Personalized Multimedia News Agents in a Broadband Environment

by
Wen Bin Kwan

A thesis submitted to the
school of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

M.A.Sc.
in
Electrical Engineering

Ottawa-Carleton Institute of Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
Ottawa, Ontario

September, 1996

ISBN 0-612-19979-7

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Table of Contents

Figures

Abstract

The growing demand for efficient, flexible yet cost effective services has produced a new perspective on *personalized service* as a form of asynchronous interactive computing with seamless support for communications and network. Personalization of services exists at two levels: the user and remote service provider levels. At the user level is a customized interface that provides access to networked services unlimited by distance. At the remote service provider level, the aim is to offer services based on individual needs as expressed in the user customized interface.

The gap between the two sides is filled by the activities of *agents*. Software agents are entities that constantly adapt to the needs of the user in terms of interests, time and space limitations. Agents observe and learn from the user the art of selecting information of value to the user. Autonomous agents have the right to make decisions on behalf of the user when dealing with other agents, resources or service providers. The processing involved can be done either locally (when static) or remotely (when mobile).

This thesis presents the architecture and environment of an agent-based system. The design of news agents for personalized news delivery over a broadband network is proposed. The mobile-agent paradigm has been derived from the demands of easy remote services accessibility. A new model, namely *aclient-iserver* model, has been created to provide an interactive and asynchronous solution to the existence of mobile agents. Personalization of services is achieved by collecting user interests locally and gathering information remotely, all transparent to the user.

Acknowledgements

I would like to express appreciation to my thesis supervisor Dr. Ahmed Karmouch for his constant guidance throughout the program. Professor Karmouch's association with researchers in the industry has given me a unique exposure to current technologies.

I would like to acknowledge the cordial relationship with each and every member of the MEDIABASE team, a group of dynamic and friendly colleagues.

I would like to thank my parents and siblings for their understanding and support.

I would like to dedicate this thesis to my Dad, late Sung Wha Kwan.

Acronyms

AI	Artificial Intelligence
AMP	Agent Meeting Point
ANS	Agent Name Server
AOP	Agent Oriented Programming
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
AUM	Adaptive User Model
CAP	Calendar APprentice
COACH	COgnitive Adaptive Computer Help
DAI	Distributed Artificial Intelligence
GA	Genetic Algorithm
GUI	Graphical User Interface
IP	Internet Protocol
IPC	Inter-Process Communication
IRA	Information Retrieval Agent
KIF	Knowledge Interchange Format
KQML	Knowledge Query Manipulation Language
LAN	Local Area Network
MAN	Metropolitan Area Network
MIME	Multipurpose Internet Mail Extension
PLACA	PLAnning Communicating Agents
QoS	Quality of Service
REV	Remote EValuation
RPC	Remote Procedure Call
SQL	Structured Query Language
TCL	Tool Command Language
TCP	Transmission Control Protocol
TK	Toolkit
WAN	Wide Area Network

Chapter 1

Introduction

1.1 Motivation

With the rapid growth in networks and communications more people are connected to remote information resources now than ever. The information highway and world wide web browsing tools like Netscape have become commonplace even among naive computer users. Communication services are constantly adapting to the needs of huge and diverse classes of users. Users want to spend less time to get more information with simplified procedures. A new class of services, namely *personalized services*, has evolved as a solution to the challenges of interactive and asynchronous access to existing services.

Personalization of services exists at two levels: the user and remote service provider levels. At the user level is a customized interface that provides access to networked services which are made transparent to the user. The user simply expresses high level goals and expects equally high level results, surpassing complex and time consuming details of communications. At the remote service provider level the aim is to offer services based on individual needs as specified in the customized interface. This amounts to a transformation of high level requests from one end to low level requests on the other end.

The gap between the two extreme levels of requests is too big to be bridged by existing solutions like the client-server paradigm. One reply for each request is too simple to achieve the goals presented in the previous paragraph. Besides, it may be too time consuming to await suitable results and uneconomical to hold on to the connections established during the entire operation. This calls for a new interactive yet asynchronous mode of communications between the service consumers and providers. Traditional methodologies for solving such problems can only be interactive or asynchronous at a time but not both.

The concept of 'agents' has posed a challenging solution for the information industry. The term 'agents' is derived from Latin 'agere' which means to act or to do something. An agent is like a personal assistant who is always alert to help the user achieve goals by breaking them into tasks and subtasks, in effect acting or doing something on behalf of the user. Agents can be present at the user as well as the service

provider level, making it possible for them to customize user needs at both the levels and bridging the gap by *interagent* communication. One problem with this approach is the continuous static presence of agents on different information sites. That is, rarely used agents may exist unnecessarily and tie up resources that can be used by other agents.

A further refinement of agents gives rise to 'mobile agents', agents that can perform the same tasks as the static agents but with greater flexibility and power. These mobile agents move around the network in search for information on behalf of the user. A mobile agent can be released into a network with connections for a short period of time. After the tasks are attained, short intermittent connections again can bring back the results to the user. Therefore, mobile agents have both the flexibility of existing at unlimited number of sites and the power to harness network resources by moving computation nearer to the sources.

The University of Ottawa's Multimedia Information Laboratory is an environment of distributed multimedia databases. Various aspects of multimedia information systems like document architectures, database models [EME93], synchronization [ROD95] and storage [WAN93] have been studied. Applications like multimedia news, multimedia learning and groupware have been explored. The application of agents in such an environment presents an interesting research item. The scope of personalized agents in a multimedia environment is worth exploring, especially when mobile agents can reduce network bandwidth or traffic by avoiding unnecessary transfer of potentially useless data.

1.2 Objectives

The main objective of this thesis was to design and implement a multimedia application of agents. This can be subdivided into three major portions: First, the architecture and environment had to designed with the constraints of agent-based computing in mind. Second, a mobile agent model had to be defined to test for tasks that it performed on behalf of the user. Finally, an application had to be chosen to specify the tasks that a mobile agent would perform.

The news application was chosen for two reasons. First, it has a running news-on-demand prototype in the laboratory that provides a good backdrop of what technical problems to expect and a suitable comparison of two closely related applications. Second, it is optimal in complexity; neither as simple as an email application nor as complex as distance learning. There would be enough time to study the many facets of the mobile agent problem and also build a quick prototype to test new ideas.

The Mobile News Agent is designed to provide personalized news as opposed to browsing a generalized newspaper in the news-on-demand project. The user need not be aware of the news sources in the network, what they contain and how they are accessed. There is only one interface through which high level requests are communicated to the agent. News articles according to the interests of the user are searched by the agent and brought back to the user in a customized presentational format.

1.3 Thesis Outline

The rest of the thesis is outlined as follows. Chapter 2 introduces the concept of agents from some important perspectives. A background knowledge of what led to mobile agent research and in what areas are presented. Definitions and meanings of terms used in agent-based software engineering and artificial intelligence are compared. The theories, architectures and languages of agents are discussed. The transition from static to dynamic (or mobile) agents and from monomedium to multimedia agents is projected. A survey of related work is presented for better understanding of the progress made in this area.

Chapter 3 develops the Multimedia News Agent System covering the design of the news architectures. Issues like the *multimedia profile*, profile manipulation and news selection criteria are described with respect to the multimedia newspaper. This chapter shows the shift from the news-on-demand project to the agentized news project.

Chapter 4 concentrates on the mobile agent and the environment in which it functions. The structural and functional components of a mobile agent are presented along with its entire life cycle. Mobile agents for multimedia news and other innovative applications like multimedia news documentaries are explored.

Chapter 5 reports the implementation of a mobile agent for multimedia news. An environment suitable for filtering and retrieving news articles by mobile agents is created. A sample session shows how mobile agents are created by the user and dispatched with a

task of retrieving news documents as specified. This implementation includes the complete life cycle of a mobile agent. Conclusions are made in Chapter 6 with suggestions for future research.

1.4 Main Contributions

The main contribution of this research is the design and implementation of a *mobile* agent for *multimedia* news. It travels the network in search of news articles in a personalized manner, that is retrieving articles of interest and importance to the user. The news articles themselves are in multiple media like audio, video, graphics, images and text. This work has identified and partially solved some critical issues in mobile agents as applied to multimedia applications.

Innovative use of mobile agents for manipulating media types has been explored to complement the monotonous task of finding suitable news articles. This introduces the possibility of applying pro-active agents in interesting scenarios like the creation of news *documentaries*. An agent system has been built that made it possible to test some new concepts and methodology. A mobile agent has been created that performed a complete cycle from its initiation to its successful remote execution.

The architecture of the news agent system is an important aspect of this research. The corresponding environment allows further extensions for incorporating other features of a mobile agent. In fact, it was designed without a specific agent in mind, the news application is only a concrete case of verification. Similarly, the mobile agent has been

defined with generic purposes in mind. The structure of the mobile agent can be extended with more details to add other features and capabilities.

New Tcl commands have been defined for user/agent and agent/service-provider interactions. They were implemented to complement the core commands of the Tcl/Tk package. The number of these new commands can easily be expanded to incorporate into other applications. In short, the work described in this thesis laid down the groundwork for future research and development in the area of mobile agents for multimedia applications.

Chapter 2

Agent Concepts and Issues

2.1 Overview

The concept of *agents* is not new but rather has a forty year old history in Artificial Intelligence. AI aimed at creating agents that exhibit some form of knowledge-based intelligent behavior. Until the mid 1980s researchers from mainstream AI gave relatively little consideration to the issues surrounding agent synthesis. However, since then there has been a burst of interests in agents by people from computer science to those working in data communications, concurrent systems, robotics and user interface. Agent-

based computing is perhaps the next significant breakthrough in software development. A UK-based consultant firm named Ovum has predicted that the *agent technology* industry would be worth some US$3.5 billion by the year 2000 [HOU94].

Researchers from academia, industry and business are keen on making the most out of this new technology. In the academic world, there are many well known laboratories like the MIT Autonomous Agents Group, UMBC Laboratory for Advanced Information Technology, Stanford Nobotics Group, University of Washington Softbots Research Group and UMASS Distributed Artificial Intelligence Laboratory, to name a few [UMB96]. In industry and business, some serious players include AT&T, General Magic, IBM, Agents, Inc. (a spin off of the Media Lab group that did HOMR) and Microsoft Research User Interfaces group.

Software agents are traditionally knowledge-base or rule-based. Knowledge-based agents have a knowledge repository in specific domains to help users attain certain goals. This repository has to be built by a knowledge engineer who has the expertise in the specified domain. Rule-based agents are controlled by 'if-then-else' rules which guide their behavior. The user is often involved in inputting the necessary and relevant rules. A more dynamic kind of agents, called the learning agents, came into the scene in more recent years. This kind of system starts with a rudimentary knowledge of the user, her preferences and work patterns. As the user work, these agents learn about the user's level of expertise, programming style and areas of personal interest.

One aspect that has not been dealt with in the above approaches is the *mobility* of the agent. It may change the way we communicate and work. Mobile agents will enhance telecommunications for the mobile worker. Users can access network services at different points of the network and at different times. What is more important is the fact that the user need not understand how the agent accomplishes tasks. The agent could travel throughout the network and route itself to the correct destination for achieving certain goals. At any point the agent could be observing any new development that would be of interest to the user. In effect the mobile agent will represent the user in mundane *negotiations* and *transactions*. IBM's Intelligent Communications and AT&T's PersonaLink [REI94] are two examples of commercial mobile agent technology.

All technology is built on old technology in one form or another. Agents represent an extension of current technology with the purpose of removing the drudgery of redundant and time consuming activities associated with interactive on-line computing. With the introduction of mobile agents we have moved into the third generation of on-line services, the agent-based services. The first generation was character-based and command-line-oriented like GEnie and CompuServe. The second generation was graphics-based like America On-line and Prodigy. Similarly, computer technology moved from host-based to client/server-based and now to agent-based. The transition from host-based to client/server-based evidently gave more power to the user. What power agent-based technology can unleash on the user (or agents themselves ?) is yet to be seen !

The rest of this chapter presents agent concepts and issues in several sections as follows. First, the background knowledge behind agent-based technology that is bringing about a transition in software development is given. Various reasons that led to a better approach each time are briefly described. Second, common terminologies used in the agent world are explained in line with a comprehensive idea of the whole picture. Different views are put into proper perspectives for later usage. Third, the vast area of research in agents is presented to show the extent of influence agents can have. Fourth, a technical discussion on agent theories, architectures and languages follows. This provides the theoretical basis of the work presented in this thesis. Fifth, a comparison is made between static and mobile agents and then the transition from monomedium agents to multimedia agents is discussed. Sixth, related work from other researchers are briefly described and critiqued. Finally, important observations about the development of agents are brought out from the previous sections of the chapter.

2.2 Background Knowledge

Client/server model serves as a popular form of service provision across the networks. The client and the server communicate via message passing; the client 'sends' a request message to the server, which 'receives' and processes the request and sends a reply message back to the client. Typically the reply message from the server is the result of the request, for example, it could be a technical report from a database.

The send and receive primitives used in the client/server model can be classified into two categories: (a) blocking or nonblocking and (b) synchronous or asynchronous. Blocking refers to the primitives that do not return control to the caller until the message is successfully sent or received. Non-blocking refers to the primitives that return control immediately whether or not the message is successfully sent or received. Synchronous means that the send primitive returns only when the recipient of the message issues a corresponding reply. Asynchronous means that the send primitive returns right after the message is sent to the destination address where the message is buffered until the recipient issues the corresponding reply.

Message passing is a low level manipulation, therefore, it gives the programmer more power and flexibility. However, this requires the programmer to know how to handle low-level details like keeping track of corresponding request/response and system errors. The message also implicitly carries the protocol defined for communications between the client and the server; thus conversion of data between client and server formats has to be dealt with.

Remote Procedure Call (RPC) was designed to hide the low-level details of message passing. The client can use RPC to invoke a procedure on the server just like a procedure call in a local program. Typically, the client places the procedure name and parameters into a message and sends the message to the remote server. On receiving the message, the remote server extracts the procedure name and parameters and invokes the

appropriate procedure. RPC handles the low level 'send' and 'receive' functions which are transparent to the programmer.

However, one problem with traditional RPC is its inability to send incremental or huge results from the server to the client. For example, a server is restricted in the amount of data it can transfer to the client. This problem is solved by the introduction of 'pipe' concept. A 'pipe' is a connection between the client and the invoked remote procedure that exists for the duration of the RPC. This 'pipe' allows incremental results and bulk data to be transferred.

Another problem with RPC is that a distributed application has two parts, namely the client and the remote procedures. This means the client is limited to the operations provided at the server. Since the server operation may not meet the client's exact needs, the client has to make several RPCs. Intermediate data is transferred across the network on every call and if it turns out to be useless, a significant amount of bandwidth is wasted.

To overcome this last problem, a new approach of Remote EValuation (REV) had been suggested by researchers. In REV a subprogram is sent from the client to the server. The subprogram executes on the server and returns its results to the client. The procedure can be transferred as source, intermediate or compiled code, depending on the language, security level and heterogeneity of the network. The restriction of REV is that the procedure must be self-contained, that is all functions and variables referenced in the procedure must be provided at the server or sent along with it. The semantics of a procedure call is defined in the passed subprogram.

Transportable agents is an extension of REV schemes where autonomous programs migrate from machine to machine. Most of these systems allow a program to suspend its execution at some arbitrary point and resume execution on a different machine. Others allow subprograms to be sent to a remote site but not their internal states. The common feature among all these systems is that the programs, which are called agents, are mobile and they are expressed in high level scripting languages like Tcl, Custom and Telescript [WHI94].

There are several schemes that fall between REV and transportable agents. MIME/Safe-Tcl embeds Tcl scripts in email messages and these scripts are executed automatically when the messages are received or viewed. Apple Script is a scripting system that allows scripts to be sent from one application to another. IBM's Intelligent Communications Network sends Intelligent Objects, which possibly contain procedures, from one application on one site to an application on another site [REI94]. Oracle Mobile Agents associates an agent with each mobile user but the agents are not transportable.

Before the invention of transportable or mobile agents, ideas about agents existed in various perspectives. In Marvin Minsky's "Society of Mind" he viewed an 'agent' as a machine that accomplishes something without the user knowing how it works [MAR85]. He used the term 'agency' to suggest the image of an office or an organization that is composed of several interacting subagents. Other researchers used the metaphor of personal assistant to describe an agent. A well known example is John Sculley's great futuristic promotional video called "The Knowledge Navigator" produced by Apple

Computer. Phil, the agent, is portrayed as a smart teacher who assists a human user in resource management.

Personal agents are further classified into Advisory, Assistant, Internet and Communicating agents [IND95]. Advisory agents are those that offer advice and instruction to help the user but do not carry out the tasks. These agents have expert knowledge in particular domains. Assistant agents assist the user in accomplishing tasks even without direct intervention or feedback from the user. These agents are more powerful but there are many unresolved technical and social issues. Internet agents are tools for managing the vast amounts of available information on the Internet. These agents gather information and sometimes present an integrated view of the Internet as a whole. The agents described so far act as intermediaries between the users and resources. Communicating agents use a common communication language to interact and collaborate among themselves instead of the resources themselves. This brings in added complexity and privacy issues.

Over time different researchers had different viewpoints about agents, giving rise to many characterizations of agents in various fields. Everyone seems to be having his own share of notion in this area of research, adding up to a host of definitions and meanings. Therefore, it is important to have a comprehensive or panoramic view of 'agents' before we put them in perspectives relevant to this thesis.

2.3 Definitions of Terms and Perspectives

The question "what is an agent ?" draws heated debates whenever researchers try to explain their work as "agent-based". It is to some extent embarrassing for the agent community just as the question "what is intelligence ?" is embarrassing for the Artificial Intelligence community. Although the term "agent" is being used by many people working in closely related fields, it does not have a universally accepted definition.

Since the notion of agents stemmed out of AI, the AI community tend to give strict definitions of agents and discard others as "noise" terms. This could be justified especially when someone starts misusing and abusing the term by even calling a compiler an agent. However, there is a tremendous amount of work that can be classified as agent-based but not necessarily AI-based. Agents in the most general sense are characterized by the following properties [MAS90], [GEN94]:

- Autonomy: agents control their own actions and internal state as opposed to direct manipulation by human users.
- Social ability: agents communicate or collaborate among themselves via some kind of agent communication language.
- Reactivity: agents perceive the changes in their environment and respond to those changes.
- Pro-activeness: agents not only react to changes but also take initiatives to achieve goals.

The term "agent" has a stronger and more specific meaning for a computer system that, in addition to the properties mentioned above, is either conceptualized or implemented using concepts that are more usually applied to humans. For example, AI characterizes an agent on the basis of mentalistic notion, such as knowledge, belief, intention and obligation. Some AI researchers have gone further and considered emotional agents in terms of human-like mental states. Those interested in human-computer interfaces have given agents human-like attributes by representing them visually with a cartoon-like graphical icon or animation.

There are some terms that refer to other attributes of agents. *Mobility* is the ability of an agent to move around a computer network. *Veracity* is the assumption that an agent will knowingly communicate false information. *Benevolence* is the assumption that agents do not have conflicting goals, and that every agent will therefore always try to do what is asked of it. *Rationality* is the assumption that an agent will act to achieve its goals and not hinder its goals.

Having put into perspectives the notion and features of an agent, the issue of "intelligence" has to be discussed. What distinguishes an "agent" from an "intelligent agent" is a question worth asking. Agents may give rise to systems which appear intelligent because they engage the concerns of the people using them. But there is no need for individual agents to appear intelligent. There are a few ways to look at "intelligence" in an agent:

(a) Minsky's "Society of Mind" treats agents as individually very simple, but they give rise to "intelligence" when acting together, in certain very special ways, in societies [MAR85].

(b) Shoham's paper on "Agent-Oriented Programming" [SHO90] sees agents as having beliefs, commitments, capabilities, etc., without any explicit implementation of intelligence.

(c) Dennett's "The Intentional Stance" reminds us that we attribute intentionality (intelligence) to what we don't understand at a physical or a design level. The attribution of intelligence to a system, whether or not it consists of one or more agents, is a matter for the observer.

(d) By no means all agent research is going into anthropomorphic entities that somehow imitate humans. On the contrary, agent-based software, often using speech acts for intercommunication between agents, implements situated interactions more than intelligent behavior in isolation.

(e) By no means all agents use one or more of first-order logic, Horn clauses, semantic nets, natural language processing, plans, knowledge bases or any other AI intelligence concepts. Those that do, are not intelligent simply by virtue of their containing such algorithms.

2.4 Areas of Research

The term "agents" has been used to refer to the following technologies:

(i) Asynchronous remote procedure calls.

(ii) Asynchronous queries and updates (usually of the non-SQL type).

(iii) Predicate Calculus or rule-based systems or simply expert systems.

(iv) Mobile programs.

(v) User generated scripts which perform a job for the user.

What is really new and different in agent technology appears to be in the area of distributed interaction. Points (i) through (iv) attempt to solve the following problems: scaleability, wide area distributability, heterogeneity and disconnected operation of entities that have distributed interaction. Similar approaches are applied in different areas of specialization like operating systems, languages, databases and knowledge-based systems.

Operating system researchers work toward the support of mobile workers and hence mobile processes. They come up with the methods to support asynchronous RPCs and mobile processes. Database researchers work toward supporting information integration across autonomous, widely distributed, heterogeneous databases. They come up with a way to support asynchronous and mobile queries. The knowledge-based researchers have the same goals as database researchers but they come up with a way to support asynchronous assertions and shipping of rules or predicate calculus expressions.

In all of these areas of research, the achievement is the support of asynchronous interaction where computation moves closer to where it will be required or utilized. This computation happens on heterogeneous targets like operating systems, interpreters, data

and knowledge bases. In the distributed interaction area, all these researchers share the same high level goals, except being specialized in their own areas of research.

The applications of agent technology range from cooperative problem solving to interface agents. Distributed Artificial Intelligence (DAI) applies a group of agents to cooperate in solving problems and tries to coordinate the activities of such a group. DAI researchers have applied agent technology in air-traffic control [STE88], intelligent document retrieval, telecommunications network management, computer integrated manufacturing, power systems management, particle accelerator control, patient care, concurrent engineering, transportation management, job shop scheduling and steel coil processing control.

Interface agent uses the metaphor of a *personal assistant* who is collaborating with the user in a particular application environment. The personal assistant virtually watches over the shoulder of the user to observe and help her perform some routine tasks. Some prototype applications of interface agent include NewT, a USENET news filtering agent system, entertainment selection, email handling and appointment scheduling [MAE94].

Agents applied to search information sources are called *information agents*. They are able to collate and manipulate information obtained from these sources in order to answer queries posed by users and other information agents. The information sources may be of various types like traditional databases as well as other information agents. An

example of an information agent is the prototype system called Information Retrieval Agent (IRA) [UMB96].

2.5 Theories, Architectures and Languages

2.5.1 Theories

Agent theory is concerned with the question of what an agent is or what properties an agent should possess, and formalize the representation and reasoning [TAN90] about the properties of agents. Agent theories are essentially specifications for agents. Agent theorists develop formalisms for representing the properties of agents and using these formalisms, try to develop theories that capture desirable properties of agents.

A complete agent theory, expressed in a logic with these properties , must define how the attributes of agency are related. It will need to show how an agent's information and pro-attitudes are related; how an agent's cognitive state changes over time; how the environment affects an agent's cognitive state; and how the agent's information and pro-attitudes lead it to perform actions. Accounting for these relationships is perhaps the most significant problem faced by agent theorists.

2.5.2 Architectures

An agent architecture is a particular methodology for building agents. It involves the decomposition of an agent into a set of component modules and the specification of how these modules interact. The module interaction defines the current internal state of

the agent and the actions it takes. The architecture therefore includes techniques and algorithms that support the methodology. It can be designated by boxes with arrows indicating the data and control flow among the modules.

The classical approaches use the so-called *Deliberative Architecture* based on symbolic AI paradigm, that is the physical-symbol system hypothesis. A physical symbol system is defined to be a physically realizable set of physical entities (symbols) that can be combined to form structures, and which is capable of running processes that operate on those symbols according to symbolically coded sets of instructions [GES87]. The hypothesis says that such a system is capable of general intelligent action. Therefore, a deliberate architecture contains an explicitly represented, symbolic model of the world, in which decisions, like what actions to take, are made via logical reasoning based on pattern matching and symbolic manipulation.

Knowledge representation, automated reasoning, speech understanding and learning, among others, is not anywhere near solved. Even trivial problems such as commonsense reasoning have turned out to be extremely difficult. The underlying problem seems to be the difficulty of theorem proving in even very simple logics and the complexity of symbol manipulation in general.

The problems faced in Deliberative approach led to the development of *Reactive Architectures*. It does not include any kind of central symbolic world model and does not use complex symbolic reasoning. Brooks [BRO90] from MIT propounded three key theses:

(i) Intelligent behavior can be generated without explicit *representations* of the kind that symbolic AI proposes.

(ii) Intelligent behavior can be generated without explicit *abstract reasoning* of the kind that symbolic AI proposes [GEO87].

(iii) Intelligence is an emergent property of certain complex systems.

Brooks identified two key ideas that have formed his research:

(a) Situatedness and embodiment: 'Real' intelligence is situated in the world and not in disembodied systems such as theorem provers or expert systems.

(b) Intelligence and Emergence: 'Intelligent' behavior arises as a result of an agent's interaction with its environment.

Maes from MIT developed an Agent Network Architecture where an agent is defined as a set of competence modules [MAP91]. These modules resemble the behavior of Brooks architecture. The *relevance* of a module in a particular situation is specified through an *activation level*. The behavior of an agent is influenced by higher *activation level* values of modules. During an agent execution, various modules may become more active in given situations and may be executed. Competence modules are defined in declarative terms, therefore, their meanings are easier to know than that of a node in neural network architectures; the node only has a meaning in the context of the net itself.

Beside the two extreme architectures, many researchers follow a median approach called the *Hybrid Architecture*. Examples of this approach include Procedural Reasoning

System (PRS), TouringMachines, Cosy, etc.. These hybrid architectures have some advantages over both purely deliberative and purely reactive architectures. However, a potential difficulty with such architectures is that they tend to be ad hoc in that while their structures are well-motivated from a design point of view, it is not clear that they are motivated by any deep theory.

2.5.3 Languages

An agent language allows programming hardware or software computer systems in terms of some of the concepts developed by agent theorists. The language is expected to have some structure corresponding to an agent and some attributes of agency like beliefs, goals and other mentalistic notion. This subsection presents some languages and their general features.

(a) Agent-Oriented Programming

Agent-Oriented Programming (AOP) by Shoham programs agents in terms of the mentalistic and intentional notions that agent theorists have developed to represent the properties of agents [SHO90]. A fully developed AOP system has the following three components:

- A logical system for defining the mental state of agents.
- An interpreted programming language for programming agents.
- An 'agentification' process for compiling agent programs into low-level executable system.

(b) AGENT0

The corresponding implementations include AGENT0 and Planning Communicating Agents (PLACA) [THO93]. The latter was a refinement of the former. It addressed one severe drawback of AGENT0, that is the inability of agents to plan and communicate requests for action via high-level goals. However, both of these languages cannot truly execute the logic associated with the interpreted programming language. MetateM language developed by Fisher has a stronger claim in this respect. A Concurrent MetateM system contains a number of concurrently executing agents, each of which is able to communicate with its peers via asynchronous broadcast message passing.

(c) April and Mail

April and Mail are two languages for developing multi-agent applications [HAU94]. The former was designed to provide the core features required to realize most agent architectures and systems. Therefore, it provides facilities for multi-tasking, communication, pattern matching and symbolic processing. Multitasking is done via processes, which are treated as first-class objects, and a UNIX-like fork facility. Communication is supported with powerful message-passing facilities for network-transparent agent-to-agent links. April's primitives are general enough to allow building of an agent or system architecture from scratch. In contrast the Mail language has a rich collection of predefined abstractions including plans and multi-agent plans.

(d) SodabotL

SodaBot is a general purpose agent user-environment and construction system with an agent operating system [COE94]. The agent programming language that accompanies the system is called SodaBotL, whose high level primitives and control structures are designed around human-level descriptions of agent activity. It abstracts low level details of agent creation like system calls, mail servers, sockets and X-windows. Two kinds of agents can be built with SodaBotL, namely, personal-assistant and application agents. Personal-assistant agents can be programmed to filter incoming email or notify user of specific events. Application agents, also called SodaBots can be created for different services like conducting a poll among a group of people.

SodaBotL primitives include Contact Agent, Request Agent, Load, Mail, Reply, Save for interaction among agents using email as the form of communication. Contact Agent and Request Agent are used to issue requests to each another. Load primitive loads the contents of a file into a variable. Mail sends a string to a given address while Reply returns a message string to the current mail message. Save primitive saves a string into a specified file. The GUI primitives include Display, Get Response and Query. Display presents a window containing a specified string on the user's current display. Radio buttons are used to let the user select or choose possible responses. Get Response window allows the user to enter a textual response to the specified prompt. Query window allows the user to enter a one-line response to the given prompt.

(e) Telescript

General Magic's Telescript is perhaps the first commercial agent language [WHI94]. It is a pure object oriented language similar to Smalltalk. Telescript is an interpreted language accompanied by a Telescript Engine. There are two language levels, High Telescript and Low Telescript. High Telescript has an object oriented syntax and is compiled to Low Telescript which has a postfix syntax for stack-based interpretation. The basic network configuration is to run a Telescript Engine on each node in the network. The network of interworking Telescript Engines provides an abstract homogeneous environment in which distributed systems can be built.

The most important class in the Telescript language is the Process. The language supports preemptive, prioritized multi-tasking of Process objects. A Process instance can be thought of as an object with a life of its own. Place and Agent are two important subclasses of Process [WAY95]. A Place object represents a virtual space in which other objects can interwork through local communication and each Telescript Engine can support a number of places. An Agent object is a Process object which can migrate between Places on the same Engine or different Engines. Hence, the Telescript notion of a distributed system is a number of distinctly located places and a number of Agents which move between these Places. Places provide meeting locations for Agents where they can exchange information and perform computation.

(f) Tcl and Tk Toolkit

Tool Command Language (Tcl) is an interpreted scripting language [OUS94] accompanied by Tk toolkit for building window widgets. Tcl scripts are in the form of command strings which consist of one or more *words*, the first word being the command name and the rest being arguments to that command. These commands are evaluated in two steps; first the Tcl interpreter parses the command string into words while performing substitutions, then a corresponding command procedure executes the command, processing the words previously to produce a result string. All the commands are separated from the next command by a newline character.

Tcl scripts are invoked only when a Tcl application is running. The freely available package in the internet provides a simple Tcl shell application called *tclsh* which accepts command line scripts. The corresponding shell for Tk is the *wish* shell which also encompasses the *tclsh* when running, that is Tcl commands are also accepted. Other Tcl applications can be written in C to enrich the sixty built-in core commands [WEL95].

New commands can be defined in either an *action-oriented* or an *object-oriented* style. In an action-oriented approach each command corresponds to an action that can be taken on an object, which is the argument to the command. In an object-oriented approach, each command corresponds to an object, whose name is the name of the command. The first argument to the command specifies the operation to be performed on the object. These two features are very suitable for a news-agent application where user

actions are to be monitored. An event command can be invoked in an action-oriented fashion while a Tk's widget command can be implemented in an object-oriented fashion.

Both Tcl and Tk are C library packages that provide a collection of procedures that can be invoked from an enclosing application. Tk's C interface is commonly used to build new kinds of widgets. The full interface of an application is normally a combination of Tk's built-in widgets and the newly built widget classes. The interface with C paves the way to implement new Tcl commands.

The newly defined commands provide the structures in C for managing large complex scripts which cannot be handled by the little structure present in Tcl. For example, access to network sockets is done at a lower level with C. The codes for creating application specific commands have three parts: (a) Procedures that implement the new Tcl commands; (b) Initialization codes such as registering the application's new commands with the Tcl parser; (c) Command loop that collects Tcl scripts and passes them to the Tcl parser for evaluation. The major part of the codes is contributed from the new commands.

(g) KQML

Knowledge Query Manipulation Language [FIN92] has three layers, namely, Content, Message and Communication layers. The Content layer uses a communication language agreed upon by the communicating entities. That is two agents using the same internal language can use that language as the communication language in the protocol.

The content could be a query, a response, an assertion, etc.. The Message layer encapsulates a content expression that an agent would like to transmit to another agent. The Message layer encapsulation specifies the nature of the content (query, assertion, etc.). Finally, the Communication layer encapsulates the Message layer message into a so called "package" which specifies communication attributes such as the sender and recipient of the message. The Communication layer thus handles the routing of messages. A KQML message is called a *performative*, that is the message is intended to perform some action by virtue of being sent.

(h) JavaAgent Template

JavaAgent Template [STF96] is written entirely in the Java programming language [SUN94]. It provides a structural framework for creating autonomous software agents which can be executed as either stand alone applications or as Java Applets via a browser. Specialized agents can be created by subclassing the generic classes provided in the source code. Agent naming and addressing are handled by an Agent Name Server (ANS). Each of these agents has a graphical user interface for user interaction and a socket-based communication interface for the exchange of KQML messages with other agents.

The agent template automatically handle the asynchronous exchange of KQML messages between agents distributed over the Internet. It is assumed that all messages passed between agents are KQML messages and that they will have the following

minimum set of fields: performatives, sender, receiver, language, ontology and content. When an agent is first connected to the network, a ServerThread is spawned that creates a server socket on the local port specified in the agent's address. Whenever a client socket attempts to bind to the server socket, a new ServerThread is recursively created to wait for more client connections. Whenever an agent tries to send a message, a ClientThread is spawned that first checks for an address for the message receiver. If an address is found, then a client socket is created and connected to the receiver's server socket. Otherwise, the message transmission is blocked and a request message is sent to the ANS.

There are a few known problems with JavaAgent Template. It does not work in Netscape due to security restrictions. Instead, it requires a browser which allows applets to make arbitrary socket connections. System messages can appear in a random order since they are sent by different threads.

2.6 Static versus Mobile Agents

Interface agents are typically static agents while information agents likely fall into the same category. They are static in that the agent processes do not move from node to node in the network but rather their tasks are achieved while they are in a fixed location. They may however access remote resources via 'ftp' or 'telnet' etc.. These may suffice in applications where not much information is collected by the agent except for those that exist locally. The aim in those cases is to customize the user locally. This approach may

not be suitable for applications that require to download huge amounts of data, most of which turn out to be irrelevant.

With the rapid advance in information systems and networks we are often faced with huge amounts of information to process. Very often we cannot afford to wait for search results or to overload our network with the unnecessary transfer of data. To address both of these issues, we need to employ asynchronous communications with intermittent connections and to delegate information processing tasks to the *background* while we work on something else.

Mobile agents have emerged as a possible solution to these issues [CHD94]. Agents can be *injected* into the network to perform information processing. They travel intelligently through the network in a *pseudo-connectionless* fashion - i.e. upon arrival at each node they release the network socket they have used. However, they can find their way back to their master user if the assigned task requires so. The mobility of the agent allows asynchronous communications and their intelligence reduces the amount of network bandwidth used to exchange superfluous data - i.e. data that has limited or no value to the user.

Mobile agents can accommodate the use of mobile clients, namely mobile devices like laptop and notebook computers. They are only intermittently connected to the network while accessing the services provided by a server. Even while they are connected, they have relatively low-bandwidth connections. Since these mobile devices usually have limited storage and processing capacity, it is advantageous to engage mobile

agents in both information retrieval and filtering at a server and return only the relevant information.

Mobile agents also have specific applications in multimedia - and in particular, multimedia news. It is one application where the advantages of reducing network traffic and remote searching/filtering become obvious. Moreover, mobile agents can be exploited in innovative ways to make multimedia computing ubiquitous.

2.7 Monomedium versus Multimedia Agents

Much research has been done with respect to the behavior of agents as discussed in the section on agent theories, architectures and languages. The works described therein deal with monomedium or single-medium environment. Although we are in the initial stages of moving agents from a monomedium environment to a multimedia environment, this trend is only natural and probably here to stay.

One can hardly name an application where only one medium is used and will likely be continued in the future. On-line internet services, digital libraries, news-on-demand and other database services are all surging ahead in multimedia. We can only meet the challenges posed by multimedia communications if we keep multimedia agent research abreast or parallel.

Recent works by MIT's Media Lab produced agents that selected music and books. Bellcore's LyricTime, a music entertainment system incorporated text, graphics

and audio. Although we are far from achieving truly multimedia agents like Apple's Navigator video, we can approach the problem in two steps. We should first focus on how agents can manipulate multimedia information and then make a transition into multimedia user-agent interactions.

In this thesis, agent technology is introduced into a multimedia platform which is described in Chapter 4. The agent is posed with the problems of handling multiple media when assigned to any task. In particular, the agent is dispatched to retrieve and filter multimedia news documents stored in a distributed multimedia database. The representations of user interests become complex by virtue of being in multiple media. The agent's adaptation to the ever changing user interests become more dynamic. The presentation of documents to the user calls for a real time multimedia presentation.

Beside the mundane task of keeping track of user interests and an updated database, the agent can be employed in producing multimedia documentaries to help the user maintain a logical and hopefully semantic line of events. These agents may be released into the network to scourge for possible news articles which will be of interest to the user and possibly combine the bits and pieces of various events into a single story presentation. This gives rise to a *mobile* agent system in a *multimedia* environment.

2.8 Related Work

Traditionally there are two ways of building intelligent agents, knowledge-based and rule-based. The former approach is widely used in the field of Artificial Intelligence.

UCEgo [CHI91]] is one such agent that helps the user in using the UNIX operating system. In this approach there is a knowledge engineer that inputs large amounts of domain-specific knowledge about the application and the user into the agent. This approach creates an agent that is static and does not adapt to the behavior of the user since it is not programmed by the user. Besides, it is domain-specific and each domain requires a different knowledge base to function. The learning problem is not solved by instantiating a system with a complex, large knowledge base, which cannot easily be added to, deleted from or reorganized in any significant way [YUE90].

In the latter approach, the user had to input rules to guide the agent, that is, the end-user programs the agent with rules. The Oval system by Malone and Lai [LAI94] consists of semi-autonomous agents that are a collection of user programmed rules. Tasks are performed on the basis of these rules. This approach evidently needs a large input of rules from the user since she initiates the way tasks are performed and this requires an experienced user.

Ted Selker's COACH (COgnitive Adaptive Computer Help) [SEL94] builds an Adaptive User Model (AUM) by observing the user's actions. The AUM is a set of user model frames for syntactic and conceptual parts of the working domain. Domain Knowledge and Coaching Knowledge are employed in the system. Domain Knowledge is modified by the AUM created by reasoning which in turn is controlled by Coaching Knowledge. The system provides an interactive on-line guide or help for programming in

LISP. It is designed for text-based interfaces and inherits the drawbacks of a knowledge-based system.

Both of these approaches do not help agents to evolve themselves over time. That is, the agents tend to be static. Preferences can be specified by the rules or knowledge base but the relevance of information is not taken into account. Both of these approaches also have restrictions pertaining to the human. A knowledge engineer cannot be depended upon by a wide spectrum of users and the application domain is highly specialized. On the other hand, users are not only burdened with defining rules but also the technical details of the agent.

In recent years a so called "learning approach" also arising out of *Artificial Life* has been introduced into the scene [SHB94]. Artificial life dwells on the mechanisms by which organisms organize themselves and adapt in response to the changes in their environment. Agents based on this approach are meant to be autonomous, that is, they program or evolve themselves.

The difference of this approach from the previous two is centrally on the learning and adaptation aspects of the agent [MAP94]. Here the agent's knowledge is not programmed by fixed rules, much less by complete prior knowledge in a specific domain. Instead the agent is given only a minimal amount of background knowledge and it learns to acquire relevant knowledge for adapting to the user's habits. It makes use of the fact that the application involves a substantial amount of repetitions which can differ for

different people. Examples of this approach include LyricTime prototype system and CAP (Calendar APprentice) as described later.

There are four ways in which an agent can learn about a user (refer figure 2.1). Two of them are direct instructions from the user, including user examples and user feedback. For example a user may show the agent an instance of an article being saved in a file or indicate dislike for an article by giving it a bad feedback. The other two ways of learning by an agent are indirect observations. The agent tries to imitate what the user does while he is interacting with the application. It can also interact with the application to learn about the possible responses of the user. Finally, the agent could consult with other agents (belonging to friends of the user) to seek advice when it comes across unknown circumstance.

LyricTime prototype system [LOE92] developed by Bellcore is a personalized music selection system. It has a collection of 1000 songs in a database from which songs can be selected on the basis of song descriptions, listener profile and listener feedback. Although the system uses audio, image and text, the main medium is audio and the others are supplementary. For example a still image stays on the screen while a song is being played. Mood (romantic, calm, cheerful, etc.) is the main criterion for selecting the song. This kind of criterion cannot be defined in a news filtering system. Besides, the class of users is restricted to the casual users.

CAP (Calendar APprentice) [MIT94] presented by Tom M. Mitchell et al is a calendar scheduling agent similar to the meeting scheduling agent. It learns the users'

scheduling preferences from past experience, i.e., by observing users' actions through routine use. The learning method involves decision tree induction. However, this system is still rule-based except that the creation of rules are automated. Many rules are involved and they change with time. In the news domain we are more concerned with user interest representations than using rules. There is actually only one general rule in the news domain, i.e., filter and retrieve articles that match user interests.

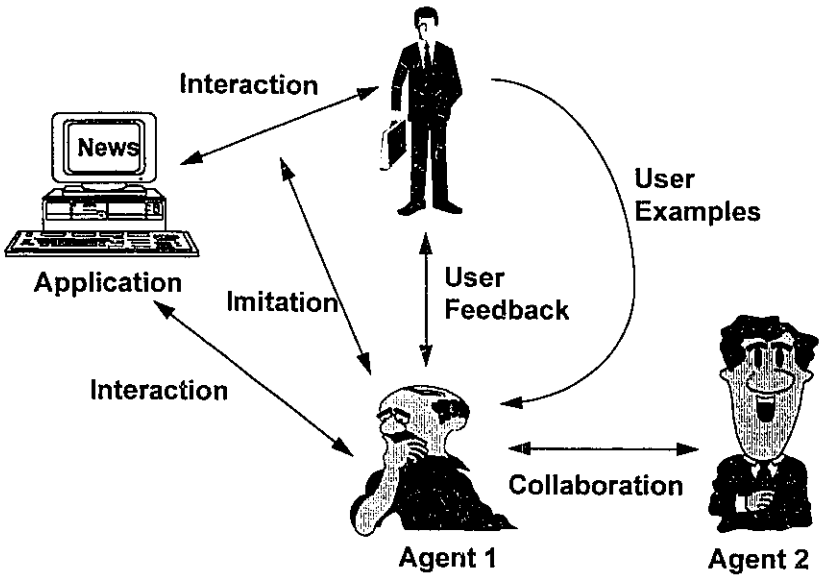


Figure 2.1: Ways in which an agent can learn about a user

MIT's Media Lab built some prototype agents that provided personalized assistance with meeting scheduling, email handling, electronic news filtering and entertainment selection [MAE94]. Memory based learning was used to keep track of the emails and schedules of meetings (accept/reject/negotiate etc.). A family of evolving agents in NewT [SHE93] based on Artificial Life techniques -- Genetic Algorithm

[YUE90] consisting of crossover and mutation operators-- was used to select news from the USENET netnews. Entertainment selection is based on "social filtering" which means the agents of different users "socialize" among themselves to make recommendations instead of correlating the individual user's interests with the contents of the available items. All these systems are based on machine learning techniques to acquire knowledge for assisting the users.

The work presented here is related to the electronic news filtering system (NewT) developed at MIT. However, there are certain inherent differences in terms of the platform and application. Our platform is one of distributed database and we consider the delivery of proprietary multimedia newspaper in a broadband network. We have tried to avoid Artificial Intelligence methods of learning like the Genetic Algorithm (GA) approach. This is due to the fact that crossover and mutation operators cannot be applied to our newspaper articles which are not classified into newsgroups as in USENET netnews. Thus, computation of profile fitness has been greatly simplified. Besides, our intention is not to validate AI techniques but rather to build an application capable of performing personalized filtering of multimedia information such as the multimedia newspaper.

We also do not depend on keywords to provide complete description of user interests, instead we have used multimedia representations. Other implicit user feedback, like time and media used, have been applied in our system. We have incorporated the

mobility of agents that has brought out interesting issues. Below are a couple of ongoing commercial projects that are related to the work presented in this thesis:

(a) IBM's Itinerant Agents

IBM laid down an abstract framework for itinerant or mobile agents for implementing secure, remote applications in large public networks. Client devices like laptop or desktop PC are used for launching an itinerant agent to a Yellow Page server. This server proposes relevant servers to be visited to accomplish the user's request. The agent execution environment is called an Agent Meeting Point (AMP) where after completion of its task, the agent collects its state, information acquired and request to be transported to another server before termination. The functional decomposition of AMP is shown in figure 2.2 [CHE95]. An AMP is a set of object oriented classes for providing the set of base services that can be used to register the services that a specific AMP will use.

An itinerant agent has three styles of interaction with the servers it visits that can be represented by three corresponding models. An Information Dispersal/Retrieval model is based on ask/receive paradigm between and itinerant agent and a static agent. This model is like a client requesting for updates from a server. A Collaborative model refers to a group of itinerant agents meeting at an AMP to exchange knowledge in order to reach a combined solution by compromise and evaluation. A Procurement model is

meant for an open-bidding auction where a group of itinerant agents attempt to bid for services offered by a static auctioneer agent.

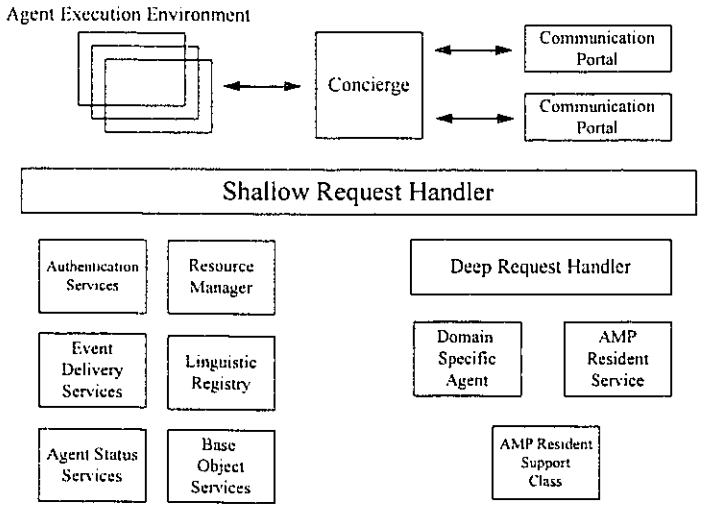


Figure 2.2: Functional level decomposition of Agent Meeting Points © IBM

The itinerant agent framework has touched upon several major issues including agent programming language, knowledge representation language, agent execution, security and privacy. This abstract design is yet to be implemented as such. Ongoing work is being done on some other issues like graphical user interfaces for launching and receiving agents, static agent support, agent status management, electronic cash services and virus control services.

(b) AT&T's PersonaLink

PersonaLink is a network service for email, information retrieval and on-line shopping [REI94]. Since the service is built around Telescript, it requires Telescript

enabled devices and software for access. Initially the network consists of centralized servers accessed by 800-number service or wirelessly through the Ardis packet radio network, although it is designed to be scaleable and distributed. One of the ways in which PersonaLink is different from other services is its multimedia component. It supports multimedia message types including voice, graphics and annotation.

A message is created using a hand-held device running the Magic Cap operating system. Then a mobile Telescript agent is formed by combining the message with a script that instructs it where and how to go. The script executes locally until it encounters a 'Go' operation when it prepares to transmit the agent by saving all the variables, pointers and stack values. For agents heading outside the local machine, they are wrapped up in a message, encrypted by a local RSA-like public key and sent across the network.

At the receiving service node, the agent is processed through an authentication and security server. On passing the security barrier, it goes to a node transport subsystem which checks the recipient's mailbox address from a database. The mailbox which is a stationary Telescript agent itself decides to accept or reject an agent. In the former case, the message is unwrapped and the script continues to execute. Interaction between the mobile and stationary agent agents occur when a 'Meet' command is encountered.

2.9 Observations

Agent technology is being considered leading edge technology not for no reason. It has turned computer industry into a relatively new direction for solutions to real world

problems in operating systems, languages, databases and knowledge-bases. As a complement to present technology, it has opened seamless venues for improving services. Email packages, for example, are trying to incorporate the functionality of agents in order to provide greater degree of personalization. Additions to software packages like this offer attractive solutions for future demands as information overloads and time constraints increase.

At times the agent approach becomes highly autonomous, meaning the agents are given ample power to make decisions on behalf of their human masters. This raises the issues of *trust* and *competence* on the side of agents and *acceptance* on the side of human users. How much we can trust the agent's ability to make non-conflicting decisions has no easy answers. Psychologists are studying the *emotions* involved in having autonomous agents function as 'personal assistants' [BAT94]. It will be a long time before we reach any conclusions, but the truth is, agents can have great impact on those who use them.

Agents promise a viable solution for mobile computing. In the past decade, telecommunications created the environment that made it possible for office workers work from home. With the advance in wireless communications workers have become mobile but have limited bandwidth access to computing resources. Now the introduction of agents has bridged that gap. Mobile workers do not need high bandwidth access anymore, instead they can delegate jobs or tasks to the agents to accomplish. This calls for intermittent and low bandwidth accesses with greater flexibility since the user is to be

relieved of telecommunications details. The agent keeps track of the user and provides necessary background support around the clock everyday.

Static agents have been created to keep track of any development of new events so that the user is relieved from tedious tasks such as searching through information sources. They have been turned into mobile agents for moving computation near to the sources of information in an effort to save time and network bandwidth. Agents can now follow up on events locally as well as remotely. Another difference through the development of agents is in the media used. Initially, monomedium agents were created to work with a certain medium like text in most cases. Now, agents are being made to work with multiple media. The trend for mobile and multimedia agents continues to gain popularity in the face of improved services.

There are three aspects of agents that need to be studied for complete understanding of the problem and corresponding solutions. First, the theory of agents can be understood in terms of specifications laid down by the users. The behavior and nature of the agents, as fixed by these specifications, constitute the problem at hand. Second, an agent architecture defines the solution to the problem. It forms the basis on which the problem is viewed and eventually solved. Finally, a language has to be chosen for agent communication. This language will guide user/agent, agent/service provider and agent/agent interactions.

Although still in its infancy, software agent technology is making great strides in the corporate world. It has the potential to enhance electronic marketing and commerce

over wide area networks. A host of World Wide Web agents like WebCrawler, World Wide Web Wanderer and Lycos already exist in the internet [IND95]. Network service providers like AT&T, IBM and General Magic are moving to agent-based WAN services. Telecommunications, the Internet and on-line communities are experiencing a period of transition.

Chapter 3

Multimedia News Agent System

3.1 Introduction

With the advances in high-speed broadband networks, powerful processors, storage and compression techniques, multimedia communications surged ahead in a matter of about a decade. People began to exchange information in multiple media with more ease. Services like video-on-demand and video conferencing are present reality. As these services become common, new ones continue to emerge in the research world as well as commercial world.

Although multimedia research is still in its infancy, we get a clear picture of the progress if we look at the advances made so far. The first generation of on-line services was character-based and command-line-oriented like GEnie and CompuServe. The second generation was graphics-based like America On-line and Prodigy. Many of these second generation services are already multimedia in nature.

The introduction of agents have moved us into the third generation of on-line services, the agent-based services. Communication technology has moved from host-based to client/server-based and now to agent-based. The transition from host-based to client/server-based gave more power to the user. Agent-based technology will unleash even greater power and flexibility on the way we communicate, especially when combined with multimedia computing.

Agent-based applications began mostly with email which was relatively known and popular among a large number of users. Email was chosen for some good reasons. Since it enjoyed widespread usage, researchers could study the role of agents in maintaining the mailbox for a wide spectrum of users. It provided a number of fairly easy yet complete test cases for understanding the users' habits. These included habits like how a user might want to view and respond to emails. Email's inherent nature of being able to transfer 'messages' from end-to-end provided the groundwork for mobile agent testing. Thus the agent evolved from being a static agent to a dynamic agent. This is in the form of active emails that can be executed in the mailbox of the recipient.

News application began to catch up with agents following relative success for email agents. Known works to date revolved around static news agent which maintained news items much like emails. News agents were dominated by non-multimedia interests including learning curve, adaptive nature and so on. Moreover, they tend to emphasize on the knowledge-based aspect of the agents.

However, the multimedia news application can provide a much more complicated and dynamic domain. Multimedia news calls for manipulation of various media types not defined or encapsulated as in email MIME. This introduces a host of interesting agent features that requires intelligence and semantics from the part of the agents. Although this agent aspect is not yet a reality, it is certainly a prospective research item and some of these possibilities are discussed in the next chapter. Beside multiple news sources, a particular news source can be distributed over the network.

In this chapter, an overview of a distributed multimedia information platform is given. One specific application, which is the multimedia newspaper, is presented next. In the context of the news application, the multimedia news agent application is developed. The description of the news agent application includes the overall design of the agent architecture, interest representations in the form of *multimedia profile*, profile manipulation by interest calculation, multi-agent collaboration and specialized services as offered by the agents.

3.2 Distributed Multimedia Information Platform

MEDIABASE is a distributed multimedia information platform [KAR93] developed at the Multimedia Information Research Laboratory of the University of Ottawa. It is composed of various user-sites and applications connected to different computer networks. MEDIABASE is a heterogeneous platform where user-sites are UNIX workstations, DEC alpha stations and Pentium PCs. These user-sites are connected by Ethernet and ATM as local area networks. These are in turn connected to OCRInet, the ATM metropolitan area network. The major applications built on this platform include multimedia newspaper, multimedia groupware and TeleLearning courseware.

3.3 Multimedia News Delivery System

The news delivery system architecture as shown in figure 3.1 is based on the client-server paradigm. Clients spread out in the Ottawa-Carleton region can access the hypermedia news database via OCRInet [FAB95]. Users are provided with a tool to browse through the news articles in the database. The News Production Center draws from the news content server and database during the creation of documents. Editors from different geographical locations collaborate in creating news articles. Completed documents are stored in the news database which supports "aging", that is old articles are stored for future reference.

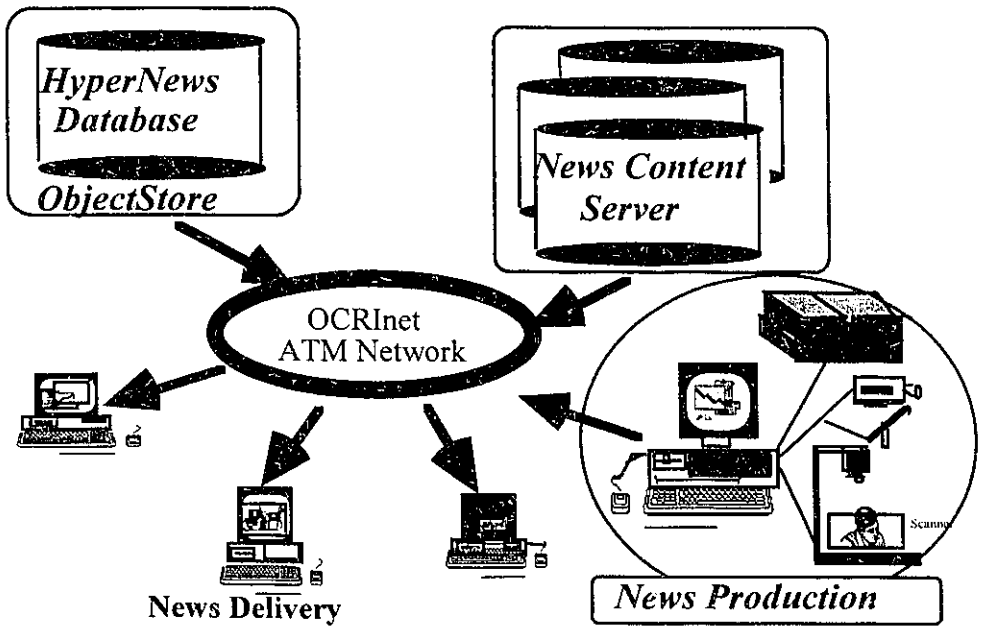


Figure 3.1: The generic architecture of the news system

Articles in this multimedia newspaper contains multiple media like text, graphics, images, audio and video as opposed to the traditional newspaper containing text and graphics only. The news documents are in HTML format containing hyper links and they are viewed on a Mosaic-like interface. The main interface allows the user at the client side select from the available databases or news sources to browse. A name list of news articles are retrieved for selecting a specific one to be actually retrieved. Finally, the entire article is downloaded to the client side for viewing.

Beside the usual selection and viewing of news from the database, the users are offered customization features by the system (see figure 3.2) [FAL96]. A user can put links to documents in a *scrapbook* much like the bookmarks in Mosaic or Netscape. The

difference in the news system is to allow the user continue viewing the document at the point where it was last read, as opposed to the indication of sites visited in the bookmarks. The scrapbook can also be used to note down any comments the user might have while viewing the documents. Links to other documents can be obtained by the *follow-up* feature of the system. A name list of related documents are returned by clicking on the follow-up button. Just as document names can be browsed, the list of keywords and the keyword tree in which the keywords are structured, can also be browsed.

Figure 3.2: Interface for multimedia news-on-demand

The strongest feature of the news system perhaps lie in *video browsing*, which allows a video clip to be browsed frame by frame on a *video tile* [FAB96]. A video tile

decomposes a video clip into twelve segments and present the first frame of every segment on the screen. Each segment can be further zoomed in or out based on the frames shown. This allows the user browse through a video clip without having to watch the entire video clip. It also allows the user view the video segments of interest by deciding on a frame.

The entire operation of the news-on-demand system is manual, that is the user selects manually what is desired and waits for the result to return. New browsing techniques are provided in an effort to customize the user but to a limited extent. The decision to transfer a document entirely rests with the user on the basis of document names. Very often a user may find the downloaded document unexpectedly not as useful. Besides, network traffic has been increased unnecessarily by the time a document is found to be useless.

3.4 Agentized News System

3.4.1 Overview

Moving a step ahead of the client/server paradigm, personalized agents can be applied to the multimedia newspaper, creating an Agentized News System [KWA95]. The agents search the database to find articles appropriate to the users' interests. The onus of finding and retrieving potentially interesting articles is shifted to the agent. Each user is presented with a "personal edition" of the newspaper based on individual preferences. The users may also choose to view the "unfiltered" edition of the day's newspaper as offered in the former application.

The Agentized News System presents two distinct aspects of the role of agents in acquiring multimedia news. The agents could be static as well as mobile [CHD94]. In a static setup, two specific agents combine to complete the cycle of updating the user with the latest yet appropriate news items. The two agents are (i) Filtering agent for filtering & retrieving and (ii) News agent for presentation. In a mobile setup, an agent can transport itself from site to site for gathering news articles [KWA96]. The traveling agent is capable of handling the complete cycle of retrieving and presenting news to the user. In the following subsections the static setup is mainly described while the mobile setup is discussed in the next chapter.

3.4.2 Architecture

The layered architecture of the agentized news system is shown in figure 3.3. There are four symmetrical layers on each side of the network. On top of one side is the user and on the other sides are the service providers. News Agent is the top layer which is the central control of the system. This is where user needs, likes/dislikes and habits are evaluated and updated to adapt the overall functions of the personal agent. Special requests like "find out the disasters of earthquakes in Japanese history" are handled here. The statistics maintained by this agent at this level portrays the interests of the user.

Based on the interests of the user the News Agent communicates the kinds of news sought for, to the Filtering Agent which searches the database for relevant articles. The latter also constantly observes the changes in the news source. The database access protocol defines the communication for filtering and retrieving from the news database.

The second layer is the Agent Environment, a generic layer which allows agents to execute and bring about different results. In the mobile news system this layer hosts the running of incoming agents, as a result of which news articles are retrieved on one hand and presented to the user on the other hand. Agents are also dispatched at this top level from across the network. The protocol between these layers is based on the aclient-iserver paradigm which is discussed in the following chapter on mobile agents.

The messages between News Agent and Filtering Agent are passed through the Interagent Communication and Network Communication layers and subsequently sent over the network. The Interagent Communication layer facilitates communication among agents. This layer defines the protocol used among the agents, the agent scripts being written in Tcl/Tk here. The Network Communication layer is concerned with the network requirements and uses TCP/UDP. It participates in negotiations for transfer of data across the network by taking into account various cost factors like QoS, so that multimedia documents can be presented to the user.

After relevant articles are retrieved from the database the News Agent arranges the order of the articles and presents them to the user. The presentation format of the articles can be changed or adjusted by the News Agent. For example a user could choose to have the articles or documents arranged in user defined groups represented by icons or have the headlines and stories appear side by side and so on. In the case of a mobile agent, it is responsible for negotiating the resources required to present the articles.

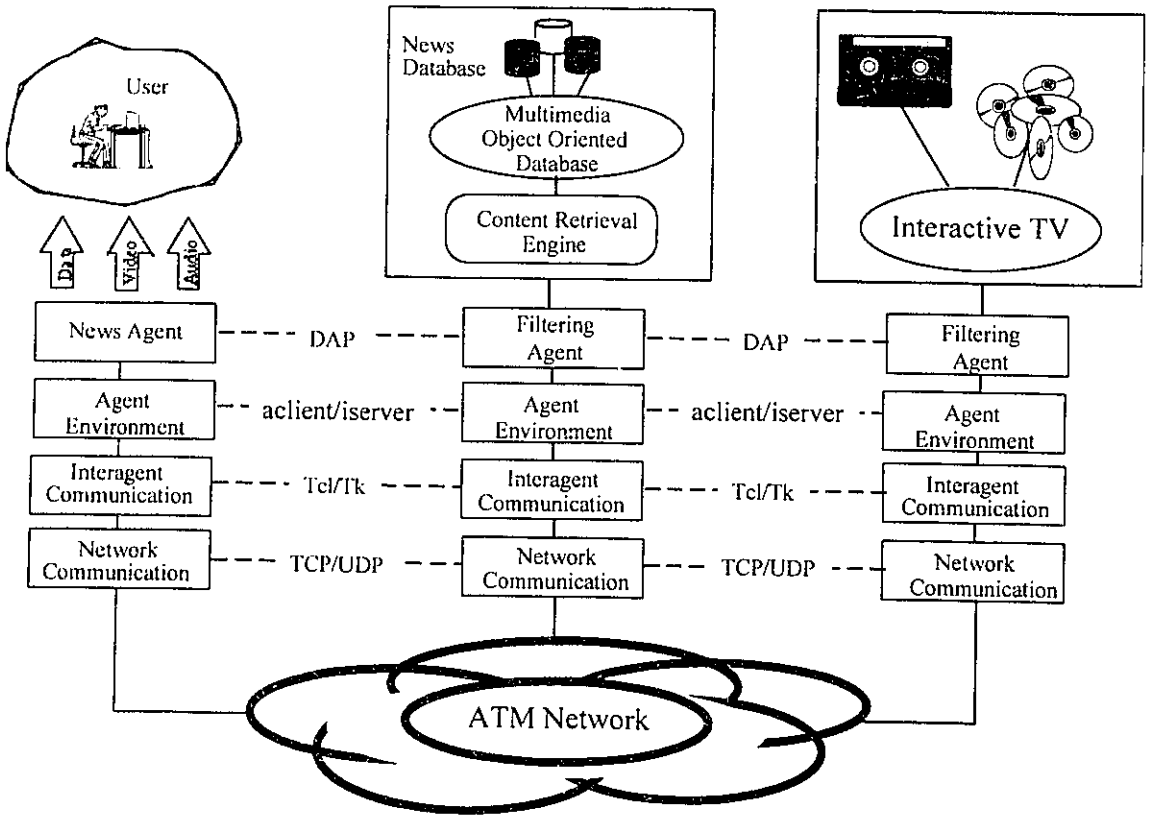


Figure 3.3: Layered Agent Architecture

It is to be noted that although the description here in focused on the news application, there is no restriction that prevents the inclusion of other applications. There could be other services like interactive television, multimedia learning and so on. Interactive television would be very similar to the news application, except that the newscast could be in live telecast. An agent can be assigned to watch the news like stock exchanges constantly and capture events that may be of value to the user. The captured event would finally be stored in a database for later retrieval.

Another point to note here is that the network itself can be expanded to a wide area network to accommodate more users separated geographically. These are not mere assumptions in view of the fact that services like news, stock exchange, archives and entertainment on the internet are becoming ubiquitous each day. With the expansion of the network to internet for example, where there is no uniformity among the connected nodes, many issues complicate matters. There are unanswered questions like how does the agent decide where to go and how far to go. There is also the risk of too many agents unnecessarily jamming the internet. Therefore, some form of architectural network uniformity constraints as described here must exist for the success of agents.

3.4.3 Multimedia Profile

A profile is a file which records the interests of a user in a predefined structure. A single interest profile represents the user's interests in a certain area. A group of interest profiles then represents the total interests of the user as much as possible. These interest profiles are maintained and used by the news agent to adapt to dynamic user interests. The adaptation using interest profiles is automated on the basis of certain factors while the user is viewing the news. These factors include time spent on an article, the frequency of viewing it and the media used in the process etc.. However, there is an interface for the user to directly communicate his/her interests to the agent. This interface also provides access to the internal representation of the interest profiles.

Multimedia Profile is an interface that summarizes user interests. It appears in a single screen with four sections as shown in Figure 3.4. The profile index section shows

the news categories being used and their corresponding profiles. Each category is made up of a number of interest profiles. Figure 3.4 also shows Science's interest profiles as an example. The content of each of these profiles can be expressed in multiple media as explained below.

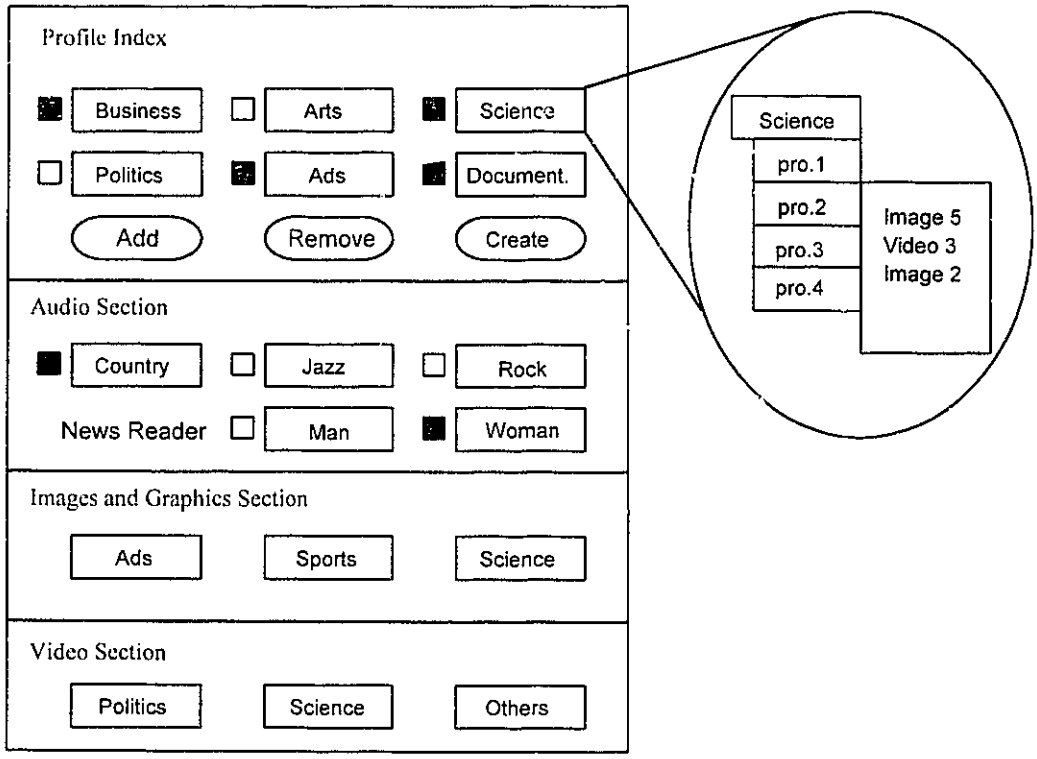


Figure 3.4: Multimedia Profile

The remaining sections of the multimedia profile screen allows the user express his/her interests using audio, images, graphics and video. In the audio section she/he can choose the kinds of music that are preferred. This will not only be used to select news on events like the Grammy awards but also for the option to play background music while

the user is reading pure text, silent advertisements etc.. The user can also choose male or female voice to read out the portion of the news that appear in text.

In the images and graphics section, each category has a so called "*Picture Board*" (refer figure 3.5). The pictures contained therein are made available by the news production system for the user to choose the ones that appear appealing. Each picture is indexed to a group of related articles in the database. The group of articles increases in number when related events occur. The *picture board* provides a powerful method of expressing interests. It is particularly useful for handling advertisements. These are neatly classified and users may or may not choose to view them. The *picture board* is not meant to supplant but to supplement keywords which would still be good at expressing certain types of concepts like abstract and counterfactual ones. The pictures should be simple and clearly on a particular theme.

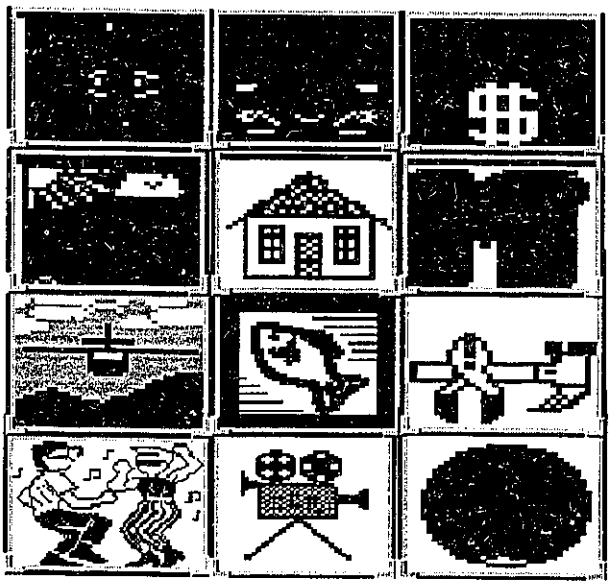


Figure 3.5: A Picture Board

In the video section each category has an attached list of video clips provided by the production system. The user can select the ones he/she prefers. As in the case of the picture board, a list of indices are associated with the video clips. Video clips provide a "live" description of the kind of articles that could be of interest to the user. A video clip of a heart surgery for example contains more content information than a long list of keywords.

A profile may contain more than one medium according to the relevance of the areas covered. Thus, a profile may contain an image of a baseball field and a video clip on a baseball match. Since the images and videos are named uniquely, the profiles record the names which we shall call 'keys'. The user distinguishes the media types by the color of the keys. For example, image keys are in blue, video keys in red and text keys (keywords) in black.

Finally, the profiles can be manipulated by the user. The manipulations include removal or addition of keys from the profile. A profile can be created as well as deleted manually. If the user does not want to keep track of the profiles and their contents he/she can just make his/her media selections and the agent automates the necessary steps. For example, on the user's selection of a picture from the picture board, the agent tries to fit it into an existing profile. If there is no suitable one that fits, a new one is created. Similarly on deselecting a picture, a profile is deleted if that were the only content. Therefore, by profile manipulation the Multimedia Profile serves as an easy, direct and uniform representation of the user's dynamic interests.

3.4.4 Profile Manipulation

The personal agent maintains a statistical table (figure 3.6) to compute a user's interests based on various attributes of reading the newspaper. While the user is reading news the agent keeps track of how it is done, such as the time spent, the media used and so on. The user's explicit feedback like "keeptrack", "good", etc. are also recorded. The profiles are given weights ranging from 0 through 1 (denoted by w_{pi}). The importance of each attribute (denoted by a_i) is dynamically set by the agent. For some users, for example, video may be more important than audio. The average of these weights comprise the fitness of a profile ($0 \leq f_p \leq 1$). Everyday the fitness of every profile may change according to the user's actions while viewing the news. A profile with a fitness below a certain value (say 0.2) may be deleted and on the other hand new ones can be created either specifically by the user or when a key(s) cannot fit into any existing profile.

It is to be noted that the profile fitness is not determined solely by the media used in presentation. Rather, content is also taken into account by the direct user feedback like 'good' and 'bad'. In fact the fitness of a profile reflects the content more than the number of media used. If an article has rich media but 'bad' content, its profile will not receive a high score. In other words, a fit profile means 'good' content which is itself manifested in the various media.

Attributes	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Time							X			
Audio						X				
Video								X		
Freq.					X					
Depth										
Keeptrack						X				
Others										

$$\text{Profile fitness } (f_p) = 1/N \sum_{i=1}^{N} a_i \times w_{pi}$$

where N = Total number of attributes, a_i = i^{th} attribute weight

w_{pi} = profile weights corresponding to i^{th} attribute

Figure 3.6: Calculation of a profile's fitness

3.4.5 News Selection

News articles are classified into categories as seen above. Each category's articles are retrieved using the profiles specific to that category. In the beginning there are three ways of creating profiles. First, the user can directly create profiles by category according to his/her interests. This is the most direct way of learning a user's interests from scratch. Second, the personal agent offers predefined sets of profiles from which the user chooses the set that appeals to her/him most. This is quick in producing results but may not be as accurate as the first method. Finally, the agent automates the creation of profiles by observing the pattern of user behavior while viewing the unfiltered editions of the newspaper. This may end up with better accuracy than the second method but may take more time to adapt to the user's interests.

After the initial profile selection the agent uses the third method mentioned anytime thereafter. In the process of calculating the fitness of the profiles, some may be deleted because of being 'unfit', others may be added because there is no suitable profile to accommodate the representation of an interest. New profiles need to be generated to adapt to the dynamic interests of a user as well as to unveil new interests to the user. The former need can be met with the computation of profile fitness. We can provide two ways for the latter to occur. First, the agent randomly selects articles to be presented to the user. Second, since both the picture board and video clips cover a wide area of interests, the articles thus offered may be varied but not random. Together the agent can meet the requirements of adapting to user interests as well as offer help in exploring new areas of interests.

3.4.6 Specialized Services

(a) News Synthesis

The news agent envisioned here offers services other than the mundane filtering and retrieval of news articles. It helps the user in 'news synthesis', that is keeping track of related events and reporting them together whenever there is any development. These events could be seen from more than one point of view (for example from different news sources) or attached to different aspects of the events. When the agent reports these events it puts them into one single *synthesized article* for giving the user a single appearance of related events. While the user reads such an article she gets the entire picture of the events without browsing through multiple articles, either contemporary or past. An example of a synthesized article is shown in figure 3.7 where four articles of a

terrorist or an extremist nature occurred prior to the new event. All these articles have been matched by the same profile, so when the latest event takes place the agent puts them together in one document. The number of articles that qualify for synthesizing into a document is arbitrary and should be a user option.

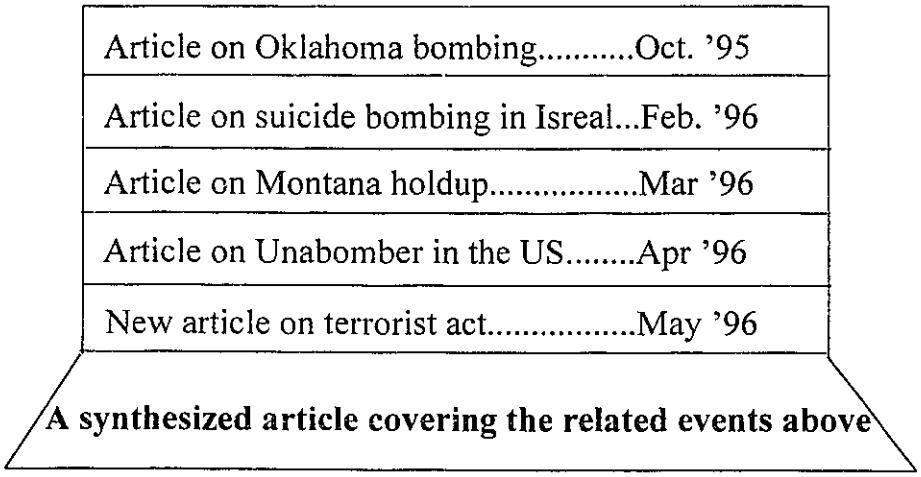


Figure 3.7: A synthesized article created by an agent

The news agent does not add or delete any part of an article to maintain its integrity and semantic. This is not meant to create or edit new articles but rather present existing articles in a helpful and personalized manner. A much more complex and interesting specialized service would be to employ the expertise of agents in the formation of entirely new documents. In the next chapter, the role of a mobile agent in creating 'multimedia documentaries' is described. These so-called documentaries are composed of various media parts of different articles collected by the mobile agent from a distributed source, thus the agent functions as a composer or editor of news.

(b) Multimedia Presentation

After the synthesis of news articles or creation of documentaries, the agent collects relevant information for making a *multimedia presentation*. The information could include the various hardware and software resources available at the time. On checking if the user is logged into the system, the agent notifies the user of an upcoming presentation. The user may choose to accept the offer immediately or specify a later time to view the presentation. This could also involve keeping track of a mobile user and it takes a mobile agent to follow the footsteps of the user. The user may want to get information in the form of an email or a facsimile or a cellular phone call depending on user convenience.

The user may send a mobile presentation agent to another user for showing certain articles of common interest. This kind of communication is especially useful for people working closely in a project. For example, a group of business consultants would want to know of latest business opportunities around the world. Information discovered by any of their agents can prove to be useful by sharing among the group members. Instead of writing emails, a user may as well send the agent with the new article for providing first hand knowledge.

(c) Interagent Communication

Specialized services may also include communications among agents in a multi-agent environment in a collaborative manner [LAS94]. Interagent communication could

be very powerful since the agents then would represent humans in their decision making strategy. The agents can collaborate to overcome their inability to otherwise perform certain tasks. For example, an agent would like to have an article on a particular topic which it could not find from any of the known sources, for a multimedia presentation. It can seek the help of other agents in acquiring the article.

Since not all the services would be free, the agents may have to negotiate in their dealings among themselves. Continuing from the previous example, the agent seeking help in finding an article may need to *negotiate* with other agents. The negotiation could be in the form of electronic payment or a substitute service in return. To extend the idea tied to such a situation, an agent will possess some digital purchasing power of the user. To summarize, an agent is authorized by a user and it autonomously perform tasks on behalf of the user in providing personalized services.

Chapter 4

Mobile News Agent

4.1 Mobile Agent

The mobile agent is designed to be a generic agent that can travel or transport itself from node to node in a network. It should be able to route and re-route itself until a preferred destination is arrived at. The routing of a mobile agent, however, is not totally dependent on the agent itself but also on the node at which the agent happens to exist. The information required by the node to route an agent should be embedded in the agent for facilitating proper routing.

At each node that an agent is routed through, there is some form of processing that brings it closer to the task it had set out to do. The processing could be as simple as transferring the agent onto another node or as complex as further filtering the already filtered (by the service provider) information. Whatever the processing, the aim is to tailor the information suitable to the interests of the user. After the necessary processing, the agent is returned with the results to the originator of the agent.

Throughout the entire 'journey' of the agent, it may encounter different situations which may affect its very existence. Therefore, whenever an agent is received a log is maintained about the agent. Information like the originator, purpose of the agent's visit, present execution state etc., are logged so that in the event of the agent's destruction for any reason, we may be successful in reviving the agent.

However, not all errors are easy to detect and/or avoid. Debugging such unforeseen error situations is a daunting task. At most we can program an agent with the ability to deal with all possible errors we can think of in advance. Inspite of that, there can be situations that will render the agent extinct. For example, the crashing of a network may destroy enough information about an agent that will make it impossible to regenerate the agent or revert to what it was doing before the crash.

We can partially compensate for such undesirable conditions by making the agent report to the user of its latest achievement. At every major step of a task, the agent would send back a message indicating what it is doing. Although this will keep the user informed of the latest development, it may not be very appealing to some users. These

users may not be interested in knowing the status of the agent but rather the results brought about in the end. In such cases, the user should be able to direct the agent not to send back complete details of its status at every step but a few prominent ones. The user may also choose to completely surpass every step except the results.

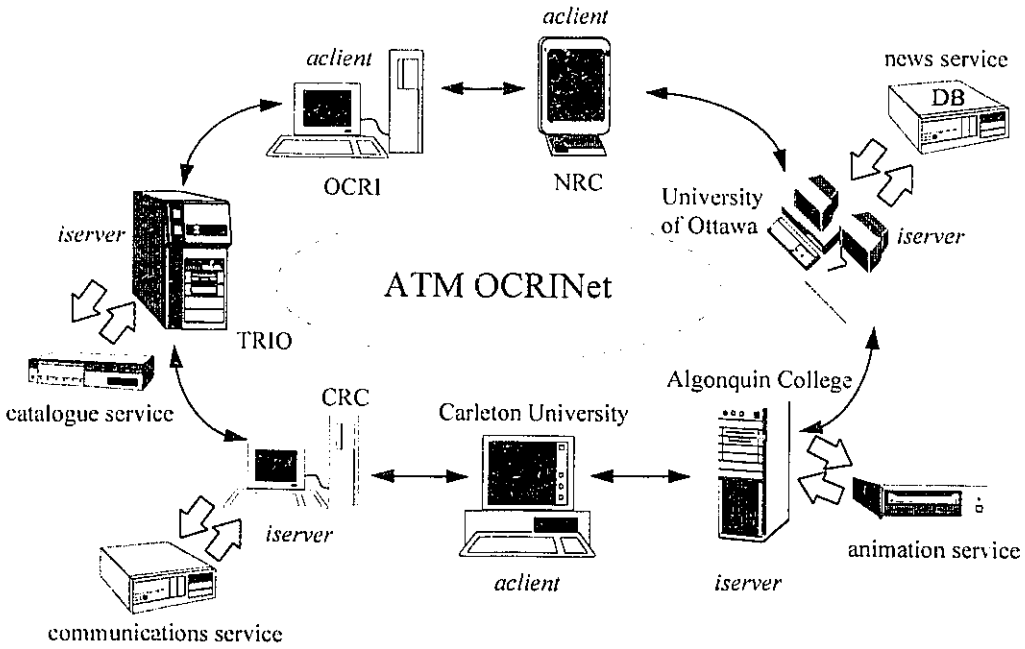
4.2 Environment for Mobile Agent

The environment that supports mobile agents needs to be different from the client-server model. We shall refer to it as the *aclient-iserver* model. The corresponding client site is called an *aclient* and the corresponding server site is called an *iserver*, since the client is now asynchronous and the server interactive. The *aclient-iserver* setup facilitates the launching, receiving and execution of an agent at any site. In particular, upon execution (when the agent accesses the database) the *iserver* itself acts as a client to the service provider.

Figure 4.1 depicts a possible configuration of the aclient-iserver environment in the OCRInet. Although the figure suggests each laboratory site to be either an aclient or an iserver, aclients and iservers can be located in all the sites. The various example services include news, catalogue, animation services etc. with a server for each. The hosting iservers allow agents to interact with the individual servers. Therefore, new sites can be added to the network and new services further defined.

A mobile agent can be transported from site to site in order to accomplish certain tasks assigned by human users. At any site the agent need not complete execution. It

may stop in some point and continue thereon at another site. The environment must provide the facilities to do so. Execution stacks and variable values are recorded for each agent. This would mean the receiving site maintains a log of an agent's activities so that necessary steps can be taken to regenerate and send the agent to another site. In the case of a news agent, incomplete execution could result when a service provider does not possess any article on a certain topic. Then the news agent has to be transferred to another service provider for further negotiation.



Legends: CRC = Communications Research Centre
NRC = National Research Council
OCRI = Ottawa-Carleton Research Institute
OCRINet = Ottawa-Carleton Research Institute Network
TRIO = Telecommunications Research Institute of Ontario

Figure 4.1: Environment for mobile agents

4.3 Structure of a Mobile Agent

The mobile agent has an encapsulated structure as shown in figure 4.2. The structure is similar to the message packets used in message passing, except that the structure here comprises of exactly one agent. The agent is in turn encapsulated by header information used in the network layer required for traversing the network. The header is defined according to the layered architecture of the mobile agent news system. The agent script portion of the header consists of various information shown and described below.

Figure 4.2: Encapsulated agent structure

Each script has destination and source addresses (DA and SA respectively) for the end points of communication. The source address points to the originator of the agent whereas the destination address refers to the eventual site where the agent wants to visit. These information are useful in determining what to do with an agent when it finishes execution or has encountered a problem. In the former case, it is obvious that the agent be sent back to the originator. In the latter case, the originator may be contacted for further instructions as to what remedies should be taken.

The number of 'hops' left to be traversed is recorded in the 'HP' field. Each 'hop' here refers to the nodes at which some form of processing other than routing has been done. If an agent is routed through three nodes while being sent from node 'A' to node

'B', the number of hops is counted as one. This convention derives relevance from the fact that an agent is interested in visiting certain number of service providers irrespective of the number of nodes it is being routed through.

Routing information (RI) indicates how an agent is to be routed. For example one may specify that the agent should stay within a metropolitan or wide area network. RI also provides the options to take when there is an unforeseen event like network failure and loss of agent information. This could suggest that the originator not be contacted and end all transactions or give directives to some specific administrator for advice and so on. In an unlimited case where the originator does not mind losing the agent and its efforts for any reason, this flag may be turned off to signify the preference.

'PP' specifies the purpose of the agent's visit to the destination. The purpose could be seeking service or even providing information about services that are available. The latter is a form of advertising for products or services. This field sets the ground for further negotiations for services. The agent is accordingly processed when the purpose of the agent is made known.

'ST' carries the status of the agent with respect to its execution state. An agent can be partially executed and wait for further processing or it can be fully executed once and for all. Therefore, when the agent arrives it is imperative to know the present status so that the environment that receives the agent knows how to deal with individual situations. In view of the possibility of partial execution, the agent has to record the variables that define its state at any time. The values of these variables will determine the

next execution step for the agent. A part of the ST field should also carry the status of the originator. The originator may be an authorized user or an anonymous guest.

The executable script (XS) in Tcl/Tk is executed when all the other fields are checked. These Tcl/Tk scripts are interpreted by interpreters created at the remote site. At each of the nodes visited, the agent is closer to its task being accomplished. The final execution of the agent produces results that may or may not be further processed before they are returned to the originator.

The final field of the agent structure (IG) provides the information for agent regeneration to send back the agent with proper headers to the originator. IG is a very important part of the agent for the completion of its 'mission'. RI as described above outlines the intentions of the originator when an agent is faced with an unforeseen event but IG provides the technical details required to materialize the intentions.

In summary an agent is sent from an *aclient* site to an *iserver* site to accomplish a certain task - for example retrieving a news article from a database and presenting it to the user. Based on the routing information carried in it the network routes the agent until it reaches its specified destination. On each hop the relevant fields of the agent script are opened for inspection and the executable script is executed if required. Assuming successful execution, the results are sent back after appropriate processing which may, for example, include further filtering. Otherwise, the agent is regenerated for carrying the error message back to the originator.

4.4 Applications of Mobile Agent

The most common applications of agents include email, scheduling and news filtering. Of the many applications, perhaps the most useful and important one is on information access and processing. Digital library and news filtering are good examples. How a mobile agent is applied to any application is another interesting issue. It can be very innovative and powerful depending on what and how we delegate an agent to do. In the following subsections, some of these possibilities are introduced.

4.4.1 Mobile News-Agent System

A mobile news-agent system is a step further than a news-on-demand system. In both the cases, the aim is to provide news services. A news-on-demand system has authorized clients which are given the right to access services offered by news servers. Typically, the user is provided with a browsing tool to search through the news database. The filtering and retrieval of any news articles are actually done by the user, although there can be other features in the tool that help the user in keeping track of what he reads.

But in the mobile news-agent system, news agents are launched into the network to seek services from news servers and they carry back the results with them to the user. Where a mobile agent finds relevant news articles and how it brings them back are transparent to the user. In effect the user delegates the task to the agent and work on something else instead of waiting for a synchronous reply as in the news-on-demand system.

Once the agent is released into a network, it finds its way to the news server and filters the database to obtain news articles of interest to the user. It may have to visit a few servers before it is successful. Whenever it has been successful or it has exhausted its search of all the available servers, the agent returns with the results. The results can be presented to the user in various manners like email, fax, voice mail, etc..

The functions of a mobile news agent can be much more complicated than just locating the right article. It may be used in other collaborative tasks like composing articles and creating multimedia documentaries. In such situations, the mobile agent is responsible for putting together facts that it finds. This requires the agent to have some form of 'intelligence' and comprehend the semantics of related stories. Both of these areas are still under research and development.

4.4.2 Agents as News Composers

Mobile agents can be seen as *composers* or editors of news articles. In a multimedia news context where articles may consist of several different media like a text passage, several images/graphics and a video, it is important that a mobile agent not only collects relevant media but also assembles them into a meaningful presentation. The agent has to first collect resource and media information over the network.. This part will involve searching the databases distributed in the network. After collecting the data required, the agent employs a knowledge base to compose them into a news documentary as depicted in figure 4.3. The knowledge base has such information as media preferences of the user, available hardware and software support, network constraints, etc..

To accomplish the task of a news composer, the agent needs knowledge of (a) temporal and narrative structure of a news documentary and (b) prejudices and biases of the user, for example in a profile of the user. The composed article will be an asynchronous document, that is a dynamic document with a sense of time and possibly authored branches in the storyline where the viewer can interact with the documentary to change it's course (see [FAE95] for details on dynamic asynchronous documents). The transitional points in a storyline indicate the points of deviation from its original course.

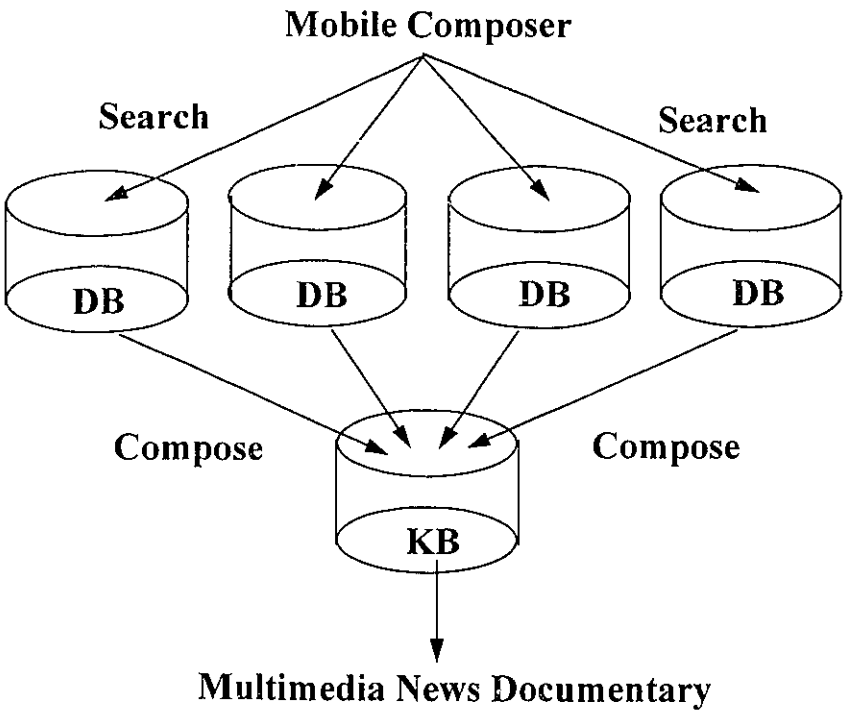


Figure 4.3: Agents as mobile composers

4.4.3 Multimedia News Documentaries

A multimedia news documentary, as created by a mobile agent composer, is a story based on a particular media event. In the best case it is an entertaining, informative

multimedia presentation. The degree to which it is, however, is based upon the composer's knowledge. For instance, it must know that a documentary should have the following structure: (a) context (b) presentation of central idea and (c) summary and conclusion.

Also, narratives have a basic structure [LAU91] in which action rises to a climax followed by a resolution; this is what makes stories "feel good" in the end. Hence, the mobile composer travels from node to node in the network in search of media that will satisfy both the preferences of the user and the particulars of documentary and story structure. This is a very difficult task - we may approach this problem by using heuristics and fuzzy systems along with a knowledge base.

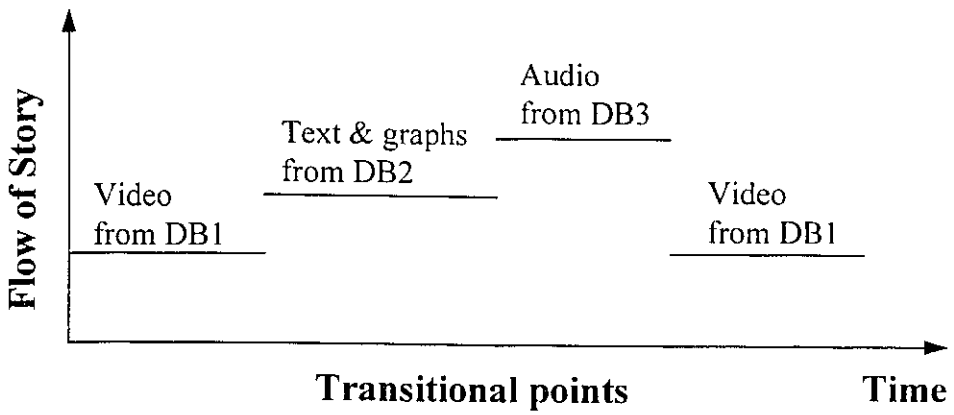


Figure 4.4: A multimedia news documentary

Because of the difficulty in this task the composer will not always make the right choice. When it is not "sure" (in the fuzzy-sense):

- how to include a media in the current documentary

- how to attach what seems to be a "related" topic
- how to "finish" the current documentary

the composer will create a transitional point in the current documentary. A transitional point is a point in time wherein the user, while watching, may influence the story flow in some way. This non-committal approach tries to include more media than necessary rather than take a chance. The number of transitional points is proportional to the confidence of choices. Figure 4.4 shows a possible multimedia news documentary in a two dimensional graph, with time represented on x-axis and the media used in the storyline on y-axis.

4.4.4 Remote Presentations

An equally interesting task as composing news documentaries is the remote presentation of these documentaries. There are two possible scenario for remote presentations: (a) the user is away from his usual work environment and (b) the user wants to send a documentary to his friend. In either cases, the mobile agent has to acquire information about access to available remote resources, after which it has to carry the composed news documentary for a multimedia presentation. The agent has to check whether a particular user is logged onto the network and if not, at what interval it should try checking again until the user is logged on.

This form of communication among the users is more than just email. In the process of the presentation, the agent may gather indirect user feedback such as the extent to which the user is interested. Direct responses can be requested with a short

questionnaire at the end of the presentation. A variation of this will result in a form of advertisement that a company may use for its customers. The company creates (as opposed to an agent creating it) and sends out mobile agents with a presentation about its new product in order to get customer responses.

4.4.5 Collaboration

In the process of creating a documentary for the user, communication and collaboration between various parties is essential. For example an agent may communicate with other agents that it may encounter at any node. This can be advantageous to the end-user in the following ways:

- An agent may learn from another agent where to find particular media that will assist in the completion of a documentary
- An agent may improve it's movie-making "style" by exchanging and comparing knowledge from another agent

Alternatively, an agent may communicate directly with its user to:

- allow the user to begin viewing the documentary - in fact a documentary can begin its playout before it is resolved. The agent will try to complete it concurrently to the viewing
- allow the user to dynamically change parameters such as the rating, perspective, genre, and so on.

Regardless, the exchange and communication of knowledge and data is not simple. There are many issues that challenge the role of an agent as a news composer.

Security is an important aspect of mobile agents that merits separate research. The problem exists for both end-to-end as well as node-to-node authentication. The final recipient of the agent would like to be sure of the original sender and that each time a mobile agent is transferred, it is not corrupted or modified in any way. Some form of encryption and digital signature will probably solve the problem of proper identification of agents.

Privacy is another issue that calls for special attention. An agent may carry information on a user that should not be compromised at any price. Nevertheless, some of the sensitive information would be required for revealing while making decisions or negotiations at some point.

Other difficult issues are related to the network, such as the best path to take for reaching a destination and keeping track of existing agents in the network. This leads to congestion control and performance evaluation in the management of networks. If we hold up the network with hung processes or unnecessary transfers of a multitude of agents, the purpose of reducing network traffic and better usage of bandwidth is defeated. These problems should be targeted in the design of an agent to reduce the complexity that arises due to the behavior of the agents.

Chapter 5

Implementation

5.1 Introduction

The implementation of a Mobile News Agent prototype is described in this chapter. It includes the agent environment, mobile agent and mobile news application. The components or modules of the environment with respect to the *aclient-iserver* paradigm are first explained. To accommodate such an environment, additions to the Tcl scripting language has been made and described. Then the life cycle of a mobile agent

from its initiation to its eventual return is discussed. Finally, the GUIs for specifying and communicating with the agent are presented for the news application.

The prototype has been implemented on SUN Microsystems Inc. SPARC workstations. Database servers and *iservers* were implemented on SPARC/10 while *aclients* were implemented for SPARC IPCs and IPX workstations. All workstations are connected by 10 Mbps Ethernet, while some are also connected by OC3 155 Mbps ATM as shown in figure 5.1.

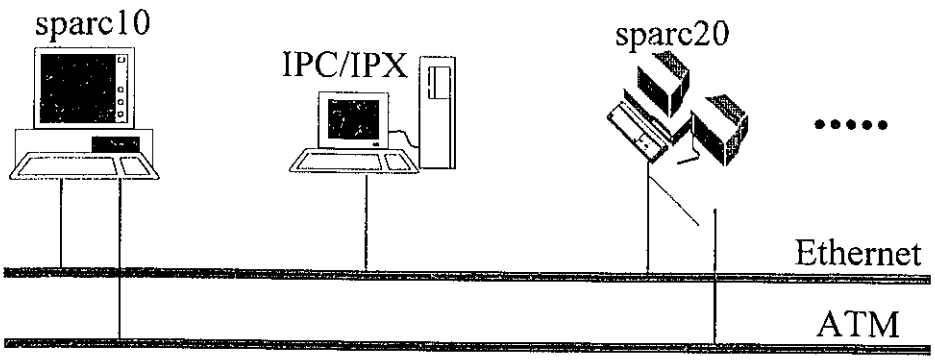


Figure 5.1: Hardware setup for implementation

The operating system on all of these workstations is UNIX running under the SUN Microsystems Inc. Solaris version 2.4. C language [STV92] is used to implement the *iserver* and *aclient* environment. The mobile agent is defined as scripts in Tool Command Language version 7.4., a string-based command language [OUS94]. The mobile news application features are constructed with the help of the interface between C and Tcl. The GUIs are built with Tk toolkit version 4.0 [WEL95]. Tcl/Tk is freely available in the internet and is widely used for quick prototyping.

5.2 Aclient-iserver Model News System

A modified version of the classical *client-server* model has been implemented. The environment for agents to 'live' and 'travel' is based on the *aclient-iserver* model as described in chapter 4. Both the aclient and the iserver programs have been written in C. The aclient process as well as the iserver process serve as the launching and landing pad for the agents. The aclient process is responsible for handling the result of an agent 'trip'. The result may be reporting either a failure or a success. The iserver also acts like a client to the service provider, the database server in the news application. In the event of an agent wanting to filter and retrieve information from the database, the iserver process allows interaction with the database server to provide the requested information.

Communications among the aclient, iserver and service providers take place via TCP/IP. Agents are also transferred between aclient and iserver sites using sockets. Both the environment processes run in the background 'listening' for requests for connection. The number of connection requests that can be accepted is arbitrarily specified in the programs. Once an agent is received the connection with the sending site is released and the agent is subsequently processed at the receiving end.

The process that receives the agent creates a Tcl interpreter to interpret the scripts in the agent. Each line of the agent script is interpreted sequentially until completion. At the *iserver* site, the execution typically results in interaction with the database server. The interaction involves filtering the database of news documents. When the requested information is retrieved, it is sent back to the originator of the agent. At the aclient site,

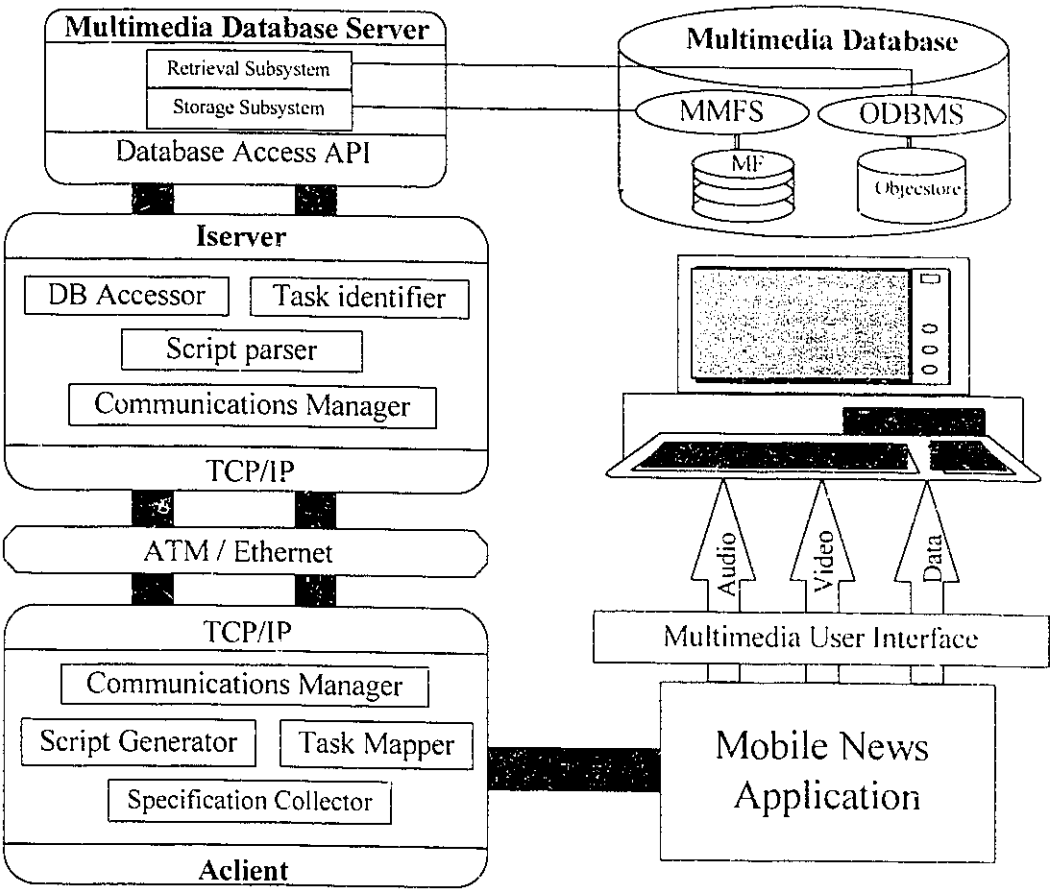
the execution of an agent may be either a message from the agent or the actual results of the database search.

Figure 5.2 shows the internal structures and configuration of a mobile news system based on the aclient-iserver model. The modules at the aclient and iserver sites are described in the following sections. The iserver communicates with the database server as its client and hence accesses the database via the application programming interface for database access. This interface provides the various primitives available to other clients accessing the database of news articles. The difference here is that the primitives are made available in Tcl.

The database in use is the commercial object-oriented ObjectStore database which stores the metadata for the actual media data. The media data themselves are stored in the proprietary media file system of the Multimedia Information System at our laboratory. Therefore, an access to our database implies collecting metadata from ObjectStore before accessing the media data. The documents or news articles in the database are arranged in a hierarchical tree structure where the nodes of the tree may refer to a news category, keywords and document name, etc. Media preferences can be reflected in the nodes by including the dominant media type present in a document. The leaves of the tree contain the actual content of the articles.

In this implementation a separate database was created to gain experience in database storage and update. This has helped giving a hands-on knowledge of the database structures and properties. The mobile news system mainly uses the retrieval

subsystem of database server, therefore, creation of a new database touched upon the storage aspect of the database. The eventual testing of the system was done on the new database although it coexisted with the database from the old prototype of the news-on-demand system.



Legends:
MF = MediaFiles
MMFS = Multimedia File System
ODBMS = Object-oriented Database Management system

Figure 5.2: Internal Structures of a mobile news system

5.2.1 Modules at Aclient-site

The modules between the user and the network at the aclient site are shown in figure 5.3. Users' specifications are collected through a graphical user interface. The interface has been built with Tk [WEL95], a high level interface toolkit that allows quick but powerful widget constructions. It is designed to simplify the creation of graphical user interface applications. A more important characteristic of Tk is its full compatibility with Tool Command Language and UNIX C. This provides a convenient mapping mechanism for converting user specifications into Tcl script commands defined in C.

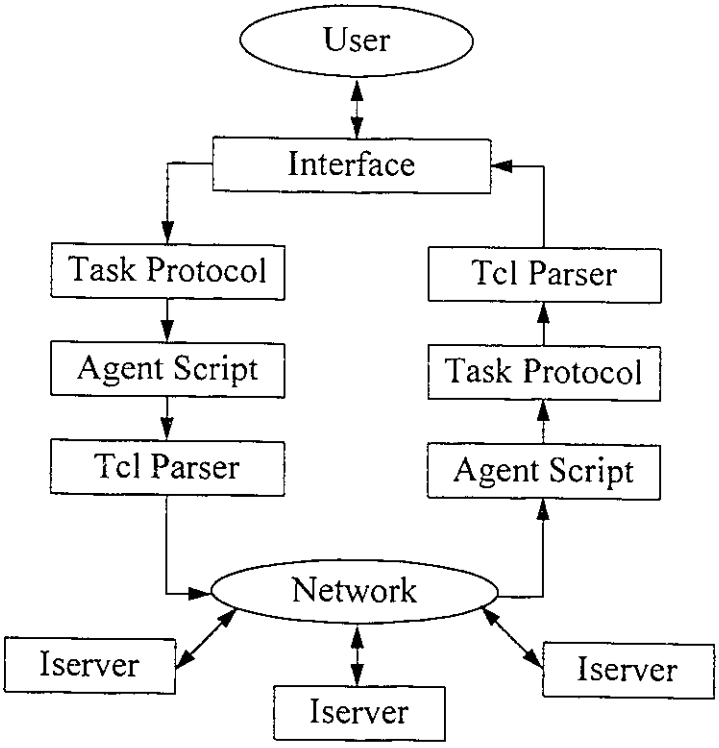


Figure 5.3: Modules between the user and network at the aclient site

These user specifications are then extracted by *Task Protocol* module into scripts with predefined format. This module defines the protocol for agent interaction with the aclients as well as remote *iservers*, database service providers and other agents. The generated agent scripts are in Tcl. The script commands in the Tcl package have been found to be useful but not sufficient to test the news application. Therefore, additions have been made to Tcl version 7.4 to incorporate the protocol definition of the tasks by creating new commands which are defined in C. These commands are briefly described in the following section.

The application specific commands are incorporated into the Tcl package so that the created interpreter has access to the new command procedures. The Tcl parser can thus execute the agent scripts with the additional commands. Figure 5.4 shows how the scripts are parsed while the agent is executed in the *aclient* module [OUS94]. Procedures that implement the new application commands are initialized first and then the scripts are passed through a command loop for parsing. The parser then has access to the core commands of Tcl as well as the mobile-news commands in this implementation. The following piece of code create interpreters with a new command:

```
/*** Creation of interpreters ***/

int Tcl_AppInit(interp)
   Tcl_Interp *interp;                  /* New interpreter */

{
   if (Tcl_Init(interp) == TCL_ERROR)
       return TCL_ERROR;

/*** Creation of new Tcl command ***/
```

```
int  Ag_ret(ClientData clientData, Tcl_Interp *interp, int argc, char *argv[])
{
message_type *message;
int sock;
FILE *fp;

/***
    fill in codes for a new command Ag_ret here.....
***/

/*** Register command for retrieve ***/

   Tcl_CreateCommand(interp, "ag_ret", Ag_ret, (ClientData) NULL,
                     (Tcl_CmdDeleteProc *) NULL);

/*** Startup script filename ***/
   tcl_RcFileName = "~/.agentenvrc";
   return TCL_OK;
}
```

5.2.2 Additional Tcl Commands

All of the above commands are of the following general format:

ag_command argument1 [argument2] [argument3] [argument4] .....

where *ag_command* stands for the various commands as described above while *argument2* and onwards are optional.

ag_send

This command sends an agent from one site to another. The sending and receiving sites run the agent environment in the background. The parameter passed to the command in

question is the destination site identifier. The creation of a single internet domain socket is initiated. At this point the decision is made as to what mode of network transmission is used, for example ethernet or ATM. This command differs slightly from site to site since the ports at which the receiving sites wait or listen for incoming connections differ. To avoid confusion in the implementation, this command had an extension in the name like *ag_send_station1( )* to distinguish among the different ag_send commands.

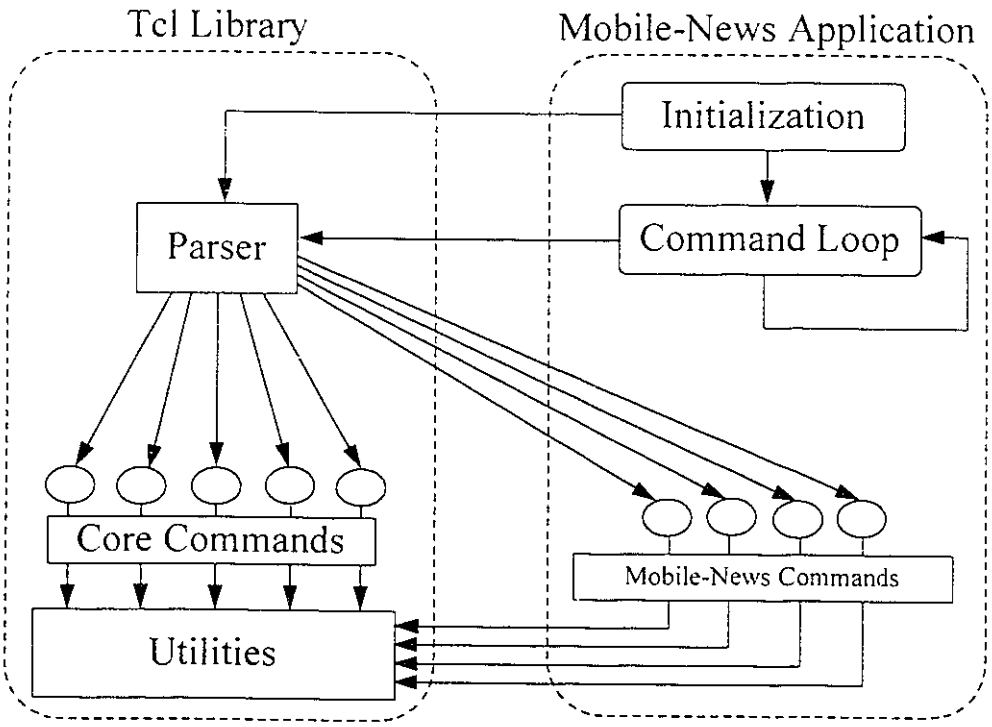


Figure 5.4: Additional commands in relation to the Tcl parser

ag_filter_db

This is a database command for filtering the news database. The present prototype supports filtering by keywords, titles, authors, subjects and dates. These are the

parameters passed to the command in conjunction with the database access protocol. The database API is clearly defined and filtered documents are passed back in memory. These documents are filtered further by taking into consideration other customization of the user likes/dislikes. This step is described in the next command.

ag_filter

A user may not want an article longer than say one thousand words. Or she may want articles with images and video clips instead of pure text. These preferences are reflected in the filtering done at the second level by *ag_filter*. This command parses the articles obtained by *ag_filter_db* to finalize on the articles to be actually sent back to the user.

ag_browse

Sometimes the user may want to browse the database instead of letting the agent take its own decision while filtering the documents. This command provides the user direct access to the database content without any hindrance from the agent. The agent in this case helps the user browse the database transparently. The database is too large to be browsed at once, therefore, keyword, titles, authors, subjects and dates act as parameters to the command.

ag_retrieve

Based on previous browsing and filtering of the database the agent is aware of the articles of interest to the user. Then when the user wants to view these articles, the agent directly retrieves them from the database with this command. The parameters passed are the

names of the documents as obtained earlier through *ag_filter_db* and *ag_browse* commands.

ag_return

At present the best way of tracking down the agent in case of an error is to have it return with an error message. The error in this sense refers to the error of no match in documents for example. The error message is returned to the user in the form of a popup menu or email. This command is also used to return other messages like its present status.

ag_email

Email is a simple yet most widespread form of electronic communication. Therefore, it is imperative to incorporate this feature of having the agent send back the results by email. This allows the user to read the articles whenever and wherever she wishes. This command is given the user's email address and invoked from the UNIX shell. The user may specify the email address to which the results should be directed. This means an email can be requested to be sent to friends.

ag_present

While the user is working on her computer (workstation or PC) the agent brings back the results of its search and presents them on a popup screen. However, the user is first asked for permission to present the results. The parameter given to *ag_present* is the name of the file which holds the results. It then invokes the Tk 'wish' shell to show the article. This command can also be employed in presenting an article to other users provided their login names are given as parameters.

ag_pause

At times the agent may need to wait while processing other commands. But while the scripts are executed sequentially, a command may need the a previous command to be completed for correct execution. In such situations the *ag_pause* command pauses the agent execution until the required commands are complete. How long does the agent pause is self adjusted by the agent.

The advantage in defining new commands as above provide script independence from service providers. The script command *ag_retrieve* for example need not be defined at the aclient site, nor does the agent know how the command is implemented at a particular site. All that it implies is the generic retrieval of documents. Each service provider site then implements the commands its own way. The commands described are recapitulated in Table 1.

5.2.3 Modules at Iserver-site

The modules at the *iserver*-sites are similar to those at the aclient-site. The changes arise due to the inherent differences of the setup at these sites as shown in figure 5.5. At the *iserver*-site, the agent is perceived as being received through the network. The agent essentially consists of scripts which are inspected with the Task Protocol and parsed through a Tcl parser. This prepares the agent to be executed or processed in the *iserver* module. In this implementation the Database Access Protocol defines access to the news database. In other implementations this could be replaced with another protocol to gain access to the corresponding service provider. The corresponding module on the

reverse direction is the Network Access module which allows an agent to be sent back to an aclient site. The control flow of the iserver modules is discussed in the next subsection.

Command Names	*Command Functions*
1. ag_send	send an agent from one site to another
2. ag_filter_db	filter the news database
3. ag_filter	further filter the articles retrieved from database
4. ag_browse	browse the news database
5. ag_retrieve	retrieve articles after browsing database
6. ag_return	return messages to the user
7. ag_email	send email to the user
8. ag_present	present results to the user
9. ag_pause	wait while processing another command

Table 1: Recapitulation of new commands

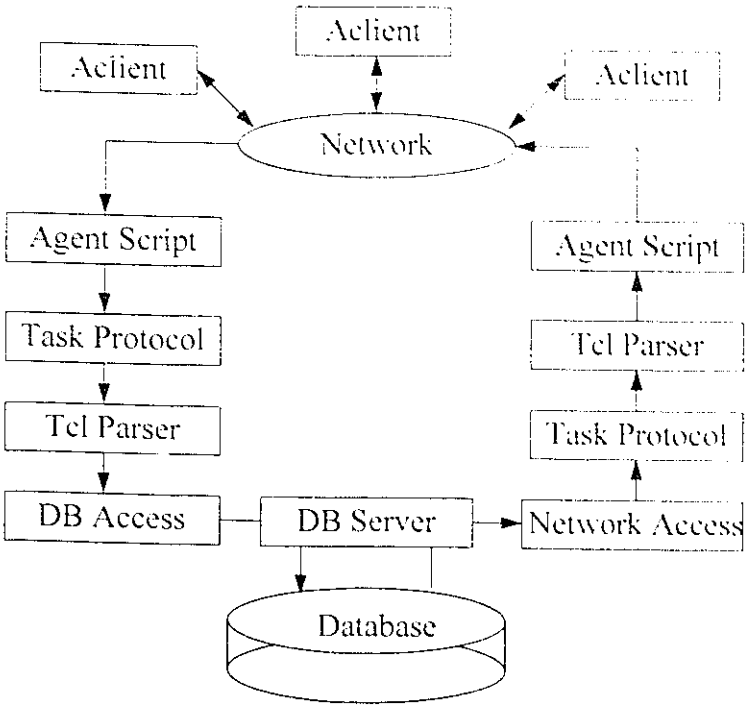


Figure 5.5: Modules between the network and database at the *iserver* site

5.2.4 Flow control at *aclient* and *iserver*

The controls of both the *aclient* and *iserver* processes can be represented in a flowchart to explain their roles. These processes run in the background when initialized. The uninitialized programs are represented as 'idle' in figures 5.6 and 5.7. The *aclient* process prepares to either send or receive agents when it is initialized or ready. On receiving an agent it is first saved as a named file called an *agent-file*. The agent-file is then opened for reading and Tcl interpreters created for parsing the script content. In this implementation the execution of the scripts displays the message carried by the agent. The messages could be articles retrieved from the database or simply messages sent by

the agent about its status. After either sending an agent or displaying a message, the 'ready' condition is reverted again.

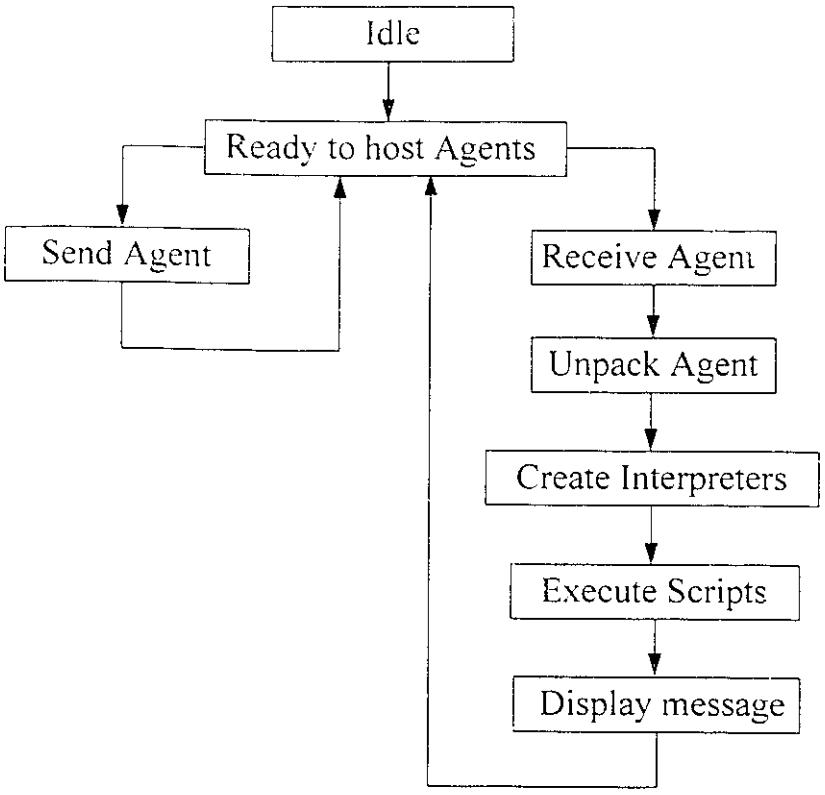


Figure 5.6: Flowchart of *aclient*

The algorithm used at the aclient sites is outlined below:

```
/* Record preferences */

if (specification not complete) V (has error) then
        (ask user refill template) else

/* Construct scripts */

while (a request exist)
     {
       if (not default destination) then (record specified address);
       (identify service requested);
       (check corresponding info like keywords field, etc.);
       (select preferences as arguments);
       (create command scripts);
       (record scripts in agent-file);
     }

/* Send agent-file */

if ((initialization of socket) != error ) then
     {
       (open agent-file);
       (send agent-file to destination);
       (close socket);
       (listen for connection);
     }
else  (print error message);
```

In the case of an *iserver* process, after initialization the process waits for agents to arrive. The unpacking of agent, creation of interpreters and execution of scripts are similar to that of the *aclient* process. Here the execution of the scripts is generally not for displaying messages but rather interaction with the database and sending the result of that interaction to the originator of the agent. There is an error recovery step for reporting to concerned parties about any failures or errors arising out of an agent execution.

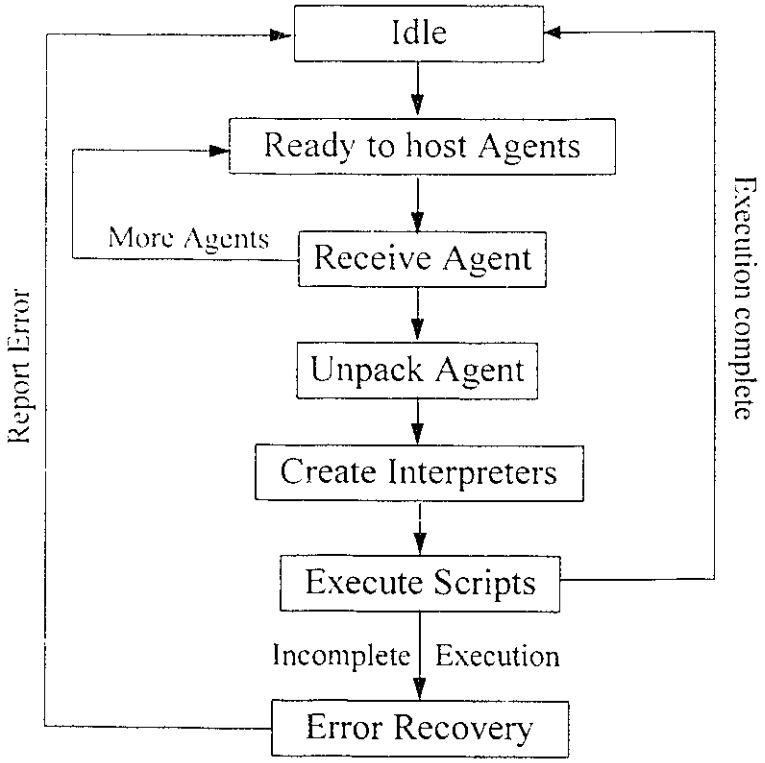


Figure 5.7: Flowchart of *iserver*

The algorithm used at the iserver sites is outlined below:

```
/* Iserver process waits for aclient connections */

if (request for connection exists) then
  {
   (open connection);
   (receive agent-file into buffer);
   (fork a child to attend to agent);
   (continue to listen for connections);
  }
else (listen for connections);

/* Child process executes agent */

(read agent-file from buffer);
(create Tcl interpreter);
while !(end of script commands)
       { (execute scripts sequentially);
          if (message or result to be sent back) then
            (goto send agent-file);
       }

/* Send agent-file back to aclient */

if ((initialization of socket) != error ) then
    {
     if (agent message to be sent) then
        {
         (create agent-file corresponding to message);
         (send agent-file to destination);
        }
     else (send result);
     (close socket);
     (listen for connection);
    }
else  (print error message);
```

5.3 Agent Specification and Construction

The tasks for an agent are specified with a graphical user interface as shown in figure 5.8. The user communicates her interests to the agent through the GUI. First of all, the *iserver* site has to be assigned by either specifying a known service provider site or setting it to default. In the latter case, the agent will search for the articles in different sites until it meets with success or completely exhaust the network database resources.

At the specified or successful database site, the agent requires certain information to filter for relevant articles. The 'agent mission' could be to search for documents or browse the database. For searching or browsing documents in the database, the user specifies keywords (authors, subject areas, dates etc.) for text searches. She can also indicate her preferences of media like video, audio, images and graphics. In the event of many documents meeting the keywords criteria, media is taken into account for selecting the best fit documents. For retrieving documents in the database after browsing, the user makes use of known document names.

Retrieved documents can be asked to be presented in a popup window, sent back by email or fax, stored for later usage or emailed to someone else. For emailing to other users, their email addresses has to be specified to the agent. Otherwise the agent uses the default user addresses to communicate the results back. The user may also want the agent report back on its latest status.

The tasks for the agent are finalized by clicking on 'OK' which records the GUI representation into a script file according to the Task Protocols described earlier. The script file constitutes the agent and its behavior or tasks. Finally, the agent is released into the network by clicking on the 'GO' button, which invokes the *ag_send* command. See figure 5.8.

5.4 Agent Whereabouts

After the agent is released into the network, the agent finds its way to the necessary database site or sites. At the agent environment there, the agent script is unpacked and executed. The execution results of the agent are recorded into a log file for future reference in case of any unforeseen circumstance like crashing of the computer system. This will help debugging at a later time. To keep the user aware of what the agent is upto, at each major step of the agent's life cycle, it reports back to the user of its latest status. However, the user may choose to disable this feature while specifying the agent task.

In this implementation, the agent is capable of sending back information like where it is presently located, what it is working on or what it has done. These information are spawned to the user in various popup windows or sent back as emails. This provides a 'loose' form of control to the user since the agent is not completely independent while going about its task. A further step can be developed to allow the user launch a new agent for communicating her desire to the original agent. For example, an agent may report back its failure to find a document at a particular site and the user may

launch another agent with the message for quitting. Then the user is truly in control of what the agent does or decides to do. The new agent should be able to be launched from the popup window which carried the original agent message.

5.5 A Sample Session

A typical sample session is presented in this section to show how the system works overall in terms of the possible events and timings. First of all, a user creates or specifies an agent on the main interface which is like filling a form in this implementation. The specifications for the mobile agent include simple basic descriptions of what document the user would like to retrieve from the news database.

In this example we assume we do not know the names of the database service provider and documents. Therefore, we select the 'default' option for the destination and 'retrieve document' for the agent mission. Other media options include text and images. We give a list of keywords like baseball and sports to match the documents. The agent is asked to present any document found and also send a copy by email. The main interface filled with the above specifications is shown in figure 5.8 .

On completion of the specifications, the 'OK' button is clicked to indicate acceptance of the user. At this point an agent is formed internally in the system. To launch the agent, the user clicks 'GO' and the agent is sent over a socket in the agent environment. The hosting environment on the other end subsequently receives the agent and allows it to execute.

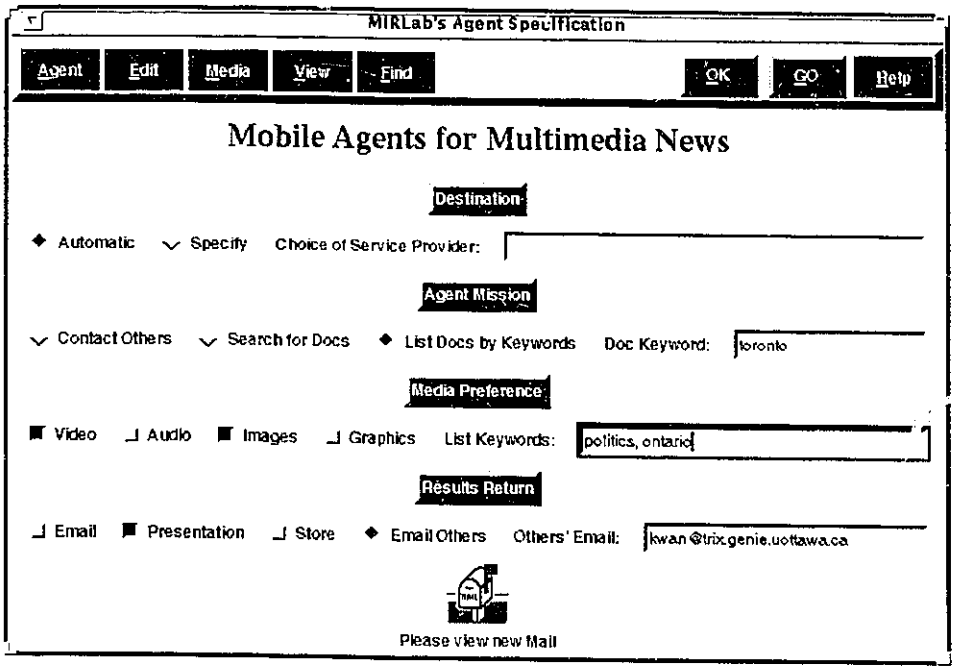


Figure 5.8: Interface for agent specification

After arriving at the hosting environment, the agent reports back to the user that it has arrived at a particular site. The popup window that presents the message is shown in figure 5.9 . The user clicks 'QUIT' to acknowledge the message.

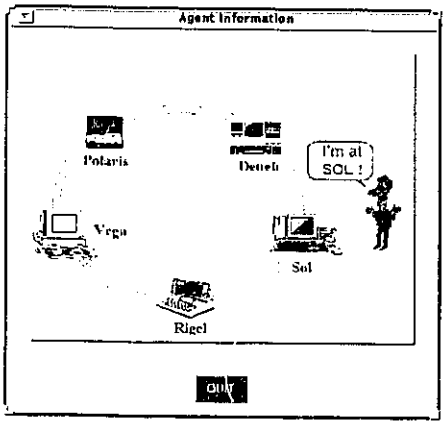


Figure 5.9: An agent arrived at its destination

In a few moments, the agent sends back another message indicating that it is filtering the database for relevant documents matching user specifications. The message shown in figure 5.10 is returned. If the agent mission had been browsing the database for the list of document names, figure 5.11 would have been the message instead.

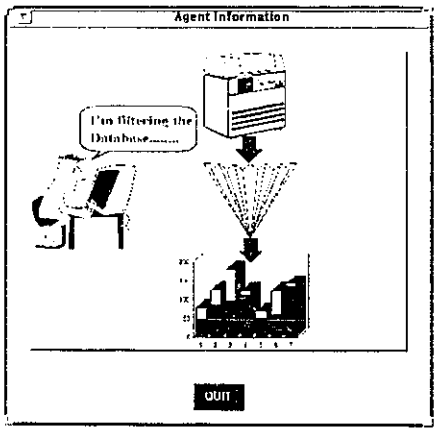


Figure 5.10: An agent filtering the database

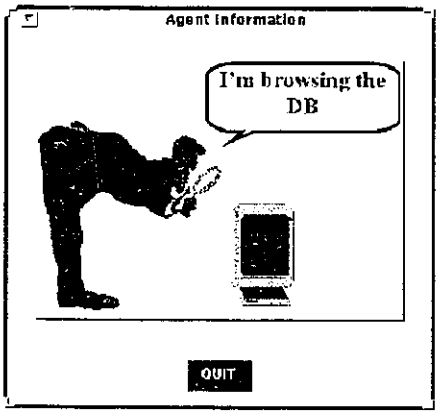


Figure 5.11: An agent browsing the database

The agent is successful in retrieving a document and prepares to return with it. Since the user asked for the document to be also sent as an email, the agent emails himself as well as specified friends before coming back with the document. Figure 5.12 notifies the user that an email has been sent and figure 5.13 informs of the agent's return.

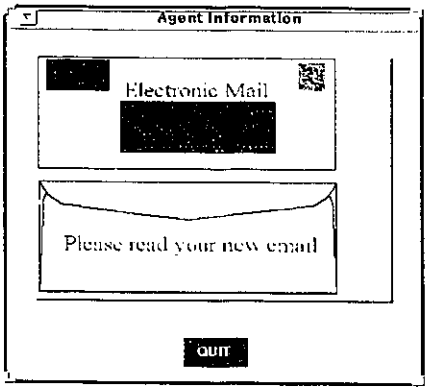


Figure 5.12: Notification of email sent by an agent

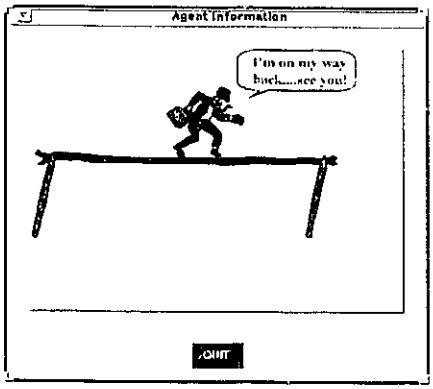


Figure 5.13: An agent on its way back to its originator

Finally, the retrieved document is presented to the user for viewing (figure 5.14). The user may choose to read and quit, or save it for the future. By this time, the email form of the document would have arrived and can be viewed with the standard system mail tool.

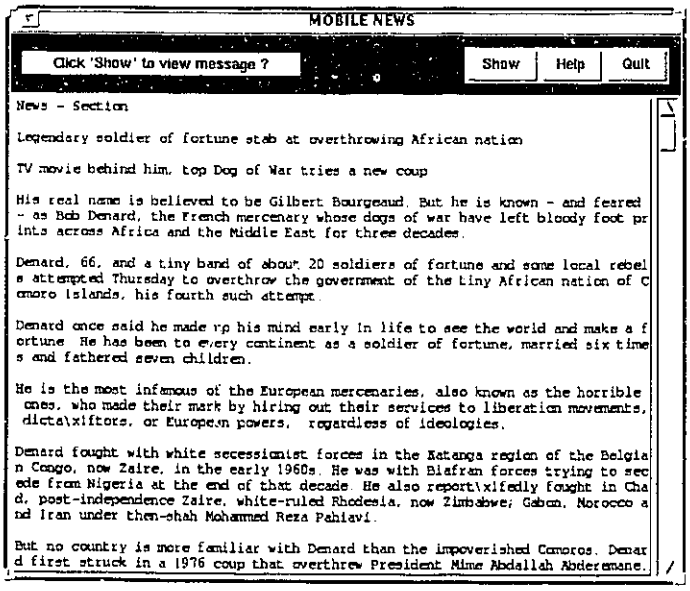


Figure 5.14: A sample document retrieved by an agent

Chapter 6

Conclusion

6.1 Summary

In this thesis, the agent paradigm has been presented as a solution to personalized services. The application of agents in other areas of research was discussed with respect to their behavior. Contemporary prototypes were studied to enhance the knowledge of agent-based computing. Classifications of agents were made and presented from various important perspectives. The work presented in this thesis focuses on the application of agents to personalized multimedia news filtering and retrieval. Specifically, the mobile

aspect of the agent has been implemented to test the theories of interactive and asynchronous communications.

Four components were designed and implemented in order to realize a mobile news agent system. They are the agent environment based on the aclient-iserver model, a mobile agent model, the protocols for agent interaction with the database news server and the user interfaces for the news application. A mobile agent's complete life cycle from its initialization through its successful remote execution had been tested and described.

The aclient-iserver model was created to provide a suitable environment for hosting mobile agents in particular. It bears resemblance to the traditional client-server model with a few modifications. The major differences are in the asynchrony of the aclient and the interaction of the iserver. An aclient can launch as well as receive agents while an iserver can, beside launching and receiving agents, interact with service providers like the database servers. Mobile agents are transferred between the aclient and iserver modules via intermittent connections. Beside the transfer of agents, the environment allows the execution of agent scripts.

A corresponding mobile agent structure has been proposed for the agent environment. Since the mobile agent is released into the network without constant control, it is embedded with appropriate control and error recovery mechanisms. It knows what it wants, where it wants to go and how it wants to achieve its goals. It is unpacked in the agent environment and the scripts are sequentially executed. The receiving site thus knows the intentions of the agent and provide necessary resources.

The interaction of a mobile agent with the news server has been defined in the API of the database server. There are predefined primitives for agent access to the database with mutually accepted protocols. These features include all those present in the present prototype of the news-on-demand project, that is an agent have equal access to the news database as any clients in the client-server paradigm. The only difference now is the conversion of the primitives into scripting commands as understood by the agents. These primitives have been implemented as Tcl application commands.

Graphical user interfaces for mapping user specifications into corresponding scripts have been implemented with Tk toolkit. The specification interface is like a form for collecting information on the type of news articles of interest to the user. Transformation of specifications into scripts were achieved through the compatibility of Tk with Tcl and C. Other interfaces including the cartoon depiction of what the agent is doing or where it is located, are meant for user/agent interactions. The agent can also communicate these messages and the results with the user via email in this implementation.

6.2 Suggestions for Future Work

This thesis led to the identification of some interesting issues pertaining to the mobile news agent. It has also opened up many avenues for further studies and future research. This section offers some suggestions for the improvement of the mobile news system. Most of the improvements are aimed at extending the capabilities and features of the current implementation.

The current agent environment based on the aclient-iserver model has the basic functionality of hosting mobile agents. It can be extended to include other tasks that makes the environment more powerful. Error recovery mechanisms should be included for regenerating lost or destroyed agents due to system failures. A name server for the agents can be added to the environment to keep track of all the existing agents in the network. Monitoring all the hosted agents and avoiding 'agent-overflow' would then be undertaken by the environment. This will shift the task load away from the mobile agent to the environment support, thus maintaining a 'lightweight' agent.

The mobile agent itself has great scope of improvement. A more detailed protocol structure would allow an understanding among agents and service providers. Thus, a service provider may get a fairly good idea of what an agent is seeking by simply parsing through some fields or flags in the agent structure. Information exchange and negotiation among agents will be made easy. This will also be more efficient in maintaining the privacy of agents and their users.

Security has not been dealt with in this implementation, instead the restriction arising out of the commands available to the agents along with the database access protocol indirectly provided a secure environment. The agent environment can involve itself of security and reduce the restrictions applied indirectly. Such an improvement will see more powerful and useful agents with greater autonomy.

Work needs to be done in other agent applications like TeleLearning and digital library to verify and test agent behaviors for a wider spectrum of usage. These

applications should be built with the agent architecture in mind to facilitate the relatively ease of attaching the agent to the new applications. There are a few points to bear in mind in the interest of smooth implementation. A common graphical uses interface should be built to include every kind of agent that would be constructed. This will help the user accustom to the agent irrespective of its added features. The communication between the user and the agent can be enhanced by including voice and text recognition features in the user interface. For example, a user may then 'talk' to the agent instead of typing commands on the keyboard.

A multimedia agent managing media types can be extremely difficult but interesting. The personal computer platform under Windows 95 would probably provide greater support to this kind of implementation. Although handling media with semantics is still a dream of the future, it would draw us closer by first dealing with raw media at the presentation level. That is, let mobile agents prepare the required media types for a multimedia presentation.

Appendices

A. Publications

(1) Kwan, W. and Karmouch, A., "*An Intelligent Agent for Multimedia Newspaper*", Proc. of the Canadian Conference for Electrical and Computer Engineering, Montreal, pp. 594-597, Sept. 1995.

(2) Kwan, W. and Karmouch, A., "*Multimedia Agents in a Broadband Environment*", *Proc. of the IEEE International Conference on Communications*, Dallas, TX, Vol.2 pp.1123-1127, June 23-27, 1996.

Bibliography

[BAT94] Bates, J., "The Role of Emotion in Believable Agents", *Communications of the ACM*, Vol.37, No.7, pp.122-125, July 1994.

[BRO90] Brooks, R.A., "Elephants Don't Play Chess", in Maes, P., ed., *Designing Autonomous Agents*, The MIT Press, Cambridge, MA, pp.3-15, 1990.

[CHE95] Chess, D., et. al, "Itinerant Agents for Mobile Computing", *IEEE Personal Communications*, pp.34-49, October 1995.

[CHD94] Chess, D.M., Harrison, C.G., and Kershenbaum, A., "Mobile Agents: Are They a Good Idea ?", *IBM Research Report, RC 19887*, October 1994.

[CHI91] Chin, D., "Intelligent Interfaces as Agents", In J. Sullivan and S. Tyler, eds., *Intelligent User Interfaces*, ACM Press, N.Y.,1991.

[COE94] Coen, M.H., "SodaBot: A Software Agent Environment and Construction System", *AI Technical Report 1493*, MIT AI Laboratory, June 1994.

[EME93] Emery, J., and Karmouch, A., "A Multimedia Document Model for Continuous Media", *CCECE CCGEI*, Vancouver, BC, pp.640-643, 1993.

[FAL96] Falchuk, B., and Karmouch, A., "Feature Set and Architecture of a Multimedia News System Over an ATM Network", *Multimedia Tools and Applications*, Kluwer Academic Publishers, Vol.2, No.1, pp.11-33, January 1996.

[FAB95] Falchuk, B., and Karmouch, A., "A Multimedia News Delivery System Over an ATM Network", *Proc. International Conference on Multimedia Computing and Systems*, pp.56-63, Washington D.C., May 1995.

[FAE95] Falchuk, B., Hirzalla, N., and Karmouch A., "A Temporal Model for Interactive Multimedia Scenarios", IEEE, *Multimedia Magazine*, Vol.2 No.3, September 1995.

[FIN92] Finin, T., McKay, D. and Fritzson, R., "An Overview of KQML: A Knowledge Query and Manipulation Language", *Stanford University, Computer Science Dept.*, March 1992.

[GEN94] Genesereth, M.R. and Ketchpel, S.P., "Software Agents", *Communications of the ACM*, Vol 37 No.7, pp.48-53, July 1994.

[GEO87] Georgeff, M.P. and Lansky, A.L., eds., "Reactive Reasoning and Planning" *Proc. of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pp.677-682, Seattle, WA, 1987.

[GES87] Genesereth, M.R. and Nilsson, N., eds, "Logical Foundation of Artificial Intelligence", *Morgan Kaufmann Publishers*, San Mateo, CA., pp.325-327.

[HAU94] Haugeneder, H., Steiner, D. and McCabe, F.G., "IMAGINE: A Framework for Building Multi-agent Systems", in Deen, S.M., ed., *Proc. of the 1994 International Working Conference on Cooperating Knowledge Based Systems (CKBS-94), DAKE Centre*, University of Keele, UK,1994.

[HOU94] Houlder, V., "Special Agents" in *Financial Times*, pp.12, 15 August 1994.

[IND95] Indermaur, K., "Baby Steps", *Byte Magazine*, pp.97-104, March 1995.

[KAR93] Karmouch, A., "A Multimedia Information and Communication System: MEDIABASE", *Proc. Multimedia Communication'93*, Banff, April 1993.

[KWA96] Kwan, W. and Karmouch, A., "Multimedia Agents in a Broadband Environment", To appear in the *Proc. of the ICC 1996*, June 23-27.

[KWN95] Kwan, W. and Karmouch, A., "An Intelligent Agent for Multimedia Newspaper", *Proc. of the Canadian Conference for Electrical and Computer Engineering*, pp. 594-597, Sept. 1995.

[LAI94] Lai, K., Malone, T. and Yu, K., "Object Lens: A 'spreadsheet' for Cooperative Work", *ACM Trans. Office-Inf. Systems* 5 (4), pp.297-326.

[LAS94] Lashkari, Y., Metral, M. and Maes, P., "Collaborative Interface Agents", *Proc. of the National Conference on AI*, MIT Press, Cambridge, Mass.,1994.

[LAU91] Laurel B., "Computers as Theatre", *Addison Wesley Publishing Co.*, New York, 1991

[LOE92] Loeb, S., "Architecting Personalized Delivery of Multimedia Information", *Comm. of the ACM*, Vol:35(12), pp.39-50, Dec.1992.

[MAE94] Maes, P., "Agents that Reduce Work and Information Overload", *Comm. of the ACM*, Vol:37(7), pp.30-40, July 1994.

[MAR85] Marvin, M., ed., "The Society of Mind", *Simon and Shuster*, NY, 1985.

[MAS90] Maes, P., "Situated Agents Can Have Goals", in Maes, P., ed., *Designing Autonomous Agents*, The MIT Press, Cambridge, MA, pp.49-70, 1990.

[MAP94] Maes, P., "Social Interface Agents: Acquiring Competence by Learning From Users and Other Agents", in Etzionii, O., ed., *Software Agents -- Papers from the 1994 Spring Symposium (Technical Report SS-94-03)*. AAAI Press, pp.71-78.

[MAP91] Maes, P., "The Agent Network Architecture (ANA)", SIGART Bulletin, 2(4), pp.115-120, 1991.

[MIT94] Mitchell, T., Caruana R., Freitag, D., McDermott, J. and Zabowski, D., "Experience with a Learning Personal Assistant", *Comm. of the ACM*, Vol:37(7), pp.81-91, July 1994.

[OUS94] Ousterhout, J.K., ed., "Tcl and the Tk Toolkit", *Addison-Wesley Publishing Company*, Reading MA, 1994.

[REI94] Reinhardt, A., "The Network with Smarts", *Byte Magazine*, pp.51-64, October 1994.

[ROD95] Rody, J.A. and Karmouch, A., "A Presentation Agent for A Distributed Multimedia System Over High Speed Networks", *Proc. International Conference on Multimedia Computing and Systems*, pp.223-230, Washington D.C., May 1995.

[SEL94] Selker, T., "COACH: A Teaching Agent that Learns", *Comm. of the ACM*, Vol:37(7), pp.92-99, July 1994.

[SHE93] Sheth, B. and Maes, P., "Evolving Agents For Personalised Information Filtering", *Proc. of the Ninth Conference on AI for Applications*, IEEE Computer Society Press, pp.345-352, 1993.

[SHB94] Sheth, B. "A Learning Approach to Personalized Information Filtering", *SM Thesis, Dept. of Electrical Engineering and Computer Science, MIT*, 75 pages, Feb1994.

[SHO90] Shoham, Y., "Agent-Oriented Programming", *Technical Report STAN-CS-1335-90*, Computer Science Department, Stanford University, Stanford, CA, 1990.

[STE88] Steeb, R., Hayes-Roth, F.A., Thorndyke, P.W. and Wesson, R.B., "Distributed Intelligence for Air Control" in Bond, A.H., and Gasser, L., eds, *Readings in Distributed Artificial Intelligence*, pp.90-101, Morgan Kaufmann Publishers, San Mateo, CA.

[STF96] Standford web site "http://cdr.standford.edu/ABE/JavaAgent.html", 1996.

[STV92] Stevens, R.W., "Advanced Programming in the UNIX Environment", *Addison Wesley Professional Computing Series*, 1992.

[SUN94] Sun Microsystems, "The Java Language: A White Paper", *Sun Microsystems*, 1994.

[TAN90] Tanimoto, S.L., ed., "The Elements of Artificial Intelligence Using Common Lisp", *Computer Science Press*, NY, 1990.

[THO93] Thomas, S.R., "PLACA: An Agent Oriented Programming Language", *Technical Report STAN-CS-93-1487*, Computer Science Department, Stanford University, CA, 1993.

[UMB96] UMBC web site "http://www.cs.umbc.edu/~cikm/iia/submitted/viewing/thompson.html", 1996.

[WAN93] Wang, R., and Karmouch, A., "A Multimedia File Structure for Continuous and Discrete Media", *CCECE/CCGEI*, Vancouver, BC, pp644-647, Sept. 1993.

[WAY95] Wayner, P., ed., "Free Agents", *Byte Magazine*, pp.105-114, March 1995.

[WEL95] Welch, B.B., ed., "Practical Programming in Tcl and Tk", *Prentice Hall PTR*, Upper Saddle River, NJ 07458, 1995.

[WHI94] White, J.E., "Telescript Technology: The Foundation for the Electronic Marketplace", *General Magic Inc.*, Mountain View, CA, 1994.

[YUE90] Yues Kodratoff and Ryszard Michalski, eds., *"Machine Learning -- An Artificial Intelligence Approach, Vol. 3"*, Morgan Kaufmann Publishing, Inc., CA, 1990.