A Multiple Autonomous Agent System for Negotiating Valued Information Exchange on the Web.

Larry O'Neill,
BA (mod) MSISS

A dissertation submitted to the University of Dublin, Trinity College, in partial fulfilment for the degree of Master of Science in Computer Science.

September 2000

Declaration

I, the undersigned, declare this work has not previously been submitted as an exercise for a degree at this or any other University, and that, unless otherwise stated, it is entirely my own work.

_____

Larry O'Neill,

September 2000.

Permission to Lend and/or Copy

I, the undersigned, agree that the Trinity College Library may lend and/or copy this dissertation upon request.

_____

Larry O'Neill,

September 2000.

Acknowledgements.

Many thanks are due to my supervisor Mr Padraig Cunningham for his help, humour and feedback throughout the project. Also, to my friends, family and fellow NDS classmates, thanks for your support and encouragement.

Summary.

The amount of detailed personal information being recorded electronically about individuals is growing rapidly. The increasing commercial pervasiveness of the Internet is a major factor in this growth. This information has a significant economic value yet it is only relatively recently that data subjects have begun to expect and receive some reward for the surrender and/or use of their information. It is considered likely that this trend will continue, producing a growing uneasiness over the erosion of personal privacy. In time, legislation or acceptable practice may oblige data users who wish to gather consumer information, to pay data subjects to provide that information.

The mechanics and logistics of data subjects supplying their own information to data users necessitates an automated system to carry out the transactions. Any such system has to be able to cope with the widest variety possible of data formats and quantities. It should allow the data sellers to specify at a very fine-grained level which information should be sold as what price. It should allow data users to stipulate what type of 'souls' they are seeking, how many and how much they want to spend.

This dissertation describes a system called the Soul Seller System that attempts to provide such facilities. The system is based on mobile agent technology called Aglets. Data subjects provide information about themselves via a web page. This data is passed to a selling Aglet (a 'soul seller') that then represents the data subject in transactions with the 'soul buyers', other Aglets who each represent a data user seeking information. When initially created, both types of Aglets dispatch themselves to a central Aglet server, the 'marketplace' where soul buyers and sellers can meet each other. A conversation is initiated where soul buyers tell soul sellers what information they are seeking. The soul sellers respond with the price this information would cost. If the price is considered suitable, money and information are exchanged. Both data subjects and data users can track what transactions have been carried out in their name via a web page.

While the Soul Seller System works well on a small scale, problems are experienced when attempts are made to scale it up. This is due to the limit on the number of Aglets that can be run on just one Aglet server. Two possible solutions include using a database to represent the soul sellers or to permit multiple marketplaces thereby enlarging the capacity of the system. It is concluded that basing the transactions exclusively on Aglets is useful when developing a prototype but would not be appropriate for the full application version.

Table of Contents.

Table of Figures.

1 Introduction.

"You cannot think about thinking, without thinking about thinking about something."

Seymour A. Papert, MIT Media Lab

1.1 Introduction to the Problem Domain.

The general problem domain being addressed is the area of consumer profiling. Competition is forcing businesses to develop relationships with individual consumers. Where once marketing and product offerings were made to large groups of consumers, now the trend is increasingly to tailor both marketing and products to individuals. Some Internet based firms such as Amazon.com can offer customised recommendations to individual customers based on what they have bought before. It is known as mass-customisation. In order to develop this sort of relationship with a consumer, some information must be known about them in advance. Traditionally, acquiring individual consumer information was often no more sophisticated than purchasing a mailing list from a database-marketing firm. This situation has been transformed by two developments:

1) There is a requirement for a much greater level of detailed knowledge about consumers to do business on a mass-customised basis.
2) The increased connectivity of consumers allows new methods of obtaining information about and from consumers.

It is the contention of this dissertation that consumers will not only be increasingly incentivised to provide information about themselves but that they will come to expect and demand recompense for doing so. If this is true then some mechanism will be required to get data from consumers to businesses and to get the compensation or reward from the businesses to the consumer. Schemes such as supermarket loyalty points, with their associated discount or cash back benefits, are only suitable for tracking certain types of information and in certain situations. Similarly, online consumer profiling such as Amazon's only captures a small subset of possible useful information. The ideal solution is to have all information about a person available, but available at a possibly high price. The origin and control over usage of that information must reside with the consumer himself. This work is intended to address the logistical problems inherent in such a system. It does this by providing an application that facilitates data subjects (consumers) in selling their information to data users (commercial organisations).

1.2 Aims and Objectives of the Project.

There are four objectives for the overall work:

1) To establish that there is a requirement for an application that will allow ordinary consumers to sell information about themselves, such as transaction data, to commercial (or otherwise) data users.
2) To create an application that fulfils the requirements described. The application must provide data subjects with an easy to use, flexible method of providing the information about themselves, which they are willing to sell. Conversely the application must also allow data users the ability to search for their target type of data subject and purchase specific information from them. This must be done without compromising the private nature of the data subject's information.
3) To create a suitable information representation structure that can deal with different data formats, data types and volumes.
4) To test the suitability of agent technology for this application.

1.3 Structure of the Report.

Chapter 1 gives a brief introduction to the problem area being addressed, leading on to the aims and objectives of the project.

Chapter 2 discusses the problem area in more detail, building up the evidence for the assertion that the Soul Seller application is already or soon will be required. Some current methods of data collection and the rise in data mining are examined. An argument is made that personal information has a significant economic value and evidence is used to support this claim. It is then posited that data subjects will increasingly seek incentives to surrender or permit use of their personal information. A number of arguments are advanced in support of this. The logistical problems involved in bringing data subjects and users into contact in order to carry out exchanges are identified. Finally, the likely characteristics of a solution to these logistical problems are considered.

Chapter 3 reviews agent and related technologies. The contentious issue of exactly defining just what are agents, is discussed and largely dismissed as being unnecessary. The issues that affect all types of agents are then examined. The subset of agent research known as mobile agents is then addressed. Rothermel's taxonomy of agent mobility is used to distinguish between the varying kinds of mobile agent. The advantages and disadvantages of agent mobility are identified along with the reasons for the selection of Java as the most common development language for mobile agents. Different agent development environments are briefly examined. IBM's Aglet API is reviewed more thoroughly with emphasis on the security model and design

patterns. XML is then briefly discussed as a possible enabler of greater information interoperability.

Chapter 4 deals with the design phase of the project. The problem domain is reviewed from the point of view of identifying user requirements. Alternative options to using agents as the basis for development are examined. The unusual characteristics of the problem domain are discussed in the context of their influence over the design process. The design outline is then given, highlighting the different components of the system. Lastly, individual design decisions and issues are discussed.

Chapter 5 provides the particulars of the implementation of the system, broken down into the three major sections - Seller, Buyer and Marketplace. Specific details discussed include the structure of the buyer-seller conversation, the use of the escrow agent, the calculation of the matching statistic and how the selection of sellers to buy from is made.

Chapter 6 evaluates the work and how successful it is in meeting the project objectives. The application is assessed and future work is identified.

2 Review of the Problem Domain.

"Civilization is the progress toward a society of privacy."
Ayn Rand, 'The Fountainhead'

The features and issues of the problem domain are discussed in this chapter. Assertions are made as to the likely developments in the domain and these are supported by a number of arguments. The motivation for the application developed is thus shown.

2.1 Not Quite Big Brother.

The notion that information about practically every move and transaction we make is stored in some perfectly cross-referenced and easily accessible, omnipotent computer, is one of Hollywood's favourite conceits. This is evident from films such as 'The Net', 'Hackers' and 'Sneakers'. Apart from being a useful, if somewhat lazy, plot device, it preys nicely on the paranoia of those not au fait with the power of information technology and on the interest of those who are. Typically in such films, the identity, financial, medical or other records of the hero or heroine can be easily erased or edited with a few keystrokes by the bad guys, for their presumably evil purposes. Anyone who has ever changed address and tried to inform the various relevant utilities, financial and retail companies of the fact, soon experiences the rather more mundane, frustrating reality of instant information updates.

The Hollywood all knowing computer fallacy, however, is only partly untrue. A vast amount of information about ordinary people *is* held by all sorts of organisations. Every time a credit card is used, a bank transaction made, a loyalty bonus point earned or a web site visited, data accrues on some computer. Unlike the Hollywood fantasy, this information is certainly not centrally maintained, cross-referenced and accessible by some shadowy 'Big Brother' like organisation controlling the masses. It is held in quite isolated data pockets by companies that merely wish to sell more goods and services. By engaging in a practice known as data mining, where patterns in stored transaction data are identified and then exploited, companies can raise sales, cut inventories, reduce costs and identify new products and markets. The oft-quoted example given is that of grocery chain store that noticed a pattern occurring on Friday evenings, when there were high sales of both baby nappies and six packs of beer to young males. By moving the nappies and beer closer to each other in the aisle, sales of both were increased.

Data mining is not of course possible without the raw transaction data. Shops initially gathered this through Point of Sale (POS) systems that stored information about each product purchase made. POS systems did not however allow multiple purchases over time to be tracked. Shops had no way of knowing if the buyer of one basket of goods one week was the same buyer of another the next. The solution was to introduce loyalty card schemes. Shoppers registered with the scheme, were issued with a numbered electronic swipe card and could then accrue valuable loyalty points for every purchase made, as long as it was registered against their scheme number. Ostensibly, this was to gain competitive advantage by rewarding customer loyalty, but the real benefit was in the amount of customer information it generated. Such schemes are not cheap to set up or run. Substantial initial investment is required as well as the ongoing costs of redeeming the loyalty points earned, which can be between 1% and 2% of turnover. Nevertheless, supermarkets, airlines, bookshops and web sites happily bear the cost, because the return on their investment is high. The personal information they gain for the cost of some money off vouchers, free flights or percentage discounts is a veritable goldmine. Not only can businesses improve their own operational efficiency, they can understand their customers better, meet their needs better and build a real sustainable competitive advantage [D'Arcy, 1995]

2.2 Personal Information Has an Economic Value.

Before justifying this statement, it is worth considering just what is personal information. For the purposes of this work, personal information is any sort of information about a person that is specific and unique to him or her. A person's name, address, phone number, likes and dislikes, sports team preferences, shopping habits, job description and credit history can be classed as personal information. Ryanair's summer timetable; Manchester United's Premiership statistics; the number of mobile phones in Ireland are evidently examples of non-personal information. Not all of a person's personal information is of interest or value to any one person or entity but all of it is of potential interest to someone or entity. The fact that a person does not list 'gardening' in their leisure pursuits is useful to a gardening centre because knowing this can save the company the cost of sending that person an unwanted garden catalogue.

Is it justified to claim that personal information has an intrinsic economic value? It is the contention of this dissertation that it is. It costs money to gather and maintain. It provides benefits to its holder. It is treated as an asset by regularity authorities. For example:

- Companies spend money on loyalty schemes, customer surveys and questionnaires to gather personal information on their customers. They would not do this if they did not get something of value to them.

- The U.S. Securities and Exchange Commission ruled [Lumeria, 2000] that companies that gave away 'free' shares to people who registered with their web sites, requiring disclosure of name and contact information were actually selling shares. Something of value (the information) was being exchanged for the shares.
- Companies sometimes buy other companies for their personal information databases. In 1999 online advertising delivery company DoubleClick bought the off line database direct marketer Abacus Direct for $1 billion. This was primarily for their 2 billion record Abacus Alliance Database of consumer catalogue transactions. DoubleClick intended to match up their online consumer data with the acquired company's 'real world' consumer databases, until controversy over privacy concerns made them curtail the plan.
- It is entirely common practice for many Internet companies to provide free access to services in return for consumers surrendering a certain amount of information about themselves. There are numerous examples of such companies including Hotmail.com, Email.com, guru.com and RealTimeQuotes.com.
- The sole raison d'étre of some companies is to facilitate the exchange of personal information. Mailing list brokers purchase collections of personal information, slice and dice them into appropriate formats and sell them on to companies looking for new potential customers. Typical sources of the mailing lists are magazine subscription lists, conference attendance lists and the customer lists of other companies. According to Direct Line Marketing, an Irish direct marketing company, mailing lists are sold at an average cost of £150 per thousand names and addresses for a single use. So at a bare minimum a name and address is worth 15 pence each time it is used.

Therefore, it would seem a reasonable assertion that personal information has an economic value. Users will determine what is and is not useful information, while the market will set the price paid. Data subjects may set whatever price they like for the information but it is up to data users to choose whether to pay that amount.

2.3 Data Subjects Will Increasingly Demand Recompense.

Two reasonable assertions have been made with regard to personal information:

1) Commercial organisations are gathering an ever-increasing range and volume of data about individual consumers.
2) This personal information has an inherent economic value to a variety of entities.

A third, much less easily justified assertion is required to fully explain the motivation for this work. It is that at least a proportion of the value generated by a person's personal information

will or should accrue to that person. There are a number of arguments of varying strength that can be made in support of this:

2.3.1 Personal Intellectual Property.

The argument here is a quasi moral / legal one. It is based on the assumption that because the information about a person originates from that person, they own, in some legal or moral sense, the rights to that information. The information should be given the same legal copyright protection as, say, a song or a novel. This argument may be viewed as somewhat weak. After all, information about someone is not created in the same way as an artistic work. The copyright protection afforded to a song is partly to encourage creative works and partly to protect their creator's livelihoods. The sort of personal information at issue here is just a side effect of normal transactions. Indeed, in the case of a consumer's grocery shopping patterns, the actual consumer could probably not describe those patterns in a useful way. It is only by the use of a supermarket's POS system can that information be captured. The privacy argument puts forward a much stronger case for the protection of personal information. However, the actual rights and wrongs of the situation may not be the issue, the *perceived* rights and wrongs and public opinion may be the real driver of this argument.

2.3.2 Economic and Market Theory.

David McCourt of the McKinsey consulting group devised a theory of market surplus to identify which of the participant sectors (producers, retailers or consumers) in a market has the greatest degree of power. Essentially in any market, there is a finite amount of profit or surplus over costs, which is competed for by each sector of the market. The theory does not relate to the competition within the same sector of the market, for example between different producers, but to competition between sectors. Each sector tries to maximise the value added in its part of the market process.

Better access to better information is one means by which one market sector can gain advantage over others. The Stock Exchange is seen as an equitable and fair market where neither buyers nor sellers have an inherent advantage. Equality is maintained between both groups by forcing them to operate using the same publicly available information. Any party privy to additional non-public information is specifically disallowed from operating in the market as this 'inside information' confers an unfair advantage. However, in the insurance market for instance, there might be a large information imbalance between the insurance companies and the consumers. The insurance companies could know a lot more about the consumers, for example the nature

and size of their risks, than the consumers know about the insurance companies, for example the speed of their claims handling. Customers are thus not equipped to make the best choice of policy for them, but insurance companies can price the policies in a way advantageous to them. The market surplus would then be primarily with the companies.

Most markets for normal goods and services operate in this way, with built in information imbalances. It is generally true that producer companies are large entities with greater resources available to them than the individual customers they serve. These resources allow them to both gather information about their customers and to dispense 'favourable' information about themselves via advertising. Producers can carry out extensive market research to find out information about the consumers and their preferences. Advertising on the other hand pushes non-objective information to the buyers influencing them to be more sympathetic towards the advertiser. An information imbalance thus results, giving the producer side of the market an advantage over the consumer part of the market.

The argument in favour of data subjects being paid a fee for the use of information about them thus states that because this information gives additional advantage to the data user (the producer or retail segment), the consumer or data subject should be compensated.

2.3.3 Privacy, Law and Acceptable Practice.

The essence of this argument is that the right to privacy affords people the right to choose freely under what circumstances and to what extent they will expose themselves, their attitudes and their behaviour to others [Westin, 1967]. So the fact a person subscribes to a certain magazine should not be revealed to others unless the subscriber has given permission for this information to be shared. Permission to share or use is a key concept in this area.

The legal right to privacy is by no means a universally accepted or defined right. Some jurisdictions do not even explicitly mention it in their constitutions or laws. However, the national courts in most of those countries have found the right in other provisions [Privacy-International, 1999]. There are also International Agreements on Human Rights such as Article 12 of the 1948 Universal Declaration of Human Rights, which declared, "No one should be subject to arbitrary interference with his privacy, family, home or correspondence." It was only in the 1960's and 1970's that the notion of the right to privacy was linked to the computer domain. The growing availability of sophisticated and powerful information technology led to concerns about its surveillance potential. The ability of computer systems to search and cross reference information from multiple sources made the monitoring of people's lives much easier

and immediate than ever was possible with purely paper-based systems. In 1970, the Land of Hesse in Germany was the first to introduce Data Protection legislation. Since then similar legislation has been introduced and harmonised across Europe with the introduction of the 1995 Directive on Data Protection [European-Parliament, 1995].

This directive defines the duty of care organisations (whether governments, companies or whatever) must take with the data they hold on computer as well as the rights of people ("data subjects") to have access to the information held about them. In particular all data must be:

- Obtained fairly and lawfully;
- Used only for the original specified purpose;
- Adequate, relevant and not excessive to purpose;
- Accurate and up to date;
- Accessible to the subject;
- Kept secure; and
- Destroyed after its purpose is completed.

It is not difficult to see how the strict implementation of such laws would quickly have a profound effect on Internet companies hoping to capture and use the personal information about their customers. In deed, this has been a source of conflict between the E.U and the United States. The conflict arose because under the terms of the directive member States have an obligation to ensure that personal information relating to European citizens can only be exported to countries that have an "adequate level of protection." The United States has no similar protective legislation. Eventually, under a 'Safe Harbour' agreement, export of information was allowed from Europe to America if the organisations using the information undertook to adhere to the terms of the directive.

While there may be no similar data protection legislation in the United States (other than for minors), there is a great deal of private concern over the increased loss of privacy from the rise of an all-encompassing Internet based economy. There is much evidence of a growing general awareness amongst web users of the threat to privacy from online profiling. Quite a number of privacy rights promoting organisations such as www.privacy.org, www.epic.org (the electronic privacy information centre) and www.privacyplace.net have been established. They are joining longer established organisations such as the American Civil Liberties Union (ACLU) and the Electronic Frontier Foundation (EFF) in a bid to inform, raise awareness and gain support for strengthening online privacy rights.

There is a trend amongst commercial web sites to display their privacy policies, stating what can and cannot be done with the information gathered from customers and visitors. There is however no legal enforcement of these policies. If the policies are infringed, the customer has no right to compensation. Even if they had it would be irrelevant as the policies can be changed at will. The U.S. Federal Trade Commission (FTC) reported to the U.S. Congress on Online Privacy [Pitofsky, 2000]. In the report the FTC recommended the introduction of regulation of consumer oriented web sites to protect the privacy of their users. Whether this recommendation is adopted or not commercial pressures may eventually lead to a tightening of the actual privacy practices of Internet companies. Mention has already been made of the controversy over the DoubleClick plan to cross-reference their online database data with the Abacus 'real world' database data which eventually forced a climb down, at least until 'proper privacy standards' are in place. It remains to be seen if a similar backlash will affect Amazon who very recently changed their privacy policy to allow the resale of customer data to third parties in the event of the company being sold. Certainly if the digital chattering classes at the popular online discussion site, SlashDot.com, are in anyway representative of the general online population (unlikely), Amazon may have a problem. Already the aforementioned Electronic Privacy Information Centre has discontinued their book selling agreement with Amazon.

So, while technological advances in computing power and data mining may be expanding the data acquisition options available to data users, legislation whether domestic or imported and commercial pressures, restrict them. In order to gain the full benefit of the personal information they possess or can obtain, the privacy argument contends that organisations will increasingly have to obtain the permission of the data subject. Data subjects will need to be enticed to give that permission. A recent survey of web users [Andersen, 2000] found that three-quarters of respondents would reveal additional personal information in return for some reward. Hence, recompense will be paid for the use of personal information.

2.3.4 Better Source of Information for Data Users.

The fourth argument in support of recompensing data subjects for the use of information about them is that obtaining the information directly from them will result in better quality information. The theory is that who would know a person's opinions, feelings or behaviour better than the person himself would?

This may be true for certain types of information but is unlikely to be true for all. There are two problems with this hypothesis, the first with accuracy and the second with honesty. In the absence of any automatically recording mechanism, who could accurately recall, for example,

what groceries they bought last month? This problem could be applicable to many categories of desirable information. Alternatively, the data subject may know the required information accurately but be unwilling to share it, perhaps because it is embarrassing. Hence an inaccurate answer might be given. Alternatively, an answer might be given that the data subject wants the questioner to believe about them. The act of asking a question affects the answer given. This is the well-known Hawthorne Effect. Indeed, by paying for such information there may even be an increased distortion. Nevertheless, for certain types of personal information there is no other way than to approach the originator of the information. How else can opinions be obtained?

Future technology such as personal digital assistants (PDAs) or web avatars will make possible the capture and storage of detailed transaction information by data subjects. Access to these personal databases on an ongoing basis would provide the very best source of information possible. After all, disintermediation, cutting out the middleman, has long been one of the promises of the Internet. By going to the source, data users would no longer have to rely on retailers and market data bureaux for data that may have already been filtered or adulterated in inappropriate ways. The direct marketing industry standard response rate for unsolicited postal mailings is only 1%, i.e. 99 out of 100 junk mails are wasted. Any innovations that can help cut this cost would be very much welcomed.

2.3.5 The Particlized Market.

'Mass customisation', 'One to one marketing' and 'Particlized markets' are all business buzzwords highlighting the fact that markets are becoming increasingly fractured. Whereas once businesses could rely on there being no more than a few clear cut divisions in any market, there may now be hundreds. Popular music is a good example. Dance music used to be a single, clearly defined market segment, now it has splintered into a few dozen subdivisions - garage, trance, hard-core and so on. The same phenomenon is repeated across markets as diverse as computers and cars. Dell famously builds their PCs to individual customer specifications. BMW have started to the same thing with their cars. Wacker and Taylor in their book about the future of business [Wacker, 1997] state that mass marketing, pushing the same message to a large group of people, was appropriate to an era of mass production. But now, markets are being increasingly 'particalized' – broken down to smaller and smaller sized markets until there will only be single individual sized markets. Marketers will have to come to individuals for their information, that is the only way they will be able to get to know them sufficiently well to sell to them. Wacker and Taylor predict that consumers will be waiting for them, hands out.

2.3.6 Discussion.

The contention that organisations will be willing to pay compensation to consumers to obtain their information is a fundamental assumption of this dissertation. Arguments have been presented in support of the validity of this assumption. While none of the arguments may be wholly convincing on their own, collectively they represent at the very least a reasonable case. It is apparent that the benefits and advantages of the Internet bring with it the necessity to sacrifice some personal privacy to make full use of it. As Wacker and Taylor point out, to be willing to sacrifice some privacy is not to be willing to sacrifice it all. In privacy as in most things, the rarer it becomes the more people value it.

2.4 Requirement for a Solution.

If the assertions made so far are accepted, inevitable questions arise. How could such transactions be carried out? What mechanism or system would be necessary? How would it work?

If a manual method of operation is considered, immediate problems can be foreseen.

- How will initial contact made with someone whose contact details are not already available? This is the availability or cold calling problem.
- How will the actual information be transferred? A telephone conversation might be suitable for obtaining simple opinion information but how would detailed shopping pattern data be exchanged in conversation? This is the detail or granularity problem.
- What type of incentive will be used to induce the surrender of personal information or permission to use it? Will the incentive offered be sufficient to justify the time and effort to (a) make a decision about whether to accept or reject the incentive and (b) find and transfer the information? This is the assessment cost problem.

It is very difficult to imagine any manual mode of operation being acceptable to either buyer or seller of the information. The logistical problems are just too great to overcome. It is perhaps why much personal information is presently gathered from data agglomerators such as magazine subscription lists. Ideally, a solution should at least address the three issues mentioned above. That is, the solution should have the characteristics of:

Availability: The information should be optionally available to data buyers with which the data subject has no existing relationship or contact – facilitating cold callers.

Granularity: It must be possible to ask for and grant permission for very specific pieces of information and also to define precisely how it could be used. For example, a data subject may permit use of his email address for a once off commercial email but not permit transfer of his physical address to a third party. Distinctions may also be made by the sellers between different buyers. A data subject may be more likely to give their details to a charity organisation than to a direct marketing company.

Autonomy: The process should not require the involvement of humans as much as possible. Thus, inconvenience to the data subject is avoided, as they do not have to constantly assess what information they wish to reveal, how and to whom. A once off policy would be set that would then be applied to all requests on the data subject's behalf. This would also apply to the payment of incentives. From the data users (or collectors) point of view an automated method is far more efficient, faster and accurate.

What is being sought is an agent to represent data subjects in their transactions with data users. Wacker and Taylor refer to such an entity as a 'privacy agent' describing its task as being "To make sure you get paid for the information you give up about yourself." The use of the term 'Agent' here is coincidental to the typical computer science or AI meaning of the term. Petrie in [Petrie, 1996] and others refer to the fact that the label 'Agent' is overloaded for a variety of contexts. It is used here in the same sense as, say, a literary agent or real estate agent. That is, agent means someone/something that represents another. It is (mostly) coincidental that 'Agents', in the computer science sense, are used as part of the solution proposed in this dissertation.

3 Review of Agent Technologies.

"Knowledge always desires increase; it is like fire, which must first be kindled by some external agent, but which will afterward propagate it."
Johnson.

In this chapter, a review is made of the various technologies used in the application and its development. These include software agents, Java, agent development environments, IBM's Aglet technology and XML.

3.1 Software Agents.

3.1.1 Defining Agents.

Agent research became a 'hot topic' for computer science in the early 1990's. Linked with the hype surrounding all things connected to the Internet, 'agents' became a buzzword in the popular computing press around 1994. Agents were being touted as 'the next big thing' in computer science. A 1994 Ovum report entitled "Intelligent Agents: the New Revolution in Software" [Ovum, 1994] predicted that the market for agent software and products in the USA and Europe would be worth $3.9 billion by the year 2000. Obviously, this has not happened. Failing to live up to the hyperbole surrounding a research field does not mean the research field itself has failed. [Nwana, 1999] remarks that the field has clearly matured since this time, with the publication of certain key papers, the establishment of several annual conferences and workshops, the founding of agent standardisation initiatives such as the Foundation for Intelligent Physical Agents (FIPA) and the launching of dedicated Agent journals.

Even a cursory examination of the literature from the study of agents would reveal that it is traditional, if not in fact ritual, to refer to the lack of consensus agreement amongst researchers on what the generic term "agent" means. Some [Wooldridge, 1995] dismiss this as hardly mattering; others [Franklin, 1996] and [Bradshaw, 1997] extensively review, seeking to find common ground amongst the agent community. There is, however, little consensus amongst agent researchers about the nature of agents and agency. It may be useful to remember that there is no shared common definition of intelligence amongst the AI community, yet it continues to expand as a research field. Russell and Norvig make the insightful observation that "The notion of an agent is meant to be a tool for analysing systems, not an absolute characterization that divides the world into agents and non-agents" [Russell, 1995].

This dissertation therefore does not intend to present a full discussion on what an agent may or may not be. There are plenty of papers that fulfil that need, such as [Nwana, 1996]. For the purposes of this work the following definition has been adopted from [MEITA, 1998] "An independent software program that runs on behalf of a network user." In the context of this work, "selling" agents operate on behalf of the data subjects while "buying" agents act on behalf of the data users.

3.1.2 Issues Affecting All Agents.

Nwana and Ndumu in their rather gloomy review [Nwana, 1999] of the progress or otherwise of agent research identify a number of challenges that affect multi-agent systems but this analysis can be applied to all types of agents, whether 'intelligent', 'autonomous', 'mobile' or 'information retrieval'. They define the issues and give their assessment of the progress made in resolving them.

The Information Discovery Problem: Some method is required to discover what relevant agent resources exist and where they are located i.e. an agent equivalent of a domain name server. This would need to map resources to physical addresses in an automated fashion and, crucially, be kept up to date. Nwana & Ndumu maintain that there is little work addressing the general form of the problem within the context of a truly open environment.

The Communication Problem: Agents from different sources need to share a common communication language and protocol in order to inter-operate. General-purpose content languages like Knowledge Interchange Format (KIF) have been developed but the agent community, in Nwana & Ndumu's view, do not take them or initiatives like FIPA entirely seriously.

The Ontology Problem: Ontologies specify the vocabulary used in a particular domain. This allows two agents, used in a travel environment for example, to agree on a common definition of a flight, an airfare or a cancellation. Domain specific ontologies are being created but there is no standardisation between them – ontologies are not addressed in KQML or FIPA ACL. This is a key issue in the context of this dissertation. Although ignored by Nwana & Ndumu, XML (eXtensible Mark up Language) may be go someway towards addressing this issue. XML is discussed later.

The Legacy Software Problem: In order for agents to be useful in the real world they will have to interact with existing systems based on older technologies that possibly were never designed to inter-operate with other systems. Three potential approaches are suggested: Rewrite the old software, use a transducer to act as an interpreter between the agent and the legacy system or use a wrapper technique to augment the legacy system with code in order to enable it to

communicate with the agent. Their concern is that little effort is made towards automating this task.

The Reasoning and Co-ordination Problem: Agents that hope to carry out useful real world tasks such as the well known travel booking example needs to be able to reason, plan and satisfy constraints in order to carry out their tasks. Nwana & Ndumu believe this to be one area adequately tackled by current agent researchers.

The Monitoring Problem: All of the above problems crop up again when the issue of post-plan monitoring is considered. Agents need to be able to monitor the external environment, evaluate changes in that environment, determine if the changes are relevant to their situation and take appropriate action if they are. The lack of suitably rich ontologies means that while some domain and application specific approaches have been taken no general-purpose total solution is possible.

In an interesting aside, the papers' authors observe that the key ingredient in the successful agent projects they highlight is that in each case there were real world problems to be solved. They speculate that much of the current agent research does not address real problems. Furthermore, they opine that much research is too concerned with the 'theoretical' aspects of agents and not enough with solving actual problems.

3.2 Mobile Agents.

Mobile agents are defined by as agents that "Can travel to multiple locations in the network" [MEITA, 1998]. Alternatively, Nwana defines them as "Computational software processes capable of roaming wide area networks (WANs) such as the WWW, interacting with foreign hosts, gathering information on behalf of its owner and coming 'back home' having performed the duties set by its user." Essentially, the agent program can start execution at one location, suspend itself at an arbitrary point, transfer both its code and data and/or state to another location and resume execution from the same point in the code with the saved data and/or state, depending on whether weak or strong mobility is required.

As with general software agents, not all of what are claimed to be mobile agents are actually proper mobile agents. [Rothermel, 1997] created a taxonomy of mobility to distinguish between its different degrees.

Remote Execution / Code on Demand: This is where an agent program is transferred to a remote note before it is activated. This transfer only happens once and the code runs until termination. Code on demand (e.g. Java Applets) differs from remote execution in that the

destination initiates the transfer of the program code. This is really code mobility rather than agent mobility.

Weak Migration: This is where only the agent code and its data can be transferred at an arbitrary point in its execution.

Strong Migration: This is where the agent code, with its complete state can be transferred at an arbitrary point in its execution.

Strong migration, while very attractive from the programmer's point of view, is difficult to achieve as few languages can externalise state at a high level. Telescript and AgentTcl are able to do this. However consider capturing and transferring the complete agent state of a multi-threaded agent – this could be a very expensive operation. Hence weak migration, where only the data state is transferred, was developed. This transfers the onus to the programmer to ensure all the relevant execution state of the agent is made consistent and captured in the program's variables. Weak migration is the type of mobility used in many Agent Development Environments and Toolkits such as the Aglets Software Development Kit (ASDK) from IBM and Concordia from Mitsubishi.

3.2.1 Benefits of Mobile Agents.

What benefits does allowing processes to migrate give? Nwana gives some of the potential benefits of mobile agents including:

Reduced communication costs: For example it is much more efficient to transfer a small program to an image repository (such as satellite photographs) to filter and choose appropriate images rather than transferring all of the images back to base to pick them. This benefit is particularly relevant to the 'Soul Seller' application.

Support for local clients with limited resources: Clients with only a relatively small amount of memory and processing power, for example mobile devices such as cellular phones and PDAs, could make use of mobile agents for more complex operations beyond their local capabilities.

Asynchronous computing: The mobile agents can be initiated and perform their task while their owner does something else. They may operate when their home host is not even connected to the network. In the 'Soul Seller' application this allows information searches to be initiated, a period of time spent disconnected can pass and the results can still be collected when a connection is next made.

Flexible distributed computing architecture: A unique distributed computing architecture is provided by mobile agents.

Radical rethinking of the design process: Agent orientated design represents an opportunity to radically rethink the general design process.
In his analysis of mobile agents [Harrison, 1995] finds that while there is no individual advantage of mobile agents that cannot also be done by other means, there is no one technique that supplies all the advantages of mobile agents.

A further advantage not mentioned by Nwana or Harrison but experienced during the course of the project is increased security of information. This may seem paradoxical, given that security is often a key problem with mobile agents but in certain circumstances, it can be true. Confidential information is more secure if it does not have to travel over a network to be reviewed. The use of, properly vetted, mobile agents can ensure that such information does not leave its 'home base' while still being available for inspection. By inspecting the code of the agent, it can be verified that it returns to its sender with only the information it is allowed to have and nothing more.

3.2.2 Drawbacks of Mobile Agents.

In almost every analysis of mobile agent systems, the key drawback identified is security but this is by no means the only drawback. Rothermel identifies three main issues with mobile agents: agent security mechanisms, control structures and transactional support. [Wayner, 1995] lists the following major challenges: transportation, authentication, secrecy, security and agents spending their owners money to which Nwana adds performance issues and the problem of interoperability amongst differently created agents.

Rothermel identifies four areas where security issues can arise. These are (1) inter-agent security, (2) agent-host security, (3) inter-host security and (4) security between hosts and unauthorised third parties. Existing cryptographic techniques can be used in areas (1), (3) and (4) but area (2) is new and specific to mobile code systems. Security between agents and hosts is a two way street. Agent servers or hosts need to be protected from malicious or malfunctioning agents while agents themselves have to be protected from malicious or malfunctioning hosts. The first problem is usually addressed by what is known as the Sandbox security model – all potential dangerous procedure or method calls are restricted, similar to how security is implemented in Java Applets. Alternatively, server administrators could also choose to allow all operations but only accept agents from highly trusted sources. This approach obviously restricts the 'openness' of the agent system. Nor is it fully protective of the host, as a malfunctioning (as distinct from malicious) agent could still wreak havoc.

The second problem of protecting mobile agents from malicious hosts is a lot more interesting and is specific to mobile agents. [Hohl, 1998] identifies twelve separate types of attack that hosts could carry out. These are spying out code, data, control flow or the interaction with other agents, manipulation of code, data, control flow or the interaction with other agents, incorrect execution of code, masquerade attacks, denial of execution and returning wrong results of system calls issued by the agent.

Four research directions for resolving these problems are suggested by Rothermel:
The organisational approach: Only run agents on trusted hosts – such as other hosts on the organisation's intranet. This restricts the openness of the system.
The reputation approach: Only run agents on hosts with good reputations. This has the same problem with restricting openness as well as introducing a burden of maintaining reputation information.
The manipulation detection approach: Use mechanisms to detect manipulation of data or execution of code. This does not prevent 'spying' attacks.
Black-box protection approach: Use code obfuscating techniques to prevent the attacker seeing what is being done by the agent code. The agent would be protected but only for a certain time interval as the attacker would eventually analyse the code. After the expiration period, the agent and its data would become invalid.

[Hohl, 1998] adds a fifth possible approach, that of using mobile cryptography. Encrypted programs are programs that consist of operations on encrypted data. This has the advantage of not being time-limited (given sufficiently strong encryption) but it cannot be used in all programs as only polynomial and rational functions can be used with this technique. Ultimately, the problem of completely protecting agents from their host servers is intractable; hosts must always have the option to terminate the execution of an agent for which there is no remedy.

3.2.3 Java and Mobile Agent Development.

Mobile agents usually use an interpreted language for mobility and security reasons. While executable binaries could be used and would generally be more efficient, they would be tied to specific platforms. Java is widely adopted for writing mobile agent systems [Versteeg, 1998] not only because it is an interpreted language but also because it provides many of the facilities necessary for building mobile agent systems. These include object serialisation, networking libraries, security model, multithreading, remote method invocation (RMI) and its support of applets. Applets are quite similar in concept to mobile agents – the key differences being that applets can only migrate from servers to browsers and do not carry their state with them.

- **Object Serialisation**: This allows an object to be written to a serialised stream and for a serialised stream to be read into an object. This means for example an object could be written to a file, the file could be transferred to another machine and the object recreated from the file. This is obviously a very useful mechanism for agent migration. One drawback of the Java Virtual Machine architecture is that it does not allow access to programs to directly access or manipulate the execution states for security reasons. This means that the agent heap cannot be serialised and hence strong migration, as mentioned previously, is not possible.
- **Multithreading:** Java not allows the use of multithreaded programs but supports them with built in synchronisation primitives. Each agent can therefore run in its own process or thread which is essential for allowing an agent to execute autonomously of any other agents at the same location.
- **Security**: Java's security is one of the key features that has made it so successful. The Java Security Manager mechanism allows untrusted programs, such as applets to be executed safely. For each application, it is possible to create a customised, very fine-grained security policy using the Security Manager. It allows differentiation between individual operations so that, for example, an application may be allowed to accept an incoming socket requests but could not make its own outgoing socket request. For dealing with untrusted agents migrating to a host, this feature is essential.

Other programming languages such as Telescript or AgentTcl provide more facilities specifically aimed at agent development such as support for strong migration and better resource control. Java, however, remains the language of choice for most agent development, possibly due to greater number of developers already with Java experience.

3.3 Agent Development Environments.

A number of Agent Development Environments (ADE) and toolkits are now available that make the creation and deployment of mobile agents a great deal easier. Versteeg evaluates four such systems; IBM's Aglets, ObjectSpace's Voyager, Mitsubishi's Concordia and General Magic's Odyssey. Other toolkits available include Panasonic's Beegent and Plangent, the FIPA backed Grasshopper system and Stanford University's Java Agent Template (JATlite).

With the exception of JATlite, most ADEs or agent toolkits have several features in common. JATlite does not provide inbuilt abilities to migrate agents from one host to another. The remaining ADEs all provide some kind of agent server that hosts the mobile agents. The server

executes the agents in a security-controlled sandbox. The toolkits provide different API classes (which therefore make the agents from one toolkit incompatible with those from another) containing some base class from which all mobile agents are derived. This base class provides a common interface to all mobile agents. Agent migration is also similar amongst the toolkits. A method is supplied that when invoked causes the agent to migrate to another host. As only weak migration is supported, methods are provided that can be called just before migration or after arrival. Therefore the agents can have their states saved before migration and can be initialised again when it arrives. The toolkits also provide several mechanisms for the agents to communicate with each other. Messages can be synchronous, asynchronous, multicast and future reply, though not all toolkits support all of them.

In the choice of which ADE to use, Aglets was chosen as the most suitable. Voyager and Odyssey at the time of development were no longer being supported by their creators. Beegent and Plangent had only recently been introduced and hence there was little research available about them. Concordia was also considered but IBM's Aglets toolkit was chosen as it had the simplest API that still covered all the essential features such as mobility and messaging. It also has good supporting tools and documentation as well as a small but active development community.

3.4 Aglets.

3.4.1 Aglet Background.

Danny B. Lange created the concept of 'Aglets' while working as a visiting scientist as IBM's Tokyo Research Laboratory [Lange, 1998]. Aglets are Java objects that can migrate from one Aglet-enabled host to another Aglet-enabled host on a network. An Aglet Software Development Kit (ASDK) was available as a free download from IBM's Aglets site [IBM, 2000]. IBM announced in August 2000 that the Aglet source code would henceforth be available under an Open Source type license. The ASDK consists of the Application Programming Interface, documentation, example code and an Aglet server called Tahiti. Version 1.0.3 was used in the development of the 'Soul Seller' application. Although a more up to date version was available, it had a time-limited license that was due to expire during the course of development.

One drawback of using Aglets was that only Java version 1.1.x could be used and not version 1.2 or above. The developers of Aglets announced they intend to support Java 1.2 and above, eventually but did not indicate when this would happen.

3.4.2 The Aglet Environment.

The Lange book gives a thorough overview of the Aglet development environment. An 'Aglet' is a Java program that inherits from the *Aglet class*. The Aglet class provides methods that allow an Aglet to carry out actions such as dispatching itself to another location, sending and receiving messages to and from other Aglets and disposing of itself. The Aglet model that underlies the Aglet API relies on several abstractions. Aglets run on an Aglet server. A server is a process engine of which several can run on a single machine or network node. The Aglet server can host several contexts. A context is a stationary object that provides a uniform execution environment for one or more Aglets. Each Aglet has its own proxy, which both shields the Aglet from direct access of its public methods and provides location transparency for the Aglet. An Aglet has a globally unique identifier. The identifier cannot change during the lifetime of the Aglet.

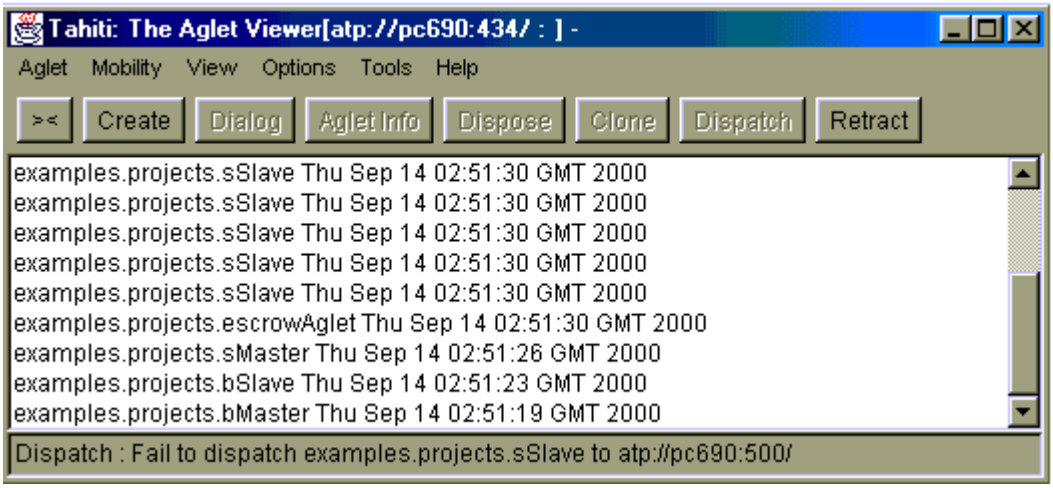


Figure 3.1 The Tahiti Aglet Server.

Three more elements are necessary to fully understand Aglets: Aglet Operations, Aglet Events and Aglet Messages. Aglet operations consist of creation, cloning, dispatching, retraction, activation/deactivation and disposal. These are self-explanatory. The Aglet event relies on listeners to fire particular actions once an event is caught. The three kinds of listeners are mobility listeners (catches changes in Aglet location), persistence listeners (catches changes in the activation/deactivation state of the Aglet) and clone listeners (catches cloning events). All of these listeners can be customised to carry out various types of action depending on the event being caught. Lastly, Aglet messages can be synchronous, asynchronous, multi- or single cast, require a reply or not. Messages are not sent to Aglets themselves (which can change address) but to their proxy (which cannot). Location transparency is thus provided. Messages can be of user-defined kinds, so that they can be distinguished from each other. They may carry a simple

atomic argument, such as a Boolean or an integer or an argument table such as a key-value pair hashtable. The Aglet messages model while simple to use, is sophisticated. It also provides a message prioritisation facility.

3.4.3 Aglet Security Model.

Aglet servers allow execution of Aglet code from their own local code bases as well as those from other Aglet servers that have migrated to the server. All the potential mobile agent problems discussed previously also apply to Aglets. The Aglet security model supports security policies, which relate permissions (what can and cannot be done) to the principals of the system.

The notion of a principal is important. In the general definition, a principal is any entity whose identity may be authenticated by any system the principal may try to access. Specifically to Aglets a principal is a program, person, company or other legal entity that originated, controls or is responsible for some constituent part of the overall Aglet system. Karjoth, Lange and Oshima [Karjoth, 1997] list the principals in the Aglet security model:
Aglet: An instantiation of an Aglet program.
Aglet Manufacturer: The author of the Aglet program. Maybe a human, a company or similar.
Aglet Owner: The initiator of the Aglet program, the person or company that launched the program.
Context: The interpreter that executes the Aglets.
Context Manufacturer: The author of the context program or product that the Aglet is running in.
Context Master: The owner or administrator of the context program or product.
Domain Authority: The owner or administrator of the domain or group of contexts.

Each principal can define their own security policy, but the key policies are defined by the Aglet owner and the context master. These define respectively what can be done by the specific Aglet anywhere and what can be done by any Aglet within in the specific context. There is a hierarchy of security policies so that an Aglet owner can be overridden by the context master and the context master by the domain authority and so on. Hence, in a conflict between Aglet and context policies the context master's policy would prevail. For example an Aglet's security policy may allow it to have only read and write access to all files in a certain directory, but the context policy may further restrict it to just reading HTML files.

Permissions define what actions, such as read, write, create and so on, can be carried out on specific resources. The Aglet API has several types of permissions:

File permission: Aglets can be granted or denied read or write access to the local file system, whether named files or entire directories.
Network permission: Aglets can be granted or denied access to network resources. For example, an Aglet may be allowed to connect to certain hosts but not others. A blacklist of known rogue hosts could thus be maintained.
AWT permission: Defines whether Aglets can open a window or not.
Context permission: Defines whether methods provided by the context can be invoked. For example, an Aglet might be allowed to create (or dispatch or clone) some types of Aglet but not others.
Aglet permission: Defines what methods can be carried out on an Aglet. For example an Aglet can be given permission to carry out the dispose() method on another Aglet owned by a named principal, but may have no permission to interfere with other Aglets.
System permission: This was proposed as part of the Aglet security model but has not yet been implemented in any available ASDK. It defines permissions for systems resources such as memory or CPU usage.

The security of the Aglets used in the 'Soul Seller' application is obviously a key issue. It can be seen that the level of security control offered by the Aglet system is fine grained. Security policies are easily implemented by allocating permissions to principals. All of the Aglets owned by person A might be granted full file access right while those from an another, untrusted or unknown person may only be granted read rights. It is unfortunate that it was not possible to implement the system permissions proposed as this could prevent 'denial of service' type attacks where a large number of Aglets are dispatched to a server and overwhelm its resources. The reason for this is that Java provides no way of limiting access to processor or memory resources to certain objects or threads.

3.4.4 Aglet Patterns.

Lange states in his book on Aglets that one of the goals in creating the Aglet API was that it would be lightweight, adding that it was tempting to call it the RISC of mobile agents. While this has benefits in terms of ease of learning and efficiency it does mean some omissions have been made such as the lack of built-in support for any agent communication languages such as ACL. A developer working with Aglets has to do a certain amount of wheel reinventing. Lange aimed to counteract some of these omissions using design patterns, several of which are implemented in the ASDK code samples or as code accompanying the book.

Originating from object-orientated design, design patterns take solutions to frequently occurring software design problems and try to reapply and reuse them in new applications. In the context of agent and specifically Aglet applications, design patterns are particularly useful. Three general types of patterns are described: travelling, task and interaction. Of the task patterns the most useful for the Soul Seller application was the Master-Slave pattern. In this pattern, a master Aglet is created whose purpose is to delegate tasks to the slave Aglets it creates. The slave Aglet can then (optionally) change location, carry out the task and return the result to its master. This pattern was found to be a particularly useful method of handling the agents required to represent buyers and sellers in the application.

3.5 XML.

The eXtensible Mark-up Language (XML) is a standard originally proposed by the World Wide Web consortium. It is not a language in itself but a language for creating languages. It is actually a subset of the Standard General Mark-up Language (SGML). It is designed to make it easy to exchange structured documents over the Internet. The format of XML documents is thoroughly discussed in [Bryan, 1997].

The significance of XML is that it allows documents to be not just machine-readable but machine understandable. A domain's terms and documents can be standardised so that entities can exchange information in a correct format without ever having to agree on anything other than the appropriate Document Type Definition to use. For the purposes of this dissertation, the exchange of personal profile information can only occur if both buyer and seller of the information use a both a common language for negotiation and exchange standard formatted documents. XML was not ultimately used in the implementation of the application although a servlet was designed that could generate buyer and seller version HTML forms from a common XML document. It was also considered as the format that data entered from the HTML forms would be stored in. The problem was that only Java version 1.1.8 could be used with Aglets while Java's support for XML did not begin until version 1.2 and 1.3.

With the exception of XML, knowledge all of the topics covered in this chapter was necessary in developing the Soul Seller application. The next chapter will show how all these different components were put together to create the system.

4 Design.

"Pessimists have already begun to worry about what is going to replace automation."
John Tudor.

In this chapter the design phase of the project is discussed. The problem is reviewed, broad solution alternatives are identified and analysed. The design process is discussed. The chosen solution is then examined. Finally, detailed design decisions are discussed and addressed.

4.1 Problem Review.

An application is required that will satisfy the criteria identified in section 2.4. The application is necessarily divided into two broad segments, a selling part that caters for data subjects and a buying part that caters for data users. The application must be capable of satisfying the needs of both types of user. Their needs are outlined next.

4.1.1 Requirements of Data Subjects.

Data subjects require the application to allow them to enter information about themselves into a repository, from which information can then be sold to data users. The application should be capable of handling arbitrary amounts and types of information. Data subjects should be able to specify the selling price of individual pieces of information. They should also be able to set usage policy regarding how the information can be used and by whom. The application should continue to operate even when data subjects are connected to it. Once the information is entered into the system, the data subject should not have to be involved in any sales decisions – the application can autonomously decide whether or not to sell and at what price. The data subjects should be able to review what transactions have been carried out with their data. It should be possible to add to, delete from and edit information in the repository. Only purchased information should be revealed to any third party, security is essential.

4.1.2 Requirements of Data Users.

The application should allow data users to specify both the type of person they wish to buy from and the information that they wish to buy. Searches should be flexible enough to include arbitrary numbers of criteria or desired information fields. Search criteria should allow comparison operators such as equals, greater than, less than and range values. While initiation

of searches can only be performed while the data user or buyer is connected to the system, it should be possible to continue searches while disconnected with the answers being made available when connection is next restored. It should be possible to rate the importance of both search criteria and desired information fields. The data user should be able to stipulate the desired number of responses ('souls') as well as the budget available for doing so. The application should attempt to obtain the best compromise between the quantity, quality and price of the information. The exchange of money for information should be secure and consistent, there should be no possibility of reneging. The potential broad solutions to these requirements are now discussed.

4.1.3 Browser Embedded Standard Profile Solution.

Standard profile systems include the now defunct Open Profiling Standard (OPS) or the proposed Consumer Profile Exchange (CPEX). Information about the user is held in a standard format by the users browser, which can then be passed to websites when the user visits them. The systems were primarily designed to allow the easy exchange of data between user and site, including data about the privacy preferences of the user. The benefit to the user for surrendering this information is that (a) the amount of form filling would be minimised (b) the site could be customised to their preferences and (c) their privacy preferences would be respected. A solution based on standard profile data has both advantages and disadvantages. On the plus side having both data producers and consumers adhere to a common standard when exchanging data makes sense. Also allowing users to set a usage policy about their data is also a welcome innovation. However, a browser-based system does not address the issue of availability – when a user is not connected, their information cannot be accessed. The motivation of browser-based systems is not to motivate users to give information about themselves but merely to make it easier for them to do so. There is hence no facility to directly compensate or motivate users for giving up additional information or permission to use the information. Finally, by definition, a fixed standard format for information makes it more difficult to exchange non-standard or arbitrary additional data. This solution was consequently rejected.

4.1.4 Web / Database Solution.

A combined web and database solution would allow the entry of the data subject's information into a central database via a web page. Data users could then query the database from the web and pay for any information they consequently use. The advantages of such an approach are that the information would be persistently available, that buyers and sellers need not have any prior contact and that a variety of arbitrary data could be held if an object database was used. The

disadvantages include inflexibility of database structure if a standard relational database is used and the problem of buyers not being able to sample the goods before buying. Buying information is different from normal transactions in that until the buyer knows what the information is, the buyer does not know if he wishes to buy it. However, if the buyer already knows what the information is, then there is no need to buy it! By allowing buyers the choice of whether to buy or not after they have seen the data, the risk is of buyers abusing the system. Conversely, if buyers are not able to view the data before buying, buyers may be unwilling to buy the information.

4.1.5 Web / Agent Based Solution.

An agent in this context is similar to an object. It contains both data and the methods to use the data. Agents can have the additional characteristics of being autonomous (capable of deciding its own actions without intervention) and mobile (capable of moving through out a network). A web / agent based solution shares many similarities with the web / database solution. Data subjects can enter their information via web pages, which is then stored by a type of autonomous agent. Data users also create their searches via a web page and the search query is also represented by an agent but one of a different type. The agents are designed to carry out transactions when they encounter each other. The agent approach has the same advantages as the web / database solution but with the additional benefit of code mobility which solves the 'try before buy' problem. By allowing logic to be executed remotely, the quality of information can be verified before a commitment to buy is made, without compromising the secrecy of the information if a purchase is not made. In addition, bandwidth can be saved if a lot of data is involved. Rather than transferring all of the data back and forth between servers, an agent is transferred which can choose which data records to obtain. The disadvantage of the agent approach is there is a potential scalability problem. If each buying agent must interact with each selling agent, the number of interactions grows exponentially as the number of agents increases.

4.1.6 Choice of Solution.

The browser embedded standard profile solution was rejected as being unsuitable because it did not meet the set criteria of availability or granularity. The choice was therefore between the database and agent solutions. There was not a great deal to choose between the two solutions, both met all of the criteria required. In the end, the fact that using agents was a relatively novel method of developing an application was the deciding factor. This decision is discussed in more detail in Chapter 6.

4.2 Design Process.

Once the broad type of solution was chosen, the method of designing the overall system and its component parts had to be considered. The scope of the application was much more open ended than would be normally typical, for a number of reasons:

1) The problem being addressed was anticipated rather than already existing. This meant that often the problem itself was changed by the choice of options available.
2) The application was novel and developed as a proof-of-concept for this problem domain. Certain assumptions about the problem domain, such as that about the availability of digital cash, could thus be made, as they were not key to the concept being demonstrated. Similarly, aspects of the system that would be critical in a commercial application but not central to an academic work, were not addressed.
3) The use of software agents as a basis for development meant that there was greater uncertainty as to what could be accomplished. The capabilities of the chosen Agent Development Environment were not fully known. Hence, the inability of agents to handle data that could not be serialised put an obvious restriction on the system resulting in the design having to be changed to accommodate it.

These factors combined to influence significantly the design and development process. Normal design methodologies such as those using Use Cases and UML were not appropriate to use for such an open-ended system. In order to give a structure to the project an initial basic scenario of how the system might work was created to illuminate the design issues. This was then refined as the system was developed. Developing a scenario allowed the mandatory and optional features of the agents to be revealed. In addition, by assigning roles to different types of agent such as a soul buyer or a soul seller, it was possible to think in terms of the agent's actions rather than the underlying objects, data or methods. In later planning, several dimensions along which the system could be developed were also identified. These dimensions (content handling, agent mobility, monetary logic and added features) helped to explore what additional functionality could be built into the application, given the time available.

4.3 Design Outline.

The general design approach adopted was for all transaction processing to be done centrally, while all data input to and output from the system was decentralised. The design allows input and output from clients of any one of many Internet connected servers. Where possible the same design is used for both the selling and buying parts of the system. Individual Aglets are created that contain each seller's supplied information (their 'Soul') or each buyer's search criteria. The

Aglets of both types travel to a single Aglet server (the 'Marketplace') where transactions can take place between selling Aglets and buying Aglets.

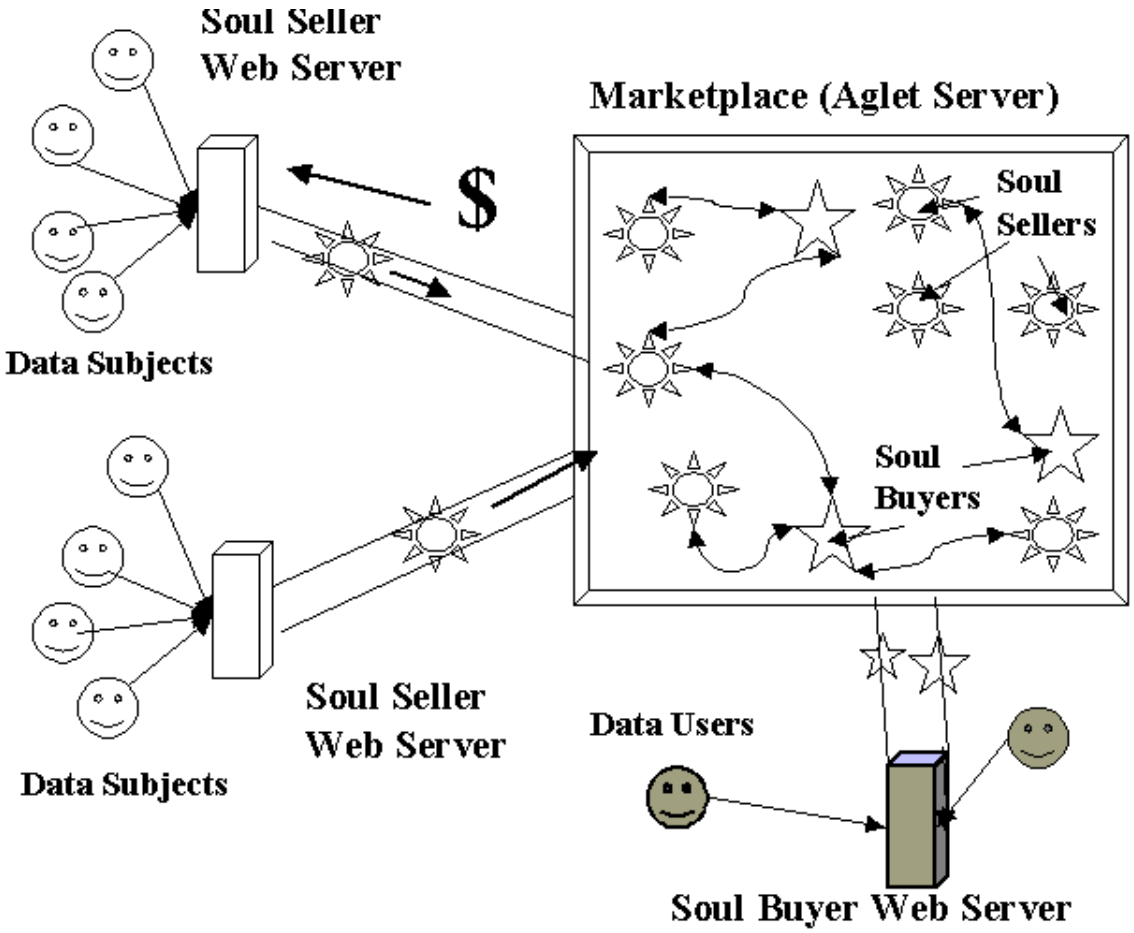


Figure 4.1 Design Diagram of the System

The interface for both buyers and sellers is web based. Using HTML forms is the easiest way to input information into the system. While it is desirable that generated information (such as web site history files or a list of transactions from a Java Wallet) could be included it is not an essential feature. JavaScript is used to ensure the form input is verified at the client side before being processed. Web pages are also used for displaying the results of transactions. The information from the HTML forms is handled by Java servlets. These servlets check the input and produce an object that contains all of the passed data. Different types of objects are produced by the selling and buying sides of the system – called sInfo and bInfo respectively. Both type of object can handle arbitrary amounts and type of data by storing all information in vectors and hashtables. The objects are then serialised and stored in a directory accessible to the Aglet programs. The structure of the objects is shown in Figure 4.2.

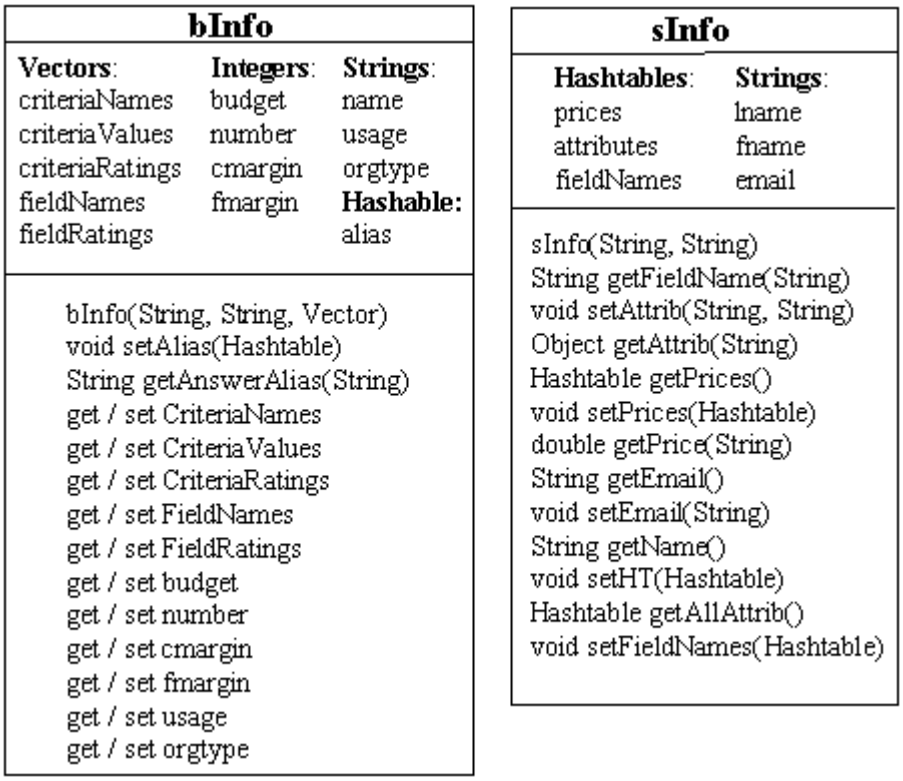


Figure 4.2 Object descriptions of the object passed to the buying and selling Aglets, respectively.

The Aglets used in the system are based on the Master / Slave pattern described by Lange and Oshima in [Lange, 1998]. Code by Todd Papaioannou implementing the pattern, available from the Aglets Portal web site [Papaioannou, 2000] is used as a basis for the Aglets. The Master Aglet, of which there is one per server, scans the directory for the addition of any new serialised objects by the servlets. When a new object is found, the Master Aglet creates a new Slave Aglet and passes the object to it. This object corresponds to the Aglet's 'mission'. The new slave Aglet then dispatches itself to the marketplace to carry out its orders. Up to this point, buyer and seller Aglets behave in the same way. When they reach the marketplace, seller Aglets wait for conversations to be initiated by visiting buying Aglets. Seller Aglets persistently remain at the marketplace while buyer Aglets visit and then leave when their transactions are completed.

The details of each conversation between buyer and seller are stored by both sides in a separate conversation object. All Aglets maintain a vector of conversation objects, meaning that multiple conversations can be conducted simultaneously. Conversations are identified by concatenating the Aglet identifiers of both parties to the conversation. Once a transaction is completed, the sales Aglet sends notification back to its master Aglet via a message that contains an object. A buyer Aglet does not send back any information to its master until all of its conversations are

completed. At this point, it sends a message back requesting for it to be retracted. Once it arrives back at its home server, it delivers all of the information it has bought to its master. Both the buying and selling masters store the information returned as files. Users can invoke servlets via a web page that displays the contents of the files as web pages. In this way, information about the transactions is returned to the user.

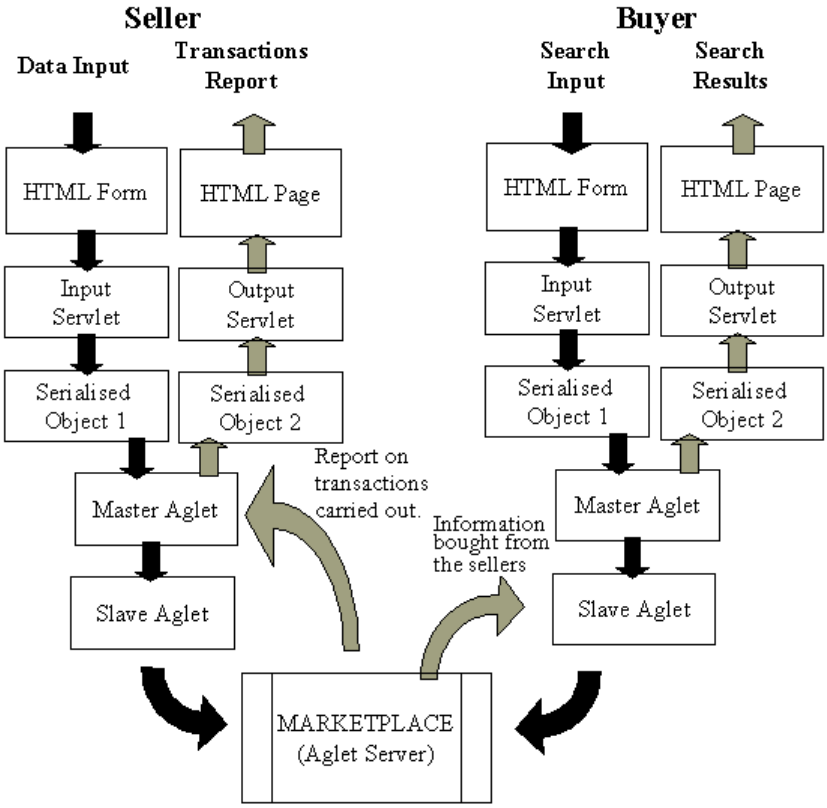


Figure 4.3 How information flows to, from and between buyers and sellers.

4.4 Design decisions.

Over the course of designing and implementing the system, many minor decisions and judgements about various issues were made. Some of the more significant decisions are discussed below.

4.4.1.1 One or multiple marketplaces.

The decision was made to only handle a single marketplace. This was done primarily for reasons of simplicity but also as this is how it is believed such a system would really operate in a commercial environment. With a single marketplace, all information in the system can be held in one place – avoiding the need to make decisions with only partial information. In the situation where there are multiple marketplaces, a buying agent would have to visit all the

different marketplaces twice, once to collect the information (such as prices and quality) and once to carry out the transaction. Agents do not need to maintain an itinerary of destinations if there is only one destination.

However, a single marketplace does suffer from several disadvantages. There is an inherent scaling problem with carrying out all transactions on a single server. There is a single point of failure. From an economic point of view, the absence of competition between marketplaces would reduce pressure to continually improve standards. As a single marketplace by definition must cope with all participants, there would be no opportunity to specialise in certain types of souls. This drawback is related to the scaling issue, because the ability to identify a cluster of similar souls makes the searching process more efficient. For example, souls containing data about cars might be all on the same server, making the search for car information simpler.

4.4.1.2 Serialised objects.

Serialised objects are used to transfer information from the servlets to the master Aglets and from the master to the slave Aglets. This was partly because all information the Aglets handled had to be capable of being serialised and partly because it is easier to deal with unformatted information using objects. The alternative option would have been to try to store the information in structured files or a relational database.

4.4.1.3 Information Representation.

A key issue was whether to hard code one or more data format standards into the system or to remain neutral, allowing the possibility of multiple data formats. There are several acknowledged standards such as the previously mentioned OPS or CPEX standards that could be used. Alternatively, the creation of a data format specially for the application was also considered.

It is much simpler to write code that just caters for one or more known data formats but it would be less useful. Given the nature of the problem being addressed, many data formats, as yet unknown or uncreated, may be applicable. For this reason a format neutral approach was taken. This required that the both the servlets dealing with the data input and the Aglets representing the users could handle arbitrary amounts and types of data. The data structure is imposed at the web page level by varying the number, type and name of the form elements. The web forms must adhere to certain standards such as always including fields that will identify the buyer or

seller or starting form element names in a defined way. It must be ensured that the web forms for both the sellers and buyers are consistent with each other.

One way to ensure consistency between seller data and buyer searches would be to derive both from a common source such as an XML document that defines the information format. This would require a servlet to create the seller's input form and the buyer's criteria search form (both HTML pages), from a Document Type Definition (DTD). This would make the application fully flexible but unfortunately, due to time constraints it must be listed in the future work section, rather than amongst those implemented.

4.4.1.4 Transaction repudiation.

If buyer and seller deal directly with each other for the entire transaction with no 'objective' third party, there is a risk that a rogue buyer or seller could try to renege. For example if a buyer makes an agreement to buy some information in return for some payment either party could try to renege by not fulfilling their part of the bargain, the buyer may not send payment or the seller may not send the information. For this reason an 'escrow' Aglet was created. At the outset, both parties must agree on the use and choice of the escrow Aglet. When the negotiations reach the final stage where information and money is about to be exchanged, both parties send their information or money to the escrow Aglet instead of directly to each other. When it has received both the money and the information, it forwards the money to the seller and the information to the buyer. In this way, agent fraud is prevented. This is also useful if there is a failure by either side to the transaction. It does require that the escrow agent is fully trusted by both buyer and seller. It should therefore be independent of both.

4.4.1.5 Enforcement of usage policies.

When the seller provides some information to a buyer, the price settled on depends on what usage the buyer is going to make of it. It would be expected, for example, that a lower price would be charged for a buyer that only wishes to use the information once than one who will use it repeatedly. A sender of spam could purchase the right to use an email address once and then bombard it with unwanted bulk email. How this usage policy is enforced is a key issue for sellers. A couple of methods for addressing this problem are discussed in the Future Work section of Chapter 6.

4.4.1.6 Honesty and / or Accuracy of Seller Information.

One of the problems with asking people for non-factual information such as opinions is that the very act of asking may change the answers they give, the aforementioned Hawthorne Effect. This problem is compounded when incentives are offered to obtain the information, as this raises the issue of information being given solely to obtain the reward. Some dishonest users may try to register many fictitious identities in the hope of gaining rewards for doing so. This is an important issue as even a small proportion of people doing this would damage the credibility of the system with data users.

Fortunately, several methods and approaches can be taken to discourage such practices. Log in passwords could be mailed to the postal address given at registration. Identities could be verified by the use of credit card numbers or bank account details. Duplicates of anyone already registered would not be allowed. Nevertheless, it is very difficult to adequately address this problem within the application itself. There is thus no attempt made to verify the information given by the seller within the application.

4.4.1.7 Potential Problem with Buyers.

There is a potential issue with buyers trying to obtain more information than they have paid for by abusing the system. For example by obtaining criteria information and then terminating the transaction, no payment would be due. To counter this the buying Aglet deletes any information not actually purchased before returning to its home server. The system as it is does not allow Aglets to be modified but it would be desirable to eventually relax this restriction. When this happens, it could not be guaranteed that the same practice would continue. This is problem is one example of the more general Aglet verification and security problem discussed in Chapter 3.

4.4.1.8 Coping with Failure.

The time at the initiation of every conversation between agents is noted. All conversations are expected to reach a terminal state (acceptance or rejection) within a certain period (which can altered by the local system administrator). If the conversations do not reach certain points (such as receiving a price message) within that time, they are deemed to have failed and are abandoned. When this happens the conversation object for that conversation is flagged as being deleted.

4.4.1.9 Exchanging Money between Aglets.

The application does not provide any real money exchange mechanisms. Electronic money would be an ideal solution but incorporating one of the available digital cash systems was beyond the scope of the project. Instead ‘cash strings’ are transferred between buyers and sellers as representatives of what would actually be used if a real digital money scheme was implemented.

5 Implementation.

"The goal of Computer Science is to build something that will last at least until we've finished building it."
Anonymous

In this chapter, how the constituent parts of the system were implemented is discussed in detail. The chapter is divided logically into three areas of functionality: the Soul Seller, the Soul Buyer and the Marketplace. Finally, the development environment used is described..

5.1 Soul Seller.

This module covers the entry of information by the data subject into the system, its conversion into a selling slave Aglet (a 'soul seller') and the subsequent recording and display of all transactions involving the data subject. The information flow is from a HTML web form to a Java servlet where an object, the data subject's 'Soul', is created and serialised. This object is then deserialised by a master Aglet, which passes the information to a created slave Aglet. This slave Aglet or soul seller dispatches itself to the marketplace. As transactions are carried out, information is sent back to the master Aglet, which makes the information available to a Java servlet, which can be called from a HTML web form.

In the Soul Seller module, there are several web pages, servlets and Aglets. These are numbered and listed here. Each element is referred to in the text by a number in brackets.

1) HTML input form – *sss.htm*
2) Java servlet – *storeDetails.java* which creates and stores the sInfo object.
3) Serialised object – based on *sInfo.java*, saved as a file with its name ending in '.sss'
4) Aglet – *sMaster.java*
5) Soul Seller Aglet – *sSlave.java*
6) Serialised object – the record of results are saved as a file with its name ending in '.ssr'
7) Java servlet – *getSResults.java*, which returns information as a HTML page
8) HTML form – *sss_results.htm*

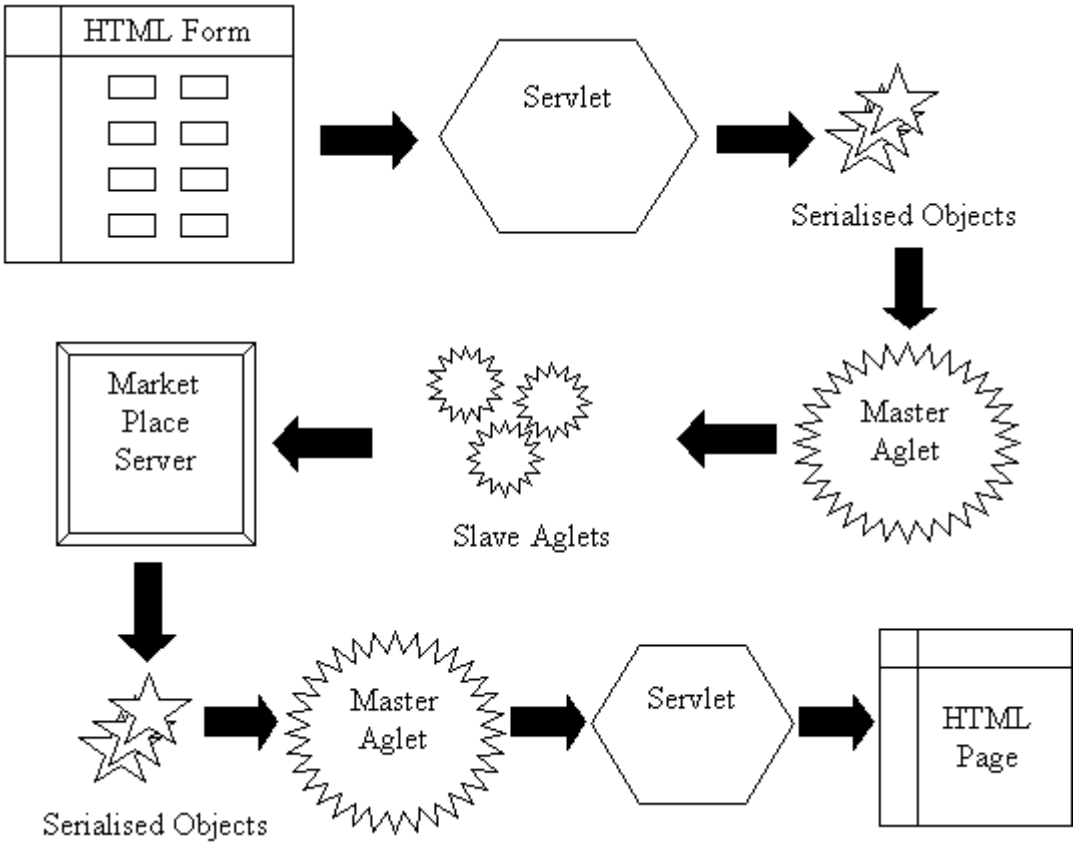


Figure 5.1 Cycle of control through the system.

All data subject information is initially input into the system via HTML web forms (1). This data is one of four types. The first type is the data necessary to identify and contact the data subject, data such as name, address and email address. These fields are mandatory. The second type of data is the answers to system specified questions such as “Are you employed?” or “Do you have a mobile phone?” The user can answer as many or as few of these as he desires. The third type of data are free form, user defined labels and answers. For example, the user may choose to give details of his hobbies or sports preferences. The fourth type of data is the prices set for all the other types of information. Users can set whatever price they like for each individual piece of information. They can also set pricing policy to determine what multiple or fraction of the standard price is charged depending on how the information is used. The standard price is presumed to be for a commercial organisation using the information internally and for a once off purpose, for example a single mail shot. Users can choose to increase or decrease the price depending on whether the information is used multiple times, by other organisations or by non-commercial organisations such as charities.

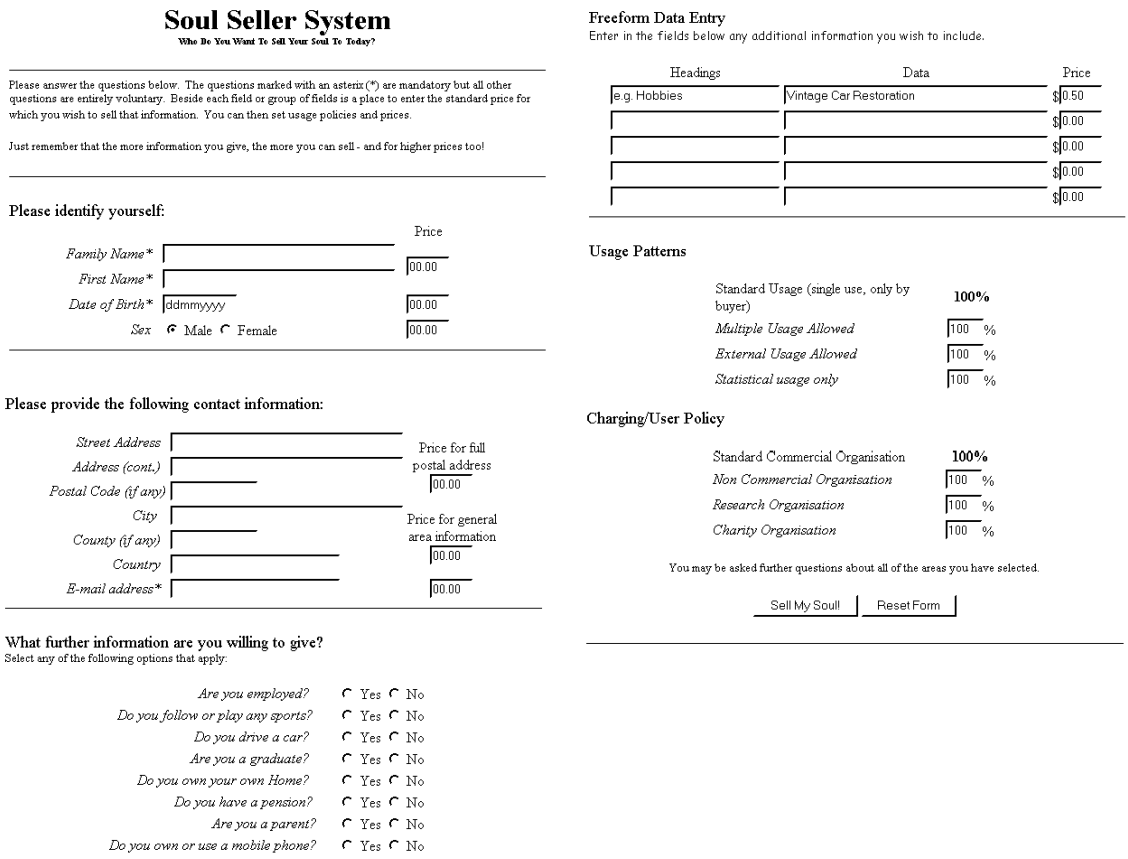


Figure 5.2 Screenshot of the Soul Seller Information Entry web page

Once all of the required and voluntary information has been input, the information is checked and processed by a Java servlet (2) when the submit button is pressed. This Java servlet resides on the same server as the web server but could reside anywhere. The purpose of the servlet is to check the incoming data for completeness (although some client side checking is done using JavaScript) and to create an instantiation of the sInfo object.. This object is then serialised (3) and stored in a place where the sMaster Aglet, the selling master Aglet (4), checks for new objects regularly.

The sInfo object contains all of the information necessary for a sSlave Aglet (5) to carry out its task. It carries the name and all other information, pricing and policy data from the HTML form as well as the methods to properly access and change it. The sMaster Aglet resides on the Aglet server running on the same machine as the HTTP server, though it too could run anywhere. When this detects a new serialised object it deserialises the object and recreates the sInfo object. It creates a new sSlave Aglet and passes it the sInfo object as well as the destination address of the marketplace Aglet server as arguments Aglet.

When the new Aglet is created, it checks where it is and if it is at its home location, it dispatches itself to the Aglet server at the destination address passed to it. What happens when it moves to this new address is covered in the Marketplace module. Its parent, the sMaster Aglet, has functions other than just creating sSlave Aglets. When it is first activated, it creates an Escrow Aglet that is dispatched to the marketplace Aglet server to act as a trusted go-between for the selling and buying Aglets. When a transaction is completed using the Escrow Aglet, it forwards details of the transactions to the sMaster Aglet, which stores the transaction persistently (6). When the original data subject wishes to see what transaction have been carried out on their behalf they invoke another Java servlet (7) that gathers this information and displays all the details as a web page (8).

5.2 Soul Buyer.

This module covers the entry of the buyers' criteria into the system, the conversion of this information into a buying Aglet and the subsequent return and display of the gathered information. It is very similar in composition to the Soul Seller module, containing web pages, Java servlets and Aglets. These are listed here and are referred to in the system walk through by the numbers in brackets.

1. HTML input forms – *sbs.htm*
2. Java Servlet – *storeCriteria.java*
3. Serialised object – based on *bInfo.java* stored as a file with its name ending as '.sbs'
4. Aglet – *bMaster.java*
5. Soul Buyer Aglet – *bSlave.java*
6. Serialised object – the results are saved as a file with its name ending as '.sbr'
7. Java servlet – *getBResults.java*, which displays the information as a web page.
8. HTML form – *sbs_results.htm*

There are three parts to the HTML form where all the details of the soul search are entered. The criteria part allows the data user to specify what type of person is desirable from which to gather information. For example, the ideal person for a particular search might be an employed male who has children and a car but no pension. The data buyer is able to set how important each criterion is in a scale from Mandatory, to Very Important, Important and Nice to Have. In addition, an overall tolerance level can be set so the degree of match could vary from very close to much looser. The second part of the form allows the data user to choose which information is to be bought.

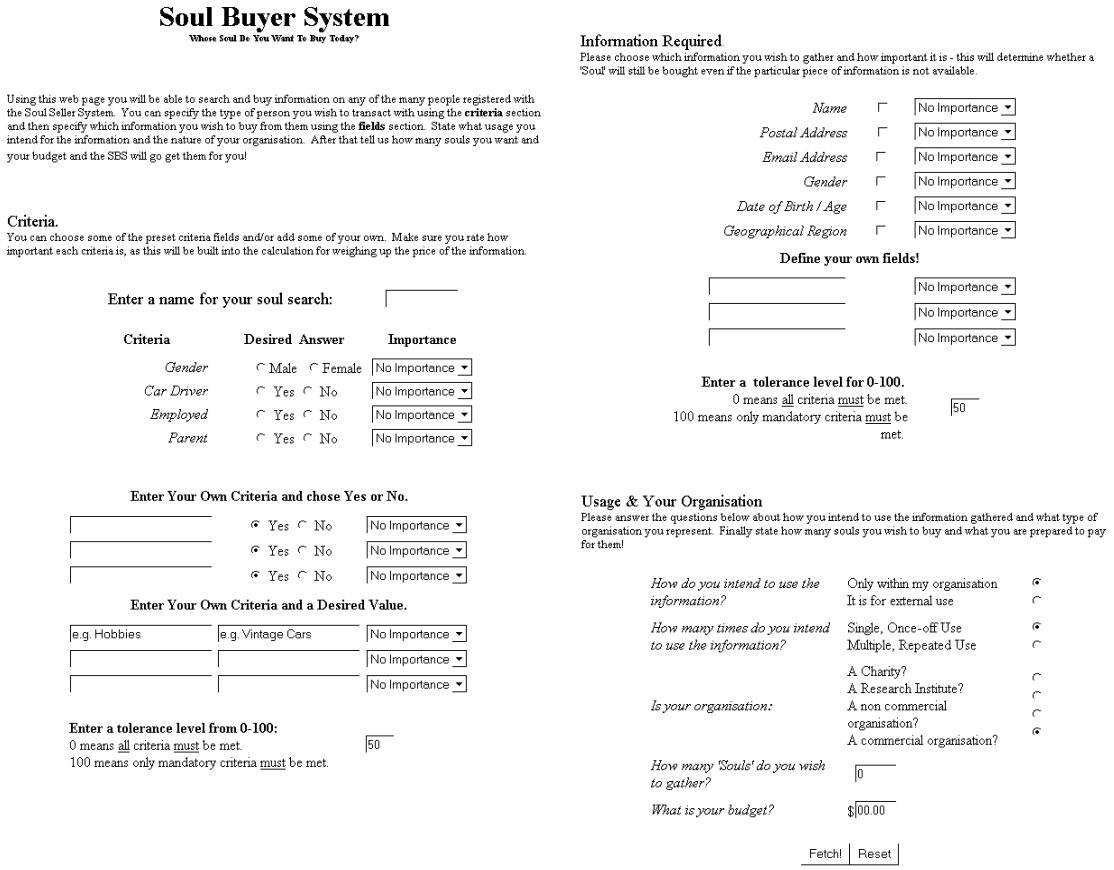
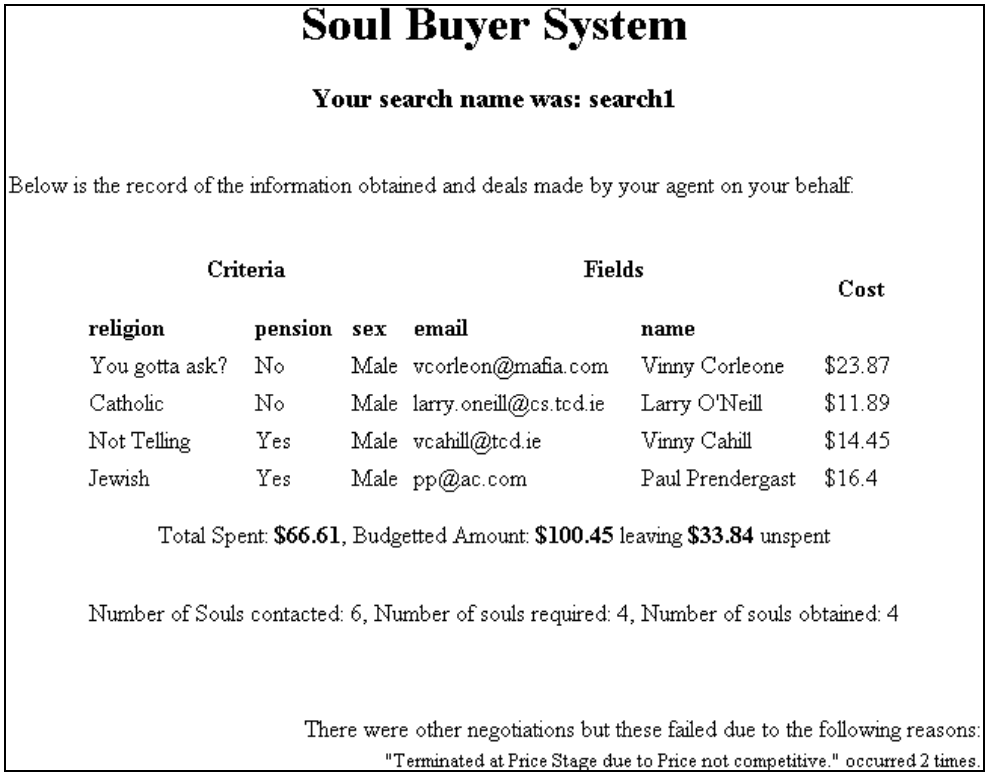


Figure 5.3 Screenshot of the Soul Buyer Search Entry web page

Some fields such as those for name, address or email address can be selected using a tick box, others require a field name to be entered as free form text. In the same way as for criteria, individual fields are assigned ratings, and an overall tolerance level can be set.
The last part of the form asks the data buyer to indicate how the information is going to be used. They must indicate if the information obtained will be used only internally or externally, singly or multiple times and whether by a commercial or charitable organisation. Finally, the desired number of souls is requested along with the budget allowable for buying them.

Once all of the Soul Buying information is entered and the 'Submit' button is clicked, a servlet (2) is invoked which instantiates an instance of the bInfo object (3) and stores the data from the form in it. The bInfo contains all of the 'mission' information necessary for the Soul Buyer Aglet such as the required criteria, fields, ratings and usage policy as well as the budget and number of souls to be bought. The servlet then serialises the object as a file with an extension that would cause it to be picked up by the bMaster Aglet (4). This Aglet scans the directory where such objects are saved for new additions. When a new object is created, the Aglet deserialises it and passes the object to a newly created bSlave Aglet (5).

When the Soul Buyer Aglet is created, it first checks where it is and if it is at home, it dispatches itself to the marketplace. The details of what it does at the Marketplace are covered in the next module but once it has completed its business there, it makes a request back to the bMaster Aglet to retract it back to base. When it arrives back, it gives all of the information it has gathered to the bMaster Aglet and disposes of itself. The bMaster Aglet saves this information in a file that can be read by the Java servlet (6). The servlet can be invoked from a web page when the data user wishes to see what information has been bought on his behalf. The servlet displays this as a web page and gives details of how much has been spent and how much of the original budget is left.



Soul Buyer System

Your search name was: search1

Below is the record of the information obtained and deals made by your agent on your behalf.

Criteria		Fields			Cost
religion	**pension**	**sex**	**email**	**name**	
You gotta ask?	No	Male	vcorleon@mafia.com	Vinny Corleone	$23.87
Catholic	No	Male	larry.oneill@cs.tcd.ie	Larry O'Neill	$11.89
Not Telling	Yes	Male	vcahill@tcd.ie	Vinny Cahill	$14.45
Jewish	Yes	Male	pp@ac.com	Paul Prendergast	$16.4

Total Spent: **$66.61**, Budgetted Amount: **$100.45** leaving **$33.84** unspent

Number of Souls contacted: 6, Number of souls required: 4, Number of souls obtained: 4

There were other negotiations but these failed due to the following reasons:
"Terminated at Price Stage due to Price not competitive." occurred 2 times.

Figure 5.4 Screenshot of the web page returning results to a data user.

5.3 Marketplace.

The marketplace module covers what happens when the Soul Buyer and Soul Seller Aglets are brought together on the same Aglet server. The main components, the Escrow, bSlave and sSlave Aglets, were all briefly introduced in the previous sections. They are dealt with in more detail here. In addition, a new object called a Conversation object is used as well. The marketplace itself is the standard Aglet server, called Tahiti, which is supplied with the Aglets

Software Development Kit (ASDK). It allows Aglets with arbitrary code to migrate to and execute on it. It provides security restrictions that only allow certain types of instructions to run.

In the development of the system, all Aglets being used were created by the developer and hence known not to be malicious. Security could be left relaxed and open. For example, all Aglets on all servers were allowed to both read and write to all local files. This would not be permissible if the system was implemented in a real situation. Security policies were set by assigning privileges (such as file access) to either certain named Aglets or by creating a general 'trusted' or 'untrusted' divide, with trusted Aglets being granted greater rights. This was done within the Tahiti server.

The Aglets used in the system are all based on the Master-Slave pattern as described in [Lange, 1998]. This is a scheme where a stationary master Aglet can delegate a task to a mobile slave Aglet. Both the sSlave and bSlave classes inherit from an abstract, general Slave class. They share methods to listen for messages, to handle invalid Messages and to request retraction, amongst others. Both type of Aglets are created by their master Aglets and are passed an object, which contains their operating instructions. They both then dispatch themselves to the marketplace, the Tahiti server. The selling Aglet does not do anything further until it is contacted by a buying Aglet. The buying Aglet first obtains a list of all other Aglets on the server. If there are no selling Aglets to deal with it deactivates itself for a while, periodically waking to recheck. Eventually if there are still no suitable Aglets available, it returns to its home server and disposes of itself. However, if some suitable Aglets are found, a conversation is initialised with them, which follows the pattern shown in Figure 5.5.

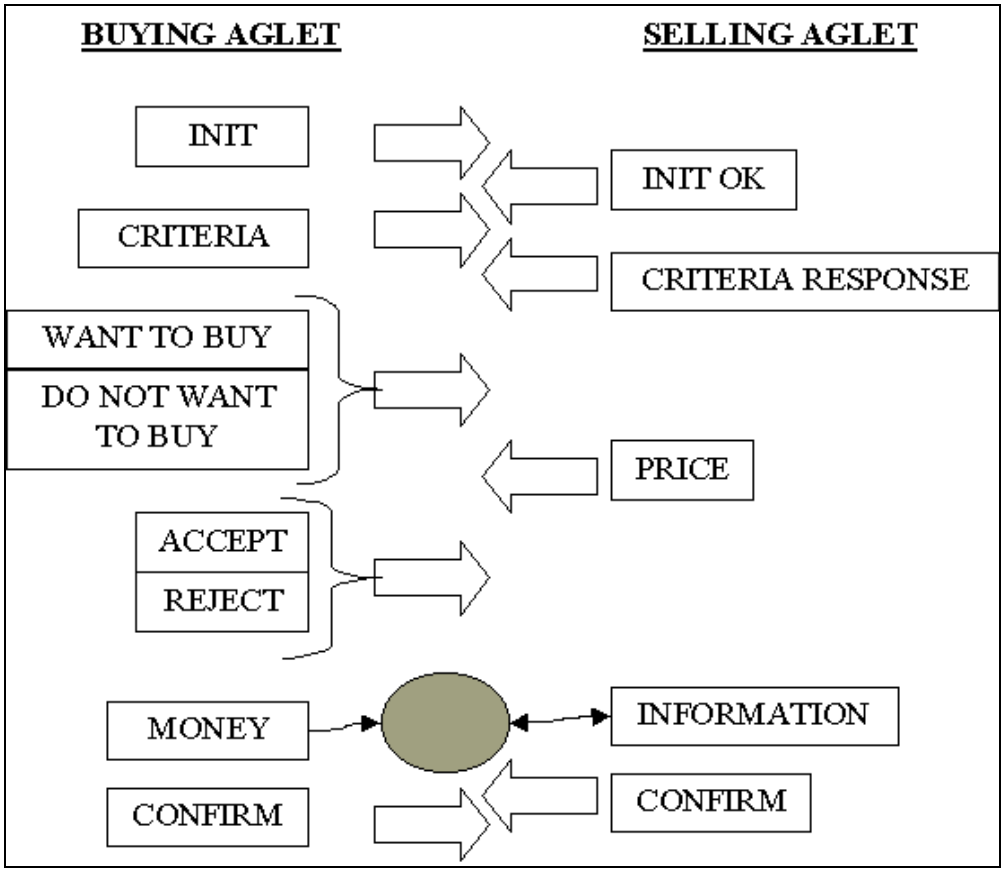


Figure 5.5 Conversation pattern between buyer and seller Aglets.

5.3.1 Conversation Process.

The buyer is the initiator of all conversations so the conversation process is described here from its point of view. The first thing a soul buyer Aglet does when it arrives at the marketplace is to make a list of all the soul seller Aglets present in the same context. It then initiates conversations with all of them by sending each of them an INIT message . The selling Aglets should respond with an INIT OK reply message. At this stage, a new conversation object is instantiated by both Aglets. A unique identifier is required for each buyer-seller pair. This is created by combining the identities of both the buying and selling Aglets, which are themselves unique, into a conversation ID which is held by the conversation object. This conversation ID is then sent with every message between the bbuyers and sellers. Each buyer and seller maintains a record of all conversations they are involved in, in a vector of conversation objects called the conversationRecord.

The soul buyer Aglet, once it receives the INIT OK message, sends its CRITERIA message to the responding soul seller Aglets. This contains a vector of field names, for example [gender, city, employed, pension], that it will use to determine if the soul is the type it wishes to purchase.

The seller Aglet should respond with a CRITERIA RESPONSE message. This message contains a vector of responses to the criteria, for example [male, Dublin, Yes, Unknown]. The buying Aglet goes through a process, described in section 5.3.2 that determines if the responses returned by each selling Aglet are of a sufficiently good match with what it requires, to continue the conversation.

All of the soul sellers that meet the criteria requirements are sent a “WANT TO BUY” message. Those that do not are sent a “DO NOT WANT TO BUY” message and these conversations are then discontinued. The WANT TO BUY message includes a vector of the field names that the buying Aglet wishes to purchase, such as [name, email, salary, hobbies]. The response, a “PRICE” message from the soul seller, indicates which of those fields requested are available from the selling Aglet, for example [true, true, false, true]. It also states how much this information will cost.

When the Soul Buyer receives a PRICE message, it first checks if the appropriate number of fields are available to buy. A similar exercise as for rating the criteria response is used to determine if sufficient information is available to warrant purchasing. If this is not the case, a REJECT message is sent to the Soul Seller. If this is the case, the soul buyer must then decide which of the eligible souls to purchase. No decision about this is made until all of the conversations in which the buying Aglet is involved with, have reached this point. The buyer then carries out the process, described in section 5.3.4, of choosing which of the suitable souls to purchase . The chosen sellers are sent an ACCEPT message while the rest are sent a REJECT message and the conversations with these Aglets are discontinued.

At this stage, both the buyer and the selected sellers are ready to complete the transaction. The seller has made an offer of information at a certain price and the buyer has indicated that this is an acceptable price. To complete the transaction, the seller Aglet must send an INFORMATION message containing the agreed data to the buyer, while the buyer must send a MONEY message containing a string representing the appropriate amount of digital money to the seller. For reasons explained in section 5.3.5, the messages are sent via a third party Aglet called the escrow Aglet. The escrow Aglet does not immediately forward on the INFORMATION or MONEY messages to their proper destinations, the soul buyer and soul seller Aglets respectively. Only once it has received both messages does it forward them on. Both parties then send a message to the escrow Aglet confirming receipt. The transaction between that particular buyer-seller pair is then completed, though neither may have completed their transactions with other Aglets.

When the buying Aglet has completed all of its conversations, whether successfully or unsuccessfully it sends a message to its master Aglet asking it to retract it. None of the buying Aglets are retracted until all of the outstanding Aglets from the same master request retraction, at which point all of them are retracted. The buying master Aglet can then start sending any new soul buyers to the marketplace that have been created in the meantime. When the soul buying Aglets return home they each send a message to the master Aglet with the details of all the transactions each has carried out. Each then disposes of itself.

It is worth noting that both types of Aglet can carry on multiple conversations simultaneously, so that each conversation is interleaved with others. This is the reason for the use of the conversation object, to record the details of each conversation. For the soul buyer each stage of the conversation process is designed to filter out unwanted souls sellers so that the complicated process of choosing which souls to buy is limited only to those eligible. Discontinued conversations have a flag marked as false in their conversation object.

5.3.2 The Rating / Scoring Process.

The same process is applied to both the criteria responses and the information field responses. When the Soul Buyer web page is used to create a new search, there is a requirement to select the importance ratings of each of the criteria or fields chosen. Each of these ratings is assigned a value that will subsequently be used to calculate a figure or score that represents how close a match a given answer or response is to what is desired. The scores are based to the following figures.

Mandatory - 1000
Very Important – 100
Important - 50
Nice to have – 20
Not important - 0

In addition, an overall tolerance level is set for all of the criteria and all of the fields. When the criteria or information response vector is received back from the soul seller, each response given is compared in turn with the desired answer. If the two match the figure corresponding to the rating for that answer is added to a running score. When all answers have been dealt with the score is compared with the total possible if all answers were as required. If the average difference between them is greater than the tolerance level specified by the data user, which is a number between 1 – 100, then the soul seller is accepted, otherwise it is accepted. The ratio of the score to the total possible is then stored as an indicator of quality of fit between what is

desired and what is available. This quality indicator is used later when choosing which of the eligible souls should be bought.

5.3.3 Calculating the price.

When the soul seller receives a WANT TO BUY message, it includes a vector of the fieldnames that the soul buyer wishes to buy. Not all of the desired fields may be available so a response vector is created indicating which of the fields are offered. The price of each of these available fields is then totalled. To this figure, a random amount between 0 and 1 dollar for each criterion is added, as such information should not be given freely. A random amount is used to help obscure the actual price of each field, so to prevent the individual cherry picking of fields. It also helps in varying the souls chosen by buyers so that the same souls are being sold repeatedly.

The final figure that will be returned with the PRICE message is calculated by applying the pricing multipliers in regard to how the information is to be used and by whom. This may increase or decrease the price depending on the situation. A wider usage policy for example, would typically carry a higher price. A data subject may have chosen to charge charities only 50% of the normal price. The price returned reflects these policies. A data subject may use this mechanism to effectively prevent certain usages of his data by assigning them a prohibitively high multiplier.

5.3.4 Choosing which Souls to Buy.

Once all souls have been defined as either acceptable or unacceptable, a decision must be made about which of the acceptable souls are to be purchased. Different approaches are taken depending on the situation. Briefly, if the soul buyer has a large budget relative to the price of souls for sale, the souls are chosen in order of quality, regardless of price. If the budget is small relative to the price of the souls available, they are chosen in order of lowest cost first. If the budget available is somewhere between these extremes then souls are chosen in order of the ratio of their quality to their price. The exact meaning of what a ‘large’ or ‘small’ budget is determined by the average cost of the souls compared to the total budget per soul required.

The problem of choosing the optimal selection of souls to buy, given several variables of price, budget, quality and also time is intractable. The number of calculations required to work out the optimal mix is a combinatorial function that explodes in value as the number of candidates increases. This method is by no means the most efficient or effective, but it gives a reasonable

performance in a variety of situations for a low overhead. One drawback to this method is that it requires all information to be available before any decision can be made.

5.3.5 The Escrow Aglet.

There is a large issue of trust between the buyer and seller aglets. Each is relying upon the other to fulfil their side of the agreement when it comes to exchanging money for information. If these messages were to be exchanged directly and simultaneously with each other, both would be left exposed to the possibility of the other reneging. For example, the soul seller could send its information but never receive payment, while the soul buyer could send the money but never receive the information. So, a trusted third party, the Escrow Aglet, which is impartial to both sides is introduced to solve this problem.

The escrow Aglet holds messages sent to it until both halves of the message pair are received. Both messages are checked to ensure the details match. For example, the string representing the digital money should correspond to the price agreed, the information should contain the required number of fields. If these tally, both messages are sent on to their proper destinations. The escrow aglet has a further purpose – it sends details of the completed transaction back to the Master sales Aglet to allow it to keep records of all the transactions carried out by the slave Aglets it created.

5.4 Implementation and Development Environment.

In the development of the application, some compromises were necessary to reconcile the target implementation environment (i.e. a heterogeneous network with multiple servers) with a viable development environment. The implementation of the application was carried out using an Apache Web Server version 1.3.9 that had the Apache JServ servlet engine version 1.0 add-in. The Aglet server, Tahiti version 1.0 revision 7 was used to host all of the Aglets. Several Aglet servers ran on the same machine, distinguished by their use of different port numbers. Aglets 'migrating' from server to server never actually left the same machine but the process involved was the same as if they had. Both Netscape and M.S. Internet Explorer were used to test the web pages and servlets. All program code was written in Java version 1.1.8 using the EditPlus text editor version 1.25 on a Windows 98 PC.

6 Evaluation, Future Work and Conclusions.

"These days, the wages of sin depend on what kind of deal you make with the devil."
Kara Vichko.

6.1 Evaluation.

There were four objectives for this work. The first was to establish that there was a requirement for the application developed. It is hoped that this was achieved by the arguments made in this dissertation.

The second objective was to create an application that fulfilled the requirements discussed in the problem domain. That is, to allow consumers to sell their information directly to data users. This required that there be a great deal of flexibility in the type and amount of data capable of being handled by the application. Data users could literally want information about any area related to the data subject. The application went some way towards allowing this flexibility, although it provided quite limited comparison test capabilities.

The Soul Seller application was developed exclusively using Java in a PC environment. It was necessary to mimic a multiple node network using just one machine by giving the various servers in use different port numbers. The application was successfully developed and demonstrated. Although much more development would be necessary to make it suitable for commercial use, the concept of its use was confirmed.

The third objective was to create an information representation structure that can store and process a great diversity of data formats and types. This was only partially achieved. Because extensible data structures such as Vectors and Hashtables were used, both of which can store objects, most types of data could be stored. However, only very limited types of data can be queried or compared. For example, criteria can only be compared as whether they equalled a single value. It is not possible to set criteria using multiple values, ranges, greater than or less than operators. This obviously restricts the flexibility of the system.

The fourth objective was to test the suitability of agent technology, in this case, IBM's Aglets, for the application. Having implemented the system only using aglets for the transaction processing module, it is difficult to see how such an exclusively agent based solution would be viable. There are specific limitations also imposed by the Aglet environment. It was found, for example, that messages between Aglets could not be larger than 64K although this was not

documented anywhere. The number of Aglets it was possible to run on a single Aglet server was also limited, depending on the size of the Aglet but a practical limit of around 20 was found for Aglets with class file sizes of around 14k. More memory would be utilised when carrying out multiple transactions.

Although it is a function of the application design there is an inherent scalability problem in requiring every buying Aglet to initiate a conversation with every selling Aglet. This results in an exponential rise in the number of conversations as the number of Aglets increases. This was evident from problems with the system slowing down, experienced even when the number of Aglets on the server was below twenty. Although having multiple marketplaces (or Aglet servers) hosting the Aglets would speed up searching as it could be done in parallel by several buying Aglets working on the same search simultaneously, the same number of transactions would occur. It was therefore concluded that using Aglets exclusively was problematic. If just the selling Aglets were replaced instead by a database that held the information of all of the data subjects, the scaling problem would be much lessened. This form of solution would retain the benefits of using the buying Aglets such as the reduced communication overhead and improved information security.

A further benefit from using Aglets was that the design process was made easier. Using Aglets allows the designer to think in quite natural terms when working on the high level design, as the agent is almost thought of as a human. This anthropomorphic quality of the agents was particularly apt as the agents were actually meant to be representing humans.

6.2 Future Work.

The problem the application was envisaged to solve is a large and complex one. The relatively simple application presented here only addresses the basic issues of automated negotiated valued information exchange. There are a considerable number of improvements and innovations that could be carried out to make the system more useful, useable and reliable. These are discussed here.

6.2.1 Allow External Aglet Development.

The Aglets developed for the application are only one of many possible Aglets that could be used in the system. While it raises obvious security issues, allowing users to define their own buying or selling Aglets would have several benefits. Firstly, users could satisfy themselves as to the impartiality, openness and security of the system. Secondly, greater functionality and

improved performance would inevitably result when a community of developers is involved compared to that achieved by a single developer.

In order to permit this, vastly improved marketplace services, beyond the provision of just an escrow Aglet, would be required. A formal definition of the mandatory standards for both types of Aglet would also be necessary to allow interoperability between Aglets. Finally a process for the testing or validation of externally Aglets would be required to protect the marketplace server and other Aglets running on it. This could be done via more fine-grained implementation of security policies, for example by reserving certain privileges for system Aglets.

6.2.2 Generic DTD Reader.

The data formats carried by the Aglets, whether buyers or seller, are determined by the HTML forms that are used to input the information. As mentioned in Chapter 4, there is a requirement to automatically generate HTML forms from Document Type Definitions (DTDs). Essentially a Java Servlet would take the DTD and construct both a Soul Seller information entry web page and the corresponding Soul Buyer query construction web page, both complete with the mandatory fields. The generated web pages would need to be compatible with the application's existing servlets and Aglets as well as each other.

6.2.3 Improved Client Functionality.

6.2.3.1 Information Editing.

There should be a facility to allow information already provided to the seller Aglet to be edited and updated. In addition, it should be possible to include arbitrary documents and files with the object passed to the representative Aglet.

6.2.3.2 Ongoing and Automated Provision of Information.

Allow links with other applications (such as browsers) that could collect information about the user in an non-invasive way, which could then be subsequently sold via the Soul Seller system.

6.2.3.3 One (or *n*) Shot Email Addresses.

These email addresses only forward the first *n* emails received (depending on how many 'rights' to email were purchased) to the sellers real email address. Once this number is reached, all subsequent email is bounced back to the sender. The temporary email addresses could be generated by the marketplace in a similar way to how unique reference identifiers (perhaps based

on the real email address) are generated. This would require the marketplace to run a mail server with a database that recorded which generated addresses matched which real email addresses and also kept count of how many times each generated address had been used. This would help alleviate consumer fears of their email addresses falling into the hands of senders of unsolicited commercial email (UCE) or spam as it is more commonly known.

This application need not necessarily be tied in with the Soul Seller application as it is likely that there is a requirement for this sort of application for general use.

6.2.3.4 More Flexible Price Negotiation.

Allow buyer agents to reformulate their field requirements when the prices initially returned are more than it can afford. The least necessary fields could be dropped and the query resubmitted.

6.2.3.5 More Sophisticated Pattern Matching.

At present the system, when matching criteria between buyer and seller, can only handle simple equals or not equals tests. The range of tests available should be expanded to include greater than or less than and multiple selection tests. It should be possible to get the same flexibility as SQL offers when querying databases.

6.2.4 Improved Marketplace Services.

There is a great deal of scope for the marketplace Aglet server to add more value in the process.

6.2.4.1 Transaction Tracking.

Provision of a formal transaction tracking service would enhance the reliability and trustworthiness of the system.

6.2.4.2 Usage Policy Enforcement.

One way to ensure buyers keep to the usage agreement would be to try to detect violators. This could be done by sending ‘dummy’ souls along with the genuine ones to the buyer Aglet. The dummy souls would contain email and contact details that are under surveillance by the marketplace. If the agreed usage policy is breached, for example by sending multiple emails when only one had been paid for, the email address for the dummy would also receive one. The errant buyer would then be caught. This could be achieved by making use of the escrow Aglet to hold information for a buyer until all of the individual profiles purchased (or a certain amount of them) are in the escrow Aglet’s possession. When the information is to be passed on, a dummy profile is added to the real ones.

6.2.4.3 Reputation Facility.

Much of the system is based on trust so any feature that enhances trust would be welcome. Individual buying and selling agents could have their own reputations based on previous transactions with them. Agents might choose to only transact with agents of known good repute thereby motivating in-spec behaviour. This facility would be made more necessary if users were allowed to edit existing or create their own Aglet code to represent them. A reputation could also be applied to authors of Aglets, where the agents of trusted authors are given greater privileges than those of unknown or untrusted ones.

6.2.4.4 Multiple Marketplaces.

The scalability issue could be alleviated by using multiple marketplace servers. If the selling Aglets were dispersed over a number of servers, parallel searches could be performed by multiple buyer Aglets.

6.3 Conclusions.

From the analysis presented of the commercial trends and technological developments, it is believed that there is a good opportunity to bring data subjects and data users into contact with each other in a way never before possible. This gives the data subjects the advantage of allowing them to profit from the use of their information while still retaining control over how that information is used, in short the benefits of connectivity without the loss of control over their privacy. It gives data users the advantage of allowing them to gather a level of detail about consumers never previously possible without huge expense. It facilitates truly meeting consumer needs on a one to one basis.

It is technological advances that make such a relationship possible in theory but only an actual developed application will make it possible in reality. The result of this work presented here represents a prototype of how such an application might work. It has confirmed that the major issues to be addressed are the data representation and scaling problems. Some contribution has been made in addressing problems such as trust between buyers and sellers, pricing mechanisms and facilitating users in pricing and rating individual fields and criteria.

The application developed used agents exclusively for its transaction processing. While this had some significant attractions, the flaws that resulted meant that developing the application just using agents is not a viable option. This is not to rule out the involvement of agents altogether, just that they should be combined with other types of technologies such as databases so as to still gain their positive features while avoiding their drawbacks. The flexibility that the use of agents

gave in the choice of matching criteria would be difficult to duplicate using simple data base queries.

The use of agents allowed the design process to be approached in a very natural way. It may be that using agents in prototype applications is a good fit. Agent systems provide easy to use but low performance migration and messaging facilities that removes the onus from developers to address these bread and butter issues in the prototype. This allows them to concentrate instead on solutions to some of the issues unique to the problem domain. Once the prototype is designed, the insight into these problems is gained. This can be then used in the development of the real system where performance, scalability, reliability and all the other normal characteristics of good development can be achieved.

The specific Agent Development Environment used, IBM's Aglets, was found to be simple, easy to learn and easy to use. It does lack some features such as the ability to restrict resource use and support for strong migration. However, the problems experienced with the Soul Seller application did not result from these deficiencies but rather from inherent design flaws.

The future work section of the dissertation provides some interesting suggestions for development, not all of which are necessarily tied to the application discussed. The one-shot (or n-shot) email system could be a useful application in its own right.

Glossary

Aglet Server: A host machine that allows agents to migrate to and execute on it. Servers implement security rules to prevent agents misbehaving. Servers are akin to the 'market' where seller agents and buyer agents meet.

Criteria: The features of the data subject that are used in selecting or rejecting them as information providing candidates. For example, their level of income, ownership of a car or an address in a particular area.

Data Subject: A person whose personal information is available for sale through their seller agent.

Data User: An organisation that wishes to obtain information about one or more data subjects, which they do through their buyer agents.

Information: Data about the data subject, which is purchased by the buying agent. For example, name, address, email address, interests – may include some of the criteria used in selecting them.

Marketplace: The main Aglet server where Soul Buyers and Sellers reside.

Master Aglet. The Aglet program that resides on the local server. Its function is to deal with requests for new slave Aglets (whether Soul Buyers or Sellers) and then oversee the deployment of them

Organisation Policy: A seller agent may deal differently with a buyer agent depending on who or what they are representing. For example, an agent from a charity may get more access to information for less cost than a normal commercial organisation.

Price: Dollars are used as the nominal unit of currency for all transactions. Individual data elements (name, address etc.) are priced by the data subject. When a buyer agent requests a set of information about the data subject (e.g. all contact details), the total price for this information is calculated, taking into account the policies and usage permission (see below) involved.

Slave Aglet: An Aglet program whose function is to represent a single Soul Buyer or Seller. They are created by Master Aglets and then dispatched to the Marketplace.

Souls: All the information about a single data subject that has been passed to a Soul Seller (selling Aglet). Data users specify how large a population size they wish to purchase by indicating how many Souls they wish to buy.

Soul Seller: The Aglet that represents the data subject. It possesses information about the data subject that it will divulge to a buyer agent if the price it requests is met.

Soul Buyer: The Aglet that represents the data user. It has a profile/criteria of data subjects about which it wishes to buy information. If the price of the information is sufficiently low, it will buy it.

Usage Permission: The actions that a data user is allowed to do with information obtained from a data subject are specified by the usage permissions. These include for example, use once and destroy, use multiple times, sell or pass on to others.

Bibliography

Andersen, (2000). Privacy Challenges E-biz, Even Without New Laws. **2000**.

Bradshaw, J. (1997). Software Agents. Cambridge, MA, MIT Press.

Bryan, M., (1997). An Introduction to the Extensible Mark-up Language - XML., SGML Centre. **2000**.

D'Arcy, D., (1995). Mine your own data. Computer Scope. **1995**.

European-Parliament (1995). "Directive 95/46/EC." Official Journal **L**(281): 31-50.

Franklin, S. G., A. (1996). Is it an Agent, or just a Program: A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages., Springer-Verlag.

Harrison, C. G. C., D. M. ; Kershenbaum, A., (1995). Mobile Agents: Are they a good idea? New York, IBM T. J. Watson Research Centre.

Hohl, F. (1998). A Model of Attacks of Malicious Hosts Against Mobile Agents. 4th Workshop on Mobile Object Systems (MOS'98): Secure Internet Mobile Computations.

IBM, (2000). Aglets, IBM Japan. **2000**.

Karjoth, L., Oshima (1997). "A Security Model for Aglets." IEEE Internet Computing **1**(4): 68-77.

Lange, M. O. D. B. (1998). Programming and Deploying Java Mobile Agents with Aglets, Addison-Wesley.

Lumeria, (2000). Fair Information Principles. **2000**.

MEITA, (1998). Mobile Agent Computing. H. S. L. Mitsubishi Electric ITA.

Nwana, H. N., Divine (1999). "A Perspective on Software Agents Research." The Knowledge Engineering Review(January).

Nwana, H. S. (1996). "Software Agents: An Overview." Knowledge Engineering Review **11**(3): 1-40.

Ovum, (1994). Intelligent Agents: The New Revolution in Software. London, Ovum Publications.

Papaioannou, T., Master / Slave Pattern. **2000**. http://www.aglets.org/downloads

Petrie, C. (1996). "Agent-Based Engineering, the Web, and Intelligence." IEEE Expert **1996**(December).

Pitofsky, C. R., (2000). Privacy Online: Fair Information Practices In the Electronic Marketplace. Washington, D.C., Federal Trade Commission.

Privacy-International, (1999). Privacy and Human Rights, Privacy International. **2000**.

Rothermel, K. H., F. and Radouniklis, N. (1997). Mobile Agent Systems: What is Missing? International Conference on Distributed Applications and Interoperable Systems, Cottbus, Germany.

Russell, S. J. N., Peter (1995). Artificial Intelligence: A Modern Approach. Englewood Cliffs, New Jersey, Prentice Hall.

Versteeg, S. (1998). "Comparison of Mobile Agent Toolkits for Java." ??

Wacker, J. T. W. (1997). The 500 Year Delta. New York, HarperCollins Publishers.

Wayner, P. (1995). "Free Agents." Byte(March): 94-95.

Westin, A. F. (1967). Privacy and Freedom. New York, Atheneum.

Wooldridge, M. J., N. R. (1995). "Intelligent Agents: Theory and Practice." The Knowledge Engineering Review **10**(2): 115-152.

Appendix: Sample Source Code.

The sSlave Aglet (sSlave.java)

```
package examples.projects;

import aglets.portal.base.*;
import com.ibm.aglet.event.*;
import com.ibm.aglet.*;
import java.net.URL;
import java.util.Hashtable;
import java.util.Date;
import java.util.Enumeration;
import java.util.Vector;

public class sSlave extends Slave {

      private AgletProxy masterProxy = null;
      private AgletProxy myProxy = null;
      private AgletProxy escrowProxy = null;
      private AgletID sAgletID = null;
      Vector conversationRecord = new Vector(10);
      public boolean repeat = true;
      private double price ;
      private String email, lname, fname, escrowCode;
      private sInfo myInfo = null;
      static boolean debug = true;
      static boolean ddebug = false;
      static int timeMultiplier = 1;
      static String Marketplace = "atp://pc690:500";

      public void onCreation(Object args) {
            myInfo = (sInfo)((Object[])args)[0];
            try  {
                  masterProxy = (AgletProxy)((Object[])args)[1];
                  escrowProxy = (AgletProxy)((Object[])args)[2];
                  escrowCode = masterProxy.getAgletID().toString();
            }
            catch (Exception e) {System.out.println("Problem with MasterProxy or Escrow message
" + e.toString()); }
          addMobilityListener(new MobilityAdapter() {
                  public void onArrival(MobilityEvent event)
                  {
                        try {
                              masterProxy.sendMessage(new Message("NewSlaveProxy",
getAgletContext().getAgletProxy(getAgletID())));
                              }
                        catch (InvalidAgletException iae) {System.out.println("sSlave: " + iae);      }
                        catch (NotHandledException ex) {System.out.println("sSlave: " + ex); }
                        catch (MessageException ex) { System.out.println("sSlave: " + ex); }
                  }
            }
          );
      }


      public void run()   {
          if (atHome()  && repeat) {
                try {
```

```
                    repeat = false;
                    sAgletID = getAgletID();
                    dispatch(new URL(Marketplace));
                    waitMessage(4000 * timeMultiplier);
               } catch (Exception e)  {System.out.println(e.toString());}

          } else if (atHome() && !repeat)    {
               uponReturnHome();

          } else {
               try  {
                    waitMessage(1000 * timeMultiplier);
                    boolean temp = true;
                    boolean proceed = false;
                    while(!proceed)
                    {
                         try {
                              waitMessage(10000 * timeMultiplier);
                              for (int i = 0;i <conversationRecord.size() ;i++ )
                              {
                                   Conversation con =
(Conversation)conversationRecord.elementAt(i);
                                   if (con.getAlive()) {         temp = false;                }
                              }
                              if ((temp) && (!conversationRecord.isEmpty())) {
                                   conversationRecord.removeAllElements();
                              } else temp = true;
                         } catch (ArrayIndexOutOfBoundsException aioobe)
                              {          waitMessage(1000 * timeMultiplier);
                              System.out.println("Problem with counting the conversations! " +
aioobe.toString());                                                    }
                    }
               } catch (Exception e) {System.out.println("Problem with sSlave " + e.toString());
          }
          }
   }


     public void uponReturnHome(){
          dispose();
     }


   public void returnHome()        {
      try {
               Message msg = new Message("RetractMe");
               msg.setArg("url", getAgletContext().getHostingURL());
               msg.setArg("id", getAgletID());
               masterProxy.sendOnewayMessage(msg);
          } catch (InvalidAgletException iae){System.out.println("sSlave: returnHome IAE" +
iae.toString());
          } catch (Exception e) {System.out.println("sSlave: returnHome" + e.toString()); }
   }


     public boolean handleMessage(Message msg) {
          myProxy = getAgletContext().getAgletProxy(sAgletID);
          if (debug)    {send("Received a " + msg.getKind() + " message from " +
msg.getArg("name") + "\n");}
       if (msg.sameKind("INIT")) {
```

```
        try {
                AgletID aID = (AgletID)msg.getArg("conversationID");
                String conversationID = aID.toString();
                AgletProxy bProxy = (AgletProxy)msg.getArg("bproxy");
                Conversation conversation = new Conversation();
                Message sMsg = new Message("INIT_OK");
                String fullConversationID = conversationID + getAgletID().toString();
                conversation.setConversationID(fullConversationID);
                conversation.setProxy(bProxy);
                conversationRecord.addElement(conversation);
                sMsg.setArg("conversationID", fullConversationID);
                bProxy.sendOnewayMessage(sMsg);
                if (debug)      {send("Sent INIT_OK message ");}
        } catch (Exception ex) { System.out.println("sSlave: INIT" + ex.toString() ); }

         } else if (msg.sameKind("CRITERIA REQUEST")) {

             try  {
                Conversation con = getConversation((String)msg.getArg("conversationID"),
conversationRecord);
                if (con != null) {
                    Message cMsg = new Message("CRITERIA RESPONSE");
                    cMsg.setArg("conversationID", con.getConversationID());
                    Vector criteria = (Vector)msg.getArg("criteria");
                    Vector responses = new Vector(criteria.size());
                    for (int x=0;x<criteria.size() ;x++ )
                    {
                        String b = (String)myInfo.getAttrib((String)criteria.elementAt(x));
                        if (b != null) {
                                responses.addElement(b);
                        }       else responses.addElement("Unanswered");
                    }
                    cMsg.setArg("responses", responses);
                    con.criteria = criteria.size();
                    con.getProxy().sendOnewayMessage(cMsg);
                    if (debug) {send("Sent Criteria Response message");}
                } else {System.out.println("Big problem at the criteria stage! " + msg.toString());}
             } catch (Exception e) { System.out.println("Problem with criteria " + e.toString());}

      } else if (msg.sameKind("I WANT TO BUY")) {
         try {
                Conversation con = getConversation((String)msg.getArg("conversationID"),
conversationRecord);
                if (con != null){
                    Vector fields = (Vector)msg.getArg("fields");
                    String usage = (String)msg.getArg("usage");
                    String orgtype = (String)msg.getArg("orgtype");
                    Vector responses = new Vector(fields.size());
                    Vector answers = new Vector(fields.size());
                    double price = 0.00;
                    for (int x= 0;x<fields.size() ; x++)
                    {
                        String fName = (String)fields.elementAt(x);
                        fName = myInfo.getFieldName(fName);
                        String b = (String)myInfo.getAttrib(fName);
                                if ((b != null) && (!b.equals("XXX"))) {
                                        responses.addElement("true");
                                        price = price + getPrice(fName);
                                        if (ddebug){ send("Field: " + fName + " answer: " + b + "
price: " + price); }
```

```
                                        answers.addElement(b);
                                    } else {
                                        responses.addElement("false");
                                        if (ddebug){ send("Field: " + fName + " answer: =
false");  }
                                        answers.addElement("Unknown");
                                    }
                                }
                                con.setAnswers(answers);
                                double priceExtra = Math.rint(((1 + con.criteria) * Math.random()));
                                price = getUsagePrice(price, usage, orgtype);
                                price = price + priceExtra;
                                con.setUsage(usage);
                                con.setOrgtype(orgtype);
                                con.setPrice(price);
                                price = con.getNicePrice();
                                Message buyMsg = new Message("PRICE");
                                buyMsg.setArg("conversationID", con.getConversationID());
                                buyMsg.setArg("responses", responses);
                                if (escrowProxy != null) {buyMsg.setArg("escrowProxy", escrowProxy);}
                                buyMsg.setArg("Price", price);

                                if (debug) {send("Sent Price message: ");      }
                                con.getProxy().sendOnewayMessage(buyMsg);
                            } else { respondToInvalidMessage(con, msg, "Incorrect ConversationID given");
}
                        } catch (Exception ex) {System.out.println("sSlave: Problem sending the PRICE " +
ex.toString()); }

                } else if(msg.sameKind("I DO NOT WANT TO BUY"))          {
                        try  {
                            Conversation con = getConversation((String)msg.getArg("conversationID"),
conversationRecord);
                            if (con != null){
                                String reason = (String)msg.getArg("reason");
                                if (debug){send("Received I DO WANT TO BUY message because " +
reason);}
                                con.setStatus("Stopped at Criteria stage due to " + reason);
                                deleteConversation(con.getConversationID(), conversationRecord);
                            } else System.out.println("Problem with I DO NOT WANT TO BUY! " +
msg.toString());
                        } catch (Exception e) { System.out.println("sSlave: Problem with DO NOT WANT TO
BUY MSG." + e.toString() + msg.toString()); }

                    } else if(msg.sameKind("ACCEPT OFFER")) {
                        try  {
                            Conversation con = getConversation((String)msg.getArg("conversationID"),
conversationRecord);
                            if (con != null){
                                Message aMsg = new Message("CONFIRM");
                                aMsg.setArg("conversationID", con.getConversationID());
                                con.getProxy().sendOnewayMessage(aMsg);
                                sendInformation(con);
                            } else System.out.println("Problem with ACCEPT " + msg.toString());
                        } catch (Exception e){System.out.println("sSlave: ACCEPT" + e.toString() +
msg.toString());}

                    } else if(msg.sameKind("REJECT OFFER")) {
                        try  {
```

```
                    Conversation con = getConversation((String)msg.getArg("conversationID"),
conversationRecord);
                    if (con != null) {
                        String reason = (String)msg.getArg("reason");
                        con.setStatus("Stopped at Price stage due to " + reason);
                        deleteConversation(con.getConversationID(), conversationRecord);
                        if (debug)    {send("Received REJECT OFFER message, because of " +
reason);}
                    }else System.out.println("Problem with REJECT " + msg.toString());
                } catch (Exception e) { System.out.println("Problem with: REJECT" + e.toString() +
msg.toString()); }

            } else if(msg.sameKind("MONEY")) {
                try  {
                    Conversation con = getConversation((String)msg.getArg("conversationID"),
conversationRecord);
                    if (con != null) {
                        String money = (String)msg.getArg("money string");
                        con.setMoney(money);  // store the money string!
                        if (debug){send("Received the money! " + money);}
                        msg.sendReply(true);
                    } else System.out.println("sSlave has problem with receiving the money!");
                } catch (Exception e) {System.out.println("sSlave: MONEY" + e.toString() +
msg.toString());}

            } else if(msg.sameKind("COMPLETED")) {
                try {
                    Conversation con = getConversation((String)msg.getArg("conversationID"),
conversationRecord);
                    if (con != null) {
                        con.setStatus("Completed");
                        deleteConversation(con.getConversationID(), conversationRecord);
                    } else System.out.println("Problem with COMPLETED " + msg.toString());
                } catch (Exception e) {System.out.println("sSlave: MONEY" + e.toString() +
msg.toString());}

            } else if (msg.sameKind("escrowCode")) {
                try {
                    escrowProxy = (AgletProxy)msg.getArg("eproxy");
                } catch (Exception e)           {System.out.println("sSlave: Escrow problem " +
e.toString());           }

          } else if (super.handleMessage(msg) == false) {
                    System.out.println("DEBUG: hM (master) " + msg.toString());
                    return false;
                }
           return true;
       }


        public void send(String s)     {
            try  {
                Message m = new Message("Message");
                m.setArg("Text", "From: " + myInfo.getName() + " " + s);
                masterProxy.sendOnewayMessage(m);
            } catch (Exception e)      {System.out.println("sSlave: SEND" + e.toString());          }
        }


        public double getPrice(String s){
```

```
        double result;
        if (s == null)       {return 0.00;
        } else    {
            result = myInfo.getPrice(s.toLowerCase());
        }
        return result;
    }


    public double getUsagePrice(double p, String u, String o)      {
        double usageMultiplier, orgtypeMultiplier;
        usageMultiplier = orgtypeMultiplier = 1;
        if (u.equals("Internal&Single")) {
            usageMultiplier = 1;
        } else if (u.equals("Internal&Multiple")) {
            usageMultiplier = getPrice("multiple") / 100;
        } else if (u.equals("External&Multiple")) {
            usageMultiplier = (getPrice("external") * getPrice("multiple")) / 10000;
        } else if (u.equals("External&Single")) {
            usageMultiplier = getPrice("external") / 100;
        } else usageMultiplier = 1;
        if (o.equals("Charity")) {
            orgtypeMultiplier = orgtypeMultiplier * getPrice("charity") / 100;
        } else if (o.equals("Research")) {
            orgtypeMultiplier = orgtypeMultiplier * getPrice("research") / 100;
        } else if (o.equals("Noncommercial")) {
            orgtypeMultiplier = orgtypeMultiplier * getPrice("noncommercial") / 100;
        } else orgtypeMultiplier = 1;
        return  p * usageMultiplier * orgtypeMultiplier;
    }

    public void sendInformation(Conversation con) {
        try  {
            Message msg = new Message("INFORMATION");
            msg.setArg("conversationID", con.getConversationID());
            msg.setArg("information", con.getAnswers());
            msg.setArg("price", con.getPrice());
            msg.setArg("usage", con.getUsage());
            msg.setArg("orgtype", con.getOrgtype());
            msg.setArg("bproxy", con.getProxy());
            msg.setArg("sproxy", myProxy);
            msg.setArg("name", myInfo.getName());
            escrowProxy.sendOnewayMessage(msg);
        }    catch (Exception e) { System.out.println("sSlave: MONEY" + e.toString());      }
    }
}
```