationships *between* syntactic or other semantic events, and are generated by the system (refer to section 3.7 on the syntactic/semantic signals of the AIT system diagram). The syntactic events fall into several categories – however, in all cases, the generation of these events is triggered directly by a single occurrence outside the scope of the application:

(i) *User action events:* these are the important basic events, which reflect the user's interaction with the system's desktop interface – opening files and folders and actively using the system;

(ii) *Database maintenance events:* if a file or folder is either renamed or deleted, and is referred to in any other part of the user model, any references to it must be either updated or deleted in order to maintain a consistent profile database;

(iii) *Session management events:* these take account of when sessions are started and stopped, thereby enabling more accurate timekeeping for access-frequency calculations.

The semantic events within the system are all concerned with connected folder-folder and folder-file access – the important difference to note is that they are generated by the system itself, in response to sets of connected events separated by intervals of time.

Event Name	Event Description
file-opened (file)	triggered when the user double-clicks a datafile *file*, causing the associated application to run, opening the file
folder-opened (folder)	triggered when the user double-clicks a folder *folder*, opening it to explore its contents
object-renamed (object, object')	triggered when the user changes the name of a file or folder from *object* to *object'*
object-deleted (object)	triggered when the user deletes the file or folder *object*
session-starts	triggered when a user logs on to the system and begins interacting with it
session-ends	triggered when the user logs out of the current session

Table 4.1. Syntactic Events Processed by the System

Tables 4.1 and 4.2, respectively, show the different syntactic and semantic event types processed and generated by the system. These tables show the event names and also, where appropriate, the attributes of each of the events. For example, the *file-opened* event has a single attribute, the name of the file that was opened.

Event Name	Event Description
navigate-folders ($folder_{parent}$, $folder_{child}$)	triggered when a folder $folder_{child}$ is opened which is a subfolder of a folder $folder_{parent}$ opened within the last ten seconds
navigate-open ($folder_{parent}$, $file_{child}$, 1)	triggered when a file $file_{child}$ is opened within a folder $folder_{parent}$ which was opened within the last ten seconds
navigate-open ($folder_{parent}$, $file_{child}$, x)	triggered when a *navigate-folders* (f_p, f_i) and *navigate-open* (f_i, f_c, *x-1*) pair is observed (that is, at least two folder levels are navigated followed by a file-open operation)
suggest-shortcut ($folder_{location}$, $file_{target}$)	triggered when a pair of identical *navigate-open* (f_l, f_t, *x>3*) events are detected within ten minutes

Table 4.2. Semantic Events Generated and Processed by the System

As a rule, an action in the user interface causes a single semantic event, of the associated type. One exception to this rule is that a folder opened as a result of selecting it from the desktop window causes a pair of *folder-open* events to be gener-

ated, one for the desktop location and one for the sub-folder, as is required to indicate the source of the desktop folder. The process is illustrated in Table 4.3.

Event Description *Internal Detail*	Event
1. User opens folder 'Personal' from desktop 'window'	
folder-opened ("\")	E1
folder-opened ("\Personal")	E2
System recognises folder-subfolder relationships (E1, E2)	
navigate-folders ("\", "\Personal")	E3
2. User opens folder 'Documents'	
folder-opened ("\Personal\Documents")	E4
System recognises folder-subfolder relationships (E2, E4)	
navigate-folders ("\Personal", "\Personal\Documents")	E5
3. User opens file 'Diary.DOC'	
file-opened ("\Personal\Documents\Diary.DOC")	E6
System recognises folder-subfile relationship (E4, E6)	
navigate-open ("\Personal\Documents", "\Personal\Documents\Diary.DOC", 1)	E7
System recognises earlier folder navigation as parent of event (E5, E7)	
navigate-open ("\Personal", "\Personal\Documents\Diary.DOC", 2)	E8
System recognises earlier folder navigation as parent of event (E3, E8)	
navigate-open ("\", "\Personal\Documents\Diary.DOC", 3)	E9

Table 4.3. Detecting navigation-selection sequences based on user events.

Event E9 [*navigate-open ("\", "\Personal\Documents\Diary.DOC", 3)*] shows that the system has spotted that the user started at the desktop and navigated through two levels of folders before opening their file. If an identical event to E9 is detected within ten minutes, a *suggest-shortcut* event is generated, which instructs the system to offer to create a shortcut between the parent folder and the file opened.

Implicit within this description is the fact that each event's timestamp is considered when deciding whether or not to generate any *navigate-xxx* event. For example, in order for the event E3 in Table 4.3 to be generated, there must be no more than ten seconds between the preceding E1 and E2 events. This should ensure that only connected folder navigation events trigger shortcut suggestions.

The structure of each dialogue record entry is that shown in Table 4.4. The *time* attribute field holds a timestamp denoting when the event was generated. The data is held as a *'time_t'*, which is actually a 32-bit signed integer, defined as the number of seconds before the event since January 1st, 1970 (a *de facto* standard introduced in the K&R Unix/C standard library in the late seventies). The *code* field, which will be implemented as an integer, stores one of a small set of values to indicate which event type the particular event relates to. The set of code values to be defined is shown in Table 4.5.

Attribute	Type	Description
time	timestamp	records the event's generation time, in standard *time_t* format
code	eventcode	records a code denoting the event's type
name1	filename	records the first filename associated with the event
name2	filename	records the second filename associated with the event
level	integer	records the number of folder levels between *name1* and *name2*

Table 4.4. Dialogue record per-event information.

code	*name1*	*name2*	*level*
SESSION_STARTS	-	-	-
SESSION_ENDS	-	-	-
FILE_OPENED	name of file opened	-	-
FOLDER_OPENED	name of folder opened	-	-
NAVIGATE_FOLDERS	parent folder name	child folder name	-
NAVIGATE_OPEN	parent folder name	child file name	number of folder levels between parent and child
SUGGEST_SHORTCUT	parent folder name	target file name	

Table 4.5. Symbolic event type code list and attribute usage.

The *name1* and *name2* fields are used to store the names of the file and folder objects referred to by the event. Table 4.5 also shows which event types use them, and for what purpose. The *level* field is used only by the *navigate-open* events,

when it gives the number of objects (folders and files) used in a particular compound navigation/selection sequence.

This concludes the 'first cut' of the abstract design for the separate components of the theoretical AIT architecture. The next step in the development process is to bring the various components together in a system outline which will integrate with a user environment.

4.4.8. Structural Design

Given these contextual requirements, a proposal for the structure of the AIMS application itself is now presented. Figure 4.4 gives an outline for the AIMS application's internal structure – it consists of a set of external communication channels between the user interface, the shell and the operating system, and a set of agents which are supplied with events. These events may be generated externally, as a result of user actions, or may be generated by other agents within the application, allowing the functionality or outputs of lower-level agents to be utilised by others.

Certain entities in the application may need to operate as separate sub-processes or 'threads' in their own right – these are denoted by the thicker lines, being the UI and Shell ports, the Event Handler, and the set of agents themselves. Threads (concurrent paths of execution within a single process, sharing the application's data) are available on most current implementation platforms, and can also be emulated, should the need arise. Alternately, it may be that true concurrency will not be required, in which case it will be prudent to use a single thread of control, which is always far easier to reason about and design with.

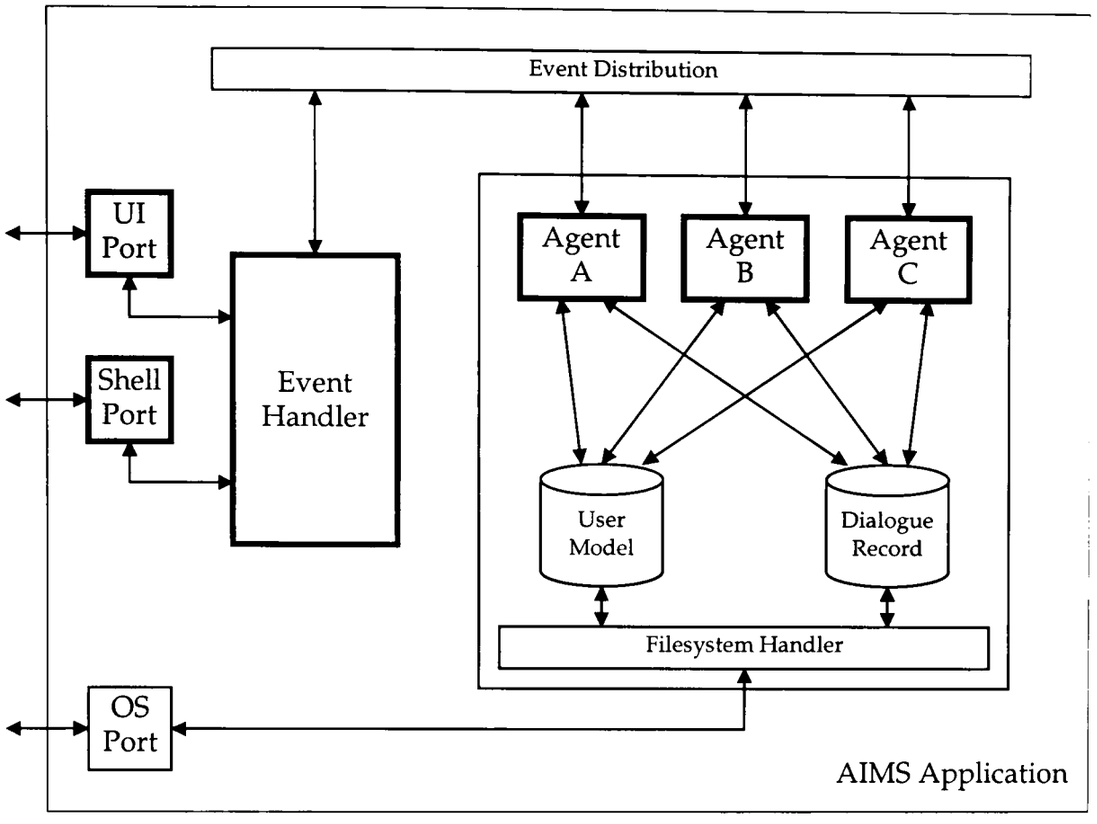

Figure 4.4. Outline structure for the AIMS application.

4.4.9. Relationship with the AIT architecture

The architecture for adaptive interfaces, described earlier in section 2.6, is echoed mainly in the shaded part of Figure 4.4. The user model (section 2.6.1) and dialogue record (section 2.6.3) components will be implemented explicitly as data stores of some kind. These will then form the resources used by the various agents to draw their conclusions about the user.

The domain model (section 2.6.2) and the remaining components in the interaction knowledge base (section 2.6.3), in this system as in almost all others, will be implemented in quite an implicit manner. The domain model's physical level will be embodied in the platform-specific portability layer, mapping between AIMS application primitives and the external environment (user interfaces, shell applications and the operating system). At a conceptual level, the domain model will be implicit in the design of the AIMS user interface and the representation of files and

folders in the augmented graphical shell. The dynamic behaviour of the various interaction knowledge base mechanisms – for inferencing, adaption and evaluation purposes – is provided for by conceptualising the system as containing agent processes which execute autonomously.

The 'implicitness' of the domain model and parts of the interaction knowledge base is really a symptom of a wider issue which can cause problems in adaptive systems design – that is, at what level of detail to model the environment, and whether or not to provide an explicit domain model. To the extent an implementation of an adaptive system must be realised in software, a domain model must exist. However, this is normally represented only in the design and coding of the final program.

It is theoretically possible to implement modelling software which would allow a domain model to be held explicitly and manipulated by the system, although this introduces considerable complexity – effectively, it requires 'meta-modelling' support. A domain model defines the extent of a system's relationship with some environment – consisting both of concrete details of software implementation, and more abstract, elusive concepts of user knowledge. A fully explicit domain model would require that the eventual software possess the ability to model any environment that could be realised, not only at the software level, but at the conceptual level as well. This provides some interesting research questions, but as it is not a fundamental issue in this study, work on it is deferred to the future (see section 7.5.2).

4.4.10. Object-Oriented Decomposition

This section illustrates how a user interface conceptualised as discussed in Chapter 2 can be expressed in OO design terminology. The adaptive interface, containing simple reactive software agents, will need to be expressed as a set of objects. The

set of objects will be instantiated from a hierarchy of classes which can be refined and augmented as dictated by the precise application.

Using the elements of adaptive behaviour identified earlier in sections 4.2 to 4.2.7, we can then develop general specifications for agents meant to respond to events in user interfaces. This work will illustrate the infrastructure requirements for such a system, especially given the event-driven nature of the interfaces involved, the requirement to transform interaction information (syntactic data) from the user interface into meaningful events (semantic information) and the integration requirements – that is, how the system 'links in' with existing software.

Referring back to Figure 4.4, the internal structure of the AIMS application has already been expressed (albeit informally) as consisting of a set of component objects. Immediately, we can see two possible classes of object – the 'Port' objects, used to communicate with the external environment, and the 'Agent' objects. Although in Figure 4.4, no descriptive names or particular functionality were ascribed to the set of agents shown, it is obvious from the figure that they must share some common characteristics – the receipt, processing and distribution of events and the manipulation and maintenance of the user model and dialogue record.

Looking at the full range of components present in the internal structure leads to the following list of candidate classes, as described here:

(i) *Port:* A link with the external environment, which acts as a source and sink for events from or to a particular source;

(ii) *Event Handler:* Responsible for accepting events from ports, distributing them to agents and communicating the responses back to the requisite port(s);

(iii) *Event:* Stores information about events;

(iv) *Agent:* An autonomous process with internal state which processes events using information from the user model and the dialogue record;

(v) *User Model:* A persistent resource which stores profile information about a particular user;

(vi) *Dialogue Record:* A persistent resource which stores historical data about the interactions between the user and the system;

(vii) *AIMS Application:* The application as a whole, represented as an object.

This initial list can be further refined, by noting some facts about these candidate object classes and relationships amongst them.

A *Port* may be synchronous or asynchronous in nature – for example, a port which receives event data from the user interface or graphical shell will be asynchronous – that is, the timing of incoming data cannot be predicted. In contrast, a port to the file system will be synchronous – a request will be followed by a response almost immediately. Thus, there could be two subclasses of *Port*, one which is used for asynchronous data and one for synchronous data. Implementations of the two might differ in that an asynchronous port might need its own thread of execution, whereas the messages passed through a synchronous port could simply take the form of method invocation.

Another relationship exists between the *User Model*, *Dialogue Record* and *Agent* candidate classes. Both the *User Model* and *Dialogue Record* objects must be persistent in nature – that is, the information stored within them should persist between sessions and not be lost when the system is shut down. *Agent* objects may similarly require their state to be persistent. A *Persistent Object* class could therefore be the superclass of these three candidate classes. Instances of this class would possess utility methods for 'suspending' the instance and placing its contents in secondary storage, then reactivating the instance at a later date.

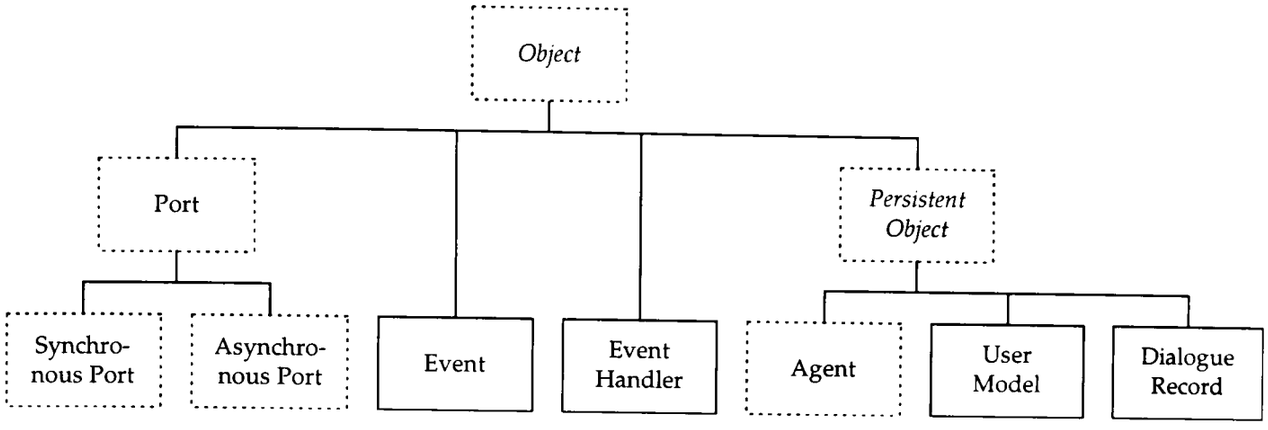

Figure 4.5. Basic adaptive interface class hierarchy.

Bringing all this together yields the class hierarchy shown in Figure 4.5. Classes with dotted outlines in Figure 4.5 are 'abstract' classes – that is, in the class 'library' which will result from this development, these classes will exist but will require further refinement in order to be fully implemented. For example, there would be little point in instantiating an *Agent* without implementing its internal state and dynamic behaviour. In most OO notations, this can be explicitly enforced by the programmer.

4.4.11. Interface-Based Architecture

The class hierarchy developed in the previous sections shows what component objects the system consists of, but does not show how these components communicate to perform the required functions. This section develops an interface-based architecture which shows how the system's constituent objects are connected together.

The term *interface*, in this context, refers to contracts of behaviour and functionality between software entities, rather than between humans and computers. An interface is a single-minded, cohesive set of methods, implemented by a particular object, which can then be invoked by another object. Box (1998a) defines interface-based software development as representing the 'second wave' of OO software

development. The first wave is typified by classical 'implementation inheritance' (Booch, 1994) in an OO system.

This style of software re-use, where objects inherit both their 'appearance' (i.e., their interfaces) *and* their functionality (or implementation) from others, often leads to so-called *white-box* re-use, where excessive coupling develops between a base class and a derived subclass. A related issue is the *fragile superclass* problem, where a change in the implementation of a superclass (although it may not alter any externally-visible details) may break existing binary client applications (see Box (1998b) for a more detailed exposition). Interface inheritance seeks to address these problems by considering the implementation of an object as a black box, accessible only through predefined set of interfaces, betraying no detail whatsoever about the object's inner construction.

As well as providing an implementation that has a minimum of coupling between its component parts, another benefit that can be gained from following an interface-based development approach is that the resulting software could then be packaged as *components*. A component is usually defined as a self-contained object which can be swapped and rearranged at will, exhibiting graceful degradation where required functionality is not present. Section 2.2.5 (concerned with features of systems meant to support PIM), mentioned that an important requirement of PIM systems was the ability of individuals to radically tailor their environment. Allowing a user to tailor their desktop environment by using different assistant agents on a 'plug-and-play' basis could therefore prove very useful.

In order to arrive at an interface-based design, a set of interfaces is required that will allow the component objects, identified in the OO decomposition, to communicate as needed. Essentially, the inter-object communication pathways denoted in Figure 4.4 will be implemented as interfaces in the target object of each pathway, and given interfaces will only be exposed to the objects that need to use

them. This is effectively an example of 'least privilege' in design, where objects are only permitted to use as much of the rest of the system as they need to.

Object	Needs ...
the AIMS application	1. to accept events from the shell; 2. to accept events from its own user interface; 3. to be able to cause events in the external operating system.
the Agents	1. to accept events from the rest of the system; 2. to generate events for propagation to the rest of the system.
the Dialogue Record	1. to record a journal of events that have occurred in the system; 2. to service requests for past events for inferencing purposes.
the User Model	1. to service requests for user preference data; 2. to service requests for user profile data.
the File Access record	1. to record file access information; 2. to service requests for access information about files.

Table 4.6. Per-object communication requirements.

Interface	Description
IUIPort	Used to communicate actions within the AIMS application's user interface to the application itself.
IShellPort	Contains methods which are used to indicate events within the operating system's shell application.
ISysPort	Contains methods to control the relevant parts of the operating system – to create shortcuts and to rank file access records.
IDialogueRecord	Contains methods to record events and to search the journalled event record back in time for specific events.
IUserModel	Contains methods to access user preference information and to gain access to the file access record store (manipulated via IFileAccess).
IFileAccess	Contains methods to update and retrieve information from the file access log.
IEvent	A event 'sink' interface – used by another object to pass event information to the object implementing the interface.
IAgent (derived from IEvent)	Used to access an agent's functionality, with methods to initialise the agent as part of the application, and to send events to the agent.

Table 4.7. List of interfaces used within the system.

Table 4.6 shows the communication requirements of the objects present in the system. Where an object is noted as accepting events or servicing requests from any other object, it is likely that an interface will be required by the accepting object.

Table 4.7 shows the list of interfaces arising from the analysis of the inter-object communication requirements[4]. The interfaces fall into two main categories – singleton interfaces (used by only one object in the system) and other interfaces used by more than one object.

For singleton objects such as the Dialogue Record, the interface will simply be the collection of methods used to access the object, as would be the case for a normal OO design. For other object types, the interface indicates a subset of the object's abilities. Essentially, the interface definitions in an interface-based design represent an object type hierarchy rather than an object behaviour hierarchy.

One subtle element to the design is the definition of the *IAgent* interface in Table 4.7. Since agents need to be able to receive event information, they should be event sinks, and therefore implement *IEvent*. However, they will also need to pass event information back to the system, and therefore need to be initialised with information about the system's *IEvent* sink. The *IAgent* interface is therefore a refinement of *IEvent* – illustrating interface inheritance, rather than implementation inheritance. Another point to note is that the explicit *XxxPort* objects in the original object hierarchy can be subsumed as *XxxPort* interfaces in the design, simplifying the object set.

From these requirements, and with the set of objects already identified, the set of components needed to build the system as a whole can be set out, as shown in Figure 4.6. The *IEvent* interface of the *CAimCore* class is shaded, denoting that it is a private interface. It is used by the agents registered with the system, so that they can communicate events (as the results of their inferences) back to the system. The

[4] The interface names in Table 4.7 are prefixed with 'I' to distinguish them from plain classes, whose names are adorned with a 'C' instead – this is a *de facto* naming convention, particularly common in COM-based (Box, 1998a) development environments.

interface is not public, as the external environment has no need to access it. The *CAimCore* class also (in this design) contains the user model and dialogue record information.

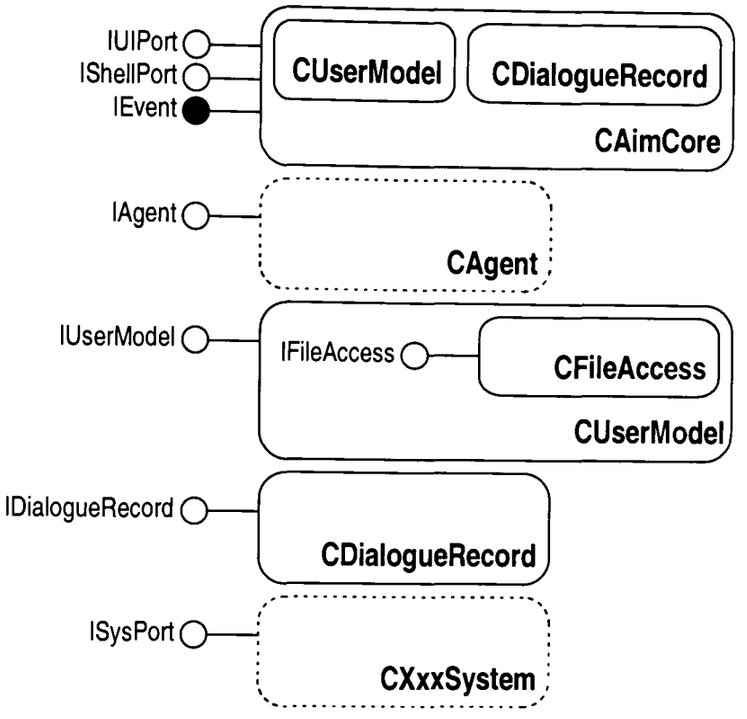

Figure 4.6. AIMS System Component 'Gallery'.

Class	Description
CAimCore	The main component of the system, encapsulating the functionality of the application, event handler and event distributor shown in Figure 4.4.
CAgent	An abstract class which will be used to implement the final agents, by deriving them from this class. Also includes private methods to create new events and propagate them back to the system.
CDialogueRecord	Implements which record events and to search the journalled event record back in time for specific event types.
CUserModel	Implements methods to access user preference information and to gain access to the file access record store (manipulated via IFileAccess, below).
CFileAccess	Implements methods to update and retrieve information from the file access log.
CXxxSystem	A to-be-implemented class which will encapsulate the operating system environment, implementing the *ISysPort* interface.

Table 4.8. Class Descriptions.

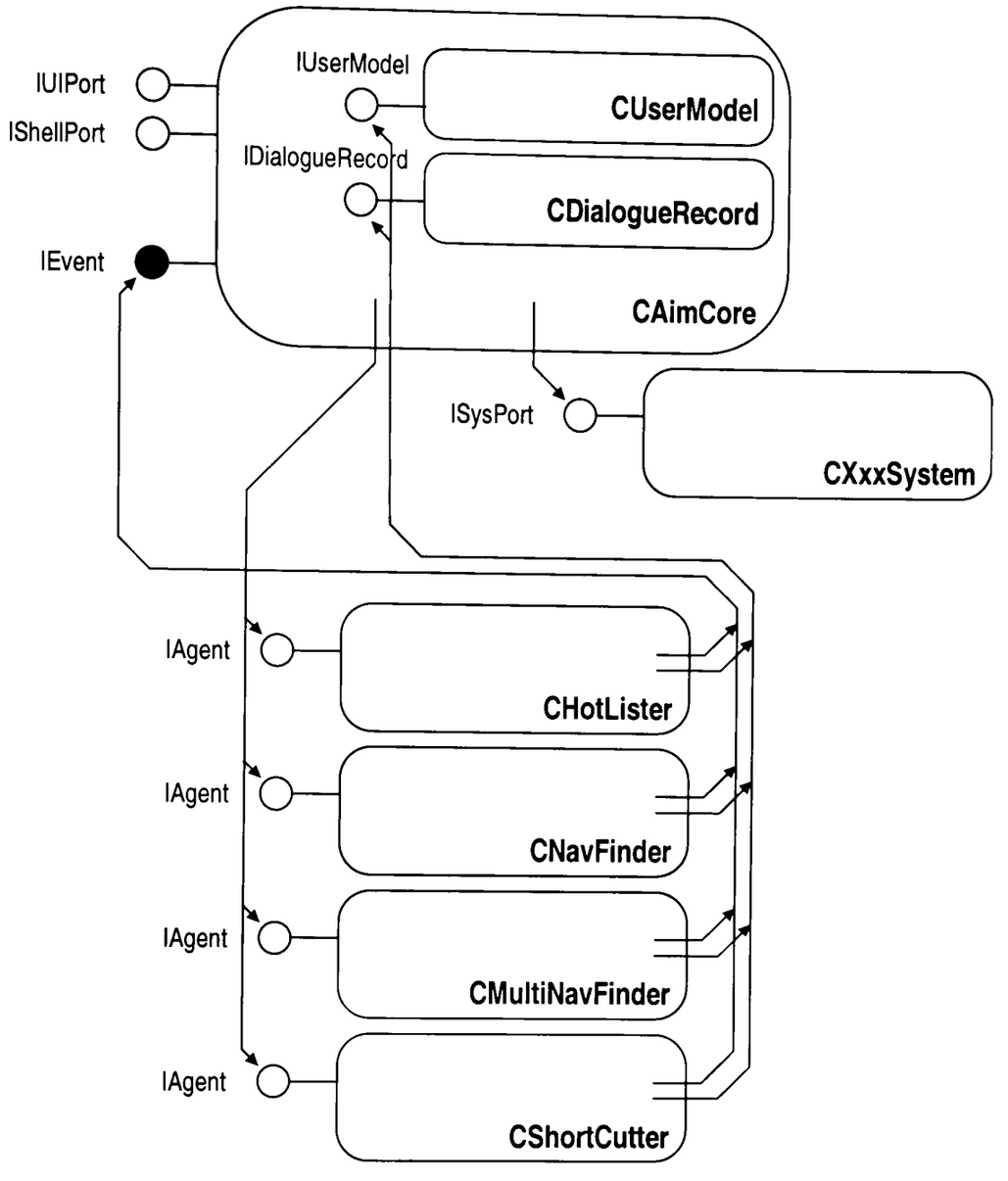

Figure 4.7. AIMS System Composition.

Table 4.8 describes the classes which need to be implemented to realise the system as a set of communicating components. The only 'missing links' are the derived *CAgent* instances (which just implement the behaviours documented in sections 4.4.6 and 4.4.7), and the *CXxxSystem* class which implements the *ISysPort* interface, to communicate with the external operating environment. The classes described in Table 4.8 are connected together to form the final system, as shown in Figure 4.7.

As can be seen from Figure 4.7, the system consists of a number of agent instances being fed event information (via their *IAgent* interfaces) from the AIMS application core. In turn, the agents may generate events as the result of inferences and send these back to the core, via its (private) *IEvent* interface. The agents may also call on the core for dialogue record and user modelling information, gained through the *IDialogueRecord* and *IUserModel* interfaces provided by members of the core.

4.5. System Implementation

This section briefly documents the implementation process followed in realising the AIMS application as a working piece of software. A choice of target implementation platform is made, which then leads to a small amount of re-design work, as some features need to be generalised slightly to cope with all eventualities.

The integration process is then discussed, showing how the platform-independent core implementation was dovetailed into an existing operating system environment. It was originally planned to integrate the software into a current popular graphical shell application – namely, the Windows NT Active Desktop Shell, as this supposedly exhibited the ability to support extensions of this kind. In practice this proved extremely difficult to do; while the Windows NT Shell does offer significant opportunities for extension, the features required by this system were not supported by it. Instead, a skeleton shell-like application (which allows users to navigate through the filesystem, open files, run applications and so on) was developed. This was then used as a source of live user event data, in order to test the application.

4.5.1. Choice of Target Implementation Platform

The final choice to be resolved before a concrete design can be arrived at is that of the target platform for the implementation of the system prototype. Several factors need to be considered when making this choice:

(i) *Installed user base:* the prototype should be implemented on a platform which is widely used, to be a representative example of how such an application would be implemented and subsequently used;

(ii) *Standardisation of user interface:* the prototype needs a target platform which has quite a standardised user interface, across various architectures and machine types;

(iii) *Availability of development tools:* in order to reduce, as far as possible, the amount of low-level user-interface programming work, the system should have a good range of tools which actively support user interface and application development;

(iv) *Extensible shell interface:* the platform should have a reasonably straightforward means of extending the default graphical shell application (thereby allowing the implementation of features such as the annotation support discussed earlier);

(v) *Good integration substrate:* it would be preferable if the target platform offered an existing mechanism for integrating software components – as it is highly likely that the extensions mentioned in (iv) will require such support.

The Windows NT platform provides all the desirable features mentioned above – it is widely used, it has many good development tools available, and from Windows 95 onwards it has had a reasonably standardised user interface. The graphical shell application has an extensible interface, based on a good software compo-

nent integration substrate (COM, the Component Object Model, another widely-used *de facto* standard). The underlying operating system provides memory and process-protection, useful in developing and debugging applications, plus a per-user profiling model.

In terms of infrastructure and integration, the system will be implemented using C++, MFC and COM. MFC, a stable platform for development work, provides a set of classes which insulate software from the specifics of the underlying operating system. Utility classes are provided (at a basic level, objects may be contained in lists and so on) and more sophisticated support (such as that required by persistent objects). Also, COM provides a simple and elegant mechanism for processes running on the same machine to intercommunicate – as required to gain user-interface event information.

4.5.2. Platform-Dependency Issues

This section considers the impact that the choice of platform has on the system to be developed. The design work up to this point has not considered any details of the environment to be used to implement the software, and there generally tend to be subtle elements of the environment which need to be taken into account for the system to work as well as possible.

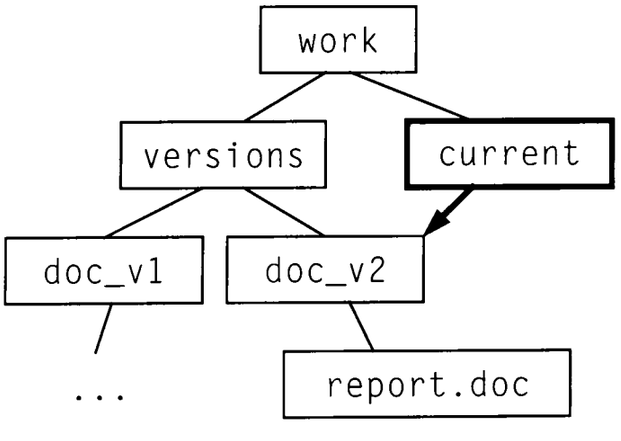

Figure 4.8. Silent shortcut translation by the operating system (Unix-style).

The nature of 'short-cuts' was not fully discussed in the preceding sections concerned with the system design (sections 4.4 to 4.4.11). The assumption was made that a 'shortcut' would be indistinguishable from a normal file, except that the operating system would translate it as required. This situation is depicted in Figure 4.8.

A folder named 'work' contains a shortcut called 'versions', which in turn contains folders 'doc_v1', 'doc_v2' and so on, for versions of a document under development. The 'documents' folder also contains a shortcut called 'current', showing the current version of the document, which is a shortcut to the folder 'versions\doc_v2'. The 'doc_v2' folder contains a report document file called 'report.doc'.

Given this situation, the design assumed that if a client application opened the file 'work\current\report.doc', the operating system would silently follow the shortcut, actually accessing the file 'work\versions\doc_v2\report.doc' without any intervention by the client application. Two (logical) files have the same underlying (physical) location. This is the case under almost all Unix-style operating systems (where shortcuts are referred to as links).

However, under Windows NT 4.0, this is *not* the case. Shortcuts are represented by special files which contain information referencing another file or folder, and which must be explicitly resolved by client applications. This introduces a problem in identifying parent-child relationships between folders and files. Figure 4.9 illustrates the underlying situation. The shortcut file 'work\current.lnk' contains a reference to the folder 'work\versions\doc_v2', conveying the same logical structure as in Figure 4.8.

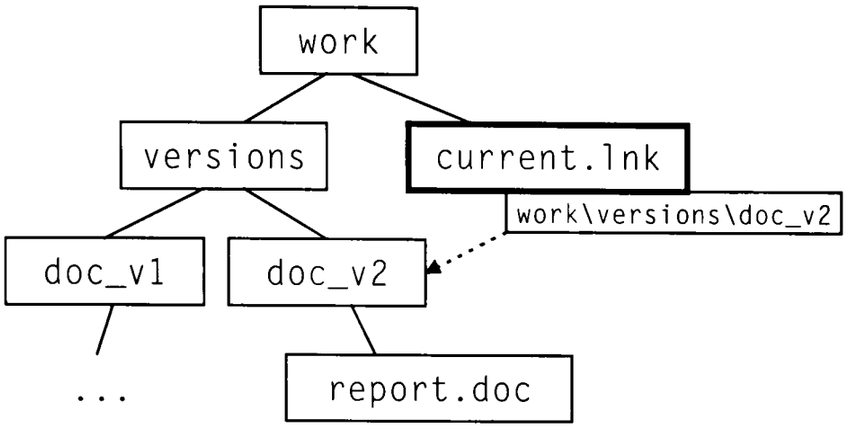

Figure 4.9. Explicit shortcut resolution by the client application (Windows NT-style).

The difference is that the file 'work\current.lnk\report.doc' does not exist, and would not be found by the operating system – the client application first has to examine the file 'work\current.lnk' using OS-supplied routines to determine that the link's target is actually 'work\versions\doc_v2\report.doc', and use that file instead.

The preceding design work assumed that, since shortcuts would be transparently followed by the operating system, a file's name alone could be used to spot single-level parent-child relationships between folders and files – for example, the folder 'work\current' is the parent directory of the file 'work\current\report.doc', which can be seen by comparing the file's path.

However, this is not the case when shortcut files are brought into the equation. In the example shown in Figure 4.9, the file 'work\versions\doc_v2\report.doc' is a subfile of the folder shortcut 'work\current.lnk', but that fact cannot be discovered simply by comparing the pathnames. The solution to this problem is that the event-handling support needs to be generalised to accept files and folders that have 'aliased' shortcut names. Instead of just recording that a file or folder was opened, if that file or folder is a shortcut, then both the target object's name *and* its shortcut alias should be stored.

The necessity for this alteration is illustrated in Table 4.9. This reflects the same kind of folder and file accesses as those in the previous example shown in Table 4.3, except that the folder and file objects in this sequence are actually shortcuts. If the shortcut aliases were not taken into account when attempting to detect parent-child relationships (as was the case earlier), these connected accesses would not be found. This would have resulted in a system where compound accesses would be silently ignored when shortcuts were involved.

Event Description *Internal Detail*	**ID**
1. User opens desktop folder 'Personal' which is a shortcut to 'Rich\Home'	
folder-opened ("\")	E1
folder-opened ("\Rich\Home", "\Personal.LNK")	E2
System recognises folder-subfolder relationships using (E1, E2)	
navigate-folders ("\", "\Rich\Home")	E3
2. User opens folder 'Documents' which is a shortcut to '\Local\Docs'	
folder-opened ("\Local\Docs","\Rich\Home\Documents.LNK")	E4
System recognises folder-subfolder relationships (E2, E4)	
navigate-folders ("\Rich\Home", "\Local\Docs")	E5
3. User opens file 'Diary' which is a shortcut to '\Journal\Today.DOC'	
file-opened ("\Local\Docs\Diary.LNK", "\Journal\Today.DOC")	E6
System recognises folder-subfile relationship (E4, E6)	
navigate-open ("\Local\Docs", "\Journal\Today.DOC", 1)	E7
System recognises earlier folder navigation as parent of event (E5, E7)	
navigate-open ("\Rich\Home", "\Journal\Today.DOC", 2)	E8
System recognises earlier folder navigation as parent of event (E3, E8)	
navigate-open ("\", "\Journal\Today.DOC", 3)	E9

Table 4.9. Detecting navigation-selection sequences based on user events with aliased names.

Table 4.9 shows the modified mechanism at work. For example, event E3 is generated because *"\Rich\Home"* has a shortcut alias *"\Personal.LNK"* which is a subfile of the previously-opened desktop folder "\". Therefore, the user has performed a single *navigate-folders* event from the desktop "\", to *"\Rich\Home"*

(even though this physically spans two folder levels, it appears to the user as only one).

Taking these alterations into account results in the modified event types shown in Table 4.10 (contrast with Table 4.4, above), and the modified event attribute usage patterns shown in Table 4.11 (contrast with Table 4.5, above).

Event Name	Event Description
file-opened (file [, alias])	triggered when the user double-clicks a datafile *file* (with an optional shortcut alias *alias*), causing the associated application to run, opening the file
folder-opened (folder [, alias])	triggered when the user double-clicks a folder *folder* (with an optional shortcut alias *alias*), opening it to explore its contents

Table 4.10. Syntactic Events Processed by the System

code	*name1*	*name2*	*level*
FILE_OPENED	name of file opened	the file's shortcut alias (if any)	-
FOLDER_OPENED	name of folder opened	the folder's short-cut alias (if any)	-

Table 4.11. Symbolic event type code list and attribute usage.

These modifications were all that were necessary to allow the system's core functionality to be implemented under the target platform's operating system environment. The next step in the process was to consider how this platform-independent system could be integrated with the graphical interface to its target platform.

4.6. Integration Process

The software developed up to this point was a standalone system, devoid of any bias toward a particular platform, having no user-interface as such – other than the binary and programmatic interfaces devised so far. Rather than build a new user interface specifically for the purpose of exercising the system, an existing

graphical shell interface was augmented. In this way, the AIMS application could be seamlessly integrated with an existing environment without having a visible effect as far as experienced users were concerned.

4.6.1. Integration Areas

A complete implementation of the AIMS application would need to be integrated into the target environment in two ways. User-interfacing support would be needed to allow individuals to use the features of the software – to attach, edit and search for notes, to accept suggestions for shortcut creation and to view regularly-accessed files. *De facto* 'standards' exist for operations such as these – for example, most users are familiar with the idea of dialog boxes with lists to choose from.

In contrast to these reasonably standard interfacing requirements, the software also had to be integrated with the graphical shell. There were two main integration areas that had to be addressed here:

(i) *The timely and accurate acquisition of information about the user's actions within the shell's user interface:* the short-cut suggestion and file tracking mechanisms rely on these events, so there needed to be a source of them;

(ii) *The ability to 'hook' into the user interface such that minor visual cues could be effected:* the file annotation support would require a mechanism to indicate the existence of a given file's annotation, possibly using icon overlays or such-like.

These areas necessitated the acquisition of some reasonably esoteric details about the target platform's operating system, Windows NT 4.0. All the available documentation for the NT Shell's application programming interfaces (APIs) and the Win32 APIs seemed to indicate that they would support extensions along these lines. There were a number of different approaches that seemed to offer possibilities for both of the above areas, but it proved to be the case that the precise nature

of the information required was too specific to be supported by the available APIs. The approaches examined and the shortcomings found with them are now discussed in the following sections.

4.6.2. Interface Event Acquisition: The Active Desktop

The latest public major release of Microsoft's 'Internet Explorer' application, version 4 (referred to as IE4) brought with it a 'Desktop Update'. This allowed the desktop screen to be generated by an HTML page (essentially, a Web page held locally). More importantly for this application, the Desktop Update provided an API to the Active Desktop itself, allowing client applications to alter the appearance of the desktop page.

The documentation seemed to imply that applications could also react to events on the desktop, in just the fashion that was required by the software under development – but this was not entirely the case. It is possible to write a small Java applet, appearing as an icon, that delivers an event when the user clicks on it – but this cannot be generalised to any icon in any window under the control of the shell.

4.6.3. Interface Event Acquisition: Internet Explorer Automation

One slightly-documented aspect of the Active Desktop is that it replaces the Windows NT Explorer (the shell application) by Microsoft's Internet Explorer, which includes file-system browsing ability. Essentially, Internet Explorer becomes the operating system's shell application[5]. Internet Explorer, in common with many

[5] As was required to circumvent a US High Court ruling that Microsoft's own IE4 internet browser could not be unfairly distributed with their operating system *as it was not a part of it*.

Windows applications, supports the Automation[6] mechanism. In IE4's case, an interested client can request to be informed when events occur in the browser window – clicking on links, and so on. It was hoped that this mechanism would allow file-management events to be received from an Internet Explorer-hosted shell window. Unfortunately, this method does not generalise to multiple windows (i.e., when a new window is opened, events are not received from it).

4.6.4. Interface Event Acquisition: Context Menu Handlers

Other methods for acquiring user interface events were evaluated. Windows NT has context menus – menus of item-specific operations. These are generated by the shell application in response to a right-mousebutton click. It is possible to register extended context menu handlers with the NT shell, so that application defined menu choices specific to a given file can be added to the system's default context menu[7].

The Windows NT documentation notes a special case where the context menu handler is called to discover the default (i.e., double-click) action for a file. This could be used to note that an object has been selected. Tests have shown, however, that the context menu handler is never called in this case. Sources from Microsoft confirm that this is true (Arnold, 1999).

---

[6] Automation is the ability for one application to control another, using a special COM interface. This is mechanism is used in scripting environments, such as Visual Basic. Automation-enabled applications ('servers') running on a machine can register themselves with the operating system. A client can look up the server it requires, access it using COM, and issue commands to it.

[7] For example, when the WinZip file compression utility is installed on a Windows NT machine, file and folder menus automatically have compression options added to them by WinZip.

4.6.5. Interface Event Acquisition: Shell Execution Handlers

In a similar fashion to context menu handlers, it is possible to install COM-based handlers which are called to determine how to execute objects. The Win32 API function `'ShellExecute()'` scans any installed handlers, passing control to them one at a time, searching for one that can handle the file specified. This mechanism would provide the information required by the file tracking support, but as directories are not 'executed' by the shell, the shortcut detector would not then function.

4.6.6. Interface Event Acquisition: Message Hooks

The preceding event acquisition methods have functioned at a reasonably high level – that is, the information that would be provided by them (had they worked) would have been instantly usable. This is obviously preferable – getting the full name of a file that has been opened is the best option – but there are ways of accessing user interface event information at a lower level.

The Win32 API provides functions to insert what are termed 'hooks' into the system's event handling chains. Two categories seemed relevant to the task in hand – journal hooks and window message hooks. Journal hooks are at the higher level of the two, processing input events in much the same way as a macro-recording system. Applications can be informed of keystrokes and mouse-clicks, but it is very difficult to relate these reliably to the visual object (window, icon) that they will eventually affect.

Window message hooks operate at a lower level, allowing the messages that are to be sent to a window to be intercepted. All visual 'controls' in Windows – such as pushbuttons, checkboxes and the like – are realised using child sub-windows, and these window 'hierarchies' are recorded by the operating system. This information could be used to allow an application to traverse the window hierarchy of an

application (such as a shell window), to locate the subwindow that a mouse double-click event would affect. However, this approach was rendered useless as shell windows do not use explicit sub-windows for their contents, using freeform graphics instead.

As well as event acquisition, a method was required to present a visual cue for the file annotation support, so that files with annotations would be obvious – perhaps using an overlaid miniature 'sticky-note' icon. The next two sections examine some methods that were evaluated, and the shortcomings that were identified.

4.6.7. Visual Cueing Techniques: IE4 Browser Control Hosting

The IE4 browser control, mentioned in conjunction with Automation in section 4.6.3, supports another access method. It is possible to create an application which hosts Internet Explorer as a subwindow of the application – in essence, re-using the facilities of IE4. IE4 exposes much of its functionality through a set of COM interfaces (one of which is the event-notification interface that was examined earlier in section 4.6.3). This was evaluated to test whether the window containing the IE4 control could be used as a filesystem browser while at the same time painting overlay graphics on it as required to indicate annotations. However, the browser control does not expose any methods for accessing the icons presented within the window, so this method was not appropriate.

4.6.8. Visual Cueing Techniques: Window Sub-Classing

Each window, in Windows, is a member of a window 'class'. Class members share common attributes – some related to appearance, and some related to the way the window processes events. Existing window classes can be subclassed to refine or alter their behaviour. Tests were conducted to ascertain whether it was possible to develop a subclass of that used in shell windows. This subclass could then be supplied with a custom event handler to record information about the window to

be used later. A special class for shell windows is registered – which shows up as being named *'SysList32'*, when a window 'spy' tool is used – but as it is not mentioned in the system's documentation or definition files, insufficient information existed to use it.

4.6.9. Integration Problems: Conclusion

This brief survey of possible – yet unsatisfactory – methods of integrating applications with the existing Window NT shell illustrates that this is a task which is not at all straightforward. This work – which spanned several months – was carried out with the aid of an employee of Microsoft, whose help was invaluable in ruling out these approaches. It would appear that there are very few people that possess the necessary knowledge to implement a piece of software such as this (Arnold, 1999). It is difficult to speculate about the possible reasons for this, although some thoughts on the subject are presented in Chapter 7.

Although not the desired outcome, for the purposes of this study the most profitable route to implementing the software is to develop a 'test harness' – a skeleton application which implements a subset of the look, feel and functionality of the Windows NT shell. The implementation of a production-quality piece of software is not an objective of this study, but some means of evaluating the final product is definitely required. The development of this test harness is therefore discussed in the next section.

4.7. Test Harness Implementation

This section documents the development of a skeleton shell program, used to provide a 'live feed' of interaction event data to the agent framework developed already. Some technical background to the development is presented, followed by a discussion of the software's capabilities, illustrated by examples of the features in use.

4.7.1. Background

The test application for this system was implemented as an application using the MFC hierarchy. MFC is a class library which supports development of applications under Windows, and is a *de facto* standard used by a large number of developers across the world. The primary development language is C++, and many features of the integrated development environment – Visual Studio – actively support the MFC framework. The environment includes a tool called ClassWizard, which provides a high-level graphical interface to the syntactic structure of a program. Programs can be browsed as collections of classes and methods – rather than just a set of source-code files which eventually compile into a program.

The MFC class library encapsulates many of the underlying system objects present in the Windows system, wrapping them with OO construction, access methods and such like. It also takes care of many of the tedious housekeeping functions encountered in GUI development. GUI elements are designed using a visual editor, and can then be connected with underlying C++ member variables, automating a large part of dialog development. The MFC application framework handles window messages, mapping them onto C++ member functions, removing the need to interpret archaic message identifiers.

4.7.2. AIMS Application: General Information

There are two components to the final implemented system as a whole: the shell windows and the application tool-bar. When invoked, the AIMS application displays a full-screen shell window with the normal Desktop icons on it (see Figure 4.10). These icons work in the same way as the normal desktop, accepting double-clicks for selection, and right-button clicks to display standard context menus.

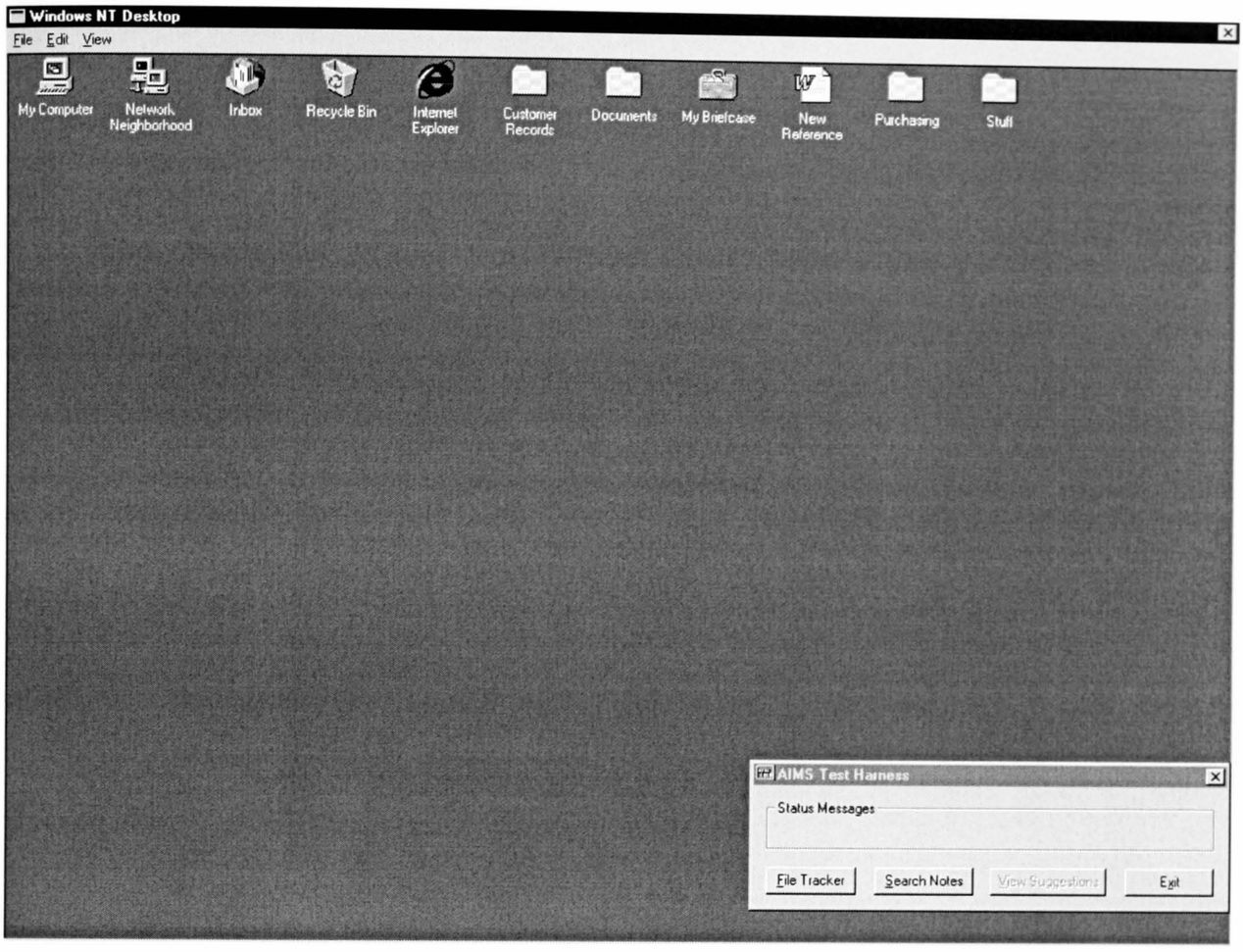

Figure 4.10. The AIMS application's pseudo-desktop window.

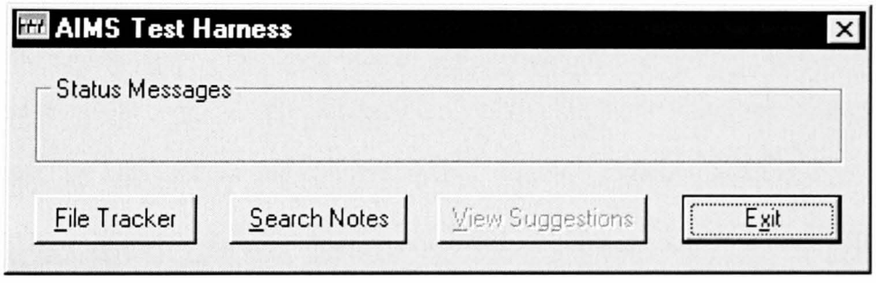

Figure 4.11. The AIMS application's pseudo-toolbar.

The AIMS application also displays a floating pseudo-toolbar (see Figure 4.11), with push-buttons on it to invoke the following operations:

(i) *View Suggestions:* displays (when enabled by the system) a list of possible shortcut suggestions;

(ii) *File Tracking:* displays a dialog which allows the user to view a filtered, sorted log of their file accesses;

(iii) *Search Notes:* displays a dialog which allows the user to search any file annotations currently stored in the system.

Each of these mechanisms is now discussed, with examples to show how they can are used in practice.

4.7.3. Shortcut Suggestion

The objective for the shortcutting mechanism is to allow the system to convey its suggestions for shortcuts to the user, for the user possibly to choose a suggestion with which they agree, and for the resulting shortcut to be created. As designed, the system issues events (of type *suggest-shortcut*) when it has identified a compound file and folder usage pattern which has been repeated within the preset interval.

The solution chosen was to allow the user to display a 'shortcut suggestions' dialog, which contains a list in plain text of the suggestions made by the system, as shown in Figure 4.12. The example shown in the figure suggests the creation of a shortcut to the file "Experiment Report.doc" in the desktop folder.

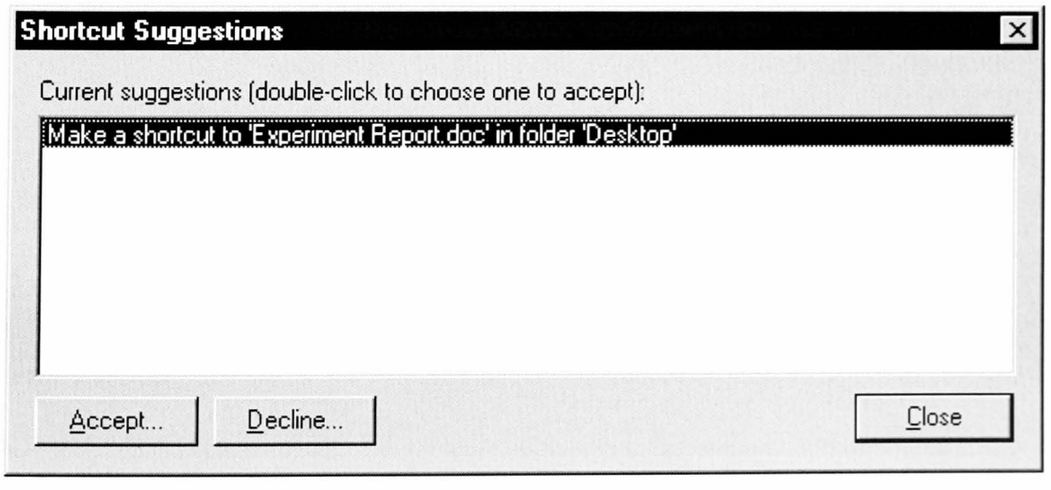

Figure 4.12. Suggestion for creating a shortcut to a file.

The user can select a shortcut to be created by double-clicking on the list, is asked whether they are sure (see Figure 4.13), and if so, the relevant operating system routines are invoked to create a shortcut.

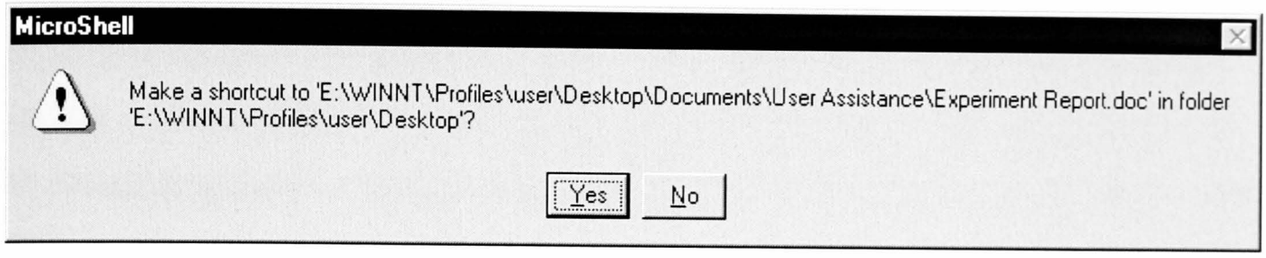

Figure 4.13. Creating a shortcut to a file – confirmation.

4.7.4. File Tracking

The system maintains a list of files, their last access times and their total access counts, as part of the CFileAccess object embedded in the user model (see section 4.4.11 and Figure 4.6). The user requires access to this information in a form which they can interpret easily, depending upon the memory of the file that they have.

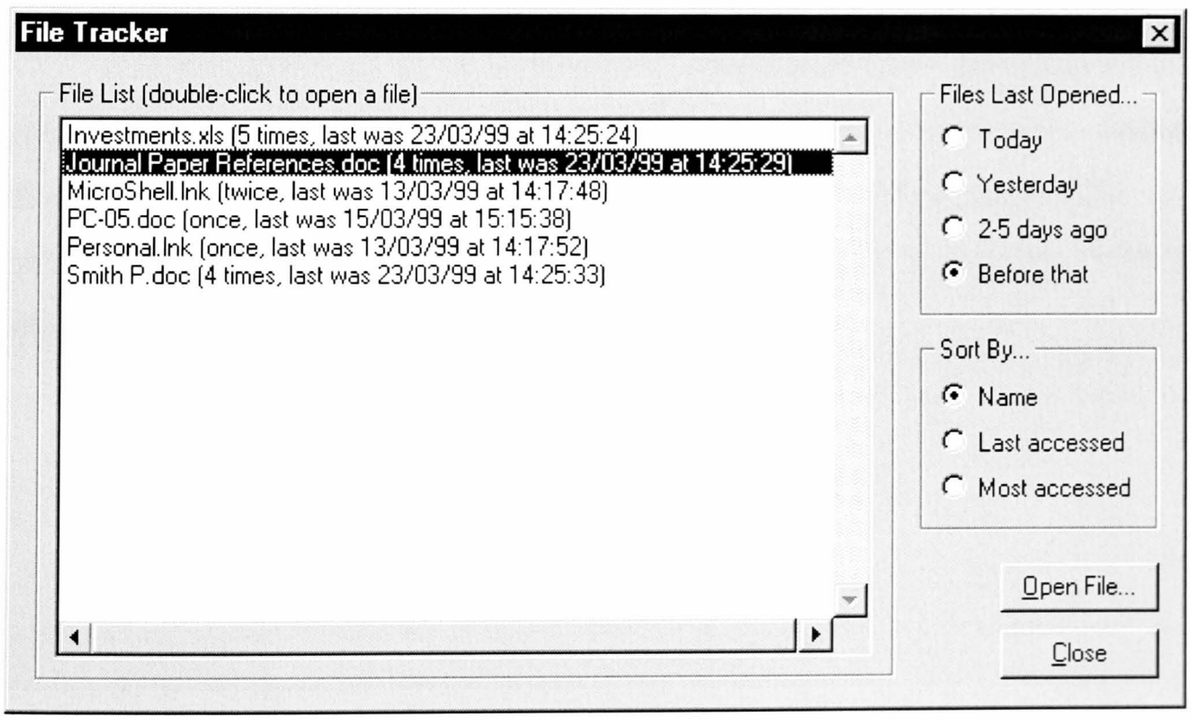

Figure 4.14. Using the file tracking dialog.

The file tracking dialog allows the user to obtain a view of the files they have opened in the past, sorted and filtered according to various criteria. The files are filtered according to their last access time – whether they were last accessed today, yesterday, between 2-5 days ago, or before that – so that not all files ever accessed have to be scanned through by the user. The user can also choose to display this file list sorted alphabetically by name, in order of last access from most recent, or in descending order of total accesses from most accessed, as shown in Figure 4.14. The user can double-click on any choice in the list, causing that file to be opened with the appropriate application.

4.7.5. File Annotation

The file annotation feature allows textual notes to be applied to any icon object in any of the shell's windows. Notes can be added or viewed using the Window NT context menu, which then has the appropriate option added to it (if no annotation exists, the user can choose to add one; if one exists, the user can choose to view it), as shown in Figure 4.15.

Figure 4.16 shows the user in the process of entering a note – a note-style dialog with a multi-line edit control (with a pale yellow background) is used. (The ghosted 'Remove' button is enabled when viewing, rather than adding, a note to allow the user to remove an annotation.)

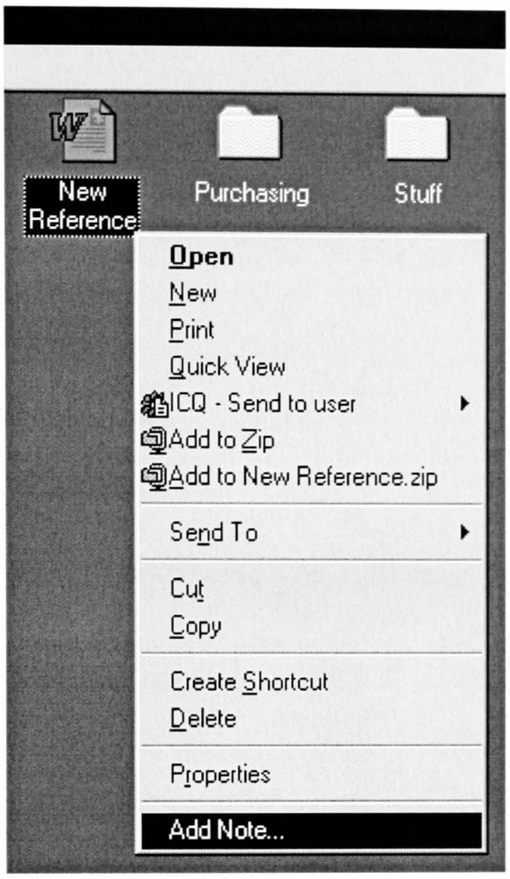

Figure 4.15. File annotation options on a file's context menu.

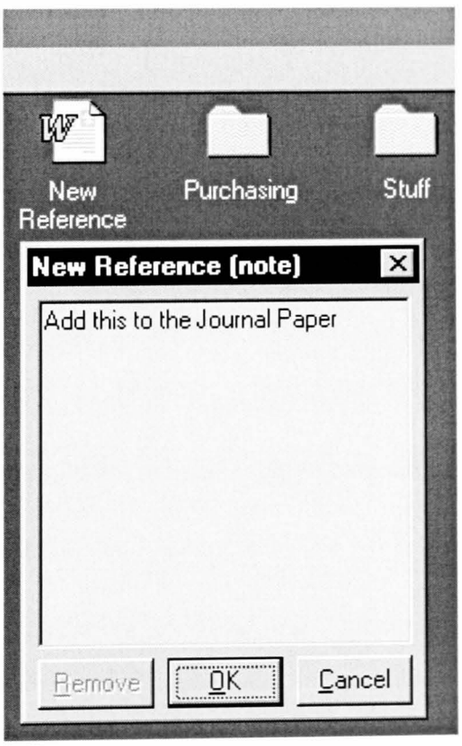

Figure 4.16. Annotating a file using the note editor.

The shell-style windows displayed by the AIMS application show the presence of an object annotation by modifying the appearance of the object's icon. Users of the Windows NT shell are familiar with the 'shortcut' overlay icon – a small arrow that is superimposed on the bottom left-hand corner of an icon, signifying that the icon represents a shortcut to the actual object. An object that has an attached annotation uses the same technique to indicate the fact, overlaying a small yellow note icon over the bottom right-hand corner of the icon, as illustrated in Figure 4.17.

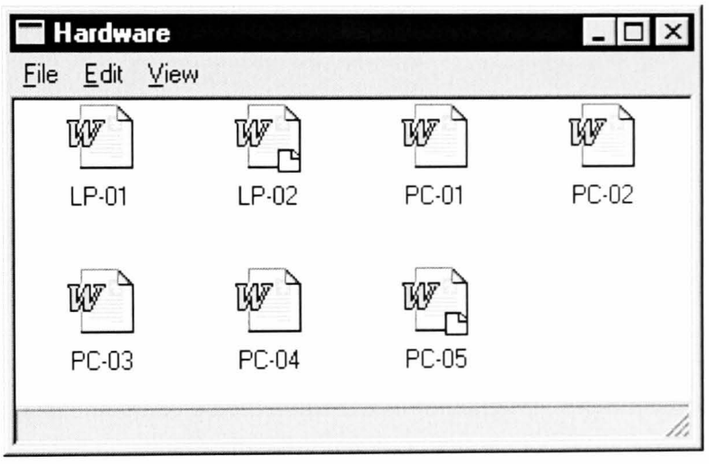

Figure 4.17. Annotated file icons in shell windows.

4.7.6. Annotation Searching

The file annotation feature in itself appears quite useful, but can be augmented further. The 'Search Notes' button on the AIMS application toolbar allows the user to search through the list of all the annotations entered, looking for a particular word or phrase. This is accomplished using the Note Search dialog, as shown in Figure 4.18.

This dialog simply allows the user to enter a word or phrase in the dialog's search field, then have the system look through all the file-annotation associations for any that contain it. A list of the matching filenames, and the first few lines of the related note, are then displayed. A double-click on a given file causes that file to be

opened using the relevant application, as if the user had double-clicked on the file in a shell window.

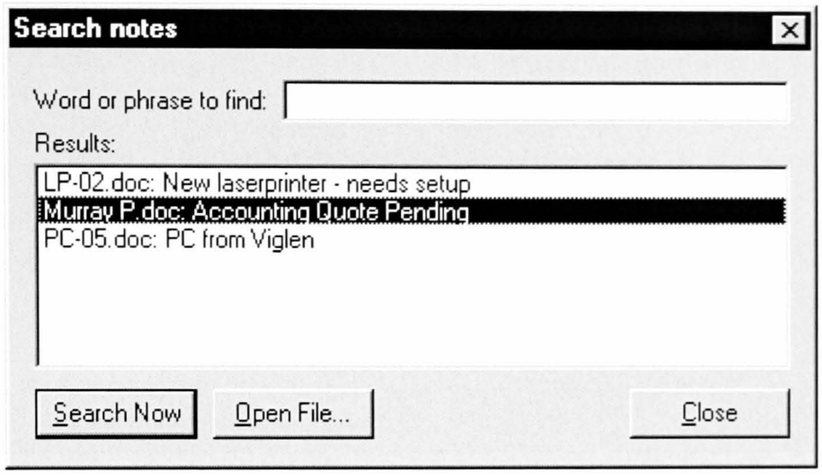

Figure 4.18. Searching for annotated files.

4.8. Conclusions

Using the informal abstract specification and design developed in Chapter 3, this chapter has developed and implemented a concrete design which took account of a particular application, platform, environment and operating system, and implemented it as a working software application. This application forms the basis for part of the evaluation work to be carried out in Chapter 5.

Chapter 5

Evaluation and Critique

5.1. Introduction

This chapter presents an evaluation of the work reported in the previous chapters. The evaluation is carried out at several levels: the application implemented in Chapter 4 is examined, by means of user trials and usability inspection; the implementation is also examined in software engineering terms; the component-based adaptive agent software architecture developed in Chapter 3 is evaluated, to show its usefulness in arriving at the design for the implementation; and the architecture for Adaptive Interface Technology (AIT) (Benyon, 1993) adopted in Chapter 2 – the theoretical framework underlying the study – is critically appraised. The findings from these evaluations are then summarised, leading on to the discussion of future work in the next chapter.

The objective of this chapter is to critically evaluate the work carried out in this study – to identify strengths and weaknesses of the final product of the research, and the methods and theories used in its design and implementation. This will lead to a set of general and specific issues arising from the research. These issues

will be used to develop solutions for shortcomings in the product itself, and indicate directions for future development of the processes and theories used.

The tangible product of the study – the software application, developed in Chapter 4 – is evaluated firstly by means of a small-scale user trial, based on a co-operative evaluation approach (Monk *et al.*, 1993). This involves a small set of individuals performing a set of tasks using a prototype interface augmented with the software developed. The trial will illustrate how the system performs some of the routine, mechanistic elements of PIM tasks, allowing the individuals to focus on higher-level issues. Usability heuristics are also used to examine the final system by inspection on a more finely-grained level, to highlight opportunities for improvement.

The system's final design and implementation are examined in technical terms, to give some insight into the output of the design process. This will illustrate how the design process was supported by the software architectures developed in Chapters 3 and 4. Software engineering issues are considered, to show how the development process may be improved to yield systems which are more elegant and easier to design and implement.

The design architectures and frameworks used to develop the system are also evaluated at a more theoretical level, by considering how they were used in the development of the specification for the system's behaviour and functions. Abstract frameworks obviously need to be reasonably application-neutral in their formulation, and the evaluation reflects this by seeking to show what concrete information needed to be added to them, in this situation, in order to give useful design advice.

The chapter concludes by providing a summary of the important issues raised in this set of evaluations to be used as the basis for discussion in the next chapter,

which will attempt to remedy problems noted in this work as a individual case, and to develop the field as a whole in the future.

5.2. Evaluation Overview

This section begins by examining the objectives of the evaluation process as a whole. It goes on to discuss a range of appropriate techniques that might be used to evaluate the system, and justifies the choice of a small number that are to be actually used. This is driven by an understanding of the structure and topics covered within the dissertation up to this point, to ensure that the reasons for choosing each evaluation technique are obvious.

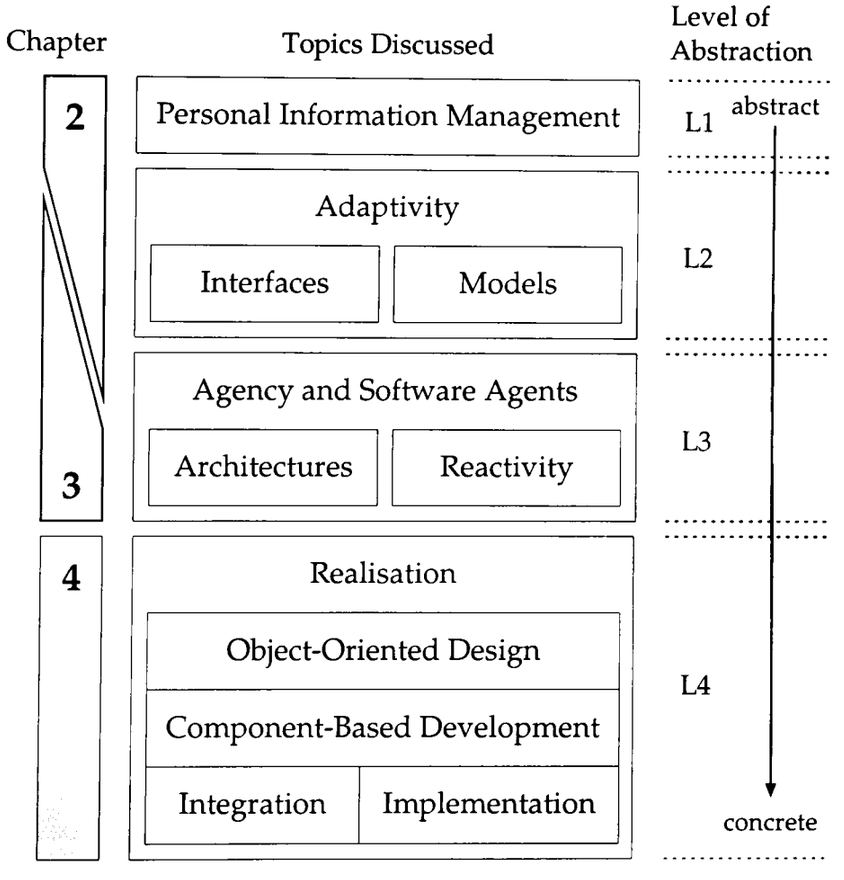

Figure 5.1. Dissertation topic 'roadmap'.

Figure 5.1 gives an overview of the preceding chapters (2, 3 and 4) of the dissertation, showing the nature of the issues addressed in each of them and how the

various topics explored relate to one another in terms of levels of abstraction, from the abstract to the concrete. The figure also shows their general location within the preceding chapters. The evaluation process will seek to critically appraise each of these areas of work in order to arrive at a set of issues to be used as the bases for the discussion and reflection in Chapters 6 and 7. With these topic areas in mind, the evaluation process begins by stating the objectives of the evaluation, which naturally lead on to a discussion of applicable techniques.

5.2.1. Evaluation Objectives

The evaluation, as a whole, has a number of underlying objectives. These are presented to give the background to the evaluation as it is essential that the reasons for attempting any evaluation process are known ahead of time. The objectives for this evaluation are as follows:

(i) *to quantify the 'successfulness' of the interface:* this aims to discover if the interface is performing as expected, and to gauge the extent to which it supports the individuals in performing their tasks;

(ii) *to gauge the 'effectiveness' of the concepts and metaphors used within the interface:* this should show whether the subjects have a good mental model of the system and its features, and if they understand the mechanisms by which it is supposed to support them;

(iii) *to examine the theory used in the system's specification, architectural design and development:* the development of the system draws upon a body of adaptive interface theory, which is appraised to find any shortcomings which might adversely affect the development process;

(iv) *to appraise the design of the system:* in software engineering terms, to examine how elegantly the system's form and structure were realised as system components;

(v) *to critique the final implementation of the system:* this will give a view of the software development undertaken during the system's realisation, to examine how easily implementation and integration was accomplished.

These five objectives lead on to the choice of evaluation techniques to be applied to different parts of the work reported in this dissertation. They can be placed into two categories: objectives (i) and (ii) above address the characteristics of the final product of the work; whereas (iii), (iv) and (v) are concerned with the system's development process. In terms of levels of abstraction, objectives (i) and (ii) apply primarily to the highest abstraction level, L1, from Figure 5.1. Objectives (iii), (iv) and (v) then apply mainly to levels L2 to L4, respectively, although the mapping is not precise (as is discussed in section 5.2.6).

Sections 5.2.2 to 5.2.5 follow the four levels L1 to L4, in discussing evaluation techniques applicable to each of the levels of abstraction. L1 is essentially concerned with the tangible product of the study, which is to be assessed in terms of usability and usefulness. L2 addresses the theoretical basis used to arrive at the requirements for the user interface and the AIMS application. L3 refers to the architectural and design techniques used to develop the system and L4 to the practical process of realising the system in software. The following sections address each level in turn.

5.2.2. L1 – Assessing the User Interface

There are a wide range of techniques that can be used to examine a user interface. In this study, we are primarily concerned with testing the validity of the resulting user interface from two different angles (essentially, objectives (i) and (ii) from section 5.2.1). Firstly, we wish to examine it to see whether it is performing as expected in helping users in simple information management tasks. Secondly, we wish to check whether useful metaphors and techniques have been used to present the augmented elements of the interface to users. A discussion of a range of us-

ability assessment methods will lead to a choice of suitable techniques to be used in this study.

Approaches to user interface evaluation can be placed into one of two categories: those which involve users and those which do not. Table 5.1 gives a brief taxonomy of system and user-interface evaluation approaches. Cognitive walkthroughs and heuristic evaluations use expert knowledge to examine a user interface without direct user involvement – they are effectively types of inspection methods. In contrast, the other three methods – usability testing, usability engineering and controlled experiments – are empirical in nature, all requiring users to take part in a study of some kind.

Evaluation Approach	Description
Cognitive Walkthroughs	A user interface is 'stepped through' methodically by experts, in a manner analogous to a program code walkthrough, to identify possible weaknesses in the interface.
Heuristic Evaluations with Usability Guidelines	A user interface is inspected by experts who use a set of broad guidelines to examine the interface for such things as consistency, error prevention, flexibility and efficiency.
Usability Testing	A reasonably general term for an approach which focuses on the observation of users performing tasks, which can inform subsequent redesign of the user interface.
Usability Engineering	A more strict testing approach where metrics – usability 'measures' of some kind – are devised, and quantitative usability goals are set and aimed for in test situations.
Controlled Experiments	The use of formal scientific experiments involving control groups, rigorous experimental design and subject selection, usually aimed at specific, low-level usability goals.

Table 5.1. Approaches to user-interface evaluation (from Baecker *et al.*, 1995, p. 82).

The approaches presented in Table 5.1 give a good sense of the spectrum of evaluation methods. Each of the categories in Table 5.1 will have a set of different sub-types – for example, there are a wide variety of methods which can be classified as 'usability testing'. At the same time, some of the methods – particularly those classified under 'usability engineering' and 'controlled experiments' – can

have quite daunting requirements. Extended periods of time can be necessary to carry them out properly and they are resource-intensive, some needing laboratories, office mock-ups and so on.

For this study, a 'discount' approach to usability testing (Nielsen, 1989) must from necessity be followed, as the evaluation scope will need to be limited due to time and resource constraints. 'Discount usability engineering' can be characterised as a hybrid of empirical usability testing and heuristic evaluation, which gives a good mix of evaluation covering both empirical user-based work and inspection-based appraisal, and is therefore the approach adopted here.

The system to be evaluated in this study is basically a prototype, although a functional one. The evaluation should reveal problems with the usability of this prototype by examining how users interact with it and gathering information about them. Some kind of interpretive approach – as opposed to the use of highly-controlled experiments – would seem to be most appropriate, as the evaluation's aims are not rigidly quantitative in nature. The users' opinions of how the system presents itself, and any means by which it might be improved, form important parts of the desired results.

A technique which can be deployed to good effect in a situation such as this is co-operative evaluation (Monk *et al.*, 1993). Co-operative evaluation aims to improve a user interface (or its specification) by detecting the problems in a prototype or partial simulation. This is accomplished by following an approach where the designer of a user interface works with a small number (typically around four) of representative subjects from the user population, so that a combination of their respective viewpoints can be used to identify potential problems and their possible solutions.

For the prototype system, an evaluation approach based on these ideas of co-operative, interpretive research would be suitable. It offers good control over the

experiment, good support for the experimental subject, a reasonable tolerance for the prototypical nature of the system, and can be sufficiently flexible to account for individuals' attitudes to the system.

As mentioned above, 'discount' usability testing also has an inspection-based, heuristic component to it. Inspection methods are useful in evaluating prototype systems (Baecker *et al.*, 1995), but there are some drawbacks that must be considered. Although a variety of several large sets of guidelines are available, such as (Smith and Mosier, 1986; Brown, 1988; Mayhew, 1992), large sets of guidelines are difficult to work with, and guidelines can be difficult to contextualise and apply (Mosier and Smith, 1986). Heuristics tend to be more widely-applicable, less specific guides to desirable qualities of interfaces, so for the prototype a heuristic evaluation approach would be more suitable.

Based on the discussion so far, the assessment of the user interface will employ two suitable techniques. Firstly, usability testing with a co-operative evaluation approach (Monk *et al.*, 1993) will be done to observe users working with the system and to record detailed data on their interactions with the system (in quantitative terms) and ask users about their opinions about the system (in qualitative terms). Secondly, a heuristic evaluation will be carried out (Nielsen, 1992; 1994) to inspect the system, with a set of published usability guidelines in mind, to gauge how well the system should aid individuals in performing simple information management tasks.

5.2.3. L2 – Assessing the AIT Architecture

This evaluation element aims to critically appraise the theoretical framework upon which the design of the system developed in the previous chapters was based. This assessment aims to examine the specification process to show any benefits realised for future designers of systems which have similar aims as this, to see how the specification process was aided by the AIT architecture, and to illustrate areas

where additional guidance was necessary. A future aim is to develop design guidelines based on the experience of this study, particularly based on the additional guidance required by this study.

The AIT architecture (Benyon, 1993) adopted earlier in the thesis (see section 2.6) provides a framework which can be used to design adaptive interfaces which are meant for a wide variety of applications. For each specific application, the various components of the architecture (the Domain Model, the User Model and so on) provide 'templates' for elements of the system. Evaluation of the architecture will appraise the process of 'fleshing out' the basic skeleton of the AIT architecture to arrive at the design for the adaptive elements of the prototype system – how much guidance the architecture provided, and where additional information would have been useful.

5.2.4. L3 – Assessing the System Design

The third area of evaluation aims to critically appraise the design of the system developed in the previous chapters from a software engineering viewpoint. The system's design embodies elements of the AIT architecture. One focus of the evaluation will be to assess how well the final implementation reflects the goals of the AIT architecture by using reactive agent systems, when the architecture itself makes few prescriptive suggestions for the active mechanisms which comprise part of it.

The evaluation will also seek to discover if the overall design of the system was elegant (in software engineering terms), and whether the software architecture developed in Chapter 3 was useful in the detailed design process. The aim will be to illustrate how some of the experience, knowledge and/or architectural details developed in this study can be re-used in the development of similar systems.

5.2.5. L4 – Assessing the System Implementation

The final evaluation element aims to critically appraise the implementation and integration of the system developed in the previous chapters. The implementation of the system is evaluated in software engineering terms, to provide an understanding of how the experience gained in this study may be generalised.

This will show how the use of a component-based approach has an impact on the ease of development and potential re-use of the system, how well the system is integrated into its host environment, and how this can be improved in interfacing terms, or eased in technical terms. We wish to find out if the resulting implementation is elegant, in software engineering terms, or if it reflects a disordered, *ad hoc* design. Again, an overall aim is to show how some of the experience, knowledge and/or architectural details developed in this study be re-used in the development of similar systems.

5.2.6. Evaluation Techniques and Dissertation Contents: A Mapping

Sections 5.2.2 to 5.2.5 described evaluation techniques broadly relevant to the four levels of discussion (shown in Figure 5.1) presented within this dissertation. However, there is not a direct mapping between the evaluation techniques and the levels to which their results may be applicable – in fact, the evaluations presented in the following sections relate to one or more of the topic areas identified above.

Table 5.2 shows how the different evaluation techniques apply to the elements of the topic roadmap. This describes how results from the different evaluation techniques relate to the different topic areas shown in Figure 5.1, and also demonstrates that the evaluations presented in this chapter completely cover the work reported in this dissertation thus far.

The next five sections provide detailed discussions of the evaluation process, showing how the principles identified earlier were applied to the system devel-

oped, and the theory used in its development. The evaluations are split into two sets – evaluations A1 and A2 focus on the usability of the prototype system, whereas evaluations B1 to B3 address the theoretical frameworks used, the design process followed, and realisation of the system, respectively. These complementary evaluation approaches are used to give a fuller picture of the system than would be obtained using just one or two analytical perspectives.

Evaluation Technique	Applies to Topic Areas (from Figure 5.1)
A1: Co-operative Usability Evaluation	L1: Personal Information Management; L2: Adaptivity (Interfaces).
A2: Heuristic Usability Inspection	L1: Personal Information Management; L2: Adaptivity (Interfaces and Models).
B1: Theoretical Evaluation	L2: Adaptivity (Models); L3: Agency and Agents (Architectures and Reactivity).
B2: Design Evaluation	L3: Agency and Agents (Architectures); L4: Object-Oriented Design; L4: Component-Based Development.
B3: Implementation Evaluation	L4: Component-Based Development; L4: Integration; L4: Implementation.

Table 5.2. Evaluation techniques and relevant roadmap topic areas drawn from Figure 5.1.

5.3. Evaluation A1 – Co-operative User Trials

This evaluation will observe individuals using the software to perform some simple PIM tasks, using a co-operative evaluation approach. The experiments seek to show how users can exploit the features of the software to help with these tasks. This evaluation does not seek to examine how individuals manage their personal information – to address this is well outside the scope of this study, and a stance on this issue has been taken and justified earlier in the dissertation (see sections 2.2, 3.3 and 3.4). Instead, we seek to establish whether the tools provided by the software are perceived as useful, and can be easily taken advantage of, by individuals familiar with the existing Windows NT graphical shell system.

The experiments are quite short in nature, and may therefore appear somewhat contrived. This is because they are meant to exercise as many of the features of the software as possible, while keeping the time necessary to perform the experiments within reasonable limits. A more prolonged study could provide in-depth results that might be more representative of actual work patterns, but this would introduce more variables into the experiment. This would make the focus of the evaluation less tight and more open to interpretation, where it would be preferable to restrict the scope.

5.3.1. Aims of the Experiment

Any evaluation process should be conducted with a particular set of aims in order to be useful and valid. The aims of this experiment are pragmatic and seek to establish the usefulness of the developed system. The experiment:

(i) aims to examine how well the software is perceived and thought about by the subjects;

(ii) aims to examine how well the software supports the set of tasks identified in sections 3.3.1 to 3.3.4;

(iii) aims to examine how well the subjects can use the software as developed.

These three aims are, from the highest to lowest levels of abstraction, meant to show the suitability of the software, its interface, and the underlying principles used to design and develop it.

5.3.2. Experimental Method

A number of subjects were selected, all of whom had been exposed to the Windows NT operating system and its user interface. The subjects were all members of the University research staff and research student cohort, and had differing lev-

els of experience with the Windows NT graphical shell – some used other computing systems almost exclusively whereas others spent the majority of their time working with Windows NT. Details of the subjects are given in Table 5.3.

Subject	Description of Subject
1	Year 2 Ph.D. student with extensive experience of Windows NT
2	Recent Ph.D. with extensive experience of Windows NT but also uses the X Window System under Unix
3	Year 1 Ph.D. student with extensive experience of Windows NT
4	Year 1 Ph.D. student with moderate experience of Windows NT
5	Year 1 Ph.D. student with moderate experience of Windows NT but almost exclusively uses the X Window System under Unix
6	Year 3 Ph.D. student with moderate experience of Windows NT but mostly uses the X Window System under Unix
7	Recent Ph.D. with extensive experience of Windows NT
8	Year 2 Ph.D. student with moderate experience of Windows NT

Table 5.3. Descriptions of experimental subjects.

The subjects were asked to perform a small set of document-based information retrieval and manipulation tasks (shown in Appendix A.1), using the prototype software to select and locate the files to be worked on. The tasks were to be accomplished in a specific order, as this would obviously have an impact on how the software responds to the access sequences observed.

Each of the tasks was meant to be accomplished using at least one feature of the software, and this was indicated to the subject through suitable guidance next to the description of the task to be performed. The subjects were told about the features of the software, but were not be given a direct demonstration of them – although they were be able to converse with the experimenter as they worked. The subjects were asked to pose any questions they wished, and to think out loud about the given tasks if they wished to do so. The experiments were carried out at the times and location types shown in Table 5.4 under supervision by the experimenter, who recorded (in note form) any points the subjects raised as they

worked. (A proforma for the observation notes is also presented in Appendix A.2).

Subject of Experiment	Date and Time of Experiment	Location Type
1	16/03/99 16:34 – 16:45	Closed-plan office
2	18/03/99 14:37 – 14:45	Closed-plan office
3	18/03/99 14:57 – 15:07	Closed-plan office
4	18/03/99 15:30 – 15:39	Closed-plan office
5	19/03/99 17:16 – 17:26	Closed-plan office
6	23/03/99 16:43 – 16:51	Closed-plan office
7	26/03/99 14:09 – 14:17	Closed-plan office
8	26/03/99 14:30 – 14:42	Closed-plan office

Table 5.4. Experiment times and location types.

Following the performance of the tasks, a short debriefing was carried out using a semi-structured interview. The experimenter asked a small set of questions designed to test how the subjects approached the tasks – having been given details of the software – and to test how they thought these features could (or should be) applied (see Appendix A.3 for details of the debriefing questions). This aimed to find out whether the subject appreciated how the features of the software could help them, how useful the subject found the software generally, and any opinions about possible improvements they might have had.

5.3.3. Results and Treatment

There were three distinct sets of results from the experiment, documented in Table 5.5. The event log and the observer notes were essentially two different views of the same thing, so the observer notes were useful in 'fleshing out' the basic event log information. This had the advantage that the observer was less concerned with noting the exact events taking place, since the software was doing this anyway, so that more attention could be paid to the subject's questions and thoughts.

Result Set	Nature and Means of Acquisition
Observer notes (see Appendix B)	Information about the subject's actions, questions and thoughts, recorded by the experimenter, as the subject progressed through the tasks.
Event log information	A timestamped 'journal' of user-interface events. The software automatically recorded the file-access actions performed by the subjects, for later perusal.
Debriefing notes (see Appendix C)	The subjects' opinions about the usefulness and applicability of the software, including any possible improvements suggested, acquired using a short structured interview after the completion of the tasks.

Table 5.5. Results from the co-operative evaluation.

The first two sets of results are used as the basis for a straightforward quantitative analysis of whether the features of the software were used as intended in the original design. The final set of results – the results of the debriefing interviews – is clearly less tightly-controlled in nature than the previous two, and is used for a qualitative review of the software's design and implementation.

Taken together the quantitative sets of results should illustrate whether or not the software is performing as intended, in supporting the simple tasks identified earlier in Chapter 3 (sections 3.3.1 to 3.3.4). The qualitative results should be useful in identifying issues not foreseen at the experimental design stage.

5.3.4. Possible Conclusions

The areas in which conclusions were to be drawn as a result of this experiment were threefold, and are presented in Table 5.6 in tandem with the questions that the results of each might be helpful in answering. All the questions presented in Table 5.6 should ideally be answered positively if the system's design and implementation accurately reflects the needs of the users, based on the stance taken on PIM in this dissertation. Any deviation from this should prove useful in later work, as it will highlight issues which need to be subject to further consideration

with respect to the presentation of the interface or the principles used in its specification and development.

Level	Questions
Conceptual	Does the system correspond to the users' ideas of how it might be used? Can they reason about the workings of the system and how it applies to the context within which it is to be used? Do the users feel that it adds value to the working environment?
Practical	Does the system provide adequate support for the sample PIM tasks previously identified? Do the users exploit the features of the system, or do they ignore them? Are the sample tasks made easier or less monotonous than they might otherwise be?
Usability	Does the system provide an obvious and logical interface to the PIM task support? Do the users feel in control of the system, or do they mistrust it? How might the appearance and function of the system be improved?

Table 5.6. Areas of interest in the co-operative evaluation.

5.3.5. Execution and Results

The experiment was carried out over a period of several days, when a sample of subjects were asked to carry out a set of simple tasks involving file manipulation and location, and were then asked a small number of debriefing questions designed to find out how they perceived the usefulness and applicability of the software.

The experiment was conducted in two parts. Firstly, a pilot study was undertaken using a single subject in order to uncover faults in the presentation of the interface and the instructions for the subject. The results of this were used to perform a small amount of re-development (which mostly consisted of adding 'Open' buttons to any dialogs which originally only used double-clicking to select items from lists). The main sample, consisting in this case of seven individuals, then carried out the experiments using the updated interface and instructions.

Table 5.7 shows basic statistical data about the utilisation of the prototype system's different features. This information was gained both from the observation notes

taken as the activities were being performed and from the machine-readable log files generated by the software. The statistics show that 90% of the activities were completed successfully by the subjects without explicit guidance from the investigator.

Feature and Activity Description	Usage
Shortcut Creation: Two repeated accesses to the same file, which should lead to the system suggesting a shortcut which could then be taken up by the user.	75%
File Tracking: Two uses of the file tracking system to locate and open files which were supposedly edited by the user several days ago.	100%
Note Viewing: A single use of a visible note overlay icon to indicate that a file has had a note attached to it, leading to the viewing of the associated note.	88%
Note Searching: A single use of the note search mechanism to locate and open a file which had a note attached, containing a particular word of interest.	100%
Average	**90%**

Table 5.7. Statistical usage data for the prototype system's features.

Most of the confusion arose over the descriptions of the activities to be performed by the subjects. A common problem was that the difference between a document and the note attached to it was not made sufficiently clear in the instructions. For example, the activities included one where the subject needed to look in a folder for a file with a note attached, open the note (for later use) then open the file itself and add some information to it. This was misconstrued by a number of the subjects, leading to some confusion. Some used the document itself instead of the note, some re-used the note overlay icon to find the information.

Another common problem was that, while the shortcut suggestion dialog could present a list of options, in the experiment it would only ever display a single suggestion. The re-designed dialog meant that the subjects could either double-click on a suggestion to accept it, or they could single-click to highlight it and then click an 'Accept' button to accept it. Most users commented that it would have been preferable to automatically highlight a singleton in the list.

In qualitative terms, the opinions of the subjects were consistent on some points, and more varied on others. All reported that they had no problems in seeing 'the idea' behind the system (that is, to use information about their activities to try to make the interaction more efficient), and several actually volunteered the opinion that it was attempting to make their 'virtual desktop' more like their actual one. All the subjects also said that that they could appreciate how the system would complement their abilities, in thinking about the specific mechanisms provided by it.

Most of the subjects (75%) asserted that, if the actual Windows NT shell contained some of the features present in the prototype, they would use them. Some subjects displayed enthusiasm for one or two of the mechanisms provided, but were not interested in others – one subject said they would definitely use the file tracking support, but would not use any of the other mechanisms. This is perhaps attributable to individual styles of desktop use, and use of computers in general.

None of the subjects attempted any form of anthropomorphisation of the interface based on their experience with it – although the interface makes no attempt to portray itself as a character of any sort. The subjects were not asked to think of the system as an assistant in its own right, and did not try to do so based on the features provided by it. This would seem to support the choice of the 'keep it simple' stance taken during this study.

5.3.6. Conclusions

Based on the results discussed above, the features provided by the system seem to appeal to users, who would then have few problems in using them alongside existing features of the Windows NT graphical shell. The overall impression gained from the user trials was that users appreciated the features provided by the system, and could see how they would use them. In terms of PIM system design theory, the findings from this study are limited, as this was not the real focus of the

experiment. However, it did tend to reaffirm the fact that different individuals can go about the same task in a wide variety of different ways, even for the simple activities used in the trial.

5.4. Evaluation A2 – Usability Inspection

This evaluation element concentrated on a literature-based inspection of the system, which aimed to arrive at a set of observations based on published usability guidelines. This provided a complementary evaluation of the system (as compared with the empirical evaluation reported in sections 5.3 to 5.3.6), allowing a more rounded view to be gained. The overall objective was to give a full impression of the system, from both a designer's and a user's viewpoint, which might not be possible with a purely empirical approach. The output from this portion of the evaluation should, ideally, have reflected the findings of the empirical work – reflecting their complementary nature.

5.4.1. Evaluation Aims

The specific aims of this portion of the evaluation were as follows:

(i) to highlight usability issues within the interface;

(ii) to show where elements of the interface are particularly useful;

(iii) to provide a guide towards the overall usefulness of the software, as far as possible, in its current form.

These aims, whilst at quite a high level of abstraction, illustrate that the guidelines to be used should provide quite broad advice, which can be refined to the specific needs of this evaluation. The next section deals with this issue in depth.

5.4.2. Choice of Guidelines

As well as establishing aims for this evaluation, the guidelines to be followed during the inspection are important. Generally, guidelines tend to be quite vague and consequently difficult to apply directly (Mosier and Smith, 1986) (although detailed guidelines can be too specific to be applicable in a given situation). Also, when applying guidelines in an inspection or critique, the context within which they are interpreted is also important – a guideline applied out of context will probably not provide useful results.

However, this fact has not deterred the HCI community from developing and publishing large sets of interfacing guidelines. There is a variety of sets of guidelines, each of which typically contain large numbers of guidelines for the interface designer to contend with, such as (Smith and Mosier, 1986; Brown, 1988; Mayhew, 1992). There will obviously be difficulty in applying large numbers of guidelines to an interface – problems in selecting a reasonable sample, problems in identifying the context in which each should be applied, and so on – so some kind of distillation of the ideas behind guidelines would be preferable.

Usability heuristics are designed to answer this need. They are typically arrived at by analysis of the application of design guidelines to system developments, noting which types of guidelines are applicable in which area of the system, and the effect they have. The usability heuristics to be used in this assessment are taken from (Nielsen, 1994) and are reproduced as Table 5.8.

The heuristics reproduced as Table 5.8 illustrate that most of the heuristics can be directly applied to almost any interfacing system. In addition, they provide concrete advice to the designer and inspector of a user interface, as well as guidelines for improving any interface that has shortcomings with respect to the heuristics.

System Area	Description of Heuristic
Visibility of system status	The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.
Match between system and the real world	The system should "speak the user's language", with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.
User control and freedom	Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and re-do.
Consistency and standards	Users should not have to wonder whether different words, situations or actions mean the same thing. Follow platform conventions.
Error prevention	Even better than the provision of good error messages is a careful design which prevents a problem from occurring in the first place.
Recognition rather than recall	Make objects, actions and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.
Flexibility and efficiency of use	Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater for both inexperienced and experienced users. Allow users to tailor frequent actions.
Aesthetic and minimalist design	Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
Help users recognise, diagnose and recover from errors	Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.
Help and documentation	Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

Table 5.8. Ten guidelines for heuristic evaluation (from Nielsen (1994) p. 30).

5.4.3. Guideline-Based Inspection

This inspection will use the ten heuristics presented above in Table 5.8 to examine the system, appraising its features in the light of each. For each of the heuristics there may be several elements of the system which are relevant, or there may only be a general sense in which the heuristic can be applied. In either case, the nature of the heuristics themselves imply that useful guidance will be forthcoming as the result.

Visibility of system status: The system uses visual indicators to show the presence or absence of file annotations, and also makes use of messages and cues to show when a shortcut suggestion has been made. In a sense, this heuristic (taken together with the next two) encapsulates the goal of any graphical shell application – that the underlying operating system should be presented to the user in a form they can easily relate to and think about.

Match between system and the real world: The 'sticky-note' metaphor is familiar to many people, providing a 'ready-made' idea of how the system works. At a trivial level, the note-entry dialog visually reflects a yellow note, reinforcing this impression. The text messages presented by the system (in terms of shortcut suggestions, and file-tracking descriptions) also use plain English.

User control and freedom: The shallow flexibility of a shell application – the provision of a consistent graphical representation of a possibly heterogeneous underlying environment – in conjunction with the prototypical nature of the system, means that this heuristic tends to be satisfied (at a trivial level) by default. The system's use of modeless dialogs (ones that do not suspend an application until they are dismissed) for the annotation support means that, if they so desired, users could transfer notes between objects or other applications – perhaps an example of not arbitrarily imposing a specific way of doing something. The requirement to offer an 'escape route', in allowing users to undo actions and re-do them if neces-

sary was not supported by the prototype – however, if full integration had been possible, the existing shell system would have supported undo and re-do anyway.

Consistency and standards: The system uses right-button menus ('context menus') for note attachment, which is consistent with the Windows NT approach. Context menus are generally available for any non-button object in the system – as a rule, the Windows NT consistency guideline is that if a graphical object responds to a left double-click operation to invoke some action (an icon, double-clicked to run, for example), a single right-click should show a menu with valid choices for that item.

Error prevention: This heuristic is applicable primarily to the process of file annotation entry, editing and deletion. The dialogs are designed to only offer sensible operations – that is, one can only 'update' a note if any changes have been made, for example – and simple questions are always asked before any annotation is deleted.

Recognition rather than recall: Although the system does not really have multi-stage dialogues as such, graphical cues (such as the presence of a note on a file) can help in recalling which file is needed. For example, the ability to show all annotated files (by entering an empty search string) can allow the user to find a file if all they remember is simply that it did have a note on it. In addition, the file-tracking support allows users to view a list of their recently-used files, allowing them to recognise the file desired, rather than having to remember it exactly.

Flexibility and efficiency of use: This heuristic would appear to be aimed more at the 'application' type of user interface, and has limited applicability in this situation. The dialogs used by the application do support keyboard accelerators, but the shell is a primarily graphical environment with only a limited amount of text entry. Having said that, the short-cut suggestion system will allow a user to cumula-

tively tailor their environment by accepting the system's suggestions from time to time.

Aesthetic and minimalist design: The dialogs used by the system are extremely simple, with a small set of options which have rigidly-defined effects. They are highly cohesive in nature – that is, each dialog is for one task only, with no other elements which can clutter the display and increase the load on the user. The graphical indicators used to signal that an object has an annotation are similarly unobtrusive, yet aesthetically pleasing.

Help users recognise, diagnose and recover from errors: The messages produced by the prototype system are all in plain English and contain no error codes. In fact, there are no error-style messages (apart from, perhaps, the 'search term(s) not found' message which appears in the Note Search dialog if a search term is not found).

Help and documentation: As the system developed was primarily an addition to an existing system, if users understand the original system they will have little trouble in exploiting the additional features provided by the prototype. Proper integration with the current NT shell would necessitate the addition of documentation to the shell's 'help' files – appropriate tools exist for this purpose, so this would pose no problem from a technical viewpoint.

5.4.4. Conclusions

Each of the ten system areas addressed by the heuristics presented in Table 5.8 has relevance for the prototype system. For the purposes of this discussion, the areas which concentrate on the visible portions of the system – and the ways in which it can be used – are the most important, particularly from the user's point of view.

The heuristics addressing these areas – specifically, those concerned with how the additional features provided by the prototype are presented to the user and can be integrated within their knowledge of the existing system – are satisfied to a great

degree. Areas which are not well supported are mainly due to the nature of the prototype, in that it does not fit the notion of a monolithic application, which is the traditional target of most usability testing.

To conclude, the system's use of graphical indicators and dialogs allow the user to see much of the system's state at a glance. The system conforms to the conventions expected by seasoned users of the environment within which it functions. The representations and metaphors used by the system match the user's real world, making it easier to think and reason about. The interaction mechanisms implemented within the system are as simple as possible. The system therefore seems to satisfy many of the criteria which constitute the heuristics used for this inspection.

5.5. Evaluation B1 – The AIT Architecture

The specification and design of the prototype system was largely influenced by an existing theoretical architecture for the development of adaptive interfaces. (See section 2.6 for the introduction and discussion of the AIT architecture, and section 3.7 for its application in the design of the system). This section examines the AIT architecture, both as a whole and in terms of its component parts, to show where it was particularly useful in specifying the system, and where guidance was lacking for this application.

One of the benefits of using an existing architecture for specifying and designing a system is that is should provide concrete advice for the developer. The architecture, as presented (Benyon, 1993), is reasonably abstract in nature, which is to be expected as it is meant for use in a wide range of applications. Although quite abstract, the architecture is mainly prescriptive in nature, providing a ready-made breakdown of the different parts of an adaptive system, and advice on how to conceptualise the system as a whole.

5.5.1. Evaluation Aims

The overall aims of this section of the evaluation are as follows:

(i) to examine the AIT architecture as a theoretical tool for the specification and design of adaptive interface systems;

(ii) to highlight areas where additional information is required in the specific application area of PIM;

(iii) to identify more general, future work in adaptive interfacing.

The issues arising from this evaluation will then be used in the formulation of suggestions for improvements that could be made to the AIT architecture and additional design guidance for adaptive systems in general.

5.5.2. The AIT Architecture – An Appraisal

The power of the AIT architecture comes mainly from its simplicity – the fact that is provides an almost 'cook-book' approach to the specification and design of an adaptive system. When approaching the design of a system that could benefit from adaptive behaviour (or any novel technology, for that matter), there is a great temptation to focus purely on the deployment of 'adaptivity' for its own sake, without paying due regard to ensuring a sound theoretical basis upon which to found the system design. The AIT architecture provides a structured, accessible 'template' for the specification of an adaptive system – a set of pigeonholes which each have a contribution to the system as a whole, but could be individually forgotten.

The different components of the AIT architecture each reflect a facet of the overall system's behaviour. The Domain Model is used in an abstract fashion to provide a scope for the possible adaptive behaviour of the system, and to give a kind of 'dictionary' of the concepts and terminology usable by the system. The User Model,

with its different levels of information about the user – psychological and cognitive data where appropriate, profile data and a learning-based 'student' model – is used as a template for a store of information about individual users. The Interaction Knowledge Base contains a history of the user's interactions with the system, and sets of mechanisms for arriving at, effecting and evaluating any changes made by the system – realised, in this application, by a set of simple software agents.

In short, the AIT architecture gives a good starting point for the specification and design of an adaptive system. It can be used purely in a prescriptive manner, as it provides ample guidance as to the components of an adaptive system and the information required by each of them. However, each application will be different – the specific area, the required adaptive behaviour, and so on – and the PIM application is no different. There will always be situations where good judgement is required, as in many design techniques – without forethought and planning, almost any method can be rendered ineffective.

As it stands, the architecture also provides little guidance as to the exact nature of the 'active' components of the Interaction Knowledge Base (those used in adapting the system's interface). These can be implemented in a wide variety of ways, using technologies ranging from simple active rules up to far more sophisticated techniques such as those generally classed as 'artificially intelligent'. Although perhaps a matter of common sense, a good match must be made between the needs of the system's users, in terms of the adaptivity necessary to support them, and the technology deployed to effect the required adaptivity. In this application, the adaptivity required was provided by relatively simple agent-based inferencing mechanisms using reactive techniques, which appeared to give a good match based on the results of the empirical study.

5.5.3. Conclusions

The preceding discussion has demonstrated the utility of the AIT architecture as a means for the specification and development of adaptive systems in general, and in particular of the prototype system developed in this study. The straightforward nature of the architecture and the consequent ease of application results in a useful theoretical framework. Although quite open in nature, relying to a degree on good judgement on the part of the designer, effective results can be achieved using it. The fact that the architecture does not prescribe specific implementation techniques for the active elements of the system has also allowed the prototype to demonstrate the usefulness of the reactive agent paradigm in the implementation of an adaptive interface – making a good match between the simple, concrete nature of the tasks supported and the adaptivity supplied.

One element of the AIT architecture which was not used in this application is the psychological level of user data in the User Model. Typically, this contains cognitive data about the user – such measures as spatial ability, for example. The discussion of the nature of PIM as a whole (see sections 2.2 to 2.2.5) highlighted that a large part of PIM tends to be based on commonsense reasoning, especially concerned with information management practices. This would imply that there might be scope for supporting more complex PIM activities, in terms of abstract cognitive processes.

5.6. Evaluation B2 – Design Process

Having used the AIT architecture as the basis for the specification of the system's adaptive behaviour, the next task was to develop a design for the system. This was done by following an object-oriented (OO) approach, using the components of the AIT architecture as a basis for the system's internals, and an interface-based development technique to reflect both the internal and external communication

needs of the system. This evaluation will assess the design process to show where these techniques proved useful and where additional support could have been taken advantage of.

5.6.1. Aims

The aims of this portion of the evaluation process are closely related to some of the key goals of software engineering in general (Coad and Yourdon, 1991; Booch, 1994; Sommerville, 1996; Pressman, 1997). In these terms, a design is good if it exhibits desirable characteristics such as: evidence of the effective use of the strengths of the OO approach; that the separate elements of the system are cohesive in nature; that the system is as free of coupling as possible; and that the system is maintainable – well structured and commented, with good use of development language and library features to aid understandability; and that elements of the system are re-usable as far as possible.

The objectives of this element of the evaluation are therefore as follows:

(i) To illustrate how the OO paradigm of software development helped in the design process;

(ii) To provide an understanding of how the application of an interface-based development paradigm can result in an elegant system which can provide opportunities for subsequent re-use;

(iii) To show how the design process was aided (or otherwise) by using the elements of the AIT architecture as a template for the set of objects in the system design.

The issues arising from this evaluation will then be used to inform any re-design activities necessary, and to provide an insight into the practical implications of developing software to be integrated into component-based environments.

5.6.2. Evaluation

The decision to use an OO approach for the design of the system was not a difficult one – casting the internal structure of a system as a set of objects, each with a set of methods, is quite natural to do, and is a widely-used technique (Norman, 1996). The AIT architecture, as discussed earlier, is presented in the form of a structured framework, which provides a natural template for a set of objects with which to implement it. From a practical perspective, the environment within which the system was to be implemented (i.e., Windows NT using the MFC application framework, which is supplied as a set of C++ object classes) would alone have justified an OO approach.

The goal of implementing the system as one that would be integrated into an existing environment had a definite impact on the design techniques that were used. A stated goal of the design process was that, as far as possible, implementation and platform-specific issues should be deferred within the design process as long as it was practical to do so. The result of that goal is that the abstract design for the system (see section 4.4) is almost entirely free from system dependencies, and could be implemented as a standalone component under any operating system which shared the currency of files and folders.

This is largely due to the use of an interface-based development process, where the usual OO principles of encapsulation are applied, but also where the object itself is *totally* opaque – other objects, even within the same system, simply have a reference to an *interface* of an object or component – a kind of 'contract', specifying available methods and their functionality. Interfaces were used to define the relationship between the system's internals and the external portions of the final implementation environment, while at the same time reducing interdependencies between them.

Interface-based development does have some drawbacks – mainly that the design of objects which could be used as components in their own right can be complicated rather by the need to have complete, cohesive interfaces. The positive aspect of this is that a component, once designed and implemented, can be re-used at a binary level without any further intervention, offering considerable opportunity for systems that can be tailored by end users.

5.6.3. Conclusions

This element of the evaluation has illustrated the complementary usefulness of the OO software development approach and the interface-based development techniques that rely upon it. The AIT architecture provided a good starting point for an 'initial cut' of the system's structure and functionality, which was then developed further. The application of a component-based approach provides benefits for the design of systems which are primarily to be integrated within existing environments, but can introduce conflicts between strictly OO principles and the practical requirements of components.

As ever, the experience of the designer is paramount in the design of easily-integrable components. As a by-product of the component-based development process, the prototype system described in this dissertation could be implemented as a COM component in its own right, using, as sub-components, the set of agents which provide the system's adaptive behaviour. This would then permit the system to be tailored by end users to the point where they could specify and implement their own agents to be integrated into the environment.

5.7. Evaluation B3 – Implementation

The prototype system was originally designed to be implementable as a set of extensions to the existing Windows NT shell – due to a range of technical factors (documented in sections 4.6.2 to 4.6.9) this was not directly possible. However, the

system's prototype implementation was realised as a skeleton application sharing many characteristics with the NT shell, and this evaluation examines the common ground between them, with particular reference to the component-based development processes used. Also discussed are the application frameworks and software development tools used, to give an insight into the technical and practical issues encountered in the development of contemporary interactive, graphical applications such as the output of this study.

5.7.1. Aims

The objectives of this element of the evaluation are as follows:

(i) To reflect upon the process followed in implementing the software, and the nature of the resulting working prototype system;

(ii) To examine the 'integration' of the system with the Windows NT shell and operating system and to appraise the current state of component-based software development and the opportunities and pitfalls associated with it;

(iii) To highlight technical issues in the development and implementation of graphical systems using the Windows system and the MFC application framework and to report on the effectiveness and usability of the tools used in the development of the prototype.

The results of the evaluation will be used to offer suggestions for the likely future development of open shell applications and to provide an insight into the current technical barriers to such developments. The strengths and weaknesses of contemporary development environments in supporting the development of these applications will also be discussed.

5.7.2. Evaluation

The design of the system, having been undertaken using the proven technique of OO design, and the emerging approach of interface-based development, reflects many of the desirable properties of a software system. The core of the system is largely platform independent, using a set of interfaces to communicate between itself and the external environment. With minimal effort, a COM-compliant implementation of the system would make these interfaces explicitly available to the outside world, allowing radical tailorability of the system's adaptive behaviour.

The (technical) failure to properly integrate the system as a part of the Windows NT shell can be seen as a side-effect of the relative infancy of component-based software development as applied to interactive systems. The techniques used to develop software have progressed from purely imperative programming systems, through OO languages and environments, toward interface-based component-based approaches. As these changes have occurred, the systems available have expressed these changes in such technologies as Microsoft's OLE, allowing objects of one type to be embedded within objects of another. Some (Brockshmidt, 1995) postulate that the next 'revolution' in computing will be based entirely around the document, and 'the application' – as it is currently known – will cease to be the focus of software development as far as the end-user is concerned.

The development environment used in this project (Visual C++, and the MFC application framework) provides a variety of tools and support for the development of graphical applications. MFC follows an OO development model, which is complemented by the tools available – visual interface design tools, and structured browsing support, allowing the developer to explore the structure of a project in terms of its classes and methods, rather than files of source code.

5.7.3. Conclusions

The implementation process has suggested that the architectures and designs used in the specification and development of the system have had beneficial effects on the final implementation process. The prescriptive nature of the AIT architecture provided a sound basis for the OO structural design of the system, and the component-based development approach enabled the communication possible between individual parts of the system to be rigidly specified. These structured approaches aided the compartmentalisation of the system's overall functionality, helping the decomposition, refinement and final implementation processes.

The choice of implementation environment was perhaps a mixed blessing – the tools and infrastructure provided by the Windows NT platform were helpful, yet the integration process (as originally envisaged) proved virtually impossible. Although the Windows NT shell does provide support for 'shell extensions', it only allows the shell to be extended in very particular ways, which did not accord with the needs of the prototype system.

5.8. Synthesis of Evaluation Results

This section will use the structure of the dissertation topic roadmap (introduced as Figure 5.1) to retrace the issues raised during the five component evaluations (A1, A2, B1, B2, B3). Each issue may have been raised by one or more of the evaluation techniques used, and each may apply to one or more of the four levels identified in Figure 5.1. The discussion will follow this four-level structure in order to relate the issues raised by each evaluation back to the relevant parts of the dissertation and, indirectly, to the literature to which the issue refers.

The output of this synthesis process will be twofold: a set of outstanding (mainly technical) issues which can be addressed by means of re-design work in Chapter 6;

and a set of more philosophical issues which will form the basis of the dissertation's contribution, which will be discussed in Chapter 7.

5.8.1. Issues/Abstraction/Evaluation Matrix

Table 5.9 summarises the output of the different evaluation approaches followed in this chapter. This table presents an overview of the issues raised during this study, characterising them in terms of the level to which they apply, and the evaluation technique from which they resulted. A brief description of the issue is given, and the subsequent columns show at which level of abstraction the issue applies (these levels relate back to the original topic roadmap, Figure 5.1), and by which of the evaluation technique(s) the issue was originally highlighted. Where issues apply at more than one level of abstraction, or were encountered in more than one evaluation technique, a large 'X' denotes the more significant level or technique.

Table 5.9 shows that the evaluation work done as part of this study has resulted in a wide range of issues being raised, which cover the different levels of abstraction – and hence, the whole body of theoretical and practical work – present in the dissertation. Given the range of topics covered in this dissertation, this would seem to vindicate the choice of evaluation approaches. The table also shows that there is a reasonable correlation between the levels of abstraction of the issues, and the evaluation technique(s) used to demonstrate them.

The following sub-sections address each of the levels of abstraction in turn, discussing the issues relevant to that level. Some issues apply to more than one level – these are discussed in detail at the higher level, and are mainly referred back to when dealing with the lower levels of abstraction, with appropriate explanation.

		Abstraction Level				Evaluation Technique				
Issue	**Brief Description of Issue**	**L1**	**L2**	**L3**	**L4**	**A1**	**A2**	**B1**	**B2**	**B3**
1	Opportunities for providing support for 'integration'.	X				X	x			
2	Applying 'appropriate information provision' principle to a filesystem.	X				X	x			
3	Provision of more complex delegation raises problems of trust.	X	x				X			
4	Support for tailorability and personalisation within the interface.	X	x			x	X			
5	Opportunities for more complex reorganisation tasks.	X	X			X	x			
6	Representation of agent 'characters' as explicit interface elements.	x	X			x	X			
7	Shortage of design guidance for adaptive systems developers.		X	x				X	X	
8	Modelling and behavioural specification using AIT architecture.		X	X				X	X	
9	Definition of criteria for user file rearrangement based on content.		X	X				X	x	
10	Use of agents as a partitioning device for adaptive behaviours.			X				x	X	
11	Tailorability of adaptive elements of the system as binary components.			x	X				X	x
12	Infrastructure required to augment AIT architecture in design.			x	X					X
13	Differences between OO designs and implementation reality.			x	X					X
14	File annotation could be widened to use COM/OLE embedding.				X				X	x
15	Using AIT as a specification template gives a non-optimal design.				X					X
16	Tool requirements for packaging/distributing components.				X					X
17	Inelegant use of local files to store dialogue record information.				X					X
18	Difficulties in acquiring event information restrict adaptive systems.				X					X

Table 5.9. Issues, abstraction level and evaluation technique matrix.

5.8.2. L1 – PIM and Interfacing: Principles and Theory

The discussion at this level is aimed mainly at assessing the validity of the work done in this study, in terms of the needs of the users of the software. While the focus in this work was not to develop theories concerned with the way that people manage their personal information using computers, this study has given some insights into the usefulness of software meant to support them in doing so. This section addresses some of wider issues concerning the nature of PIM and the impact it has on user-interface systems meant to support it, showing how the prototype system developed as part of this study satisfies some of them.

Based on a stance taken at the beginning of the study – that a profitable route would be to provide PIM systems which exhibit simple adaptivity – the relevant evaluations sought to establish that software based on this premise is useful. On the whole, this assumption has been borne out, particularly by the results of the user trials, which were promising (see sections 5.3.5 and 5.3.6). As comments from the usability evaluations show, the use of agency as an enabling technology – as transparently as possible, to avoid unduly loading users of the system – seems to be a good approach in the context of this study. The use of adaptivity – in an obvious manner, ensuring that the user maintains a sense of control – also appears to provide a good approach to the provision of PIM systems.

In Chapters 2 and 3 (see sections 2.2.1 and 3.3), some simple task elements were identified within PIM as a whole, and it was shown that these provided scope for automated support. In particular, the activities of storage and retrieval underly almost any information management system. The prototype system allows users to perform these task elements by providing most of the standard shell's functionality in a transparent manner.

As well as storing and retrieving information, PIM systems should also to be able to support the individual in organising their information, aiding the storage and

retrieval process in order to ease the recall of documents. The adaptive element of the system observes the user's file manipulations and offers suggestions accordingly, as well as providing a historical record of the set of files used. The usability evaluations showed that individuals were able to use the support provided by the system to aid location and organisation of their files, and that these ideas were easily thought about by the individuals involved. However, the trials and the heuristic evaluations did raise some issues of interest.

As mentioned earlier in Chapter 2 (see section 2.2.1), a desirable characteristic of more sophisticated PIM systems is *integration* (in the sense of combining disparate forms and modalities of information to exploit relationships between them). As it is currently implemented, the system offers little scope for this – the prototype is not well integrated (in the implementation sense) within the final operating environment, so the scope for the integration of the information used internally by it is limited (issue 1). This is demonstrated by the usability inspection, as well as (implicitly) in the design assessment. However, this appears to be the result of the stance taken in the study that simpler, more controllable systems are preferable – to address integration more fully would have required a different approach.

Similar comments apply to the characteristic of 'appropriate information provision' (see section 2.2.5) – although this would imply some kind of information-filtering application (for example, a system which can automatically redirect or dispose of e-mails based on a combination of subject and content), it can be applied to a file-based storage system (issue 2). In a sense, as a user works with the system over a period of time, it will attempt to optimise the selection of files by offering shortcuts to those which appear to be badly placed with respect to their patterns of usage, as demonstrated by the usability trials – this might be argued to be the provision of the appropriate information.

In using the prototype system, the user is effectively delegating part of the task of 'being organised' when using a computer-based storage system. Admittedly, the 'delegation' which happens in this system is very simple in nature – although this simplicity does mean that it is much easier to relate the actions of the system to the actions of the user, as mentioned in the usability inspection. If agency is applied to more complex tasks, issues of trust soon become something of a problem (issue 3). For example, instead of providing shortcuts to frequently-used files, the system could attempt to optimise the user's file store by archiving and deleting files which are not used – this would require the user to be able to trust the system much more than when purely non-destructive adaptivity was used (Milewski and Lewis, 1997).

The tailorability of the system is reasonably limited due to its nature as a prototype – the existing Windows NT shell provides some customisation facilities in terms of user preferences, but true 'tailorability' (in the sense of deferred design) is not present – this was highlighted by the usability inspection (issue 4). The prototype does provide support for personalisation, mainly in terms of the file annotation feature. Individuals each have their own set of notes – this could be useful in recalling the locations of shared files, where each user would have access to the same underlying file system, but would have a personal 'view' of it.

As demonstrated by the usability tests, and to a lesser degree by the inspection, the re-organisation attempted by this system is quite basic in nature, and it might therefore be possible to provide more comprehensive support (issue 5). While the existing system provides support for organisation of files based purely on usage patterns, an issue to be addressed is the provision of alternative ways of accomplishing this.

The usability inspection also demonstrates that the prototype system does not make any attempt to actively portray itself 'as an agent' – i.e., no cartoon charac-

ters or other anthropomorphic representations are employed (issue 6). From a technical viewpoint, it is possible to make 'agents' more explicit in appearance – an example being the Office Assistants provided by Microsoft, implemented using the Microsoft Agent API (Microsoft, 1997). In reality, however, these animated 'characters' are only used as a graphical front-end for a context-sensitive help system and a documentation search tool. The use of explicit agent characters is a difficult issue in user interfacing, as it is important to provide the user with an accurate expectation of the level of 'intelligence' possessed by these characters (Norman, 1994). As the prototype system does not attempt to be intelligent – merely to offer sensible suggestions in a timely manner – this issue is largely one for future consideration.

5.8.3. L2 – Adaptivity and Adaptive Interfaces: Frameworks

This section begins to address some of the more technical aspects of the development undertaken in this study, addressing issues concerning the use and application of the AIT architecture in the development of the system, refining and augmenting the components and models as necessary. This level of abstraction is largely concerned with the theoretical architectures used in the specification of the software as a system with an adaptive user interface.

In Chapters 2 and 3 (see sections 2.4 and 3.5), a conceptualisation, specification and design approach based on adaptivity in user interfacing was proposed, and this section examines that approach used in development of the system and its interface – in particular, the use of adaptivity – and the use of agents within a framework for adaptive interfaces. The AIT architecture effectively provides a template for the specification and design of adaptive systems, based on a set of models. This 'template' gives the designer of an adaptive system an analytical framework which can be used to deconstruct the interconnected details concerned with a

given problem, yielding a design for a set of models and mechanisms which, when implemented, will realise the adaptive behaviour desired of the system.

Essentially, the AIT architecture is applied to partition the problem into manageable components. However, this process is reasonably subjective in nature (as is the case with many design tools). To implement the generic AIT architecture for a specific case requires additional knowledge about the application. In addition to the modelling input needed to 'flesh out' the template of the AIT architecture, the addition of a set of simple reactive agents brings its own requirements. The area of adaptivity and adaptive interfaces shares some aspects of a selection of the issues noted in conjunction with the principles and theory of PIM, such as the provision of more complex delegation and the trust required by it, the opportunities for more complex re-organisation tasks, and problems concerning the representation of adaptive agent 'characters' as explicit interface elements.

The evaluation of the AIT architecture showed that there is a shortage of design guidance for developers of adaptive systems, as they tend to be quite niche-oriented. This is not necessarily a bad thing, as specific focused solutions to particular problems can be more effective than wide-ranging, shallow systems. Having said this, there is a need for the ability to generalise experience gained from point solutions (issue 7). A related point is that the modelling approach taken in this study was to use the AIT architecture as a template for the possible information that might be required, and then to partition this into the separate models (issue 8). More guidance as to how to perform this modelling and partitioning would therefore be useful.

In the previous section, opportunities were identified for providing more comprehensive support – for example, automatically arranging a user's files could be accomplished in alternative ways. A process such as this obviously requires a set of criteria for doing so – instead of usage patterns, a content-based approach could be

followed, for example (issue 9). Various techniques exist for inferring user interests from document contents – for example, much work has been done on this in the arena of Web page recommendation (Crabtree *et al.*, 1999).

5.8.4. L3 – Agency and Software Agents: Architectures

This section addresses the ideas behind agency itself, as well as the more technical details concerning the technology used to implement the agents in this study. The use of agency was driven by the idea of indirect management, provided through adaptive systems. The concept of software agents was appropriate since an adaptive system may be conceptualised as a user interface which contains a set of agents, where each agent may be responsible for a portion of the system's adaptive behaviour as a whole.

The choice of a particular agent technology was made in Chapter 2 (see section 2.9), where reactive software agents were recommended based on the fit between the qualities of reactive agents and the requirements of software meant to integrate within a user interface. The use of reactive agents in an event-driven environment such as the system developed illustrates that the reactive agent paradigm fits well with the real-time needs of a user interface.

As was the case in the previous section, an overlap exists between this area and the previous one, primarily in the shortage of design guidance for adaptive systems developers which applies equally well to those developing agent-based systems. Additionally, modelling and behavioural partitioning using the AIT architecture has an impact. Partitioning the total adaptive behaviour required of the system into component parts for implementation as a set of agents can be difficult – some guidance would be useful here (issue 10). In the case of this system, the adaptivity required was quite simple, and the approach taken was to decompose the behaviour required into a set of levels and implement an agent for each. It is

not obvious how more complex adaptivity would be handled, as the approach taken here might not prove to be scaleable.

On a more practical note, the design evaluation highlighted the possibility for the provision of tailorability of the adaptive (and possibly passive) elements of the system (issue 11). This could be achieved using a binary component model, which would then permit the swapping of components to perform different tasks. This is somewhat related to the fact that the active mechanisms of the AIT architecture (inference, adaption and evaluation mechanisms), once implemented as a set of agents, also required infrastructure in terms of event acquisition, distribution and feedback (issue 12). Both issues could be addressed by using a software infrastructure which treated the agents (and possibly other elements of the system) as opaque components.

The AIT architecture is cast in terms of a set of models augmented, in this situation, by a set of agents. With such a set of readily identifiable objects, a design process based on the OO paradigm recommended itself. However, many OO design paradigms have roots in system-based design methods, where specification and design often takes place in something of a 'vacuum' – for example, objects are largely assumed to run in parallel, whereas in a final system implementation, they might have to execute sequentially (depending upon the target platform). This mismatch was alleviated to a greater degree in this project due to the use of component-based development to formalise the intercommunication needs of objects, and using method calls to reflect event occurrences (issue 13).

5.8.5. L4 – Realisation of the Prototype System: Design and Implementation

This section discusses the final design and implementation stages of the study, where a combination of OO and component-based design approaches were used to partition the system into implementable parts. The process of integrating the software into the target platform's operating environment is also addressed. This

section contains technical details about shortcomings uncovered in the evaluations which could largely be addressed by re-design work. Some outstanding issues from the previous section which also apply here concern the tailorability of adaptive elements of the system as binary components, the infrastructure required to augment the AIT architecture in design, and the differences between OO designs and implementation reality.

As mentioned, an OO paradigm suited the implementation of the AIT architecture. The use of a component-based development process – where all communication between objects occurs via interfaces – further aided the partitioning of the system into implementable elements. The use of a component-based approach had other benefits – since inter-object communication is characterised in terms of rigidly-specified interfaces (sets of methods) with each objects' internal detail entirely hidden, integrating the system with an existing environment would have been eased from a technical viewpoint.

In addition, the use of a binary component standard such as COM (Box, 1998a) allows a level of tailorability to be provided, with minimum re-development effort. This is because components are typically cast as interchangeable binary entities, so alternative sets of adaptive elements (i.e., components containing the reactive agents used in the system), could be provided and re-configured by the end-user.

As regards the final integration process, the adaptive elements of this system were to be integrated with an existing non-adaptive graphical shell interface, because they were meant to subtly enhance the interaction, rather than be the centre of attention. This integration process posed substantial problems, due to a variety of technical and design issues. These were covered in detail in Chapter 4 (see section 4.6), but are largely the result of only partial adoption, by system providers, of the principles underlying open document-centric computing.

The design and implementation evaluations showed that while the file annotation feature currently works purely within the shell system, the metaphor could be extended to work across applications (issue 14). This could be implemented under Windows NT using a combination of the COM infrastructure and the OLE object embedding mechanism to provide embedded notes within compound documents.

In addition, while the AIT architecture provided a good starting point for the design of the system, the information needs of the different components of the resulting system meant that some of the system's design was not as elegant as possible (issue 15). This was due to such factors as the detailed intercommunication requirements between the agents within the system and the dialogue record and user models, which tended to clash with the OO design ideal of encapsulation.

There is a requirement for the use of an automated tool for packaging the adaptive components of the system and for distributing them (issue 16). This would promote integration and tailorability – the system could be constructed as a component in its own right, containing interchangeable adaptive elements, realised as components. The prototype system was implemented using an hand-coded emulation of COM, which would not provide the ability to package, distribute and reconfigure elements of the system. However, a COM-compliant implementation would be feasible, particularly given the support provided by Microsoft's ATL (Active Template Library) which supplies classes which hide component implementation issues and can be used to develop COM components directly from existing objects.

The use of flat files to store information about the dialogue record, note data and file access logs is inelegant, and makes sharing this information difficult, where features exist within Windows NT – notably the Registry, a centralised database of application information – which can be used to good effect in this situation (issue 17).

In platform-independent terms, the adaptive elements of this system were integrated with an existing non-adaptive graphical shell interface, because they were meant to subtly enhance the interaction, rather than be the centre of attention. This integration process posed substantial problems, reported in section 4.6, owing to several factors. Most importantly, differing platforms have different means of accessing information about user actions. The specific nature of the information required by this system means that it can be difficult to gain access to it (issue 18). An aim of the design work was to make the resulting system as independent of architecture, platform and operating system as possible, yet the impact of the specific nature of interaction event acquisition on the development was considerable.

5.9. Key Re-Design Issues

This section summarises the issues discussed already which will be directly addressed as part of this study, to lead to a re-working either of the system's design or the theory used in its development. These issues will then form the basis for the re-design work presented in Chapter 6.

5.9.1. Issues for System Re-Design

The issues described here are primarily technical in nature, arising from levels 3 and 4 of the evaluation, and illustrate opportunities to take greater advantage of the technological support present in the target operating system environment.

Windows NT provides 'roving profiles' (so that a user's personal profile data can 'follow' them around a networked installation). The current system uses local files to store profile data, which precludes this, and also makes it difficult for other applications to share the data. The operating system's support for the Registry will therefore be exploited to enhance the mechanisms used for profile data storage.

The evaluation highlighted the opportunity to allow the object-annotation metaphor to be extended by providing a system whereby note objects can be directly inserted into compound documents (e.g., word processor documents). The linking and embedding technology present in Windows NT will therefore be utilised to develop an OLE server which can allow note objects to be handled in this way.

Finally, the evaluation suggested that the provision of tailorability using explicit binary components would be desirable. The COM component infrastructure provides the ability to dynamically reconfigure systems using 'plug-and-play' binary components, and this technique could be used to open up the system, allowing end-users to tailor the adaptive behaviour of the system.

5.9.2. Issues for Theoretical Reworking

The issues described here are primarily theoretical in nature, arising from levels 2 and 3 of the evaluation, and illustrate opportunities to take greater advantage of theory and technological support present in the wider context of adaptive and reactive systems design and implementation.

The evaluation suggested that the system's design based on the AIT architecture was non-optimal in terms of implementation complexity and inter-object communications requirements. This re-working will seek to address the problems uncovered in implementing the AIT architecture directly in terms of objects using a component-based approach.

The evaluation also suggested that a more rigorous approach to the specification and design of the behaviours of reactive software agents would be desirable. The ad-hoc approach taken to the decomposition of the system's desired adaptive behaviour will therefore be formalised to yield design guidance for future developers of adaptive systems in the form of a level-based behavioural partitioning technique.

The problems encountered at integrating the prototype system into an existing user interface environment suggest that this issue needs to be looked at. Although perhaps an implementation issue, within a wider context, the feasibility of a general event information notification mechanism will be examined. This will seek to show how platform-independent event acquisition might be accomplished, based on existing methods used by current operating systems.

5.10. Key Issues for Further Discussion

This section summarises some issues which are either beyond the scope of this project, or serve to illustrate useful avenues for future research. These issues will then drive the discussion in Chapter 7 which aims to show clearly the contribution made by this study.

The evaluations have uncovered some interesting points about how PIM activities relate to adaptivity in user interfaces and how these can be examined in terms of the concept of agency, which will lead to a discussion of the relationship between PIM, adaptivity and agency. Additionally, the support for PIM provided by the prototype is reasonably basic, but there is certainly scope to support more complex task elements such as integration.

Addressing wider issues, there is a question concerning the role 'intelligence' has to play in controllable, predictable, trustworthy user interface systems. Allied with this is the users' perceptions of adaptive interfaces – adaptive systems change over time, but many users may be apprehensive of an interface which promises to do so. There are also opportunities for the application of the concept of agency in related areas. The concept of agency is quite simple and attractive, but this is no guarantee that it would be applicable to any given area.

5.11. Conclusion

This chapter reported the results of five separate evaluations of aspects of the work presented in the first four chapters of this dissertation. Two usability evaluations were carried out, one based around a live user trial, and one based on a usability inspection driven by published guidelines. The framework and theory used to design the system was critically examined, as was the prototype's design, and the system's final implementation.

These different viewpoints give a picture of the system's strengths and weaknesses, from practical and philosophical perspectives. The results of these assessments were summarised as a list of key issues which are carried forward into the next chapter. This will address some of the more tractable problems identified in order to provide possible solutions for a re-design of the work presented. The more wide-ranging philosophical issues identified will be deferred until Chapter 7, where they will be used (in conjunction with the other issues highlighted) as the basis for a discussion of the contribution made by this study and to frame future work.

Chapter 6

Re-Design Work

6.1. Introduction

This chapter presents some practical solutions and theoretical approaches to address problems uncovered during this study, based on the issues discussed at the end Chapter 5. This study raises issues at various levels – from the technical details of implementing open interface systems using interface-based software components, up to philosophical and intellectual issues concerning the inclusion of intelligence in interfaces and the impact this has on usability.

This chapter considers the practical aspects of the issues raised in this study, by presenting re-design work which addresses problems identified by the evaluations in Chapter 5. The re-design activity occurs at two levels. Firstly, at a practical level, some elements of the system are augmented by exploiting additional features of the target platform used in the implementation of the software. Secondly, at a more abstract level, some suggestions are put forward for ways to improve the theoretical tools available to help with building adaptive systems in general.

The more philosophical issues raised in Chapter 6 are deferred until Chapter 7, where they are examined in context of the work done in this study, as well as the wider context of the state of the art. This will aid distillation of the key issues raised in order to state the contribution made by this dissertation more precisely.

The next two sections introduce the issues to be dealt with and provide some of the background to them, in preparation for the detailed discussion of the re-design work which follows.

6.1.1. System Re-Design – Areas to be Addressed

This portion of the re-design work is primarily based on some of the practical shortcomings found, and the associated opportunities for improvement of the software itself. The result of this work will be a re-designed system which takes more advantage of the technological possibilities provided by the target platform and is therefore better integrated within it. The re-design activities are concerned with three aspects of the system, using platform-native support to re-implement elements of the current prototype: storage of personal user profiles; embedding and manipulation of notes; and providing scope for end-user tailoring of the adaptive elements of the system.

The current system uses plain local files to store profile data. Although this is the simplest and quickest route to take when designing and implementing a prototype system, it is not the preferable solution, for several reasons. The files' formats and locations are known only by the application, making the sharing of the information held within them difficult and hazard-prone. In addition, the user's profile data is held in a static location, so it would not be able to travel with them.

Windows NT provides a system database called the 'Registry', which is a structured repository of information concerning many aspects of the operating system. Specifically of interest for this system, it maintains secure per-user profile areas

which can hold application-defined information about individual users. The Registry also supports 'roving profiles', so that a user's personal profile data can 'follow' them around a networked installation without any intervention by the user or the application. Support for these facilities will be incorporated into the prototype system to remedy the shortcomings mentioned in the previous paragraph.

Another area for improvement partially revolves around the component-based approach taken during the design and development of the software, and partially refers back to tailorability, one of the desirable attributes of PIM user interfaces discussed in section 2.2. Although the prototype system was not directly implemented as explicit components, it did follow rules of interface-based design and implementation, thereby reducing coupling between modules of the system so that a component-based implementation would be possible.

Windows NT uses the COM (Component Object Model) component infrastructure for many application programming interfaces (in fact, COM underlies the OLE mechanism to be used in developing the note objects discussed earlier in this section). To be COM-compliant, a component needs to implement certain standard interfaces, and to respond to method calls on those interfaces in standard ways. The exact specifications are not important here (they will be discussed in the relevant section), however, these standard interface protocols are implemented in the Active Template Library (ATL) supplied by Microsoft.

Once implemented as components, the mechanisms provided by COM promote the ability to dynamically reconfigure systems using 'plug-and-play' binary components – that is, systems can be reconfigured under software control with no need to recompile or re-link them. By defining appropriate standard interfaces (essentially, a superset of those already defined within the prototype), and a small amount of administrative information (for example, a Registry-stored list of avail-

able agent components), this could be used to open up the system, allowing end-users to tailor the adaptive behaviour of the system.

The final area for improvement involves the icon-annotation support provided by the prototype, which proved quite popular with the test subjects, but which had some problems. Firstly, this was a new idea to those who had previously only used Windows NT, as no equivalent mechanism exists in the operating system, resulting in a few subjects confusing the document and the annotation to it. Secondly, from a practical viewpoint, the requirement for the system to be able to display its own overlay icons on the shell windows (to indicate the presence of a note) was one of the reasons why it proved to be impossible to directly integrate the prototype system with the Windows NT shell.

A possible solution to this might be to use a slightly different metaphor for annotating objects – one which revolves around the notion of a 'compound document'. This term refers (in the world of Microsoft) to a document which can contain objects of different types. Word documents are referred to as compound documents as they can contain objects such as pictures, charts and so on. If a note were to be one of the types of objects which could be inserted in such a compound document, this would provide an alternative mechanism.

Windows NT does support a mechanism which can be used to provide this sort of functionality. OLE (originally standing for Object Linking and Embedding, and based on COM) has evolved into a de-facto standard for the encapsulation of objects within compound documents, and the in-place editing of embedded objects. The OLE mechanism will be used to extend the object-annotation metaphor by providing a system whereby note objects can be directly inserted into compound documents, analogous to sticking a note on the page of a document.

6.1.2. Theoretical Reworking – Areas to be Addressed

The issues dealt with in this section do not result directly in re-implementation of the system but are instead concerned with providing some guidance for future designers and implementors of adaptive systems such as this one. There are three areas of concern arising out of the design, implementation and evaluation work, each of which will be briefly introduced before being examined in detail: using the AIT architecture in a component-based design; the feasibility and requirements of a generic architecture for acquiring user-interaction events; and the process of decomposing adaptive behaviour into mechanisms suitable for direct implementation as reactive agents.

The process of implementing a system directly based on the AIT architecture raised some issues, particularly concerning the sometimes slightly conflicting requirements of object-oriented design and the component-based approach versus the need for an elegant design. This re-working will seek to address the problems uncovered in implementing the AIT architecture directly in terms of objects.

The AIT architecture is presented as an aggregate object, comprising a set of models and a dynamic knowledge base. The active elements of this knowledge base are essentially deemed to have access to all the models' data – however, in this design, the interface-based approach required that the active elements access this information through interfaces. The final implementation of the architecture showed that while the component-based approach eliminated data coupling, the penalty paid for this purity was a more involved communications structure between the components of the system. Having said this, a useful by-product of the approach was the ability to dynamically configure a component-based system, as discussed in the previous section.

Connected with an examination of the handling of the AIT architecture is a re-examination of the way in which the reactive agents' behaviours were arrived at

for this prototype. This will examine the nature of reactive agents and attempt to formalise a process of levelled behavioural partitioning, based on the somewhat ad-hoc approach taken to the decomposition of the prototype system's desired adaptive behaviour. The aim will be for this work to yield some re-usable design guidance for future developers of adaptive systems which use reactive software agents.

The final element in the re-working addresses one of the main contributing factors to the technical failure to integrate the prototype into the existing Windows NT shell – namely, the lack of any supported mechanism to get direct access to information about user interactions. This is a requirement that would be shared by any other system that sought to assist users in the same way as this one – i.e., by monitoring user actions and attempting to use the information to good effect. This re-working will seek to examine how feasible it would be to define a generic user event notification mechanism. Existing methods used by current operating systems and other relevant technologies will be examined to see how platform-independent event acquisition might be accomplished.

The following sections now address the issues raised in section 6.1.1 which lead to some practical re-design work, discussed below.

6.2. Re-Design 1: Using the Registry

The Windows NT operating system provides a system-wide centralised database for storing named application-specific information, known as the Registry. This portion of the re-design activity aims to take advantage of the Registry database to store the prototype system's model data in a secure yet shareable form. The following subsections firstly give a brief technical background to the Registry, and then detail the design for a version of the system which uses it to store the data which, in the first version of the software, reside in flat files on the local drive.

6.2.1. The Registry – Technical Details

This section details what the Registry is and gives a rough description of how it works, to show how it can better support the system's current data storage requirements. Essentially, the Registry came about as a solution to the problem of storing per-application and per-user configuration data in a regular, structured and shareable way, as is required by almost any application of any size.

In contrast to this centralised approach, many applications used to use their own '.INI' files, resulting in non-portability, a lack of personalisation and a profusion of files in unpredictable locations, not to mention the added burden on developers to provide per-application configuration file handling requirements. Instead of this, the NT Registry provides machine-specific and user-specific data storage facilities, thereby circumventing some of the more peripheral implementation issues.

The Registry takes the form of a hierarchy of keys, each of which stores a set of name/value pairs. Keys are analogous to directories, and in fact use the same naming conventions as directory names. Certain Registry keys are guaranteed to exist by the system. For example, the key "HKEY_CURRENT_USER\Software" acts as a repository for any application configuration data which needs to be personalised to the current user. Any information stored here by an application is automatically associated with the user who was logged in when that information was stored, is permanently saved when the user logs out, and is restored when they log back in at a later date.

6.2.2. Dialogue Record Re-Design

The prototype system's implementation of the dialogue record simply writes a file containing a list of events. (Section 4.4.7 documents the required event types and their attributes – these are currently stored in a sequential list, being written out on shutdown and read back in on start-up). Instead of being stored as text in a flat

file, these events will be written into Registry keys in the structure shown in Figure 6.1.

```
HKEY_CURRENT_USER\
  └── Software\
        └── AIM\
              └── DialogueRecord\
                    ├── First(DWORD)
                    ├── Last(DWORD)
                    └── <numeric_index>\
                          ├── Code(DWORD)
                          ├── Time(DATETIME)
                          ├── Name1(OLECHAR[])
                          ├── Name2(OLECHAR[])
                          └── Level(DWORD)
```

Figure 6.1. Dialogue Record Registry Key Layout.

The keys First and Last below the DialogueRecord key store the numeric index of the earliest and latest events stored in the dialogue record. This will permit some of the dialogue record data to be expired as new data is entered, to avoid generating huge volumes of data over a period of time. For the purposes of this study, the dialogue record will need no more than a few days' worth of data, and it may be possible to have the system function well with less than that. The events themselves are then stored, indexed numerically, with their associated data.

Although the use of numeric indexes indirectly referring to file names might seem to be more complex than is necessary, plain filenames cannot be stored as part of key names – obviously, they may contain '\' characters, which have special meaning in Registry key names, which have the same naming conventions as directories.

6.2.3. User Model Re-Design

Using the same principle as for the dialogue record, the persistent data stores used by the rest of the system can be re-implemented using the Registry as the storage medium. Figure 6.2 shows, in symbolic form, the storage of the user model as Windows NT Registry data. For example, the key named:

```
HKEY_CURRENT_USER\Software\AIM\UserModel\FileAccessLog\0\Filename
```

might have the value "E:\WINNT\DataFiles\Report1.DOC", and if the key named:

```
HKEY_CURRENT_USER\Software\AIM\UserModel\FileAccessLog\0\UseCount
```

then had the value 12, this would indicate that the user's Report1 document data file had been opened 12 times.

```
HKEY_CURRENT_USER\
  └── Software\
        └── AIM\
              └── UserModel\
                    ├── FileAccessLog\
                    │     ├── Count(DWORD)
                    │     └── <numeric_index>\
                    │           ├── FileName(OLECHAR[])
                    │           ├── UseCount(DWORD)
                    │           └── LastUse(DATETIME)
                    ├── ShortCutList\
                    │     ├── Count(DWORD)
                    │     └── <numeric_index>\
                    │           ├── TargetFileName(OLECHAR[])
                    │           └── ShortCutName(OLECHAR[])
                    └── AnnotationList\
                          ├── Count(DWORD)
                          └── <numeric_index>\
                                ├── FileName(OLECHAR[])
                                └── Annotation(CHAR[])
```

Figure 6.2. User Model Registry Key Layout.

Again, the Registry automatically takes care of the possible requirement to handle profiles for multiple users, as the HKEY_CURRENT_USER key is automatically re-mapped according to the identity of the currently-logged-in user.

6.3. Re-Design 2: End-User Tailorability Using Binary Components

The COM component infrastructure provides the ability to dynamically reconfigure systems using 'plug-and-play' binary components. The following sections give a brief overview of COM and ATL at a technical level and proceed to illus-

trate the re-design of the system to use fully COM-compliant components where necessary.

6.3.1. COM and ATL – Brief Technical Background

The Component Object Model (Box, 1998a) is an infrastructure for interface-based software development. For example, it defines a standard interface through which a client application can discover the other various interfaces supported by any given component, called the `IUnknown` interface. COM also defines some standard ways of creating components, all of which are based on the idea of a 'class factory' (which, technically, is a component factory, although the misnomer is generally accepted as equivalent in the literature).

A class factory is a special object type which is responsible for constructing new instances of components of a given class. The simplest way for a client application to construct a component is to call a system function which creates a single instance of a specific class of component. This system function actually uses a system-specific method to determine the class factory for that component class, constructs it, and then uses it to construct the desired component.

Although COM strictly specifies all these mechanisms and the ways in which they must be used, it does not provide implementations for them – this is left to the implementors of the components. The Active Template Library (ATL) provides a set of C++ classes which can be sub-classed to arrive at a COM-compliant component more quickly and with less scope for errors than would be possible by hand-crafting a component implementation.

As the components which comprise the prototype system have been implemented using an interface-based approach, ATL classes can be applied to them (by redefining the inheritance of classes to be made into explicit components), to arrive at a COM-compliant system. This will then allow the reactive agents in their own

right to be divorced from the rest of the system, permitting the end-users of the system to tailor the exact adaptive behaviour of it according to their wishes, simply by using a different set of software agents.

6.3.2. Using ATL to Re-Implement the Prototype Using Components

This section describes the process of applying ATL classes to the prototype system, so that it can be said to be truly component-based. The initial version of the prototype used interface inheritance, based upon the notion that a class can effectively 'advertise' – by exposing a specific interface – that objects belonging to it behave in a certain way. The COM standard requires that any COM-compliant component object must expose a basic interface called '`IUnknown`'[8]. This interface implements a method called '`QueryInterface`', which is used to enquire about the other interfaces supported by that particular component object.

Interface inheritance differs from traditional object-oriented implementation inheritance – essentially, exposing a specific interface means that the component behaves in a specific way (without reference to any other component or object). Traditional implementation inheritance implies that a subclass is a refinement of some superclass, which may or may not inherit a set of traits from that superclass, may or may not exhibit all or some of the behaviour of that superclass, and may or may not maintain the same states as the superclass. Interface inheritance is a much purer way of defining the behaviour of a system, as it separates the implementation of a component from the interface used to manipulate it – although source-level implementation re-use is not directly possible, a well-developed component is easy to re-use.

---

[8] The name `IUnknown` arises from the fact that the underlying type of the component is initially unknown – the component could be *anything*.

The first step in applying ATL to the system is to identify the elements of the system which do actually need to be components in their own right. Initially, the classes used to implement the agents themselves obviously need to be modified, although some other items, like the system's interfaces, could be set up as components as well. The objects and interfaces used by the agents, such as the dialogue record and the other elements in the user model, also need to be borne in mind during this process. However, if the AIMS application is modified to be COM-compliant, that would then allow this to be done without great problems, as if the system itself were a component, all the other interfaces required by the agents could be obtained directly from the application component's `IUnknown` interface without any problem. (The application's component class will not need a class factory, as the system explicitly constructs it on startup, and the references to it held by the agents will be released when the system is shut down.)

In practical terms, the existing inheritance system needs to be altered so that the `CAgent` class inherits from the requisite ATL component-object class. Although COM specifies that a legal COM interface must inherit from one (and one only) other legal COM interface, a class which implements a COM component can inherit from any number, including indirectly via another component class (as long as the eventual `QueryInterface` method is overridden accordingly). A class factory also needs to be developed for the ATL-based agent component class. This can be accomplished by defining an ATL subclass, which can be used by each agent type to do this.

The result of this is that it will then be possible to realise an agent (or set of agents) as a binary component in its own right. Instead of being a tightly-linked element of the system, each agent or set of agents could be packaged in a system code module, such as a dynamic-link library. The system-specific COM support would then be able to load these on behalf of the main AIMS application, based on a

small amount of configuration data. The process of doing this is discussed in the next section.

6.3.3. Tailoring the System using ATL/COM Agents

For an end-user to be able to tailor their system, there needs to be some kind of intermediary between the user and the tailorable elements of the system. The most obvious way to do this would be to provide a simple user interface which displays the system's configuration and allows it to be changed. An important factor is that the system needs to be able to 'explain itself', in that it needs to be able to provide understandable details about what it does, and how changes will affect the status quo.

This user interface could exploit the nature of the agents, being packaged as standalone components. For example, the system could designate a specific folder to store all the agent component modules. The user-interface could then use a special configuration interface on each agent, to acquire information about what it does and the parameters applicable to it, as well as natural language descriptions of the agent(s) present in the module. Figure 6.3 shows a hypothetical system set-up where the user is configuring an agent – this shows the description provided by the agent component, and also displays a configuration set-up (again, generated by the agent).

At a more technical level, several issues arise from the requirements placed upon the agents by a user interface such as that discussed in the previous paragraph. Each agent must be able to have different configuration data, yet this must all be presented in a consistent format within the tailoring interface. The agents must therefore somehow generate their own configuration interfaces in response to a call from the interface – a technique which is already used by the Windows NT shell. One of the shell extension types is a 'Property Sheet Handler', called by the operating system to display the properties for some file system object – the name,

size and free space for a hard disk, for example. Again, a configuration property sheet handler can be stipulated as part of the agent component standard, implemented to generate the user interface elements and retrieve data from them.

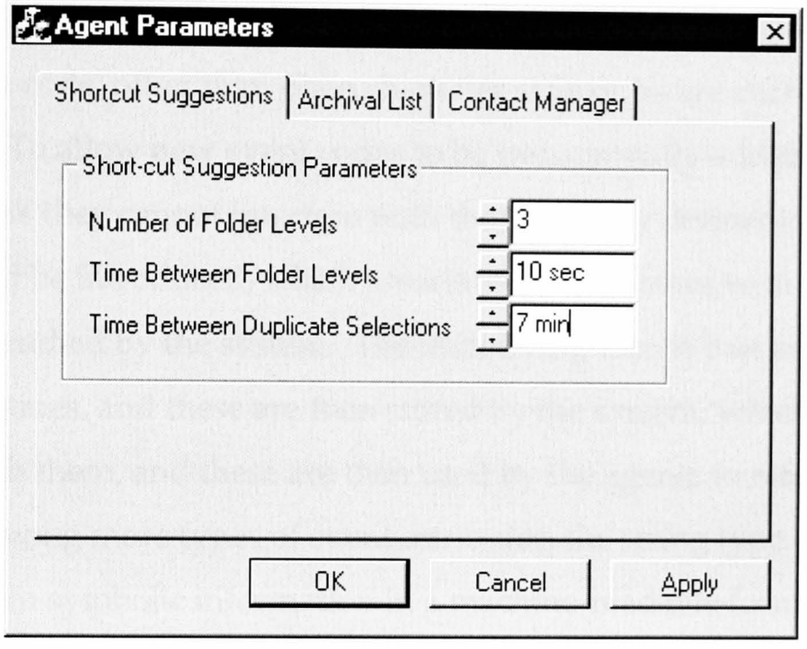

Figure 6.3. Tailoring an agent module.

As well as allowing additional agent types to be added to the system, another issue is that the current system is limited to the event types defined when the system was designed, which would prove inadequate for other applications. For example, to extend the system to handle a Web assistant would result in additional event information requirements. In addition, methods for allowing the system to accept or generate events which are not yet known would be required – for a Web assistant, there would need to be some means to communicate with any browsers running on the machine.

In fact, a partial solution already exists to this problem, in that the agent class definition within the prototype includes the ability for agents to generate their own events, as used in the levelled design for the shortcut suggestion system (see section 4.4.6). Since agents would be presented as components, there is no reason

why an agent could not contain a thread of execution which would generate internal events as the result of external happenings.

The issue of providing new event codes and different types of event information is still outstanding – that is, allowing differing types of information to be associated with each event code, other than plain character strings, as are currently used by the prototype. To allow new event codes to be transparently added to the system (in such a way as they cannot interfere with those already defined) a system of identifiers could be introduced, which associates event names with numeric codes which can be handled by the system. The underlying idea is that an agent submits a set of event names, and these are then stored by the system, which associates a set of codes with them, and these are then used by the agents to refer to the events. As regards allowing more types of event attributes, the string type could be extended to contain symbolic information in a machine-readable format. (For information on a suitable format, see section 6.7).

6.4. Re-Design 3: OLE-Based Annotations

This section will describe an alternative approach to the annotation feature present in the prototype. As mentioned earlier (in section 6.1.1), the icon-annotation support provided by the prototype proved quite popular with most of the test subjects, but there were some problems with it. From the usability viewpoint, a few subjects confused the notions of the document and the annotation to it. Secondly, from a practical viewpoint, the need to be able to display custom overlay icons on shell windows meant that the full integration that was originally envisaged proved to be impossible.

A solution to this would be to take advantage of the notion of a 'compound document', which can contain other 'objects' of different types – objects such as pictures, charts and so on. A mechanism called OLE (originally standing for Ob-

ject Linking and Embedding) is the *de facto* Microsoft standard for the encapsulation of objects within compound documents. OLE provides an infrastructure which allows objects created by different applications to co-exist and be displayed in a containing compound document, and should edits subsequently be required, defines methods for locating and executing the program originally responsible for creating that object – transparently, from the user's viewpoint.

Essentially, the OLE mechanism will be exploited to provide an alternative object-annotation metaphor, by implementing a system whereby note objects can be directly inserted into compound documents, analogous to sticking a note on the page of a document. The subsequent sections give a brief technical background to the mechanisms provided by OLE, and then show how they can be used in the design and development of an alternative annotation system.

6.4.1. OLE – Brief Technical Background

The OLE infrastructure (for our purposes here) essentially provides a mechanism which can be used to divorce the contents of a compound document from the actual document itself. The program responsible for creating and editing the document is termed the 'client', and the ancillary programs which create and maintain objects that can be embedded within compound documents are called 'servers'.

It is possible for a single program to act in both ways – for example, when Excel is being used as a spreadsheet, it is functioning as a client (in that other objects, such as charts, can be inserted in an Excel workbook). When part of an Excel spreadsheet is pasted into a Word document, however, Excel is acting as a server, responsible for all operations on that spreadsheet – display rendering, editing, printing and so on.

The important point to grasp is that although the document might be a single entity in itself, there may be any number of server programs responsible for the ob-

jects within that document. When that document is being edited, printed or whatever, the OLE infrastructure transparently manages these servers, marshalling information between them to allow the system to present the illusion that one single program is responsible for the whole document, where in fact several may be at work behind the scenes.

OLE is itself based upon COM technology – a multi-server linking and embedding scenario clearly requires communication between the client and server applications, and COM is the logical way to provide this. In fact, COM and OLE started out effectively as two facets of the same thing (driven by the need for extensible linking and embedding support), but were logically separated by Microsoft when the utility of a component-based development infrastructure in its own right was recognised.

An OLE server is therefore no more than a COM component which supports a particular set of standard interfaces defined by the OLE programming interface specification (Brockshmidt, 1995). The next section describes how this may be used in actually developing an OLE server which can be integrated within the system, allowing users to insert note objects into their compound documents.

6.4.2. Developing an OLE-Based Notes Server

This section considers the functional requirements of both the note objects themselves and the OLE infrastructure to arrive at a design for an implementable server. Some changes to the existing prototype system are also proposed, which are necessary in order to accommodate some of the practical issues encountered in writing software of this type.

The end product of this design will be an OLE server, which generally takes the form of a dynamic link library (DLL) – a chunk of program code that can be loaded on demand by the operating system, and merged into a running process.

When a compound document with embedded objects in it is loaded, OLE locates the correct servers for each of the embedded objects, and dynamically loads them using the operating system, as shown in Figure 6.4. Servers of this type are referred to as 'in-process' servers, as they effectively become part of the client application's resident set.

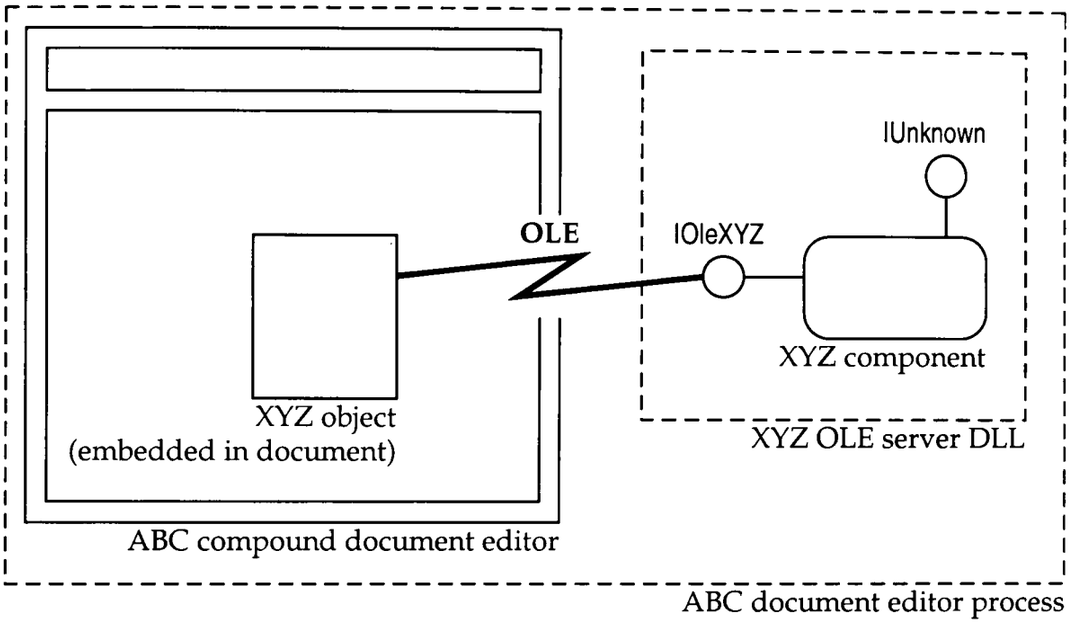

Figure 6.4. OLE in-place editing using a in-process server.

However, servers of this type present a problem for this application, because of the requirement to search the contents of notes for a given word or phrase. To do this, the centralised AIMS application obviously needs access to the contents of the notes. This would not be possible using an OLE server alone, as when loaded, the operating system insulates each instance of the server from the rest of the applications executing on the system, so that a global search would not be possible. To rectify this, there still needs to be some central point of contact that can be used by the note objects to store and retrieve their contents, which can then be used when a search is required.

The obvious solution to this problem is to make a modification to the current AIMS prototype, so that it can function as a repository for the notes' contents. When a note object is inserted into a compound document, it could then use the

AIMS program as a kind of database to store and retrieve its contents. A search of note contents would then proceed much as it does in the current version of the prototype.

To do this, the AIMS application will need to be presented to the operating system as a component in its own right – for the instances of the note objects to be able to find it, it must be registered with the operating system in the Running Object Table (ROT), which acts as a kind of directory of active components on the machine. Other applications running on the system can then use the ROT to find an instance of a required component, gain access to an interface to it, and use the methods it provides, as shown in Figure 6.5.

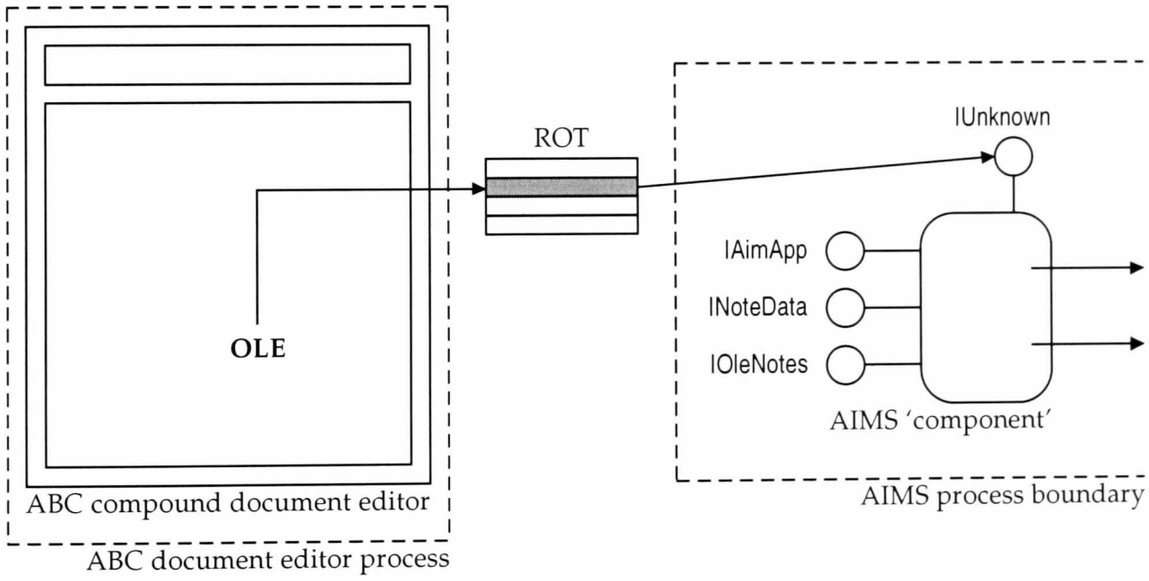

Figure 6.5. Using the ROT to locate the AIMS application.

The administrative overhead of making the AIMS application look like a component is quite small (it involves creating a COM-compliant object, which can be done using ATL), but there is one technical issue which arises from it. When the application registers itself as a component, it makes a set interfaces available to the rest of the system. Client applications (the note objects, in this case) may obtain references to these and call methods on them in any order, possibly at the same time – that is, there may be multiple threads of execution active simultaneously.

This introduces the need for some synchronisation primitives, so that the AIMS application's internal data stores cannot be rendered inconsistent by simultaneous accesses. Again, this is not a difficult problem to solve, but an awareness of it is essential to arrive at a system design which does not suffer from rare, obscure and highly unpredictable race conditions.

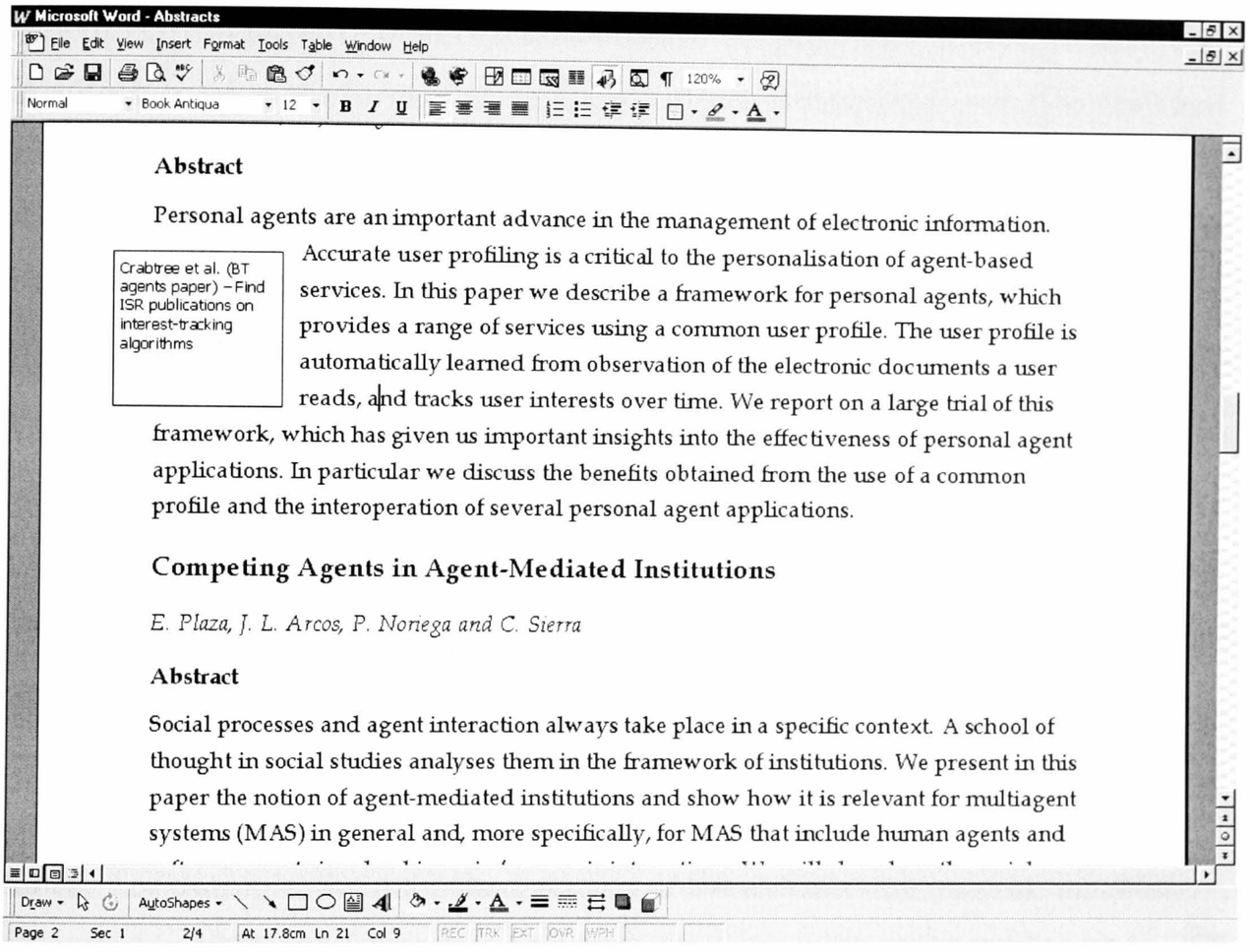

Figure 6.6. An embedded note object in a compound document.

In implementation terms, a note embedded in a compound document could just be stored as a index reference, and the note's contents plus a file reference would actually be stored by the central application for searching and container-object access. The eventual 'user interface' to a note object would be extremely simple, as was the file-annotation dialog. Following a philosophy of simplicity, a suitable interface would consist simply of a multi-line edit control, in its own window, with appropriate line imagery on the border, as shown in Figure 6.6. The text en-

tered into these notes would then be stored along with the standard icon-annotation note text, and be searchable in the same manner.

6.5. The AIT Architecture and OO/CB Design

This section addresses some of the issues raised by implementing the AIT architecture directly in terms of objects and interfaces. As presented in the literature, the architecture takes the form of an aggregate object, comprising a set of models and a dynamic knowledge base, where the active elements of this knowledge base have access to all the models' data. However, the object-oriented and component-based design approach together with the AIT architecture resulted in a final design for the prototype system which appeared to be less elegant than it could have been – while data coupling had been eliminated, the interface-based communication required between the modules of the software had at the same time been made quite complex.

There are several reasons why this increased complexity might have been brought about. One is that the problems noted in this study could have simply been the result of applying OO and component-based (CB) development techniques to a model that was developed before these techniques began to gain wider acceptance. For example, the published version of the architecture used in this study appeared in 1993, at a time when OO methods were quite well established, but CB development had yet to make much of an impact.

The increased complexity could also be seen as a mismatch between the levels of abstraction and detail of AIT architecture and the abstraction levels of the adaptive agents – the agents function at quite a low level of abstraction, whereas the AIT architecture needs to be applicable at a range of levels of abstraction – from syntactic, through semantic, up to goal oriented. Although the goal-oriented level of

information does not really play any part in the prototype system implemented in this study, the system's design would not preclude it from being used if necessary.

It is difficult to address the problem of the intra-architecture communication requirement with a view to reducing or eliminating it, as it will in fact take place anyway – whether explicitly through a component's interfaces, or implicitly as direct (unprotected) data member accesses. From a software engineering viewpoint, the former is much preferred as it at least offers the server component the chance to protect and hide its internal details from its clients. Instead, it would be preferable to accept that the communication takes place, and recast the AIT architecture to support it explicitly.

An alternative design which retains an interface-based approach to systems development will suffer from the need to implement a number of observer/mutator interfaces, but this also occurs in purely object-oriented design when the principles of encapsulation and data-hiding are rigidly adhered to. Although the ideas of component-oriented and interface-based development are not new in an academic sense, support for them in terms of integration with existing methodologies is limited, and much research is ongoing in this topic area (Pooley and Stevens, 1998; Szyperski, 1997).

Rather than presenting the architecture simply as an aggregate object, it could therefore be repackaged as a set of components in its own right. As well as resulting in a more elegant design solution in general, this would also make the active nature of the mechanisms explicit as sub-components in their own right.

There are several advantages to be gained from a natively component-based architecture for adaptive interfaces. From a practical perspective, this would promote easier deployment of adaptive systems, as they could be re-used as standalone components in applications not yet designed. There is also the usual benefit of insulating components from implementation changes – if an alternative environ-

ment existed which could be used to gain event information, the event input component and the system interface component, both of which effectively act as adaptors, could simply be swapped without reference to the rest of the system. As an added benefit, the use of published interfaces to adaptive systems would also promote the sharing of profile information. This is especially useful in systems where bootstrapping is an issue – a larger user base would improve the statistical reliability of conclusions, and improve user perceptions of the usefulness of adaptive systems in general.

As is almost always the case with any design technique, the designer needs to balance the requirements of the design approaches and the requirements of the system being implemented, to end up with a design that is as elegant as possible given the applicable constraints. Component-based solutions offer significant opportunities for the re-use of existing systems, without many of the drawbacks inherent in some purely object-oriented environments.

6.6. Levelled Design of Reactive Agents

This section will examine the nature of reactive agents and propose a process of levelled behavioural partitioning, based on the approach taken to the decomposition of the prototype system's desired adaptive behaviour. The approach taken in this study is partially formalised to yield design guidance for future developers of adaptive systems which use reactive agents as the active inferencing mechanisms.

The software agent technology used in this study was based on the idea of reactive agents – entities which possess few modelling resources, and instead exploit their nature, embedded within existing systems, to perform their tasks. However, many classical reactive systems tend to rely on a form of emergence (Nwana, 1996), where a set of sensing and responding mechanisms, when composed together, result in a compound behaviour which appears to be intelligent in nature.

Although this is useful as a result in itself (illustrating that intelligence can be an emergent property of a set of simple behaviours), it would be preferable to be able to guarantee that a given design *will* produce useful adaptive behaviour, which can be a problem with purely reactive architectures. We therefore propose an approach based on a hybridisation of reactive agents and simple models such as those used in this study, in combination with a semi-formal design technique for arriving at an implementable set of simple reactive agents which result in the desired behaviour.

The technique used in this study was to adopt, implicitly, a kind of levelled approach to deriving semantic information from syntactic events – this is essentially influenced by the different levels of information manipulated by adaptive systems (Benyon, 1993), as shown in Figure 3.1. At the lowest level, the input signals to the system give the total of syntactic information available, and these need to be combined with each other to arrive at information of some semantic value. In this study, levels of reactive agents were used, where simple inferences were made on the basis of the syntactic information and were effectively passed up a hierarchy of agents, leading to more complex inferences (in terms of temporal span or the nature of the conclusion).

To formalise this idea, the technique basically uses a behavioural specification as a starting point, decomposing compound behaviours into simpler fragments and organising these into levels or hierarchies. The result is a set of levels of simple behaviour, each of which can be implemented using one or two agents. In turn, these agents pass their results on to higher-level agents, which make more complex inferences, eventually arriving at the semantic result desired.

To apply this idea to the prototype system developed, the starting point for defining the adaptive behaviour of the system was to examine the relevant user scenario in section 3.3.1, where a user executes two multi-folder compound navigate-

open sequences (i.e., opening a folder, then a folder within it, then a file within that) twice within a short period of time. The syntactic events available to the system are just those concerned with opening folders and files.

The notion of a *compound access* is obviously important here, so we could define this as a sequence of at least one folder access followed by a file access, where the folder and file accesses are all linked by parent-child relationships. A *simple compound access* would occur when the user opens a file which resides in a folder that was opened recently. The first reactive behaviour necessary would recognise this sequence of events, and generate a simple compound access event.

The need to recognise multiple levels of folder access also exists, so we could add a second behaviour which reacts to any compound access which is preceded by the opening of the folder in which the compound access takes place. This would produce some kind of a *multiple compound access* event. This formulation is naturally recursive, in that one multiple compound access event might cause the generation of another, if it had been preceded by the containing folder.

The final reactive behaviour necessary would need to recognise duplicated multiple compound access events (i.e., those accesses which are repeated within a given time). When recognition occurred, some kind of event would need to be generated to signal this, which would be processed by the external system to (as in the case of the prototype) make a suggestion based upon it.

Agent Label	Description of Agent
Single compound access detector	Recognises a file access preceded by an access of the file's parent folder.
Multiple compound access detector	Recognises a compound access preceded by a parent folder access.
Duplicate compound access detector	Recognises a compound access preceded by the same compound access.

Table 6.1. A three-agent formulation for the compound access detection problem.

The results of this analysis are presented in Table 6.1, showing that we have arrived at a set of three reactive agents. An interesting point to note is that the prototype implementation actually used four, although the final behaviour of the two would be the same. The three agents can be arranged as a stack as shown in Figure 6.7, where the events generated by low-level agents are used by those at higher levels to make their inferences.

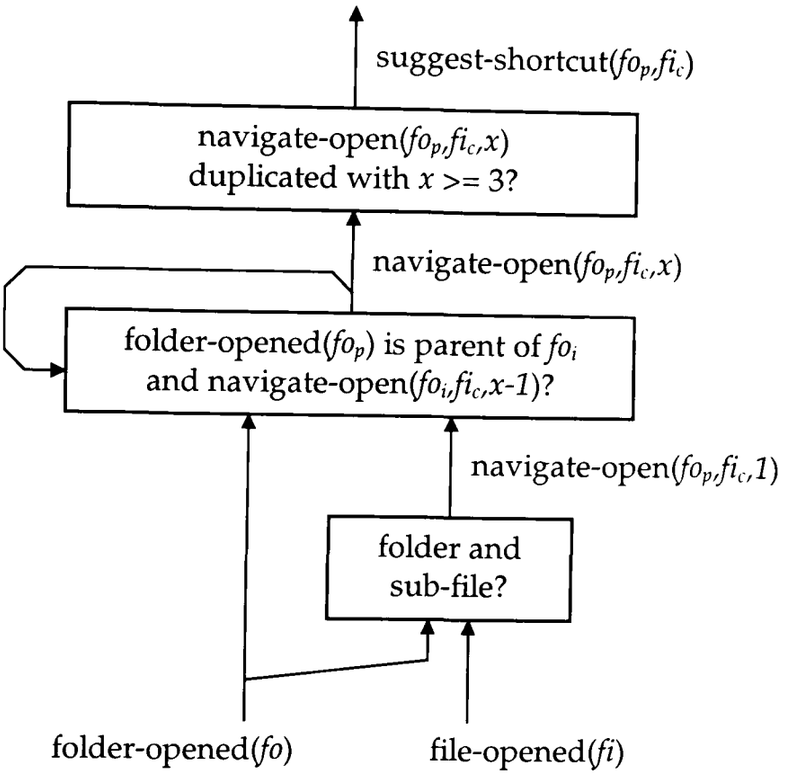

Figure 6.7. Three-level solution to compound access detection.

This kind of formulation is related to other work in the area of reactive software agents – in particular, the Agent Network Architecture (Maes, 1991) and subsumption architectures (Brooks, 1986). The former architecture uses a set of 'competence modules' with pre-conditions and post-conditions which activate, given the correct system configuration, producing an effect in the environment. Subsumption architectures were originally developed as light-weight (in computational terms) solutions to the problems of real-time control in robotics, but the principles of reactivity (simple models, if any, in conjunction with computationally-quick al-

gorithms for generating resulting behaviours) fit well with the needs of an simple adaptive user interface.

This section has shown how a semi-formal approach to the design of reactive software agents can be applied to problems such as those described in this study. However, the technique reported also gives advice on designing the set of agents, which was somewhat lacking in earlier work, which tended to focus more on the mechanisms of the agents themselves rather than the processes followed in designing the set of agents necessary (Wooldridge and Jennings, 1998). In conclusion, the behaviour of a simple, predictable adaptive interface should be specifiable using a semi-formal structure such as this one, since a similar approach to this has been used with some success in this study.

6.7. Providing Generic Event Notification

This section will seek to examine how platform-independent event acquisition might be accomplished, based on existing methods used by current operating systems. As mentioned in section 6.1.2, lack of any supported mechanism to get direct access to information about user interactions was one of the main contributing factors to the technical failure to integrate the prototype into the existing Windows NT shell. Since this is a requirement that would be shared by any other system that sought to assist users by monitoring user actions, this re-working will seek to examine the feasibility of defining a generic user event notification framework and mechanism.

There is a general lack of mechanisms for registering interest in, and receiving information about, events occurring within an operating system, and in particular as a result of user-interface events. This might be owing to the technical nature of the requirement, or the unwillingness of software vendors to expose such proprietary information to the public development community. However, one main difficulty

is the lack of a common standard for representing these events – a manifestation of this can be seen in the number of different mechanisms implemented on just one platform. For example, in the case of Windows NT, four or five different ways of acquiring different sets and types of system event information are available (see Appendix C for a description of some of these).

There are systems which do make an effort to ease the task of acquiring event information and integrating with the operating system. Two systems which partially address some of the issues in user interface integration are the AppleEvents system used on the Macintosh, and the Workplace Shell in use under IBM's OS/2 operating system. Systems such as Eager (Cypher, 1991) and Letizia (Lieberman, 1995) used the AppleEvents mechanism to receive information about the actions of users, and more recently systems such as Apple Data Detectors (Nardi *et al.*, 1998) continue to exploit the open, flexible nature of this system. The OS/2 Workplace Shell offers some of the integration features lacking in the Windows NT equivalent, providing an application programming interface which allows applications to 'hook' into events concerning application window-creation, for example.

Of course, there is great difficulty in defining a common standard – while almost all platforms do share certain syntactic concepts (such as windows and buttons, mouse-clicks and key-presses), other system elements can differ widely. There are considerable implications in designing a system to handle generic events. An initial need would be to define a basic set of events which are well-known, and whose meaning is implicit – mouse-clicks, key-presses and so on. Basic events such as these would then need to be augmented with higher-level events – like icon-selection, as in the prototype system. This introduces a problem of representing the events, their context and their actual meaning.

This actually introduces yet another problem – there needs to be a way of representing common events concisely and unambiguously, while at the same time

leaving scope for the representation of other events which may not even be known about at design time. This shares some of the requirements of the research into communication languages for software agents, where (particularly in multi-agent systems work) sets of agents need to be able to converse, unambiguously, without imposing unnecessary restrictions on the form and content of the messages exchanged.

Effectively, this is an ontological problem, in that the meaning of events needs to be explicit. This is currently a major problem, particularly in the area of inter-agent communication, although efforts are ongoing to solve this problem (Nwana *et al.*, 1999). This ontological information is implicit in the design of most applications, but could be made explicit – a partial example of this can be seen in Microsoft Internet Explorer's automation model, which exposes events and properties which can be used to control the application and get information about happenings within it. This approach could be widened to include information about the events occurring within the system, which could in fact be generated automatically as a result of the user-interface resource definitions present in the application.

The multi-agent systems research effort has produced several standards for the interchange of messages. Two related efforts are – KQML/KIF, the Knowledge and Query Manipulation Language and the Knowledge Interchange Format expressed within it (Finin *et al.*, 1997) and the FIPA ACL, the Foundation for Intelligent Physical Agents' Agent Communication Language (FIPA, 1997). Both use structured yet flexible text messages (i.e., easily machine-readable and able to be extended without rendering existing systems unable to use them) to express requests and responses, so could be adapted to express the occurrence of events. Both standards also include explicit support for an 'ontology' – that is, a set of terms of reference, so that an event is a self-contained entity which gives information as to how it should be interpreted.

The user-interface message requirements could therefore be expressed as a levelled ontology in KIF/KQML or FIPA/ACL format, which has the benefit of being machine-readable and platform independent. As a simple example of the message format, Figure 6.8 shows some hypothetical messages corresponding to user-interface events: (a) a raw mouse-click; (b) a push-button activation (which results in the dismissal of a dialogue box); and (c) an icon-selection (which results in the running of a program).

```
a) (tell :sender Desktop
         :content (mouseclick 450 760)
         :ontology Global_UI)

b) (tell :sender MS-Word.Save
         :content (buttonpush Cancel)
         :ontology Global_UI.FileDialog)

c) (tell :sender Desktop.Shell
         :content (iconselect "Microsoft Office" "Microsoft Excel")
         :ontology Global_UI.ShellWindow)
```

Figure 6.8. User interface events expressed in FIPA ACL (FIPA, 1997).

At a more general level, this discussion illustrates that to a certain extent, in common with the problems experienced in implementing the prototype system in this study, useful facilities are sometimes suppressed due to lack of technological support in the environments in which they are supposed to function. If a generic system existed which could be used to publish the structure of applications and the nature of the user interfaces and the events which occur within them, extensive use would almost certainly be made of it to provide more (and better, in terms of integration) commercial user interface assistant systems. Since the annotation system designed as part of this study would have been reasonably trivial to implement, many other applications would be found by the community.

The acquisition of user-interface event information (and other system-wide event data, for that matter) is currently a difficult task, but is nevertheless central to the development of transparent, integrated adaptive interfaces. To prevent much re-

inventing of the wheel, this is an area which needs to be looked at, and the technique proposed here is part of a possible solution to it.

6.8. Conclusion

This chapter began by examining some of the issues raised by the evaluation, using these to inform re-design activities and examinations of some of the theory used in the development of the system. The work discussed in this chapter has covered both theoretical and practical aspects. Technical work concerning the underlying software implementation of the system was covered, as was an alternative means of integrating some of the PIM support into an existing user interface and environment. The chapter concluded by examining some theoretical reworkings, in order to arrive at some re-usable design guidance for future developers of adaptive systems meant to support individuals, such as this one.

Chapter 7

Conclusions

7.1. Introduction

This chapter concludes the dissertation, providing a review of the main body of text, summarising the main points covered in each chapter, and the key issues raised by them. It also presents a statement of the contributions made during the study, by relating the issues raised in the dissertation as a whole to the wider discussions taking place in the research community.

The scope and limitations of the research reported in this dissertation are also discussed in a critique of the study as a whole. The chapter concludes by considering promising areas for future research arising from the work undertaken in this study.

7.2. Review of Dissertation

This section gives a brief review of the first six chapters of the dissertation, giving the objectives of each chapter and a summary of the key points raised and conclusions drawn.

7.2.1. Chapter 1 - Introduction

Chapter 1 introduced the study, giving an overview of the wider context of the dissertation. The chapter showed that it is concerned with aiding today's professional individuals in activities which revolve around the management of information.

The chapter began by focusing on the importance of information in general and the consequent need for information management both by corporations and individuals. The increasing 'information load' on individuals was noted – in terms of the information that forms the basis of many professionals' jobs and the ways in which this may need to be acquired, manipulated and stored. The discussion suggested that there is a necessity to support Personal Information Management (PIM). A variety of approaches exist which aim to support PIM activities, and an indirect approach was introduced, based on the idea of 'agency' and indirect management. This approach was shown to offer opportunities to aid the support of PIM, via user interfaces which could adapt to their users.

7.2.2. Chapter 2 - Managing Personal Information: Taking an Agent-Based View

Chapter 2 explored the field of PIM in more depth, driven in particular by the trend towards empowering the individual, therefore requiring more 'personal' technologies. Some traditional approaches to PIM were examined, and issues arising from them were noted. In particular it was shown that significant opportunities exist to provide automated support in many PIM scenarios. The chapter highlighted some areas in which knowledge is lacking – particularly, in terms of design guidance for PIM systems which are to be based around the user and able to adapt to her, based on patterns of interaction behaviour. The use of adaptive interfaces in systems to support the management of personal information was suggested, illustrating how the requirements of PIM systems can be fulfilled using

interfaces which can adapt themselves based on their user's working habits and practices.

The use of software agents as an approach to providing the active elements in these adaptive user interface systems was discussed in detail. Examples of current agent technology were evaluated to provide insights into both their particular strengths and weaknesses and those of the concept of agency in general. This led on to the choice of a 'reactive' software agent technology as suitable for the provision of agent-based adaptive user interfaces to support PIM. These reactive agents can offer behaviour which appears to be somewhat 'intelligent', whilst bypassing the problems associated with deliberative architectures.

7.2.3. Chapter 3 - Developing a Framework for an Adaptive Interface for PIM

Chapter 3 developed an abstract design for a class of systems which would act as examplar implementations – these systems would embody some of the principles noted in Chapter 2 as being important to systems which support PIM. The starting point was the notion that an adaptive interface meant to support the basic activities in PIM could be provided using a set of reactive agents as part of an adaptive interface architecture; both these concepts had been introduced and explored at a theoretical level in Chapter 2, and were made more concrete in this chapter.

The chapter proceeded with the development of a design for an adaptive interface which utilises reactive agent technology within a traditional interface. A sample 'adaptive information manager' application was introduced and used as the scope for a detailed design for a user interface which exhibits simple adaptive characteristics as well as providing support in more traditional ways. At the same time, the design activity was kept abstract as far as possible, resulting in a framework which, to a great an extent as was feasible, was language, platform and operating-system neutral.

Ideas covered at a theoretical level in Chapter 2 were examined more closely in the light of the scope laid down, as the requirements became more concrete. The end result was a set of functional requirements – behaviours that the user interface will be required to respond to and exhibit – and in turn, an informal set of requirements which would need to be satisfied by the reactive software agents that were to implement them.

7.2.4. Chapter 4 - Design and Implementation

Chapter 4 used the informal abstract specification and design that was developed in Chapter 3 to develop a concrete design which took account of a particular application, platform, environment and operating system, and implement it as a working software application. The application demonstrated how the principles argued for in Chapter 2 could be realised in a piece of software that provided automatic support for some simple activities encountered as parts of PIM.

Chapter 3 arrived at a set of informal agent requirements based upon a subset of user behaviours observed during file management activities. These were analysed in detail in Chapter 4 to yield a number of basic 'fragments' of adaptive behaviour required to implement a system which would support the tasks specified. These elements of adaptivity then allowed the information requirements of the adaptive system to be ascertained.

An abstract class hierarchy and architectural design was developed in the chapter, to serve as a template for a class of systems which satisfy the general requirements discussed in Chapter 3. The basic class hierarchy was then augmented, according to the requirements of the final application. A behavioural specification for the system was developed which contained details about events required to be observable – opening files and folders, for example – and actions required of the interface in adaptive terms – making suggestions to the user and adding shortcuts, and ranking lists of popular files.

At this point in the process, no decision had yet been made about the platform to be used for the final implementation of the system. An informed choice was therefore made based on several pertinent factors, encompassing technical issues both of infrastructure and implementation as well as issues concerning how the user interface to the system may be presented while not resulting in an excessive burden on users. A suitable platform – Windows NT – together with a supporting infrastructure and integration techniques was then used to implement the system, although some considerable problems were encountered in the process of integrating the software into the target environment.

7.2.5. Chapter 5 - Evaluation and Critique

The objective of Chapter 5 was to critically evaluate the work carried out in this study – to identify strengths and weaknesses of the final product of the research, and the methods and theories used in the design and implementation of it

The evaluation was carried out at several levels: the application implemented in the previous chapter was examined, by means of user trials and usability inspection; the implementation described in Chapter 4 was examined in software engineering terms; the component-based adaptive agent software architecture developed in Chapter 3 was evaluated, to show its usefulness in arriving at the design for the implementation; and the architecture for Adaptive Interface Technology (AIT) adopted in Chapter 2 – the theoretical framework underlying the study – was critically appraised.

The chapter concluded by providing a summary of the important issues raised in the set of evaluations.

7.2.6. Chapter 6 - Re-Design and Re-Hypothesis

Chapter 6 presented some practical and theoretical solutions for problems encountered during this study, based on the issues discussed at the end of Chapter 5. The chapter considered the more practical aspects of the issues raised in this study, by presenting re-design work which addressed problems identified by the evaluations in Chapter 5. The re-design activity was undertaken at two levels. Firstly, at a practical level, some elements of the system were augmented by exploiting additional features of the target platform used in the implementation of the software. Secondly, at a more abstract level, some suggestions were presented for ways to improve the theoretical tools available to help with building adaptive systems in general.

7.3. Statement of Contributions

This section states the contribution made by this work reported in this dissertation, at various levels of abstraction. The levels of abstraction are those identified in section 5.2, encompassing the principles and theory relevant to PIM and user interfacing (L1), adaptive interfaces and frameworks for adaptivity (L2), software agents and agent architectures (L3), and technical issues concerning software design and implementation (L4).

The individual elements of the contribution made by this study arise from differing elements of the dissertation; from the contextual information provided in Chapter 2, through the design and implementation work reported in Chapters 3 and 4, and the evaluation and re-design activities discussed in Chapters 5 and 6.

This section takes a high-level view of the issues raised by this study, in order to set out clearly the contributions made by this dissertation. The objective of this section is to examine the context within which the key issues were raised. This

will show how these issues were resolved, and in doing so, will highlight how they contribute to the ongoing research work relevant to each area.

7.3.1. L1 – PIM and Interfacing: Principles and Theory

In Chapter 2, this dissertation argued that adaptive systems, in combination with more traditional user-interfacing techniques, seemed to offer a profitable combination. Rather than aiming to replace a user's intelligence, it was argued that PIM assistant systems need to complement it and allow users to work in their own way, supporting them where possible.

This contrasts with much of the work in the area, where the emphasis is often mainly on the features of the technology, paying surprisingly little attention to the issue of the usability of the resulting systems. For example, in a recent review of the area of software agents (Jennings *et al.*, 1998), HCI concerns appear to warrant little more than a few paragraphs. This dissertation has attempted to provide a more balanced approach to the area, seeking to maintain usability while at the same time taking advantage of relevant technology.

In Chapter 3, this philosophy of simplicity and predictability yielded a scenario-based analysis of some simple activities which seem to be found in many elements of computer-based PIM. These were used as the basis for an interface specification which incorporated adaptive behaviour with traditional user-interfacing techniques. Once built, in Chapter 5, the results of the evaluation involving user trials appear to confirm the assertion that simplicity in PIM support appears to be a highly desirable attribute. Although as a whole PIM is complex, multi-modal and so on, the individuals in the study were of the opinion that the simple support offered by the prototype system could help them – at least initially.

All of the participants in the experiment said that they would use some of the prototype software's features if the Windows NT operating system had them as stan-

dard – although not all features appealed to all the subjects, perhaps confirming the highly personal nature of the area and the ways in which individuals work.

From a technical perspective, this dissertation has shown in the Chapter 6 re-design activity, that in current operating systems, technology does offer integration and tailorability possibilities which are not being exploited greatly at the present time. These possibilities are mainly in terms of interface integration, although the use of advanced technology (such as OLE under Windows NT) should result in possibilities to support the integration of information as well.

7.3.2. L2 – Adaptivity and Adaptive Interfaces: Frameworks

In recommending a realistic approach to supporting PIM through simple, predictable adaptive interfaces, this dissertation has also shown that there are benefits to be gained through the use of reactive agents in PIM applications. Amongst the wide variety of agent technologies, the characteristics of reactive agents suit the needs of PIM applications well, as discussed in Chapter 2. They are computationally simple and therefore quick-responding, and are consequently well-suited to the requirements of real-time interactive systems. They also do not have great requirements in terms of complex user models and theorem-proving or reasoning support, which makes them attractive from a development viewpoint.

Part of the output of the design and implementation elements of this study, as reported in Chapters 3 and 4, was a proposed architecture for platform-neutral user-interface-based agency support. This was based on an existing generic architecture for adaptive interface technology (Benyon, 1993), and illustrates that careful design can result in a system which is not tied to a specific machine platform.

The implementation phase of the study, as documented in Chapter 4, and the evaluation and re-design activities reported in Chapters 5 and 6 highlighted some of the practical difficulties in writing the prototype software. Particularly of con-

cern was the fact that user-interface and system event acquisition proved to be quite difficult, and the desired integration of the prototype software into the Windows NT environment proved impossible. The dissertation has therefore identified problems in proprietary integration mechanisms in Chapters 4 and 5, and in Chapter 6, proposed a technique for making information exchange more easily possible.

7.3.3. L3 – Agency and Software Agents: Architectures

In Chapter 2 the combination of reactive software agents and adaptive interfaces had been identified as one that might offer benefits in terms of providing PIM support systems. Chapters 3 and 4 illustrated the process of designing and implementing a system based on the AIT architecture's elements, but using reactive agent technology as the inferencing mechanisms. This dissertation therefore contributes a validation of the use of the AIT architecture, with reactive agents as the active elements within it, as a useful technique for the construction of adaptive interface systems.

To partially address some of the concerns mentioned in the previous paragraph, part of the re-design work in Chapter 6 resulted in a technique for specifying reactive agent behaviour. This takes the form of a semi-formal method for examining and analysing the behaviour desired of an adaptive interface, to arrive at a set of reactive agent specifications which can be used to implement the desired behaviour. This could be seen as a contribution either to the theory of reactive agent design in itself, or to the field of adaptive interfacing in general terms, since reactive agents have been shown to be of use in that setting.

Although the combination of reactivity and adaptivity was shown to work well in this study, the dissertation has highlighted areas where additional knowledge was needed to augment the AIT architecture. In Chapter 4, the final design of the models necessary for the adaptive system had to be accomplished in a reasonably

ad-hoc manner, in the absence of concrete advice for their derivation. This is an area where additional material was definitely needed to add to the basic AIT foundation, in order to provide sufficient design guidance. Having made this observation, however, it is perhaps clear that this problem is dependent upon the domain and is consequently difficult to formalise.

7.3.4. L4 – System Realisation: Design and Implementation

The contributions made by this dissertation at the levels of design and implementation are perhaps less important than those made in other areas, but are still worthy of some attention. The implementation work reported in Chapter 4 and the subsequent evaluation of it in Chapter 5 illustrated that the techniques used in the development of the system – object-orientation and component-based techniques, in particular – have a particular set of costs and benefits associated with them.

The implementation of the AIT architecture as a set of components using interfaces to communicate resulted, as noted in Chapter 5, in a communications structure that was more complex than would have been required if the software had been implemented as an aggregate object. However, the process did show how the reasonably novel COM-style approach offers advantages in software development terms, such as the elimination of data-coupling. It also provides an explicit way to define facets of objects' behaviour using the idea of the 'interface' as a kind of behavioural contract.

As a fringe benefit of the use of COM, Chapter 6 highlighted the fact that the component-based nature of the resulting software could be used to provide a system which could support a reconfigurable set of agents. This would then allow end-user reconfigurability, using preferences, and tailorability through a kind of "plug and play" approach to agent installation, where end-users could add to or modify the system's adaptive behaviour by installing new agents, which would then seamlessly integrate with those already present.

7.4. Critique of the Study

This section reviews the study as a whole, critically appraising the various parts of the work reported in this dissertation. It examines strengths and limitations of the work presented, the choice of methods and techniques, and shows how these relate to the scope of the study, the desired results and the eventual contribution made by the work.

The discussion in Chapter 2, which gave the study its detailed context, essentially led on from Chapter 1 in defining the scope of the research work to be carried out. While this study sought to support PIM, it only aimed to support a small part of PIM as a whole. After all, PIM does not necessarily need to be involved with IT in any way. To that extent, the study sought to provide an insight into the provision of an enabling technology which would support individuals in the management of their computer-based personal information.

This study does not attempt any high-level analysis of PIM behaviour, focusing rather on the lower-level activities which are more easily identifiable. The stance taken in this study was to acknowledge that while individuals have highly personal, often subconscious sets of information management practices, there are a set of quite low-level tasks which form the basis of many PIM activities. This stance effectively defines the scope of the practical elements of the study, in seeking to support these activities.

In Chapters 3 and 4 the effect of this stance on the nature of PIM manifests itself in the user scenarios employed in the sections relevant to the behavioural analysis of the system. Having said this, the later evaluation work in Chapter 5 appeared to confirm that the trial users of the system roughly matched expectations in their perception and use of the system.

The design and implementation work undertaken as part of the study was quite small-scale in nature. The application's codebase is not especially large – the complete system is less than 5,000 lines of code, which is small by most standards. The software is prototypical in nature, especially given the technical failure to elegantly integrate it with the target platform. While the Windows NT environment has technical strengths such as protected multi-tasking and so on, there are a lack of ways to closely integrate third-party software with the operating system.

Turning to the evaluations carried out as part of the study, the empirical user study is small, but the results gained from it are valid in terms of the co-operative evaluation approach. Around three or four subjects are generally deemed sufficient (Monk *et al.*, 1993), although a few more (eight, in fact) were used in this case due to the small-scale nature of the tasks to be performed. The resulting findings seemed to be quite consistent – some subjects found all the features useful, and all subjects found some features useful. In any case, empirical work is much needed in this area (Nwana and Ndumu, 1999).

7.5. Future Work

This section addresses issues left outstanding as a result of this study, examining them in order to identify which possible avenues of research might be profitable in resolving them. The key issues discussed in this section are those identified by the evaluation in Chapter 5 (see section 5.10), complemented by others which have been mentioned in this dissertation. The same levels of abstraction are used in this discussion as were used to identify the contribution of this work. This section therefore explicitly shows the boundaries between what was done in this study and what is yet to be done and is the subject of ongoing research.

7.5.1. L1 – PIM and Interfacing: Principles and Theory

Personal information systems offer considerable scope for future research, in a variety of directions. The underlying concept of agency is immediately appealing to most people – the idea that they might be able to delegate routine elements of their everyday tasks to an autonomous, intelligent assistant. Whilst this study has chosen to avoid a debate about 'intelligence', preferring to leave intelligent decisions to human users, there is no doubt that there will be much work which aims to provide intelligent personal support.

The relationship between PIM, adaptivity and agency is a complex one. In this study, the agency provided by the system was quite low-level in nature, whereas a long-term goal will probably be to provide systems where a more human form of agency is the goal. Of course, this raises issues of trust and responsibility and requires the designers of these systems to think very carefully about how systems like these would present themselves to their users.

The use of adaptivity in user interfaces is also an area of ongoing research – as with the provision of intelligence within an interface, there is a need to concentrate on maintaining usability when incorporating behaviour such as adaptivity into an interface.

There is also scope to provide support for some of the more sophisticated elements of PIM, such as communication and integration – the consumer electronics market demonstrates that the technological capability exists to provide integrated telephone and data communication devices – however, research is still needed to guide the development of these advanced services. There should also be opportunities to 'informate' (as opposed to automate) other activities – with an integrated telecommunications 'space', fewer barriers exist to the acquisition of information necessary.

In abstract terms, research needs to be ongoing into the concept of agency itself and the agent-based approach, as this is still a very poorly-defined area, encompassing ideas from many different fields. A key weakness exists in the over-adoption of the 'agent-based' buzzword. The viewpoint taken in this dissertation is that the focus needs to be on solving a problem using appropriate technology, rather than searching for a problem to apply an 'agent' solution to, as sometimes appears to be the case.

7.5.2. L2 – Adaptivity and Adaptive Interfaces: Frameworks

There are a variety of opportunities for research into the theoretical frameworks necessary to support and apply adaptivity within user interfaces. Perhaps most importantly, much more work needs to be done to produce empirical results which are needed to gauge user opinions and perceptions about adaptive and/or intelligent interfaces, and to gain more insight into the effectiveness of the approach as a whole. The principles of designing interactive systems around their users are not directly violated by an adaptive interface design as a matter of course, but care needs to be taken to ensure that usability is maintained as advanced features are incorporated into products.

There is also considerable scope for the development of adaptive interface theory and technologies, in the same way as this study showed in the proposals concerning techniques for the design of reactive agent behaviours discussed in Chapter 6. Existing architectures such as that adopted in this study prove useful as starting points for adaptive systems design, but there is still a general lack of design guidance for these systems. Technological support could also be improved, perhaps in the guise of CASE support for interactive systems development in general, and in particular where adaptivity is used.

Section 3.7.2 raised issues concerning the domain modelling requirements of adaptive interfaces, based on the relationship between the domain and user models –

that the user model is essentially an 'instantiation' of the domain model. In almost all adaptive systems, including this one, the domain model is largely implicit – the mechanisms used to communicate with the operating systems are fixed, and the set of semantic objects such as windows and files are also static. Future work may address this issue to arrive at a higher-order management for adaptive systems where these mechanisms and terms of reference may be more flexibly defined.

7.5.3. L3 – Agency and Software Agents: Architectures

At the more technical level of software agent architectures, future work in the area will need to concentrate on exploration of agent architectures and support for inter-agent communication, particularly concentrating on the problem of interoperability of agent systems – work on this is ongoing most obviously in the fields of multi-agent systems and agent systems focused on electronic commerce. Other work will address issues such as the techniques needed by adaptive systems to infer user interests and to track them as they change over time.

Future work is also required to address the need for toolkits for the provision of agent systems, to counter the current situation where most work involves the implementation of some kind of infrastructure upon which to build the eventual agent-based systems. Higher-level toolkits which support the provision of agent systems have been developed – for example, toolkits such as Zeus (Nwana *et al.*, 1999) and AgentBuilder (Reticular Systems, 1999) address needs such as specification and configuration management within an agent-based system. However, there still exists considerable scope for the development of further CASE support for the design, implementation and management of agent-based systems, whether they be personal or group-oriented in nature.

7.5.4. L4 – System Realisation: Design and Implementation

The process of realising the software prototype described in this study revealed some technical problems, especially concerning the original goal to integrate some adaptive interface elements into an existing user interface – the Windows NT shell. At the present time, this kind of endeavour appears to be extremely difficult, and it is also difficult to see how this will change without quite a fundamental shift in the perspective from which application software is developed.

It would seem that there is quite a political issue in 'open' software development, in that manufacturers understandably wish to protect proprietary product information. It has been suggested (Brockshmidt, 1995) that the 'document-centric' approach will represent the next major revolution in desktop and personal computing, heralding the end of the 'application' as we know it. Software will instead be supplied and used as components, on an as-required basis, invisibly to its users. While this is a utopian view of the future, some of the technology that will be necessary to realise this vision exists at this time, in the form of the component-based software environments such as COM.

The object-orientation and component-based nature of COM software (and similar component-oriented systems) could also be taken advantage of in providing solutions in the area of rapid application development and prototyping – for example, providing a library of commonly-used business components in tandem with a means of specifying and organising them within a user interface of some kind. This should herald opportunities for increased open systems integration, allowing easier customisation and 'plug-and-play' composition, in the same sorts of ways as were proposed for this system in Chapter 6.

References

Allen, K. R. (1995). *Time and Information Management that Really Works: Organization for the '90s.* Affinity Publishing, Los Angeles, CA.

Arnold, J. (1999). *Private Communication by E-Mail.* January 15th, 1999, from <v-johnar@microsoft.com>.

Baecker, R. M., Grudin, J., Buxton, W. A. S. and Greenberg, S. (eds.) (1995). *Human-Computer Interaction: Toward the Year 2000.* Morgan Kaufmann Publishers, San Francisco, CA.

Bates, J. (1994). The Role of Emotion in Believable Agents. *Communications of the ACM*, **37**(7): 122-125.

Benyon, D. R. (1993). Adaptive Systems: A Solution to Usability Problems. *User Modelling and User-Adapted Interaction*, **3**(1): 1-22.

Benyon, D. R. and Murray, D. (1993). Adaptive Systems: From Intelligent Tutoring to Autonomous Agents. *Knowledge-Based Systems*, **6**(4): 197-219.

Bliss, E. C. (1995). *Getting Things Done: The ABCs of Time Management.* Charles Scribner's Sons, New York, NY.

Bond, A. H. and Gasser, L. (eds.) (1988). *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, Los Angeles, CA.

Booch, G. (1994). *Object-Oriented Analysis and Design* (2nd ed.). Addison-Wesley, Reading, MA.

Box, D. (1998a). *Essential COM.* Addison Wesley Longman, Reading, MA.

Box, D. (1998b). The Evolution of Objects. *Microsoft Systems Journal, January 1998,* reproduced in *Box, D. (1998a) Essential COM, Addison Wesley Longman, Reading, MA.*

Brockshmidt, K. (1995). *Inside OLE.* Microsoft Press, Redmond, WA.

Brooks, R. A. (1986). A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation,* **2**(1): 14-23.

Brooks, R. A. (1990). Elephants Don't Play Chess. In: *Maes, P. (ed) (1990) Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back,* MIT Press, London.

Brooks, R. A. (1991a). Intelligence Without Reason. In: *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI'91),* Sydney, Australia: 569-595.

Brooks, R. A. (1991b). Intelligence Without Representation. *Artificial Intelligence,* **47**: 139-159.

Brown, C. M. (1988). *Human-Computer Interface Design Guidelines.* Ablex, Norwood, NJ.

Browne, D. P. (1990). Chapter 7: Conclusions. In: *Browne, Totterdell and Norman (eds.) (1990) Adaptive User Interfaces,* Academic Press, London.

Browne, D. P., Totterdell, P. A. and Norman, M. A. (eds.) (1990). *Adaptive User Interfaces.* Academic Press, London.

Chaib-draa, B. (1994). Distributed Artificial Intelligence: An Overview. In: *Kent, A. and Williams, J. (eds.) Encyclopedia of Computer Science and Technology,* **31**(16).

Chaib-draa, B., Mandiau, R. and Millot, P. (1992a). Distributed Artificial Intelligence: An Annotated Bibliography. *ACM SigART*, **3**(3): 20-37.

Chaib-draa, B., Moulin, B., Mandiau, R. and Millot, P. (1992b). Trends in Distributed Artificial Intelligence. *Artificial Intelligence Review*, **6**(1): 35-66.

Chang, D. and Lange, D. B. (1996). Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW. In: *Proceedings of the OOPSLA'96 Workshop Toward the Integration of WWW and Distributed Object Technology*, San Jose, USA, October 1996.

Checkland, P. and Scholes, J. (1990). *Soft Systems Methodology in Action.* Wiley, Chichester.

Chen, L. and Sycara, K. (1998). Webmate: A Personal Agent for Browsing and Searching. In: *Proceedings of the Second International Conference on Autonomous Agents (Agents'98)*, May 1998, Minneapolis/St Paul, MN.

Chin, D. (1991). Intelligent Interfaces as Agents. In: *Sullivan, J. W. and Tyler, S. W. (eds.) (1991) Intelligent User Interfaces*, ACM Press, New York, NY: 177-206.

Coad, P., and Yourdon, E. (1991). *Object-Oriented Analysis* (2nd ed.). Yourdon Press and Prentice-Hall, Inc., New York, NY.

Collin, N. (1995). *Knowledge Workers and Information Technology: Managing a New Organizational Paradigm.* SRI International, Business Intelligence Program, Menlo Park, CA.

Covey, S. R. (1992). *The Seven Habits of Highly Effective People: Restoring the Character Ethic.* Simon & Schuster, London.

Crabtree, I. B., Soltysiak, S. J., Thint, M. P. (1999). Adaptive Personal Agents. *Personal Technologies* **2**(3): 141-151.

Croft, C. (1997). *Time Management.* International Thomson Business Press, London.

Cypher, A. (1991). Eager: Programming Repetitive Tasks by Example. In: *Proceedings of CHI'91, New Orleans, LA, ACM*: 33-39.

Davids, N. (1996a). Personal Digital Assistants (1). *Computer* **29**(9): 96-99.

Davids, N. (1996b). Personal Digital Assistants (2). *Computer* **29**(11): 100-104.

Dhar, V. (1997). *Intelligent Decision Support Methods: The Science of Knowledge Work.* Prentice Hall, Upper Saddle River, NJ.

Dieckmann, M. (1996). Information/Technology – PDAs Get Connected. *Managing Office Technology*, **41**(5): 44-45.

Drucker, P. F. (1967). *The Effective Executive.* Harper and Row, New York, NY.

Drucker, P. F. (1968). *The Age of Discontinuity: Guidelines to Our Changing Society.* Harper and Row, New York, NY.

Earl, M. (ed.) (1988). *Information Management: The Strategic Dimension.* Oxford University Press, New York, NY.

Eden, C. and Spender, J.-C. (eds.) (1998). *Managerial and Organizational Cognition: Theory, Methods and Research.* Thousand Oaks, London, UK.

Erickson, T. (1990) Working with Interface Metaphors. In: *Laurel, B. (ed.) (1990) The Art of Human-Computer Interface Design*, Addison-Wesley, Reading, MA: 65-73.

Etzel, B. (1995). New Strategy and Techniques to Cope With Information Overload. In: *Proceedings of the IEE Colloquium on Information Overload.* IEE, London, UK: 1-10.

Etzel, B. and Thomas, P. J. (1996). *Managing Personal Information: Tools and Techniques for Achieving Professional Effectiveness.* Macmillan Business, Basingstoke.

Etzioni, O. (1993). Intelligence Without Robots: A Reply to Brooks. *AI Magazine,* **14**(4): 7-13.

Etzioni, O. (1997). Moving up the Information Food Chain: Deploying Softbots on the World-Wide Web. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence,* Portland, OR.

Etzioni, O. and Weld, D. (1994). A Softbot-Based Interface to the Internet. *Communications of the ACM,* **37**(7): 72-76.

Etzioni, O., Lesh, N. and Segal, R. (1994). Building Softbots for UNIX. In: Etzioni, O. (ed.) *Software Agents – Papers from the 1994 Spring Symposium (Technical Report SS-94-03),* AAAI Press: 9-16.

Ferber, J. (1994). Simulating with Reactive Agents. In: *Hillebrand, E. and Stender, J. (eds.) (1994) Many Agent Simulation and Artificial Life,* Amsterdam, IOS Press: 8-28.

Ferguson, P. M. (1993). *The Motif Reference Manual for OSF/Motif Release 1.2.* O'Reilly and Associates, Inc., Sebastopol, CA.

Fidler, C. (1996). *Strategic Management Support Systems.* Pitman Publishing, London.

Finin, T., Labrou, Y. and Mayfield, J. (1997). KQML as an Agent Communication Language. In: *Bradshaw, J. (ed.) (1997) Software Agents, AAAI/MIT Press, Menlo Park, CA.*

FIPA. (1997). FIPA 97 Specification, Version 2.0, Part 2: Agent Communication Language. Available as *'http://www.fipa.org/spec/f8a22.zip'*.

Fischer, G. (1993). Shared Knowledge in Co-operating Problems-Solving Systems – Integrating Adaptive and Adaptable Components. In: *Schneider-Hufschmidt, M., Kuhme., T. and Malinowski, U. (eds.) (1993) Adaptive User Interfaces – Results and Prospects*, Elsevier Science Publications, Amsterdam.

Galliers, R. D. and Baker, B. S. H. (1994). *Strategic Information Management: Challenges and Strategies in Managing Information Systems.* Butterworth Heinemann, Oxford, UK.

Gosling, J. and McGilton, H. (1995). The Java Language Environment. *White Paper*, Sun Microsystems Inc., 2550 Garcia Avenue, Mountain View, CA 94043.

Greenbaum, J. and Kyng, M. (eds.) (1991). *Design at Work.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Heller, D. and Ferguson, P. M. (1994). *The Motif Programming Manual for OSF/Motif Release 1.2* (2nd ed.). O'Reilly and Associates, Inc., Sebastopol, CA.

Henderson, A. and Kyng, M. (1991). There's No Place Like Home: Continuing Design in Use. In: *Greenbaum, J. and Kyng, M. (eds.) (1991) Design at Work,* Lawrence Erlbaum Associates: 219-240.

Höök, K. (1997). Steps to Take Before IUI Becomes Real. In: *Proceedings of the 1997 Workshop "The Reality of Intelligent Interface Technology"*, March 1997, Edinburgh, UK.

Huhns, M. and Singh, M. P. (eds.) (1998). *Readings in Agents.* Morgan Kaufmann Publishers, San Mateo, CA.

Jennings, N. and Wooldridge, M. (1996). Software Agents. *IEE Review*, (**January 1996**): 17-20.

Jennings, N. R., Sycara, K. and Wooldridge, M. (1998). A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems* **1**: 7-38.

Johnson, C. (1997). The Impact of Marginal Utility and Time on Distributed Information Retrieval. In: *Thimbleby, H., O'Connaill, B. and Thomas, P. (eds.) (1997) People and Computers XII, Proceedings of HCI'97, Springer-Verlag, Godalming, UK*: 191-204.

Jones, M. K. (1989). *Human-Computer Interaction: A Design Guide.* Educational Technology Publications, Englewood Cliffs, NJ.

Jones, S. R. and Thomas, P. J. (1996). Perception and Understanding of Personal Information Management Artefacts. In: *Proceedings of Computers in Psychology '96 Conference*, University of York, UK, 25-27 March.

Jones, S. R. and Thomas, P. J. (1997). Empirical Assessment of Individuals' 'Personal Information Management Systems'. *Behaviour and Information Technology*, **16**(3): 158-160.

Kay, A. (1990). User Interface: A Personal View. In: Laurel B. (ed.) (1990), *The Art of Human-Computer Interface Design*, Addison-Wesley, Reading, MA, 191-207.

Kling, R. (ed.) (1996). *Computerization and Controversy: Value Conflicts and Social Choices* (2nd Edition) Academic Press, San Diego.

Kruglinski, D. (1997). *Inside Visual C++ (4th ed.).* Microsoft Press, Redmond, WA.

Laberis, B. (1995). PDAs Are Still a Solution Looking for a Problem. *Computer-World* **29**(27): 36.

Lacity, M. C. and Hirschheim, R. A. (1993). *Information Systems Outsourcing.* Wiley, Chichester, UK.

Landauer, T. (1991). Let's Get Real: A Position Paper on the Role of Cognitive Psychology in the Design of Humanly Useful and Usable Systems. In: *Carroll, J. (ed). Designing Interaction,* Cambridge University Press: 60-73.

Lanier, J. (1996a). Agents of Alienation. *Hotwired (available as http://www.voyagerco.com/misc/jaron.html).*

Lanier, J. (1996b). My Problems with Agents. *Wired Magazine.*

Lees, D. Y., Meech, J. F. and Thomas, P. J. (1996). Information, Artefacts and Management Strategies: The Personal Perspective. In: *Proceedings of OzCHI'96, Hamilton, New Zealand, November 1996.*

Lewis, C. and Rieman, J. (1993). Getting to Know Users and Their Tasks. *Self-published over the the Internet, reproduced in Baecker, R. M., Grudin, J., Buxton, W. A. S. and Greenberg, S. (eds.) (1995) Human-Computer Interaction: Toward the Year 2000.* Morgan Kaufmann Publishers, San Francisco, CA: 122-127.

Lieberman, H. (1995). Letizia: An Agent that Assists Web Browsing. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95),* Montréal, Québec, Canada, August 1995, AAAI Press: 924-929.

Lieberman, H. and Maulsby, D. (1996). Instructible Agents: Software That Just Keeps Getting Better. *IBM Systems Journal* **35**(3-4): 539-556.

Mackay, W. (1990) Patterns of Sharing Customisable Software. In: *Proceedings of CSCW'90,* ACM: 209-221.

Macredie, R. and Keeble, R. (1997). Software Agents and Agency: A Personal Information Management Perspective. *Personal Technologies* **1**(2): 44-56.

Maes, P. (1991). The Agent Network Architecture (ANA). *SIGART Bulletin*, **2**(4): 115-120.

Maes, P. (1994). Agents that Reduce Work and Information Overload. *Communications of the ACM* **37**(7): 31-40.

Malone, T. W., Lai, K.-Y. and Fry, C. (1995). Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. *ACM Transactions on Information Systems* **13**(2): 177-205.

Martin, W. J. (1995). *The Global Information Society.* Aslib, Aldershot.

Mayhew, D. (1992). *Principles and Guidelines in Software User Interface Design.* Prentice Hall.

Microsoft. (1997). *ActiveX Technology for Interactive Software Agents.* Technical Documentation – Platform SDK, MSDN Library Edition, July 1998.

Milewski, A. E. and Lewis, S. H. (1997). Delegating to Software Agents. *International Journal of Human-Computer Studies* **46**: 485-500.

Minsky, M. (1994). A Conversation with Marvin Minsky about Agents. *Communications of the ACM* **37**(7): 23-29.

Mislevy, R. J. and Gitomer, D. H. (1995). The Role of Probability-Based Inference in an Intelligent Tutoring System. *User Modelling and User-Adapted Interaction* **5**(3/4): 253-282.

Mitrovic, A., Djordjevic-Kajan, S. and Stoimenov, L. (1996). INSTRUCTA: Modelling Students by Asking Questions. *User Modelling and User-Adapted Interaction* **6**(4): 273-302.

Monk, A., Wright, P., Haber, J. and Davenport, L. (1993). *Improving your Human-Computer Interface: A Practical Technique.* Prentice-Hall, New York, NY.

Mosier, J. and Smith, S. (1986). Applications of Guidelines for Designing User Interface Software. *Behaviour and Information Technology* **5**(1): 39-46.

Myers, B. (1993). State of the Art in User Interface Software Tools. In: *Hartson, H. and Hix., D. (eds.) (1993) Advances in Human Computer Interaction 4*, Ablex: 110-150.

Nardi, B. A., Miller, J. R. and Wright, D. J. (1998). Collaborative, Programmable Intelligent Agents. *Communications of the ACM* **41**(3): 96-104.

Negroponte, N. (1990). Hospital Corners. In: *Laurel, B. (ed.) (1990) The Art of Human-Computer Interface Design*, Addison-Wesley, Reading, MA.

Nielsen, J. (1989). Usability Engineering at a Discount. In: *Salvendy, G. and Smith, M. J. (eds.) (1989) Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Elsevier, Amsterdam: 394-401.

Nielsen, J. (1992). Finding Usability Problems Through Heuristic Evaluation. In: *Proceedings of CHI'92*, ACM: 373-380.

Nielsen, J. (1993). *Usability Engineering.* Academic Press, Boston.

Nielsen, J. (1994). Heuristic Evaluation. In: *Nielsen, J. and Mack, R. (eds.) (1994) Usability Inspection Methods*, John Wiley and Sons, New York, NY: 25-62.

Norman, D. A. (1994). How Might People Interact with Agents. *Communications of the ACM* **37**(7): 68-71.

Norman, D. A. and Draper, S. (eds.) (1986). *User Centered System Design*, Lawrence Erlbaum, Hillsdale, NJ.

Norman, R. J. (1996). *Object-Oriented Systems Analysis and Design.* Prentice-Hall, Inc., Upper Saddle River, NJ.

Nwana, H. S. (1996). Software Agents: An Overview. *Knowledge Engineering Review* **11**(3): 1-40.

Nwana, H. S., and Ndumu, D. (1999). A Perspective on Software Agents Research. To appear in the *Knowledge Engineering Review* in 1999.

Nwana, H. S., Ndumu, D., Lee, L. and Collis, J. (1999). ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. *Applied Artificial Intelligence Journal* **13**(1): 129-186.

Nye, A. (1992). *The Xlib Programming Manual for X11R4/R5* (3rd ed.). O'Reilly and Associates, Inc., Sebastopol, CA.

Nye, A. and O'Reilly, T. (1993). *The X Toolkit Intrinsics Programming Manual for X11R4/R5* (3rd ed.). O'Reilly and Associates, Inc., Sebastopol, CA.

Olsen, D. (1992). *User Interface Management Systems: Models and Algorithms.* Morgan Kaufmann, Mountain View, CA.

Orwant, J. (1996). For want of a bit the user was lost: Cheap user modelling. *IBM Systems Journal* **35**(3-4): 398-416.

Petrie, C. J. (1996). Agent-Based Engineering, the Web, and Intelligence. *IEEE Expert* **11**(6): 24-29.

Pooley, R. J. and Stevens, P. (1998). *Using UML: Software Engineering with Objects and Components.* Addison-Wesley, Harlow, UK.

Pressman, R. S. (1997). *Software Engineering: A Practitioner's Approach* (4th ed.). McGraw-Hill, London, UK.

Puerta, A. R. (1998). Intelligent User Interfaces. *Knowledge-Based Systems* **10**(5): 263-264.

Reilly, P. A. and Tamkin, P. (1996). *Outsourcing: A Flexible Option for the Future?* Institute for Employment Studies, Brighton, UK.

Reticular Systems. (1999). AGENTBUILDER: An Integrated Toolkit for Constructing Intelligent Software Agents. *White Paper.* Reticular Systems, Inc., 4715 Viewridge Avenue, Suite #200, San Diego, California 92123.

Rifkin, J. (1996). *The End of Work: The Decline of the Global Labour Force and the Dawn of the Post-Market Era.* G.P. Putnam's Sons, New York.

Robson, W. (1997). *Strategic Management and Information Systems: An Integrated Approach* (2nd ed.). Pitman, London.

Rogerson, D. (1997). *Inside COM.* Microsoft Press, Redmond, WA.

Roos, J. (1997). *Intellectual Capital: Navigating the New Business Landscape.* Macmillan Business, Basingstoke, UK.

Schneider-Hufschmidt, M., Kuhme, T. and Malinowski, U. (eds.) (1993). *Adaptive User Interfaces: Principles and Practice.* Elsevier, North Holland.

Schuler, D. and Namioka, A. (eds.) (1993). *Participatory Design: Principles and Practices.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Seddon, D. (1988). Experiences in IT Strategy Formulation: Imperial Chemical Industries PLC. In: *Earl, M. (ed.) (1988) Information Management: The Strategic Dimension*, Oxford University Press, New York: 147-156.

Senn, J. A. (1989). *Analysis and Design of Information Systems* (2nd edition). McGraw-Hill, London.

Shneiderman B. (1997). Direct Manipulation vs. Agents: Paths to Predictable, Controllable and Comprehensible Interfaces. In: *Bradshaw, J. (ed.) (1997) Software Agents, AAAI/MIT Press, Menlo Park, CA*: 97-106.

Shneiderman, B. (1998). *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (3rd ed.). Addison-Wesley Longman, Harlow, UK.

Shoham, Y. (1993). Agent-Oriented Programming. *Artificial Intelligence* 60(1): 51-92.

Singh, M. P. (1994). *Multiagent Systems: A Theoretical Framework for Intentions, Know-How and Communications*. Lecture Notes in Artificial Intelligence 799, Springer-Verlag, Heidelberg.

Smith, S. and Mosier, J. (1986). *Guidelines for Designing User Interface Software.* Report No. 7, MTR-10090, ESD-TR-86-278, MITRE Corporation.

Sommerville, I. (1996). *Software Engineering* (5th ed.). Addison-Wesley, Wokingham, UK.

Stamper, D. A. (1994). *Business Data Communications* (4th ed.). Benjamin/Cummings Publishing, Inc., Redwood City, CA.

Suchman, L. (1987). *Plans and Situated Actions: The Problem of Human-Machine Communication.* Cambridge University Press, Cambridge.

Sullivan, J. W. and Tyler, S. W. (eds.) (1991). *Intelligent User Interfaces,* ACM Press, New York, NY.

Sunday Times. (1997a). *"Internet Lets Shoppers Browse at Leisure"*, Sunday Times, 20th April 1997, Section 4 (Money): 6-7.

Sunday Times. (1997b). *"'Information Fatigue' Saps the E-mail Set"*, Sunday Times, 20th April 1997, Section 1 (News): 8.

Szyperski, C. (1997). *Component Software – Beyond Object-Oriented Programming.* Addison-Wesley, Harlow, UK.

Thomas, P. J., Meech, J. F. and Macredie, R. D. (1994). Information Management Using Integrated Personal Information Appliances. In: *Proceedings of the BCS Computer Graphics and Displays Group Conference on Digital Media and Electronic Publishing.*

Wayner, P. (1995). *Agents Unleashed: A Public Domain Look at Agent Technology.* Academic Press, London.

Wheelwright, G. (1995). PDAs: The Next Generation. *Personal Computer World* **18**(5): 518-522.

White, J. E. (1994). Telescript Technology: The Foundation for the Electronic Marketplace. *White paper*, General Magic, Inc., 2465 Latham Street, Mountain View, CA 94040.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, **10**(2): 115-152.

Wooldridge, M. and Jennings, N. R. (1998). Pitfalls of Agent-Oriented Development. In: *Proceedings of the 2nd International Conference on Autonomous Agents, Minneapolis, USA.*

Zuboff, S. (1995). The Emperor's New Workplace. *Scientific American* **273**(3): 202-204.

Appendix A – User Trial Materials

This appendix contains a copy of the list of activities performed by the subjects as part of the user trials (A.1), and a copy of the observation note proforma used to record details as the subjects worked through the activities (A.2). The appendix contains a copy of the list of questions asked as part of the de-briefing interviews with subjects after they had undertaken the tasks shown (A.3).

A.1. List of Activities

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

A.2. Observation Log Proforma

Subject: _____ ID No: _____

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

A.3. De-Briefing Questions

These questions form the basis for the brief interviews conducted following the experiment. The interview is therefore reasonably structured in nature, but can be adjusted to cover unexpected points raised by the subject.

Transcriptions of all de-briefing interviews are provided as Appendix C.

(i) Do you get it – can you see how the system attempts to help?
(yes ... no (why?))

(ii) Do the various mechanisms make sense, or are they confusing?
(yes ... no (why?))

(iii) What did you think about the system's interface, as implemented?
(good ... bad / clear ... unclear (why?))

(iv) If things didn't go to plan:
In activity X, I noticed that [whatever]...; why did you do that?

(v) Would you use it?
(yes ... no (why?), (would you need to make a special effort to do so?))

(vi) How can the system be improved, in your opinion?
(make it more obvious how to use (how?), make it look nicer (how?), make it do more (what?), it doesn't do x)

Appendix B – Observation Notes

This appendix contains copies of the observation notes taken during the user trials as the subjects followed the instructions shown in Appendix A.1.

B.1. Observation Notes from Subject 1

Observation Log

Subject: [redacted] ID No: 1

1. Add your name to the file 'Experiment Report' located in the folder 'User Assistance' in the folder 'Documents' on the Desktop, and close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the short-cut suggester.)*

no problems

2. Assume you updated a spreadsheet file called 'Investments' a few days ago, and you can't remember where it is now – but a change needs to be made. Find the file, and alter the price of the PRU stock to 950.50, closing the spreadsheet afterward; *(Hint: You could use the file tracker.)*

"track file" – why double-click.

3. A couple of days ago, a customer requested a quote for year-end accounting and audit services. You can't remember which customer it was, but you attached a note to their record. The quote is £250, which needs to be inserted in their record document. The documents are stored in the Customer Records folder on the Desktop; *(Hint: Look for a document with a note on it in the Customer Records folder – leave the note open for the next task.)*

found correctly

4. Add the name of the customer from activity 3 (user the note you left open) to the file named 'Experiment Report' located in the folder 'User Assistance' in the folder 'Documents' on the Desktop, and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

was news from .doc, not note
flash indicator? cleared suggestion, double-click?

5. You have received a request for the invoiced amount for a new PC – the price is stored in an account document file somewhere. The PC was supplied by Viglen – a fact you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

no problems – double-clicked OK.

6. The file 'New Reference' on the desktop (someone just mailed you with it) contains citation details for a publication currently under development. Add this to the 'Journal Paper References' document – you last edited this a few days ago. *(Hint: You could use the file tracker.)*

no problems – maybe need instructions for simple task

7. Record the current time in the file named 'Experiment Report' located in the folder 'User Assistance' in the folder 'Documents' on the Desktop. *(Hint: You could use the shortcut that should have been created in Activity 4.)*

using manual to- method made shortcut.

B.2. Observation Notes from Subject 2

Observation Log

Subject: [redacted] ID No: 2

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

no problems.

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

opened wrong file – did not read question.
got correct file. no problems

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

opened directory, viewed note OK but closed it.
added quote OK. re-viewed note OK.

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

only one suggestion. should be highlighted / invoked,

found file OK accepted (after Select) OK.
added name OK. – removed note - refresh file before icon went

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

no problems.

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

opened ref OK. found refs OK. added OK.

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

used shortcut OK. no problems

B.3. Observation Notes from Subject 3

Observation Log

Subject: [redacted] ID No: 3

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

no problems.

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

no problems - double-clicked rather than button

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

found file OK - hassles with document - confused between document/note. (instructions!)

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

Found file OK put name in ok, accepted shortcut (same prob with SK)

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

found OK, no problems

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

opened JPRefs OK, problems seeing reference file.

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

no problems

B.4. Observation Notes from Subject 4

Observation Log

Subject: [redacted] ID No: 4

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

no problems.

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

opened OK (used better) no problems.

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

found file OK, viewed note, opened OK.

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

note left OK (name?), put name in.
didn't immediately use suggestion - did eventually.

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

no problems.

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

no problems. new ref first, then JPRs.

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

no problems.

B.5. Observation Notes from Subject 5

Observation Log

Subject: [redacted] ID No: 5

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

no problems

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

no problems

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

found file eventually – tried search first. (Read hint) reopened note

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

found file OK, had left note, open. Time > 10 minutes, so no shortcut made.

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

no problems.

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

got JPR (>5 days ago, so in "wrong" category), then NP.
no problems

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

opened file OK - shortcut made.

B.6. Observation Notes from Subject 6

Observation Log

Subject: [redacted] ID No: 6

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

no problems. (word hangs again?)

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

opened using button, OK

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

opened file without using ok. re-opened note for viewing purposes

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

no problems, suggestion made OK

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

searched and found OK

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

got NR wrong OK, f'd → JPR, copy + paste OK.

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

used shortcut OK.

B.7. Observation Notes from Subject 7

Observation Log

Subject: [redacted] ID No: 7

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

no problems

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

found file ok (was in wrong category!)

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

found note – minimize icon? (open directory / file from note?)
closed then reopened – question wording again

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

hunting for the file - (wording) no problems...
(not checked s/cut yet)

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

no problems. | would prefer immediate execution, and rewording of the suggestion dialog

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

got reference - again wrong category (!)
(then jpr's) no problems

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

used shortcut. no problems.

B.8. Observation Notes from Subject 8

Observation Log

Subject: [redacted] ID No: 8

1. On the desktop there is a folder called 'Documents', which contains a folder called 'User Assistance', which has a file in it called 'Experiment Report'. Open the file and type your name where shown, then close the word processor; *(Hint: You'll need to do this as normal, but it will provide the system with information used by the shortcut suggester.)*

no problems.

2. You worked on a spreadsheet file called 'Investments' a few days ago, but you can't remember where it is. Now a change needs to be made – find the file, and alter the price of the Prudential (PRU) stock to 950.50, then close the spreadsheet; *(Hint: You could use the file tracker.)*

found – problems with question : (few ≈ >5)
no hassle.

3. A couple of days ago, a customer requested a quote for accounting services. You attached a note to their record document, which is in the Customer Records folder on the Desktop. The quote is £250, which needs to be inserted in their record document; *(Hint: Look for a document with a note on it in the Customer Records folder – view the note and leave it open for the next task.)*

found file OK - opened immediately.
re-opened note after prompting. (didn't close)

4. Add the name of the customer from Activity 3 (use the note you left open) to the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'), and close the word processor. *(Hint: If you do this by opening the document file as normal, the system should ask if you want a shortcut made.)*

found file OK, S/C suggestion made.
Suggestion accepted.

5. You have received a request for information about a new PC. The information is in an invoice document somewhere. The PC was supplied by Viglen, which you recorded in a note attached to the file. Locate the file and open it. *(Hint: You could use the note search feature.)*

found OK - no problem.

6. The file 'New Reference' on the desktop contains a reference for a publication currently under development. Add it to the 'Journal Paper References' document – pretend you last edited this a few days ago. *(Hint: You could use the file tracker.)*

Opened NR first then JPR.

7. Record the current time in the file named 'Experiment Report' (located in the folder 'User Assistance' in the folder 'Documents'). *(Hint: You could use the shortcut that should have been created in Activity 4.)*

no problems.

Appendix C – Transcripts of Debriefing Interviews

This appendix contains transcripts of the semi-structured debriefing interviews conducted after each of the users accomplished the tasks shown in Appendix A.1.

C.1. Transcript from Debriefing Interview with Subject 1

E: Did you – did you, get it, you know?

S: Yeah, yeah – it's just something to organise your work – to do some of the, er, dirty work for you, which is ...

E: What like?

S: I mean the, this ... *(indicates list of activities)* is ...

E: D'you mean the detail, of all sort of the things kicking around it, kind of obscured ...

S: Sorry?

E: So, all the documents, and the ...

S: Yeah, document organisation ...

E: ... those instructions kind of distracted from what it actually does? *(I'd got a bit confused here – perhaps I thought the subject was unclear about the difference between the instructions and the experiment)*

S: It's a, I mean, like, it's also a reminder, and you do some stuff, and you sometimes forget that you have to continue, and it reminds you what to do. That's what I think it does.

E: *(back on track now)* With the interface that it's got, you raised some points about, like, double-clicking where buttons should be?

S: Yeah, 'cos – I mean, for a user, double-clicking may not always be, their thing, 'cos they'd like to press some button – press this button, that button.

E: Right – it's a good point. So apart from that can you think of any way you might improve it?

S: If you have a messages like suggestions, or something, I'd prefer to put them like in, you know, one of those things – Bing! *(impersonates Windows chime on message box)* – would you like to do this, yes or no, yeah?

E: Yeah – a more common way of saying – I have a suggestion for you?

S: Yeah, so it's like, it's not like you have to come back to see if it has a suggestion for you. It will come to you.

E: Have you used the little Microsoft Office thing, where it puts a little lightbulb, to say 'ah, I've spotted something you've done'?

S: Yeah.

E: Do you mean that kind of thing, 'cos that's like a little window, and this symbol clears – *(indicates the message area in the 'toolbar' window)*

S: Yeah, like that's a ...

C.1. Transcript from Debriefing Interview with Subject 1

E: Did you – did you, get it, you know?

S: Yeah, yeah – it's just something to organise your work – to do some of the, er, dirty work for you, which is ...

E: What like?

S: I mean the, this ... *(indicates list of activities)* is ...

E: D'you mean the detail, of all sort of the things kicking around it, kind of obscured ...

S: Sorry?

E: So, all the documents, and the ...

S: Yeah, document organisation ...

E: ... those instructions kind of distracted from what it actually does? *(I'd got a bit confused here – perhaps I thought the subject was unclear about the difference between the instructions and the experiment)*

S: It's a, I mean, like, it's also a reminder, and you do some stuff, and you sometimes forget that you have to continue, and it reminds you what to do. That's what I think it does.

E: *(back on track now)* With the interface that it's got, you raised some points about, like, double-clicking where buttons should be?

S: Yeah, 'cos – I mean, for a user, double-clicking may not always be, their thing, 'cos they'd like to press some button – press this button, that button.

E: Right – it's a good point. So apart from that can you think of any way you might improve it?

S: If you have a messages like suggestions, or something, I'd prefer to put them like in, you know, one of those things – Bing! *(impersonates Windows chime on message box)* – would you like to do this, yes or no, yeah?

E: Yeah – a more common way of saying – I have a suggestion for you?

S: Yeah, so it's like, it's not like you have to come back to see if it has a suggestion for you. It will come to you.

E: Have you used the little Microsoft Office thing, where it puts a little lightbulb, to say 'ah, I've spotted something you've done'?

S: Yeah.

E: Do you mean that kind of thing, 'cos that's like a little window, and this symbol clears – *(indicates the message area in the 'toolbar' window)*

S: Yeah, like that's a ...

E: Something even more prominent than that, would you say?

S: Yeah, 'cos sometimes I see, that one *(the message line)* stays – does it stay?

E: Erm – what, you mean in between other options, other operations?

S: Yeah.

E: Yes it does.

S: It stays, because that one stays, if you don't see it on the spot, if you do some other editing stuff or like that it will stay until you see it, until you click it – if you don't click it, it won't go. So, it may take time to see that one, because still, in the corner, I mean you're using two... *(windows)* So...

E: Do you think that if Windows NT had features like that you would actually use them?

S: Sorry?

E: If Windows NT did actually have features like that, like you could put notes on icons, search for things, and use a more detailed file history – do you think you would use actually them? Or would you not be bothered?

S: Er – well, I mean if something like for example Viglen *(referring to note searching task)* if you, sometimes you want to search a word, a key word in a file, yes, it could be helpful, so I could ...

E: Well, with the existing system, you can, er NT will let you search for words in a file, but, I suppose my idea was that you don't always want to put words in a file, you'd rather just, y'know, like you stick a note on something, you don't want to write in a book...

S: Oh that one, yes, that's, a like visual effect, that's even better because if you know, for example, the general location of the file, and you open it *(a window)* and find a list of files, instead of going – er, this file *(mimes browsing)* oh, the name is not full, let's click and so it will expand the name, but if you found the icon then something *(claps hands together)* it's just there, that's good, a visual effect.

E: Well, thankyou very much for your help.

S: No problem, any time.

E: Wasn't too bad, was it?

S: No, no, that's no bother at all.

E: I can see you've managed to uncover all sorts of problems ...

C.2. Transcript from Debriefing Interview with Subject 2

E: There you go ... and that's the end of the activities.

S: I want one of these – it's quite good.

E: So, do you get it?

S: Yes, I think I did – I think I did. It's quite handy ...

E: So you ...

S: ... it's like, like using – I don't know – it's adding things that you use on your desk, on your proper desk, but that isn't available ...

E: So, do you see what the mechanisms are?

S: Yes, I can, you can put another, the computer's got a structure for storing the information, but you can like make up your own structure with your post-it notes and things ... so ... just a different way of referencing information, because your way of doing it, 'cos everyone has their own way of doing it. That's a way of imposing your own way of doing it.

E: Um, apart from that, the interface, the way it's presented ... what did you think of that, is there anything you'd do differently ... or found strange, at all?

S: No, I don't think so ... the, the little yellow notes, were, obviously little yellow notes ... erm ...

E: ... providing you knew that they were ...

S: Yeah, well I know ... there's nothing odd about that. The things were, the removing-it feature was obvious ... I assume I can add a note *(reaches for mouse to try it)* to something can I ... right yeah ... that's obvious *(types a note on one of the desktop files)* so that's contextual as well. It says add note and remove note *(referring to context menu option)* unless it ... good. Yes, I think that makes ... sense. Does the, can you do any other, does it catch any other suggestions? Or just that you've opened something more than one time?

E: At the moment, just the ... well, let's see, it's ... you've opened it more than, more than one time, quite close together, in time ... so less than sort of five or six minutes apart, I think, and the file's located within two directories, at least. So you know, click-click, click-click, click-click to open it, and then, done that twice within ten minutes, then it says d'you want this?

S: Oh ... so yes, that makes sense.

E: Apart from that erm ... so do you think you'd use it? If it was actually part of the system ...

S: Yes, I would. S'cool.

E: Well, that concludes the interview. Thankyou very much.

C.3. Transcript from Debriefing Interview with Subject 3

E: There you go.

S: Is that it?

E: That's, that's it for the doing this, I'm just going to ask you a couple of questions.

S: Oh, okay.

E: Did you, did you, did you get what the software was trying to do, never mind about my questions *(the activities)* did you kind of see what ...

S: Yeah, it's making a profile of me.

E: Yeah, sort of, I guess.

S: -It's being nosey, almost.

E: Well, yeah, it's trying to adapt your ... the files to you, really.

S: To what I use most? It's trying to provide a customised environment?

E: That sort of thing, yeah. Did you think the interface was alright ... any good or bad things?

S: It's just ... the usual. It's just the same, it's what you're used to.

E: Right, I guess. Well, that's part of the idea.

S: You know how to use it.

E: Erm. How do you think you might ... well ... would you use it, if NT came with features like this as part of it? Do you think? Or would you not bother?

S: I'd probably use, you know where you do the bits about when you ... when you can't remember where you've stored a document, I do that all the time, and I can never find it, so that's a good feature. Erm ... I don't know about the note thing.

E: Uh huh.

S: Is that it?

E: Yeah, it is, thankyou very much.

C.4. Transcript from Debriefing Interview with Subject 4

E: That concludes the activities, I'll just ask you a couple of questions ...

S: Did you take ... oh, OK, I won't ask questions then ...

E: ... again, I'm not testing you, it's just that ... once you'd got through my questions did that kind of make sense ... did you get it?

S: Yeah, it was really good, actually ... I like this (*indicates the 'toolbar' window*), this is really useful, and it's just there, isn't it, so you don't have to go ... I mean there is a file thing but you have to go into ... and it's just there. I like the idea.

E: And you get, like, the idea of sticking a note on things ...

S: Yeah ... and ...

E: Looking around?

S: Looking for notes, yeah.

E: And the interface, did you thing it was alright ... and good or bad points, improvements you would make?

S: It's ... it's ... no, because ... it's something that we're used to, in a way isn't it I suppose, it's not too different ...

E: Do you think that if NT actually had features like this you'd use them?

S: Yes, they're not difficult to use. Did you note down the first, the time that I started?

E: No, but that's not really what I'm looking at ...

S: You weren't timing me then?

E: No, it's nothing like that. It's not a race.

S: I see ...

E: Thankyou very much.

C.5. Transcript from Debriefing Interview with Subject 5

E: That's the end of the activities, I'll just ask you a few questions ... do you ... kind of ... get what it's trying to do?

S: Yeah.

E: The various mechanisms?

S: It's using like, like the notes is assisting, extending some of the, extending the criteria by which you can search for items within the desktop, isn't it, it's user-configurable to the extent that you can just chuck what you want in there as a key word, so that's, that's really cool ...

E: What did you think about the interface to it?

S: The interface, like ... that? *(indicates toolbar)*

E: Yeah, the ...

S: With this, particular ... I thought the button toolbar was pretty cool, but ... I think the Windows 95-y sort of Windows environment, what would have been nice would have been to have ... like these as buttons, you know, with the task bar, yeah, just to keep it ... because sometimes, lot of windows, it gets a bit ... erm ... so obviously you're using some type of passive agent for that just to pick up on, on probably some rules, so ... I think that was really neat actually what that does, and 'cos it does, it doesn't just suggest, it does it for you, which is a good thing.

E: Erm ... do you think that Windows NT had features like this in it already that you would actually use them?

S: Personally, no. Erm ... like, things like shortcuts and the desktop and so on, I tend not to customise them by much at all, in fact, I'll just like use the start menu and that's that, but I can see where, other people, like family members, might find it really useful. Definitely, I think that some of these features, like definitely, that ... file tracker was a bit ... was ... file tracker was a bit ... dunno, not sure if that's useful, search notes is really good, erm ... suggestions ... yeah, good. But the file tracker was a bit ... err. I would probably use search notes, but I doubt I'd use that one at all, ever.

E: Why, what do you do, do you just tend to keep things organised?

S: Well I keep things organised in terms of where I locate my files and stuff, so I know exactly where everything is, so yeah. But search notes is really good in terms of, like reminders if you need to do something to a particular file of whatever you can just set up key words and just do a big search and catch the particular set of files you want. It's good.

E: Thankyou very much for your time.

C.6. Transcript from Debriefing Interview with Subject 6

(The tape recording of this interview turned out to be of particularly bad quality, as the batteries had begun to fade, combined with the fact that the subject's voice was quite soft anyway – this is therefore as a good an approximation as possible.)

E: First of all, did you get it? Did you see what it was trying to do?

S: Yes ... I quite liked the erm, file tracker, the when you've last edited thing, today, yesterday, some days before ... I think, when you can't remember the name of a file you've edited, that'd be quite useful ...

(some unclear dialogue)

E: Do you think, or what do you think could be improved about the interface to it, or maybe the ideas behind it?

S: I thought it was alright, looked OK ... erm ... well I don't know really. What do you mean?

E: One example was, the suggestions dialog, well, with this experiment you only ever see one suggestion in it although it's actually a list, and some people said, erm ... perhaps that should be highlighted straight away so you didn't have to select it?

S: Yeah, well I've seen that, one choice, and I've thought, select it and click the button, fair enough. Selected it directly.

E: Erm ... if NT had features like that, in there as standard, do you think that you might use them?

S: I don't know about the notes one, just, because it's not one of those things you're used to, but the file tracker ... I'd use that, I think.

E: Thankyou very much for your time.

C.7. Transcript from Debriefing Interview with Subject 7

E: Right ... did you, sort of ... get what it was trying to go, and the mechanisms in it?

S: Yes, well it was trying to make my life easier, and just find things that I used quite often, and say that there's a much easier way to get to them and do things with them.

E: Erm ... with your, when you said rather than have it say 'oh, I've got a suggestion' and then leave you to look at it, do you think ... would you, would that interrupt you very much, do you think? *(Earlier, during the activities, the subject expressed the opinion that it would have been preferable to have a direct suggestion made when the system noticed a duplicate access, rather than having to check the suggestion afterwards.)*

S: What if a box came up, and ... ? Well, I suppose it could annoy you after a while if stuff kept coming up and saying ... I think what you've got to get used to, is probably, I'm not used to looking for status messages *(the message line in the pseudo-toolbar window)* so if you get, once, once you're used to that being there and you just keep an eye on what's going on, then that's probably quite a good way to do it, probably I just like the, the kind of phrased as a question format that says 'would you like me to do this' and you can choose to ignore it if you want to. Er ... you know, I don't know, I suppose I'm just used to, I suppose you get used to the Windows API and the sort of central message box, popping up.

E: Erm ... if, if, if NT actually had features like this in it, do you think you would use them?

S: Erm ... yes, because they're ... yeah, I think you would, just because it is, and if it's there, and it's seamless then, you know ... if it makes life easier, then yeah.

E: Thanks for your time.

C.8. Transcript from Debriefing Interview with Subject 8

E: Right. Did you, kind of get what this system's trying to help ...

S: Yes, I did.

E: ... I mean, I think the main problem with this experiment's the directions, isn't it ...

S: ... never mind about the directions, but that, that's very good. That's extremely useful.

E: What did you think about the current interface to it?

S: I liked the interface, I thought it was ... clear, concise, easy to use, not too much information came up on screen, so you weren't overwhelmed with information ...

E: Did you think ... erm, when it ... said 'oh I've got this suggestion' do you think that would, would it be better for it to say 'do you want the shortcut made' and give you the whole message, or just say 'oh, there is a suggestion' and check it later if you want?

S: You could do it either way, but quite honestly, I enjoyed having ... erm ... a about, the first bit, the .. 'do you want to' ... what ... because it was personable, it was as though it was speaking to you, which I thought was good, I liked that.

E: Do you think there are some improvements that could be made to it as it currently exists ... is there anything you would change about it.

S: No ... it sits nice and snugly, without taking up too much space, so it's not obtrusive ... is, is that a particularly big screen? What's the screen size?

E: It's a sort of ... Super VGA resolution, so not huge.

S: No, I can't think of anything I'd suggest.

E: If Windows NT had features like this in it, as standard, you know properly integrated into the system, do you think you'd use them?

S: Yes I would, yeah. Absolutely, there's far too much opening and closing of files, and going back and forth, yes.

E: Thankyou very much.