UNIVERSITY OF STIRLING

Martin L. Gill

Department of Mathematics and Computing Sciences

Combining MAS and P2P Systems: The Agent Trees Multi-Agent System (AT-MAS)

Submitted for the degree of Doctor of Philosophy

October 2005

Declaration

I declare that this thesis has been composed by myself and that the research reported therein has been conducted by myself unless otherwise indicated.

Stirling, October 2005

Martin L. Gill

Acknowledgements

This work has been funded during the earlier years by the Universal Information Technology Group Ltd. (UniTech). In the later years, by the Department of Computing Science and Mathematics, University of Stirling, and finally through the sale of my flat.

A large number of people have made this work possible. First and foremost, I'd like to thank my main supervisor Professor Leslie Smith, secondary supervisor Dr. Simon Jones, and everyone in the Department of Computing Science and Mathematics at the University of Stirling. There are too many people to name individually, but all have been helpful, friendly and provided solutions to problems which caused me many bitten fingernails!

My family have been incredible; my parents Eileen & Pat, my brother Philip, my sister-in-law Ann, and my sister Katherine, have been great. Without their support and occasional food parcel, this would have been impossible.

I'd like to offer my thanks to Audrey who never really understood why I needed to complete a PhD, but knew how much it meant to me. My love and best wishes go to her and her children; Kenny and Bex.

My friends also deserve credit; Cathy, Simes (Dude!), Scott & Georgina, Guiness, Nicki, Susan & Jim, Jeudi and Jules. Thank you for your patience - it has been very much appreciated.

Geddy Lee, Alex Lifeson & Neil Peart - collectively known as *Rush* who have provided the soundtrack to most of my life.

Finally I'd like to thank Karla and Auntie J. who was there for me during the darkest moments. You helped me understand. For that and so much more, I'll always be in your debt.

Abstract

The seamless retrieval of information distributed across networks has been one of the key goals of many systems. Early solutions involved the use of single static agents which would retrieve the unfiltered data and then process it. However, this was deemed costly and inefficient in terms of the bandwidth since complete files need to be downloaded when only a single value is often all that is required.

As a result, mobile agents were developed to filter the data in situ before returning it to the user. However, mobile agents have their own associated problems, namely security and control.

The Agent Trees Multi-Agent System (AT-MAS) has been developed to provide the remote processing and filtering capabilities but without the need for mobile code. It is implemented as a Peer to Peer (P2P) network of static intelligent cooperating agents, each of which control one or more data sources.

This dissertation describes the two key technologies have directly influenced the design of AT-MAS, Peer-to-Peer (P2P) systems and Multi-Agent Systems (MAS). P2P systems are conceptually simple, but limited in power, whereas MAS are significantly more complex but correspondingly more powerful. The resulting system exhibits the power of traditional MAS systems while retaining the simplicity of P2P systems.

The dissertation describes the system in detail and analyses its performance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis describes the Agent Trees Multi-Agent System (AT-MAS) and its relationship to other existing work.

Two key technologies have directly influenced the design of AT-MAS: Peer-to-Peer (P2P) systems and Multi-Agent Systems (MAS). P2P systems are conceptually simple, but limited in power, whereas MAS are significantly more complex but correspondingly more powerful. AT-MAS is a combination of these two different technologies but with a number of other features. The intention is that the resulting system exhibits the power of traditional MAS systems while retaining the simplicity of P2P systems.

1.1 Motivation

The AT-MAS project was partly motivated by the desire to simplify the general design concepts of these systems by replacing the mobile agents with static agents and introducing a much simpler protocol for interaction based on some of the concepts used in P2P systems. While it is unlikely that the AT-MAS system will cause a revolutionary re-design of the techniques for designing MASs, it is hoped that it may cause some consideration as to whether the current level of complexity in most systems is required and how it may be reduced in future.

The second motivation for this work is the desire to produce a functionally equivalent (or better) system to replace mobile agents, but which does not suffer from the same limitations e.g. security, control. While much of the simplicity of the AT-MAS system stems from the comparative simplicity of the protocol, the use of static agents instead of mobile agents has contributed greatly to security and to a reduction in the infrastructure required to support the agents.

Thirdly, when considering the current level of complexity of the underlying design of P2P system, it seems remarkable that little work has been done in extending the application-level facilities available to the user. By replacing the simple file-handling of the current generation of P2P systems with a more sophisticated level of data manipulation abilities based around XML elements, AT-MAS creates a usable information filtering and retrieval resource which helps to bridge the gap between MAS and P2P Systems.

1.2 Scope

There are three underlying technologies; Intelligent Agents, Multi-Agent Systems and Internet based Technologies. These elements relevant to the design AT-MAS are described in detail. Aspects which have influenced the design particularly, such as P2P networks are discussed in more detail.

The various methodologies and frameworks such as Gaia[190], JADE[16, 56], LEAP[17, 120], etc. used to create various Agent Systems are not described as they are by their nature, development tools, and as such are incidental to the study of functioning agent systems. While they do provide many benefits in terms of speed of development, they limit the possible forms that such agent systems can take.

Thus, it has been important to develop novel agent systems such as AT-MAS from scratch. AT-MAS system was both positively and negatively influenced by the design of other modern MASs. As a result, AT-MAS is an unconventional MAS, owing much of its inspiration to the development of P2P systems. Similarly P2P frameworks such as Jxta[171, pages 163-179] have also been excluded from the development process for the same reasons.

The whole purpose of the agent frameworks and development is to make the process of building agent systems easier. In doing so, they provide a pathway; a simple route to creating agent based systems, unless the final destination is radically different from that envisaged by the framework designers. By providing tools, modules, interfaces and off the shelf components to assist with creating the agents and the infrastructure to support them, these systems provide a tempting *easy solution* which may cause the original idea to become diluted.

Additionally many of the social aspects of the technology such as the ongoing legal concerns regarding P2P music downloading and copyright issues have also been excluded from this work as they are beyond the scope of this thesis. For a historical and non-technical introduction to P2P and music sharing, the reader is directed to the book 'SonicBoom' by John Alderman[4].

1.3 Contribution to Current Research

By showing that it is possible to combine P2P techniques with Multi-Agent Systems to produce a system which is simple in comparison to existing MASs, yet powerful, it is hoped that it will prompt other researchers to reconsider the complexity of their designs, leading to a simplification in the design of Multi-Agent Systems. Additionally, extending the range of information processing available to P2P system, may prompt researchers to consider more ambitious applications for their systems, leading to convergence.

Further, it is hoped that the simple AT-MAS communications protocol can be used as the basis for interaction between different P2P systems which are currently designed and implemented to function separately. While it is acknowledged that a number of Agent Communication Languages (ACLs) and protocols exist, they are, in general, more complex than required for simple information retrieval and processing tasks. Conversely, the current proprietary protocols used by P2P systems allow only file retrieval and as a result are too limited.

1.4 Structure of the Thesis

Chapter 2 starts with a discussion of the problems concerning finding a reasonable definition of Intelligent Agents. In addition to the more general definitions, three different definitions are given to illustrate the range of viewpoints that can be considered when viewing the field. This is followed by a discussion of a number of the criteria that should be considered which create an agent. Finally, a number of architectures are considered.

Chapter 3 considers groups of agents. This chapter, much like the field of research, concentrates mainly on communication between the agents. Both open and closed systems of agents are considered.

Chapter 4 concerns the environment of AT-MAS, namely the internet. Firstly, the problems caused by its scale, and lack of regulation are described. The sections following this describe the rapidly expanding range of application which it supports.

The next chapter (5) introduces the AT-MAS system. The chapter starts with an overview of the system, followed by a description of the system in operation in which key operations and design decisions are explained. The following section details the separate components which provide support for the agent along with the services that they provide, leading into a dissection of the agent. This gives further detail about the components and their functionality. Although this chapter concerns the AT-MAS system, other systems are referred to where they have directly influenced the design of AT-MAS.

Chapter 6 evaluates the various aspects of the system. Both quantitive evaluations (based on the results of tests carried out) and qualitative evaluations (based on general comparisons with other types of system) are described.

In chapter 7 of this thesis, a final evaluation of the success of the system is given. This is followed by a description of the future work that may be carried out on the system to improve its operation.

Chapter 2

Intelligent Agents

2.1 Definition of an Agent

When presenting any complex topic in detail, it is important to define the common terms. This is especially important if the terms are a new or altered definition of terms that are already in common use elsewhere. *Agent* is one term that this applies to. In our everyday lives we come across specific kinds of agent: *Travel Agents*, *Estate Agents* and we hear stories about *Secret Agents* but the question remains *What is an agent?*

As with most topics, the answer to this question depends on who asks the question. It is for this reason that three different, but closely related definitions of an agent are presented.

2.1.1 Definition 1: Autonomous Servants

According to the dictionary, the term *agent* is as follows;

agent: . . . a person who acts on behalf of another person or organisation.[49, page 15]

By replacing the first occurrence of the word *person* with *program* we are closer to a definition that applies to the computing domain, but this definition is still not close enough. The problem is that every piece of software is a program, but not every piece of software is an agent. However, when this word is replaced with the words *software entity*, and the sentence re-worded to improve clarity, the definition is as follows:

> An agent is a software entity which acts on behalf of a person or organisation.

This is very similar to a number of the more general definitions of an agent:

> Many agents are meant to be used as intelligent electronic gophers - automated errand boys. Tell them what you want them to do - search the Internet for information on a topic, or assemble and order a computer according to your desired specifications - and they'll do it and let you know when they've finished.[165]

> Intelligent agents are software entities that carry out some set of operations on behalf of the user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires.[63, page 2]

These definitions, while still valid, are too broad for all but the most general-purpose agents. Unfortunately finding the balance between a definition that is too vague and one that is too specific can be a problem. Any vague definition will often give false credibility to a number of programs that should really not be defined as agents, whereas any definition which is too specific will exclude a large number of legitimate agents.

> I find little justification for most commercial offerings that call themselves agents. Most of them tend to excessively anthropomorphize the software, and then conclude

> that it must be an agent because of that very anthropomorphization, while simultaneously failing to provide any sort of discourse or ”social contract” between the user and the agent. Most are barely autonomous, unless a regularly scheduled batch job counts. Many do not degrade gracefully, and therefore do not inspire enough trust to justify more than trivial delegation and its concomitant risks.[59]

A more sophisticated definition of an agent can be found at the start of Gerhard Weiss's book; *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.*

> An agent is a computational entity such as a software program or robot that can be viewed as perceiving and acting upon its environment and that is autonomous in that its behavior at least partially depends upon its own experience. As an intelligent entity, an agent operates flexibly and rationally in a variety of environmental circumstances given its perceptual and effectual equipment. Behavioral flexibility and rationality are achieved by an agent on the basis of key processes such as problem solving, planning, decision making, and learning. As an interacting entity, an agent can be affected in its activities by other agents and perhaps by humans.[181, page 1]

The problem is that while the definition is valid for a large number of agent systems, not all agents exhibit all of the characteristics mentioned and as a result, some agents are excluded.

> It's difficult to find a succinct definition that *includes* all of the things that most researchers and developers consider agents to be, and *excludes* all of the things they aren't.[165]

The task of choosing a definition is further complicated by the number and range of different agents; each with different attributes, different purposes and programmed to operate in different environments. As a result, they are given different names to reflect this.

> So now we have synonyms including knowbots (i.e. knowledge-based robots), softbots (software robot), taskbots (task-based robots), userbots, robots, personal agents, autonomous agents and personal assistants.[126]

In many ways, these definitions applied to the agents correspond to the roles or job descriptions in human society. Agents are defined as softbots and userbots in the same way human occupations are defined as Doctor, Lawyer, Scientist, Estate Agent, etc. In addition, even the more general-purpose agents may be known as software agents, intelligent agents, autonomous agents, or similar.

Usually it depends upon the agent's creator(s) to determine what modifiers, if any, are to be used when describing their agents - if, of course, the programs being described are actually agents.

2.1.2 Definition 2: An object with 'attitude'

As most programmers are aware, an object may be described as a computational model of its real-world equivalent. This is also the case with agents.

> But agents are *not* simply objects by another name. This is because an agent is a *rational decision making system:* we require an agent to be capable of reactive and pro-active behaviour, and of interleaving these types of behaviour as the situation demands.[186, page 29]

With agents, we are trying to create an simple abstract model of an independent living organism. Due to the incredible complexity of most living organisms, agents and agent based applications may be very complex in terms of the number of lines of code, processor and memory requirements when compared to other computer programs. However, they are usually very simplistic when compared to their real-world counterparts[1]. As a result the model *(agent)* must reflect this.

When describing a user interface containing both objects, Tom Erickson provides a simple comparison:

> ...objects and agents can be used in the same interface, but they are clearly distinguished from one another. Objects stay what they are: nice, safe, predictable things that just sit there and hold things. Agents become the repositories for adaptive functionality. They can notice things, use rules to interpret them, and take actions based on their interpretations.[54, page 94]

In practical terms, this means that while objects are called programmatically and return results, when requests are made to an agent it can perform a number of different actions. It may, for example, ignore the request, refuse the request or attempt to complete it. As a result, agents have sometimes been referred to as *objects with attitude*[24, page 382], or more recently as *complex objects with attitude*[157]. This contrasts with the sort of object that make up Java and C++ programs.

2.1.3 Definition 3: A 'rational' software entity

In the third definition of an agent, we consider it as an *intentional* system, and concentrate on its internal *mental* state rather than its anthropomorphic outward appearance.

> For some researchers - particularly those working in AI - the term 'agent' has a stronger and more specific meaning than that sketched out above. These researchers

[1]The exceptions to this are *reactive* agents which are modelled on simple living organisms such as insects. These are described in section 2.4.2.

> generally mean an agent to be a computer system that, in addition to having the properties identified above[2] is either conceptualised or implemented using concepts that are more usually applied to humans. For example, it is common in AI to characterise an agent using *mentalistic* notions, such as knowledge, belief intention, and obligation... Some AI researchers have gone further, and considered *emotional* agents.[189]

By creating agents which are modelled with human-like reasoning, we are reducing the possibility of uncertainty in their actions. This means that given an agents beliefs (knowledge) and desires (goals), we can work out what the agents intentions are. This allows us to predict with some degree of certainty what the logical course of action in the current situation will be: in other words, the agent will behave *rationally*.

One way of achieving this is for the agent to use its own knowledge and its perception of the current state of the environment to create plans in order to achieve its goals. These plans will be executed by the agent which will monitor the effect(s) of its actions. If the plan fails, then the agent will have to re-plan. In some cases the environment may change in such a way that a number of actions in the plan may not be required. When this happens, the agent should recognise this and revise its plan accordingly. If the agent is not able to complete a particular goal then it must also recognise this.

If an agent has been designed in this way it may be referred to as using a number of different terms - strong agents, rational agents, BDI (Belief, Desire, Intention) agents.

> Strong agents are defined by the same set of properties to that of weak agents, and are constructed using a cognitive approach which bases design of the agent on how a human may solve the problem, using knowledge, belief and intention.[174]

> An obvious problem is how to conceptualize systems that are capable of rational

[2]The attributes referred to are; Autonomy, Social Ability, Reactivity and Pro-activeness

> behaviour ... One of the most successful solutions to this problem involves viewing agents as *intentional systems* ... whose behaviour can be predicted and explained in terms of *attitudes* such as belief, desire, and intention ... The rationale for this approach is that in everyday life, we use a *folk psychology* to explain and predict the behaviour of complex intelligent systems: people.[186, page 30]

This use of folk psychology has extended into the communications aspect of agents. As a result, most, if not all communication between agents is carried out using Speech Acts. These are intentional statements such as TELL, ASK, REGISTER which allow an agent to communicate its intentions rather than carry them out directly.

> Briefly, the key axiom of speech acts theory is that communicative utterances are *actions*, in just the same sense that physical actions are. They are performed by a speaker with the intention of bringing about a desired change in the world; typically the speaker intends to bring about some particular mental state in the listener. Speech acts may *fail* in the same way that physical actions may fail: a listener generally has control over her mental state, and cannot be guaranteed to react in the way that the speaker intends.[189]

By defining communication in this way, it is possible for an agent to incorporate communication actions into plans. This is important when the system contains more than one agent (see chapter 3).

2.2 Definition of an Agent - part 2

The above definitions show that the word ’agent’ can mean different things to different people. From a general point of view, the idea of a simple electronic entity[3] is an appealing metaphor. It allows us to abstract away all of the details and leaves us with a simple assistant that we can delegate tasks to - as we would any subordinate.

From a more technical point of view, it is possible to make comparisons between objects and agents. This is because both are abstract computation models of real-world items. However, agents are models of independent thinking entities and the models (agents) must reflect this. As a result, agents can, and often do, refuse requests from other agents. In contrast, an object has no control over its own behaviour when it receives a message from another object.

The third definition is the most complex. In this, the agent is viewed as a rational computational entity with beliefs, desires and intentions. It must make decisions based on information that it knows or has learned, and use this information when creating rational plans in order to achieve its goals. As part of this model, the agent is given human-like thought processes, which include beliefs, desires, intentions, and in some cases emotions. Agents which fulfill the criteria of this third definition are known as either *strong*, *rational* or *BDI* agents.

Having examined the various definitions of an agent, it can be seen that the topic is a wide and varied one. It is important to note that each of the definitions is an attempt to provide an abstraction; whether it is anthropomorphic, object-oriented, or mentalistic. As such, all definitions are valid, but the most appropriate one will depend upon the context.

In his 1992 Extended Abstract ”Distributed Intelligent Agents”, Stanley J. Rosenschein comments on the use of agents as a metaphor for complex computer systems.

[3]often created as a simple computational servant

> The notion of an ”agent” is often useful in these circumstances because it abstracts away from the particulars of how information is encoded or what specific actions are produced and focuses instead on the *content* of the information to be encoded and the *goals* to be achieved by the system.[147]

So, we are left with a number of definitions which provide a useful abstraction - removing all of the detail, and giving us a number of general descriptions based on how we view the agent. However, this leaves the problem that there is still no single all-encompassing definition of an agent.

> So, what exactly is an agent? Must it be intelligent? Adaptive? Itinerant? There are almost as many opinions on this as there are agents themselves, leading to frequent debates flaring up on several Internet forums In any case, as a practical matter we should always ask the question, What is so special about an entity that it may be called an agent? or What does calling it an agent buy us? The answer would not be the same in each case, but it should be nonempty for the notion of agency to be nonvacuously applied.[83, page 1]

Unfortunately, it is unlikely that there ever will be a single complete definition. Therefore, it will be more useful to move on to discuss the attributes that an agent may possess.

> There is as much chance of agreeing on a consensus definition for the word ’agent’ as there is of AI researchers arriving at one for ’artificial intelligence’! When necessary an agent is defined as referring to a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user. However, it is would be preferable to say that is an umbrella term which covers a range of more specific agent types, and then go on to list and define what these other agent types are.[125]

2.3 Key Features of an Agent

While there is a lot of disagreement about what an agent is, there is more agreement about what an agent is not. Instead, many researchers base their definitions around a list of the attributes that they believe an agent should possess.

> It has become common to define an appropriate notion of agency by specifying the *necessary attributes* that all agents of the particular kind one has in mind are required to share . . . There has been much of debate, however, what set of properties exactly qualifies an entity, such as a single human decision maker, a firm in the market, a computer program, a robot or an unmanned autonomous vehicle, for an *autonomous* or *intelligent* agent.[173]

This next section highlights some of the more important features that should be considered when designing an agent. Unfortunately, this list does suffer from the same problem as the definition: there are many different types of agent with different abilities, attributes and domains. In fact, the only attribute which researchers agree is essential is *autonomy* i.e. the agent must be able to exhibit some level of independent behaviour.

> Essentially, while there is a general consensus that *autonomy* is central to the notion of agency, there is little agreement beyond this. Part of the difficulty is that various attributes associated with agency are of differing importance for different domains.[187, page 15]

All of the other attributes - including intelligence - are optional depending on the agent. In each case, attributes may be implemented in different ways; or to different levels.

2.3.1 Autonomy

Considering the range of agents, their possible attributes, and the struggle to find an all-encompassing definition which is universally acceptable, it is reassuring to find that there is complete agreement about the need for agents to be able to work autonomously. However, as before, there is a great deal of discussion about what autonomy implies.

Even a very simple definition of autonomy requires that the agent is able to survive and pursue its goals in a potentially complex environment without the need for continuous user assistance. The user may delegate tasks to the agent which it will attempt to complete - yet the agent must be able to work alone if required. Other tasks/goals may arise from either conditions within the environment or from the agents own needs, and these must also be acted upon.

Agents function asynchronously and as such, are not restricted to the command, response, command, response, command, response style of interaction generated by traditional software. As a result, they are always listening for commands and/or information from the user, other agents, and the environment. By working this way, an agent is able to respond to any events which have the potential to affect it.

> The agent operates without direct intervention (e.g. in the background) to the extent of the user's specific delegation. The autonomy attribute of an agent can range from being able to initiate a nightly backup to negotiating the best price of a product for the user. [29]

> The real world is complex, unpredictable and dynamic. It is simply not possible for a designer to foresee all of the circumstances that might be faced by an agent in continuous long-term interaction with such an environment. Any truly intelligent agent must therefore possess a considerable degree of autonomy. It must be capable of flexibly adapting its behavioral repertoire to the moment to moment contingencies

> which arise without being told what to do in each situation.[15, page 169]

> An autonomous agent is a computational system that has a set of goals and operates completely autonomously in an unstructured, dynamic environment. It tries to achieve its goals by interacting with the environment through sensors and effectors. An example of an autonomous agent would be a "robot pet" that "lives" in an everyday house and tries to survive and receive attention from its human house mates. Another example would be an animated figure that inhabits a simulated 3D world (e.g. An adventure world video game).[108, page 1]

Although computer games and robotic pets can be switched off, the example given hints at another important part of autonomy; the fact that the agents often have a long 'lifespans'. This persistence is one of the key concepts associated with autonomy, and with agents in general; the agent must be able to survive for long after its initial goals have been completed.

> Any "proper" computational or biological autonomous agent can also be expected to be at least somewhat *persistent*, that is, to "live on" beyond completing a single task on a single occasion. In case of software agents, persistence makes an agent different from say a *subroutine* of a computer program whose "turning on and off" is controlled from outside of that subroutine.[173]

However, this means that there will be times when an agent has no specific goals to complete for the user or for other agents. Many researchers believe that for an agent to be autonomous it must also be *pro-active* and make use of these times - to gain knowledge or perform other tasks which *may* assist it in carrying out tasks in the future. These self-assigned goals may include exploring its environment, optimizing its own internal states, etc.

While much of this work performed by an agent will be of limited use, since it is carried out when the agent would otherwise be sitting idle, the cost is essentially zero. However, there is a potential gain as the information discovered may allow the agent to respond more quickly to future requests.

However, this activity must be balanced against the possible effects on other agents and/or resources. In a distributed application, the number of query messages sent out by an agent in an effort to find out more information must be balanced against the additional network load that the agent generates. Similarly if an agent regularly locks a shared database in order to update its own information, this could have a adverse effect on the overall efficiency of the system. Therefore it is important to limit any potentially detrimental effects.

In his report "What's an agent, Anyway? A Sociological Case Study", Leonard Foner[59] describes a chatterbot[4] 'Julia' which exists and interacts with the users in the TinyMud[5] environment.

> A more autonomous agent can pursue agenda independently of its user. This requires aspects of periodic action, spontaneous execution and initiative, in that the agent must be able to take pre-emptive or independent actions that will eventually benefit the user. [Julia] carries out many independent actions in the MUD. In fact, most of her time is spent pursuing a private agenda (mapping the maze), which is nonetheless occasionally useful to her users (when they ask for navigational assistance).[59]

But this poses the question: *'If an agent like Julia didn't map the maze, or perform some other tasks in its spare time, could it still be classed as autonomous, but more importantly, would it*

[4]A chatterbot is a an agent which has been designed as an artificial person. Chatterbots have their own personalities, preferences, moods and motives. However, they are generally designed to be helpful and provide assistance when requested.

[5]TinyMUD is an example of a Multi-User Dungeon (MUD); originally an environment - combining the features of both chatrooms and text adventures - for online role-playing. MUDs typically consist of a number of virtual rooms where users communicate by typing messages to each other. Moving between rooms and performing other actions may done by typing simple commands; eg. GO NORTH, TAKE SWORD, EAT FOOD, etc.

still be an agent?' While mapping a maze is a useful task, it could be argued that this pro-active behaviour does help the case for agency, the lack of it does not always indicate a lack of agency. In this case, the perception of *believability* would be shattered if Julia were to remain motionless and inert between performing tasks for the users. Therefore it is important for Julia to be pro-active at times.

In other systems where believability is not a requirement, continuous activity may be necessary for other reasons.

2.3.2 Robustness

One of the key goals of Software Engineering is to produce programs that are robust. That is, all software should be able to cope with changes and unexpected circumstances; whether it is the lack of an expected response or an unpredictable environment change.

For agents, especially those which operate in dynamic environments, changes and unpredictable events are commonplace. Agents must be able to cope with a multitude of changing circumstances; changes to the environment; and changes to the agents goals. It is also possible that some of the actions attempted by an agent may not succeed. If this happens, the agent must be able to detect the failure and alter its plans accordingly. If the failure of an action means that a key goal has become un-achievable, then the agent must also recognise this - there is no point in continuing if an essential data item is unavailable.

In the Seaworld simulation created by Steven Vere & Timothy Bickmore[178, 177] the agent *Homer* is equipped with a camera. If the camera became damaged, then Homer would still be able to perform all of the tasks given to him, except for those which required him to photograph objects.

For reactive agents, this is especially important as their designers have made a conscious decision to make robustness a key feature. By creating artificial animals; whether robotic as in the case of the robots created by Rodney Brooks[26, 27, 28], or software based as in the work carried out by David Cliff[46], Toby Tyrrell and John Mayhew[175] researchers have been able to produce very simple, but robust agents which can survive in real world environments.

> Insects are not usually though of as intelligent. However, they are very robust devices. They operate in a dynamic world, carrying out a number of complex tasks... No human-built systems are remotely as reliable... Thus I see insect level as a noble goal for artificial intelligence practitioners. I believe it is closer to the ultimate right track than are the higher level goals now being pursued.[46]

2.3.3 Intelligence

What is intelligence? Like the definitions for *agent* and *autonomy*, intelligence means different things to different researchers. To some, an agent may be classed as intelligent if it is able to exist within the its environment and react appropriately to the events. To others, the ability of an agent to create (and execute) plans based on its own knowledge, experience and the current state of the environment in order to achieve its goals is a more appropriate indicator of intelligence.

In some circumstances, the intelligence of an agent is indicated by its ability to appear intelligent to observers. This is the case for *believable* agents (see section 2.3.8 for more information) whose main purpose is to give the illusion of life - whether real or artificial. However, this may achieved by using either of the previous philosophies, or a combination of both, and therefore believability may be considered a possible result rather than a method of achieving intelligent behaviour.

It could be argued that autonomy, when combined with pro-activeness, implies a level of intelligence, but this is not always the case. *Reactive* agents (see section 2.4.2) are autonomous and can react to events within their environment, and are pro-active when there are no events to react to. However, their actions correspond more to instincts than to higher level thought processes and reasoned behaviour. Programmatically, these agents are simple - they store very limited knowledge - if any - and their actions and responses are hard-coded.

This leads on to another important element of agent design: an agent should only ever be as intelligent as it needs to be for the domain in which it is operating. Similarly, the type of intelligence is important to consider; whether it is a fast, reactive intelligence, or a slower more deliberative approach that is required.

> The only intelligence requirement we generally make of our agents is that they can make an acceptable decision about what action to perform next in their environment, in time for this decision to be useful. Other requirements for intelligence will be determined by the domain in which the agent is applied: not all agents will need to be capable of learning, for example.[186, page 28]

One researcher, Ben Shneiderman is unhappy with the idea of "intelligent agents":

> A generally troubling issue is the choice of "intelligent" as a label for much of agent technology. The obvious comparison is to humans. But is such a comparison a good thing? The metaphors and terminology we choose can shape the thoughts of everyone from researchers and designers to members of congress and the press. We have a responsibility to chose the best metaphor for the technology we create.[154]

He continues by listing a number of points which may be summarized as follows:

1. Use of the word "Intelligence" limits our thinking by limiting our frame of reference to human terms,

2. 'Intelligent' implies creativity and adaptability, which in turns implies a reduction in control for the user,

3. The idea of an intelligent computer (or agent) absolves us of responsibility. People already blame computers for their mistakes. If the computers were classed as intelligent, this tendency would increase,

4. Machines will never be people but if we treat them as people, we may end up treating people as machines.

Although Shneiderman makes a number of interesting points, the fact remains that we already have agents which exhibit a limited (in comparison to humans) form of intelligence however we choose to describe it. The word *intelligent*, like the word *agent* is just a convenient label that we can use. Choosing a different word may change our perceptions and expectations, and as a result may change the direction of research, but who is to say that the new word will be more appropriate? Most likely, any new description of an agent will take its place *alongside* existing words such as *autonomous, intelligent, strong, weak,* and *software.*

2.3.4 Adaptability, Personalization and Learning

Depending upon the type of agent and the environment in which it finds itself, an agent's ability to adapt may have differing consequences. These may range from the ability of an agent to search three database systems instead of only two in a certain timescale, through to determining whether of not the agent will be able to survive[6].

> In a changing, unpredictable, and more or less threatening environment, the behavior of an animal is adaptive as long as the behavior allows the animal to survive. Under

[6]It should be noted that for agents, the term 'survive' means to continue to function successfully.

> the same conditions, the behavior of a robot is considered to be adaptive as long as the robot can continue to perform the task for which it was built. Now the survival of an animal is intimately involved with its physiological state and the successful operation of a robot depends upon its mechanical condition. Under these circumstances, it is obvious that one can associate with an animat - whether the term indicates a simulated animal or an autonomous robot - a certain number of state variables upon which the survival or successful operation depends.[117]

In the example of the animat[7] environment, a successful predator would have to learn not to attack a group of animates which may fight back, but would instead adapt and learn to wait until it could isolate its prey. Another example of a robot that is able to adapt to its surroundings would be a military robot, which must be able to use all of the available cover on a battlefield to prevent it from being destroyed by the enemy.

For an internet agent, adapting quickly to its environment is less important, as there are no predators. However, as stated earlier, survival is not the only reason that an agent must be aware of and able to adapt to its surroundings. For example, an agent which is given the task of trading in stocks and shares online must be able to react quickly to any sudden change in value of the shares.

Another type of agent for which survival is not an issue is the Intelligent Assistant. Instead, the form of adaption required for an agent of this type is personalizability. These agents must be able to remember the preferences, habits and personal information of other key agents or users in the environment. For example; personal assistant agents will remember the user's working preferences, internet search agents will remember locations of various resources, and email agents will remember how to deal with the different types of messages.

[7]An *animat* is an agent which is modelled on an animal. Often, the nature of the animal is unspecified, but it has predators, prey which affect its actions.

Chatterbots such as Julia will remember the personal information for the users within the environment. For example, if she is given someone's email address then she will remember it and use it as part of a description of the person when prompted.

One important way that an agent adapts is by learning; sometimes from its own mistakes, other times from observing the actions of other robots, agents and/or humans in its environment. Alternatively, the agent may be given the information directly by the user. For an agent to be effective, it must be able to make sense of this information, whatever its source, and use it to improve its efficiency.

> The machine learning approach is inspired by the metaphor of a personal assistant. Initially, a personal assistant is not very familiar with the habits and preferences of his or her employer and may not even be very helpful. The assistant needs some time to become familiar with the particular work methods of the employer and organization at hand. However, with every experience, the assistant learns, either by watching how the employer performs tasks, by receiving instructions from the employer, or by learning from the other more experienced assistants within the organization. Gradually, more tasks that were initially performed directly by the employer can be taken care of by the assistant.[109, pages 148-149]

In a system where there is more than one agent (see chapter 3), it is also possible for an agent to ask for advice from other agents.

> Additionally, the agent can learn from experience which agents are good sources for suggestions. It can learn to trust agents that in the past have proven to recommend actions that the user appreciated.[109, page 149]

2.3.5 Communication and Social Awareness

Although it is possible for an agent to be programmed with fixed goals and work autonomously without ever needing to communicate, this tends to be rare. The best examples of these systems are the Predator/Prey simulations such as that created by Toby Tyrrell and John Mayhew[175], and the robotic insects created by Rodney Brooks.[26, 27, 28]

More often, agents are created with the intention of communicating - some agents such as Homer in the Sealife simulation[178, 177] only communicate with the user to receive tasks, request information and report their progress, whereas for chatterbots, intelligent assistants, and other social agents such as the Oz agent[12, 11], communication is critical to the success of the agent.

In a Multi-Agent System (see chapter 3) the agents may be required to communicate with each other in order to cooperate to perform goals which cannot be achieved by a single agent on its own. For example, a robot may be too small to move an object on its own, but with the assistance of another robot, the object may be moved. In other cases, such as in swarm systems, (see section 3.2.2) cooperation between the agents is coordinated through changes to the environment such as scent-laying; so called stigmergic signalling.

2.3.6 Environment

An agent's environment can have a significant impact on its chances of successfully completing its goals. Complex dynamic environments can produce complex effects which the agent must be able to respond to in order to survive. However, designing a simulated environment can be a difficult task.

> A major design issue concerned the degree of realism and complexity in the environment. We needed to create a balance between several factors. On the one hand

> we wanted to minimise the time spent on programming the environment and also keep the complexity of the environment within manageable limits so that we could maintain a good understanding of its dynamics. On the other hand we wanted to make the environment fairly realistic so as to avoid the dangers of abstraction apparent in classic AI and other fields... and we also wanted to make the environment fairly complex so as to pose a difficult challenge to any behavioural strategy. Another factor was our desire to make the environment quite realistic so as not to prejudice the testbed by building into it our expectations of the solution. [175, page 264]

For some researchers, the importance of the interaction between an agent and its environment cannot be overstated.

One particular class of agents; *reactive* agents (see section 2.4.2) are very closely dependent upon their environment. Any events within the environment which are detected by the agent trigger simple reactions - instincts - and as a result these agents are able to produce a fast response to any situations which occur.

> Autonomous agents are situated in some environment. Change the environment and we may no longer have an agent. A robot with only visual sensors in an environment without light is not an agent. Systems are agents or not with respect to some environment. The AIMA agent ... requires that an agent "can be viewed" as sensing and acting in an environment, that is there must exist an environment in which it is an agent.[63]

When moving an agent from an environment in which it can function to one in which it cannot, it seems a little extreme to state that the system is no longer an agent unless the core of the agent is changed. For example, a submarine is still fundamentally a submarine when it is not in water... it is just not a very effective one. Similarly, a robot with only a visual sensor in the dark is still a robot even though it will have greater difficulty trying to achieve its objectives.

This scenario hints about another important detail concerning an agent and its environment. Most environments are complex, yet it is often the case for an agent to have been designed with a comparatively limited set of sensors.

If this is the case, then the agent will be completely oblivious to some of the possible environmental effects. For example, in a simulated world there may be heating fluctuations ranging from freezing to boiling, but if the agent is not able to detect these changes then they will only ever effect the agent indirectly - by melting ice, or freezing water.

> An agent's environment provides context and support for its abilities. Being able to hear has no great advantages in an environment that does not support sound.[107]

Essentially this leads to the important point that an agents environment is only ever as complex as its effect on the agent; whether it is a direct or indirect effect.

Another important feature of any real environment is *time*. While time can be slowed down, speeded up, and adjusted as required in simulated environments such as the Sealife simulator created by Vere & Bickmore[178, 177], this luxury is not possible in real-world systems. In other simulated environments such as the Pyrosim[149] fire-fighting simulation, real-time operation is an important requirement for the agents.

> The environment demands real-time action from the Agent. The fire is not stopping. The world is both dynamic and dangerous for Agent Goals (e.g.: survival). The Agent needs to be constantly perceiving the environment and to act accordingly. However, there might not be enough time/resources to perform an exhaustive analysis to decide the "best" action to be taken. Agents might need a mechanism to adapt their response time to environment requirements and a method to balance the amount of time spent

> on environment analysis and action control.[149, page 103]

Everything is dependent upon time and as a result, agents are limited in the planning and reasoning that they can carry out. For example, an agent acting as a computer game character, or as a chatterbot cannot remain motionless and mute while planning a course of action for too long or the illusion of believability will be destroyed. Instead it must be able to plan and react in a similar time frame to a normal human.

While many simulations require that agents react in real-time, there are many other factors of the environment which can affect the success of an agent. As a result, there are many researchers who believe that creating physical robots which operate in the real world is a better way of researching that creating simulations. This is especially true since many of the simulations will produce results which will eventually used in the real world.

> ...we believe ...the world is its own best model ...When running a physically grounded system in the real world, one can see at a glance how it is interacting. It is right before your eyes. There are no layers of abstraction to obfuscate the dynamics of the interactions between the system and the world. This is the elegant aspect of physically grounded systems[26, page 13]

This is in contrast to a model or a simulation:

> A model is always approximate to reality. This means that there will always be aspects that are not fully covered by the model...
>
> It is assumed that the human designer constructs these models. Although this is often done with great ingenuity, such a design is necessarily based on what the designer believes the task will be and what the environment is going to look like. This makes

> model-based systems inflexible and brittle. As soon as there are situations that are not foreseen by the designers they will break down.[161]

By building robots rather than simulations and software agents, researchers are attempting to remove these problems. By making a system which can function continuously within the real world, they can be sure that it is robust.

> A critical problem in the construction of both mobile robots and assembly robots proves to be the handling of uncertainties of the real world. So many systems which worked wonderfully in an ideal simulation world have foundered upon this rocky problem.[111]

In a reply to researchers such as Rodney Brooks[26, 27, 28] who believe that the only way to create robust agents is to model them as physical robots, Oren Etzioni from the University of Washington argues that the UNIX operating system provides an effective real time *software* environment in which to test agents.

> The softbot paradigm escapes these quandaries by committing to full realism at every step. Softbots operate in dynamic, real-world environments that are not engineered by the softbots' designers. In the UNIX environment, for example, other agents (particularly humans) are continually changing the world's state by logging in and out, creating and deleting files, etc. Softbots are forced to cope with changes to their environment (where did that file go?) in a timely fashion. To succeed, softbots have to make sense of the flow of information through their limited bandwidth sensors and respond appropriately.[55, page 2]

This is important since building physical robots is costly and often problematic.

> In principle, mobile robots offer excellent testbeds for AI research. In practice, building intelligent systems that successfully interact with an unpredictable physical environment is a rigorous challenge, given the current technology. The cost of such robots (including laser range finders, sonars, grippers, television cameras, etc.) is non-trivial, and the effort and expertise required to assemble and operate such an apparatus are considerable.[55, page 5]

In Multi-Agent Systems (MAS) (see chapter 3), numerous agents occupy the same environment - whether it is an internet server or simulation, etc. When this happens, the actions of one agent may have an affect on the other agents. It has been found that a number of reactive agents can produce intelligent behaviour - even when the agents themselves are simple and unintelligent. These systems are known as Complex Adaptive Systems (CAS) and are most often based on simulations of insect behaviour such as ant or wasp colonies (see section 3.2.2 for further details).

Another real-world environment which provides a challenge for agents is the Internet (see chapter 4). This is partly due to the fact that large areas of it are constantly changing, yet some areas have remained static for some time. One example of a part of the Internet which may be considered static for the purposes of an agent, is the 'Showroom' section of the Triumph Motorcycles website (http://www.triumph.co.uk). This section is usually only updated each year, or when the company launches a new model of motorbike. An example of a fast changing part of the Internet is an electronic newspaper or other daily news service, such as the BBC (http://www.bbc.co.uk), which would certainly change at least daily, but could be updated at any time depending on what news was occurring.

2.3.7 Mobility

In the terms of this dissertation, mobility refers to an agents ability to migrate from one host computer to another. Agents of this type were initially developed for use in situations where large amounts of information are stored at remote locations on a network. Rather than drag megabytes, or potentially gigabytes of data back across the network to be processed locally when all that is required is an email address; a mobile agent can be dispatched. The agent will travel from system to system searching the data until it finds specific information that it requires. It will process the information and then return with it to the user.

> Given the large volumes of data that are involved, it is desirable to perform as much analysis as feasible at the sites where the data is located and transmit only the results of analysis rather than flooding the network with data. This calls for the use of *mobile* software agents that can transport themselves to appropriate sites, carry out the computation on site and return with useful results.[193]

> Mobile Agents are programs that can migrate from host to host in a network, at times and to places of their own choosing. The state of the running program is saved, transported to the new host, and then restored, allowing the program to continue where it left off. Mobile-agent systems differ from process-migration systems in that the agents move when they choose, typically through a ”jump” or ”go” statement, whereas in a process-migration system the system decides when and where to move the running process (typically to balance CPU load). Mobile agents differ from ”applets”, which are programs downloaded as the result of a user action, then executed from beginning to end on one host.[97]

> Mobile agents offer many potential advantages over traditional approaches. By moving the computation to another host, it is possible to collocate the computation with an important database, allowing high throughput, low latency access to that database. Compared to more traditional client server approaches, mobile agents can avoid transmitting a large amount of data across the network, which is of particular

value when the network is slow or unreliable. The mobile agent can move, with partial results from one server to another until it has accomplished its task, then return to the originating host.[98]

In the early days of mobile agents, there were numerous problems of control, security and trust. Once an agent was dispatched there was no way that a user could communicate with it to obtain progress and status reports, or update its goals. In addition there was no way of knowing whether an agent (or host) was malevolent or benevolent.

In response to these concerns, mobile agent frameworks were developed.

While there is still no way of determining an agents intentions, agent frameworks include security measures which prevent an agent from accessing host facilities which they do not have permission to use. In addition to this, the frameworks also provide services which agents can access. Since these services do not have to be provided by the agent, the agent is smaller and therefore the cost of moving it from one host to another is greatly reduced.

Therefore the success of a mobile agent system depends on the ability of the agent to interact with the environment provided by the host that it travels to. Currently there is a great deal of research into open systems (see section 3.2.1). These are systems which are open in the sense that agents can enter and leave at any time. Consisting of a number of platforms, services and agents, they provide the means for new agents to both advertise services that they provide, and request services from other agents in the system. Open systems are described in chapter 3.

Key to this advancement has been the creation and updating of Agent Communication Languages such as the Defense Advanced Research Projects Agency (DARPA) developed KIF/KQML combination and the Foundation for Intelligent Physical Agents (FIPA) developed FIPA-ACL which allows agents to perform all of the actions needed to participate within the system. The only problem is that in order to implement this functionality, the complexity and therefore the size of the mobile agent must be significantly increased.

While the implementation of mobile agent systems has improved greatly, the original motivation for mobile agents has remained. Information is dispersed widely and unevenly across networks, and moving an agent to where the information resides to process it is often more cost effective than moving the information to the agent.

The uneven distribution of information means that some mechanism is required to locate the server(s) with the required information. For an agent to simply move at random across the network would be inefficient. Instead, some form of *semantic routing* is required. This is where either the agent itself, or a central repository, maintains a list containing the information about the data stored on each server. Researchers at IBM described how this process might work:

> A user requesting specific information or any other service would express his or her needs in (something like) natural language and the query would be transmitted to a consultant agent. The consultant agent would reformulate the natural language query into vocabulary and syntax of the Agent Language. It would then consult its own index and possibly the indices of other consultants to identify one or more servers likely to be able to satisfy the query. The consultant would then forward the query to these servers and the results would be returned directly to the requesting client. . . . Thus the initial query submitted by the user is routed based on its semantic content.[40]

They continue by stating that:

> Although mobile agents certainly facilitate several aspects of this process, there is again nothing here that can be performed exclusively by agents or indeed significantly better than by other means.[40]

Maintaining an index of this type increases the size of the agent which in turn means that the code takes longer to move across the network. Alternatively, this knowledge base could be maintained on the client's computer and the mobile agent would be given the relevant data before the it is dispatched. However moving the services information across the network from the servers to the client's computer would cause the same problems which mobile agents are designed to solve.

The server/services location problem is one of the many problems that must be overcome if an agent is to use software mobility. In each case, every technical problem returns to the balance of functionality verses size of the code. This is due to the fact that there is a maximum size beyond which it becomes impractical to move the agent from host to host. As a result, there is a limited amount of space available for other functionality.

This space is further reduced since mobile agents have to interact with other agents in order to gain access to the resources of the remote system or the agents currently on the system, they must have the ability to communicate. Whilst it is possible for an agent to have a simple communications protocol, FIPA-compatible agents require a comprehensive communications system (see chapter 3.3). Whatever the level of communications ability of the agent, this further reduces the space available for searching and filtering capabilities. In contrast, static agents do not have such limitations since there is no requirement to move the agent from server to server.

Whilst the researchers at IBM conclude that;

> With one rather narrow exception, there is nothing that can be done with mobile agents that cannot be also done with other means. The exception is remote real-time control when the network latency prevents real-time constraint being met by remote command sequences. [40]

This conclusion is not strictly true since a locally installed agent could provide such services. However, apart from this exception, their conclusion is valid - there are no specific tasks that specifically require mobile agents. Despite this, there is a general trend towards the use of mobile agents as part of Multi-Agent Systems (MAS). These are described in detail in chapter 3

2.3.8 Personality, Emotion and Believability

One justification for producing believable agents with accurately modelled emotions, is that users will be able to identify with the agents and as a result be more productive. This productivity may be in terms of the users interaction with the agent as a personal assistant, with the agents as the other characters in a computer game. In each case, the agent/s within the environment must sufficiently realistic that the human user is not constantly aware that they are working with a computer program.

> If humans identify with and accept an agent as human instead of machine-like, they may be more able to trust the agent and better able to communicate with it. This type of agent could then be a personal assistant, a companion to shut-ins, a counsellor, or even a nurse who actually listens to your concerns and attempts to explain things to you and comfort you. But giving a program more than a rudimentary imitation of emotion will not be an easy thing. We don't even understand how human emotions work. This, however, brings us right to the point that solving this problem is a perfect AI task. By trying to model emotions, perhaps we can learn more about them. By learning more about them, we can create more realistic models for use by the agents.[134]

In his paper, The Society of Objects, Mario Tokoro describes the *intimate computer*; a future version of a Personal Digital Assistant.

> Intimacy implies security, peace of mind, trustworthiness, reliability, and respect. The intimate computer is intended to inspire users with such a feeling. It has a

> face, and it understands natural languages, so that it presents you with a completely different user-computer interface from those we are used to today.[172]

Although the intimate computer has not been developed, many of the techniques that it will rely on, are already in use at the moment; albeit in a much simpler form. Currently it is possible to download animated characters such as the Microsoft Agents[10, 118, 77] which can then be customised by the developer. These are the same characters that can be found as part of the Microsoft Office suite of programs - these simple help wizards which are modelled on; a paperclip, a dog, a robot, and change shape to show limited emotions. For example, the paperclip, changes shape to a question mark when it is asking about input. However, anthropomorphism can become a problem for some users:

> The anthropomorphic styles are cute the first time, silly the second time, and an annoying distraction the third time.... Anthropomorphic terms and concepts have continuously been rejected by consumers, yet some designers fail to learn the lesson. Talking cash registers and cars, SmartPhone, SmartHome, Postal Buddy, Intelligent Dishwasher, and variations have all come and gone.[154, page 101]

With the current level of technology allowing high-definition screens to be created, it is possible to create computer generated characters that are realistic, but in some cases, these characters are too realistic.

The "Uncanny Valley" is a problem which was first described by the Japanese roboticist, Masahiro Mori. It occurs when models (whether they were computer generated characters or created via other media) are realistic enough to appear human at first glance, but on closer inspection are artificial. When this happens, the level of empathy from the viewer to the character drops dramatically.

> As characters become more photo-realistic, you start to believe in them more and more. With humans characters, you get to a certain point of realism. What happens is there are characters that are so realistic you want to believe they are actually human. Then you notice their deficiencies. They have very plastic skin or very wooden eyes. All of the sudden they just become creepy. They are like zombie people, rather than appealing computer people. The appeal of the character rises, then drops dramatically, then rises again as you approach photo-realism.[176]

One solution to this problem is not to attempt to make the characters too realistic. By creating a character that is definitely artificial, the "Uncanny Valley" problem is avoided. There are many ways that this can be done; from the simple cel-shaded techniques of traditional animated characters, through to detailed, but non-human entities; such as robots, aliens and other simple items that can be given personality through their actions.

2.4 Agent Architectures

The design of an agent is dependent upon both its application and the environment in which it will need to function. The two main approaches are the *deliberative* and the *reactive* architectures. Each have their advantages and disadvantages. More recently, attempts have been made to combine the two approaches to produce *hybrid* architectures which combine the positive aspects of the previous two approaches.

> Current robotic systems are deliberative (plan and use knowledge representation), reactive or hybrid. Deliberative systems rely mainly on symbolic reasoning and world representation whereas reactive systems are reflective. The speed of a response of a robotic system increases as it becomes more reactive. On the other hand, the

predictive capabilities of the system increase while the system becomes more deliberative. Also deliberative systems depend on accurate and complete world models while reactive robotic systems don't tend to use models at all.[194]

2.4.1 Deliberative Architectures

Deliberative architectures such as Homer[178, 177], IRMA[139] and SNePS[100] are based on the symbolic reasoning approach used in traditional AI systems. The agents created in this way, also referred to as Belief-Desire-Intentions (BDI) architectures, are rational and deliberative: they can reason about their goals and current state of the environment and produce definite plans to achieve their goals. Structurally they are implemented with separate components e.g. Planners, Executors, Knowledge Bases, etc.

The most common way of creating an agent is to provide it with the ability to reason. That is, the agent is able to create plans based on its knowledge (beliefs), in order to achieve its goals (desires). An agents intention is expressed by its creation of a plan. If the agent had no intention of completing the goal, then it would not create a plan.

> ...in the context of BDI agents we may assume that all relevant aspects of the environment of the agent are modelled as mental attitudes of the agent, we may assume that agents determine the course of action based on these mental attitudes.[50]

> Rao and Georgeff ... are usually credited with the first full implementation of an agent technology of this kind. A "Belief-Desire-Intention" (BDI) agent is able to monitor its environment, and maintain a database that symbolically represents the state of that environment (its beliefs). It can operate under the influence of a set of data-structures that refer to states of the agent's environment, or the state of the agent's knowledge of the environment, that it would like to bring about (its desires). Lastly,

> BDI agents can assess whether their beliefs are consistent with their desires and, if not, adopt plans of action (intentions) that are expected to bring its environment or its knowledge into line with them.[62]

However, deliberation is a time-consuming process and therefore unsuitable for environments which change rapidly. The practical implication is that while an agent is deliberating, the environment may change to a point where the beliefs upon which the agent based its plans are no longer valid.

Beliefs

An agents beliefs may come from a number of different sources. They may be based on information from the user, the agents perception of the environment, its interpretation of other agent's actions (in the past have they been; cooperative, uncooperative, hostile, deceptive), etc.

These beliefs constrain an agent's actions by reducing them from the complete list of possible actions that an agent may perform, to the list of actions that an agent may perform (with the possibility of success) at that particular time. This is important in two ways. Firstly, it may reduce the number of possible plans significantly, which in turn reduces the time taken to create the plans and as a result there is less chance of the environment changing as the plan is formed. Secondly, as it ensures that the plan formed and the actions taken by the agent are relevant to its current situation.

> We are familiar with intelligent agents from everyday life because our common-sense accounts of behaviors of other human beings are linguistically couched in terms of this model. Roughly speaking, our everyday account of human activity is centered on the collection from the world through the external senses and its use for attaining goals. The information collected is used by people to update and maintain a set of

> *beliefs* that encode what they know (or think they know) about the world, and these beliefs are identified by their propositional content or by sentences expressing that propositional content, e.g. the belief that *it is raining*.... These beliefs... are time varying collections of elements, and certain invariant rationality conditions are typically satisfied, along with certain conditions relating the states of these components to sensory inputs and action outputs.[147]

One problem concerning an agents beliefs is that some of them are transient: that is, they are dependent to a large extent on the current state of the environment. If the environment changes, then beliefs may no longer be relevant. This may be due to a number of factors; including the action of other agents - whether intentionally, or as an unintended consequence of their own actions. This is most likely in dynamic systems such as the Tileworld simulation[140] in which an agent creates plans to clear blocks from an area. If an agents beliefs are no longer valid, then the plans and goals which are dependent upon these beliefs must be re-considered by the agent.

Desires

An agents desires are its goals. That is, the new state of the environment that the agent desires to bring about. In many cases, the goals are provided by the user - an agent has an implicit objective of attempting to fulfil goals provided by the user. However an agent is also capable of creating its own goals - for example, a robot may create a goal of re-charging its internal battery when the energy levels fall below a certain value. For a software agent, self generated goals may involve exploring the environment.

> The system is guided in its activities by certain *goals* or *desires*, i.e., chosen states or behaviors that the system should attempt to accomplish. These desires or goals may be many, may possibly conflict with one another, may alter over timer, and may be determined and shaped by the environment.[65]

As with beliefs, an intelligent agents desires constrain the agent in its choice of actions.

> An Intelligent System performs actions in order to achieve a particular goal. Action not oriented to a specific goal is meaningless, and this random behavior is not a feature of intelligence. In most cases, a **goal** is described as a *desired property of the world state.* The mission of the system is to transform the current world state into a state where this property is held (goal state). [103, page 5]

Intentions

An intention is a desire which the agent is committed to achieving. This commitment is shown by an agent through its creation of a plan to complete the goal.

> When should an agent stop to *reconsider* its intentions? One possibility is to reconsider intentions at every opportunity - in particular, after executing every possible action. If option generation and filtering were computationally cheap processes, then this would be an acceptable strategy. Unfortunately we know that deliberation is not cheap - it takes a considerable amount of time. While the agent is deliberating, the environment in which the agent is working is changing, possibly rendering its newly formed intentions irrelevant. [187, page 78]

One consideration that must be made when designing a deliberative system is the frequency with which the agent reconsiders the world state, its plans, and the state of its goals. This is a fine balance between spending time in deliberation and checking the current state of its environment to ensure that the goals that it is attempting are still possible and that its plans to achieve the goals are still valid.

If the deliberation process takes too long and the environment changes (see also section 2.3.6), then the agent may have to re-plan, or in some cases abandon its goal. When an agents goals become un-achievable, then the agent should no longer be committed to them. However an agent should not perform some action in order to make the goals unobtainable thereby removing its obligation and commitment. In the paper "Intention is choice with commitment", Cohen, and Levesque[48] gives the example of a fictional household robot, which when asked to fetch a bottle of beer for its owner, smashes it.

> Back at the plant when interrogated by customer services as to why it abandoned its commitments, the robot replies that according to its specifications, it kept its commitments as long as required - commitments must be dropped when fulfilled or impossible to achieve. By smashing the last bottle, the commitment became unachievable.[48]

While this is technically a valid solution for the robot, it is clearly unacceptable for the owner. Therefore there must be an incentive for an agent to achieve the goals that it is given. However, if the robot has a number of concurrent goals, these must be prioritised. For example, if the robot was going to run out of power when collecting the bottle of beer, it should recharge/change its batteries before attempting the task - preferably after informing its owner that the task would take longer to complete.

2.4.2 Reactive Architectures

While it is possible to build *Deliberative* agents which can reason about their goals and own internal beliefs, this is not always appropriate. In rapidly changing environments, such as the real world, where the agent must react quickly to external events, there may not be the time available for an agent to perform many of the time-consuming actions such as planning, and introspection.

Instead, an agent with a *Reactive* or *Behavioral* architecture may be more appropriate.

These differ from the deliberative architecture in that agents have a very limited set of beliefs - in most cases none at all, and instead of explicitly defined goals their actions are determined by a set of behaviours which are triggered by events within the environment. The most well-known reactive architecture - the subsumption architecture developed by Rodney Brooks[181, page 49], is created as a number of behaviors[8] - which are implemented as Finite State Machines (FSMs). In her paper detailing the Agent Network Architecture (ANA), Pattie Maes describes the main characteristics of the reactive, behaviour based architecture:

> ...there are no central functional modules, such as a perception module, a reasoning module, a learning module, and so on. Instead, the agent consists of a completely distributed decentralized set of competence modules (also called behaviors). These modules do all the perception, "reasoning", learning and representation necessary for achieving a particular competence. There are no forms of consistency imposed among the modules cooperate (locally) in such a way that the society as a whole functions properly. Competence modules are directly connected to the relevant sensors and actuators and run all in parallel. This guarantees fast and robust actions from the overall agent.[108, page 115]

However, it is possible for a number of behaviours to be triggered by the same stimuli. Therefore it is essential that agent is able to choose between a number of potentially conflicting actions within a short period of time.

> Some behaviors normally take precedence over others. Some behaviors are mutually exclusionary (i.e. any behaviors which utilize the same motor apparatus for incompatible actions).[15]

[8]each behaviour may be though of as an individual *action* function[181, page 49]

The typically fast-acting, instinctive behaviour is directly suited to modelling biological systems such as the predator-prey simulations, artificial life, and other biological-based systems. In particular, swarm intelligence (see section 3.2.2) systems are a popular application for reactive architectures.

Although this level of stimulus-response between the agent and its environment has been criticised on the grounds that blind response to an event or situation cannot be considered to be intelligence, researchers have argued that responses of this type are analogous to reflexes.

> When there is a rigid relationship between a stimulus and a response in an animal, the response is referred to as a reflex response. Reflex responses provide the animal with protective behaviors. Such responses have been shown to be present in animals which have been isolated from birth and is thus considered instinctive. Reflex responses are elicited independent of environmental factors[6]

> Reflexes allow an animal to quickly adjust its behavior to sudden environmental changes. Reflexes are commonly employed for such things as postural control, withdrawal from painful stimuli, and the adaption of gait to uneven terrain.[15]

Rodney Brooks and his team at MIT have created a simple robotic insect which illustrates this philosophy.

> Squirt is the smallest robot we have built. It weighs about 50 grams and is about 5/4 cubic inches in volume.
>
> Squirt incorporates an 8-bit computer, an onboard power supply, three sensors and a propulsion system. Its normal mode of operation is to act as a "bug", hiding in dark corners and venturing out in the direction of noises, only after the noises are

long gone, looking for a new place to hide near where the previous set of noises came from.

The most interesting thing about Squirt is the way in which this high level behavior emerges from a set of simple interactions with the world.[26, page 10]

2.4.3 Hybrid, or Layered Architectures

A third approach which is becoming more popular is the layered approach. By combining the two previous approaches, it is possible to provide a reactive agent which is able to form long-term plans which responding to events in the short-term.

Hybrid Architecture are a marriage of the reactive and deliberative components of agent modelling and combining them to produce a more powerful model. It will not necessarily make the agent more "intelligent" and will in fact suffer from the same "transduction problem" (of mapping the real world to a symbolic representation) as deliberative approaches. The benefit would be in a well-defined problem domain with "static knowledge" in the environment for the deliberative part, and a reactive part that can handle the environmental events.[174]

The ${}_3T$ architecture[23], Atlantis[64] and the Touring Machines[58, 57] are based on three layers. InteRRaP[187, page 101] has two layers, but there are no definite rules as to the specific number of layers required. Usually, three layers of abstraction are sufficient.[169]

2.5 AT-MAS and Intelligent Agents

The AT-MAS system described in chapter 5 was created with agents that function as intelligent assistants. AT-MAS agents are autonomous in that they are able to receive a number of goals from different sources simultaneously and form plans in order to achieve these goals. Furthermore, the agents are persistent which allows the to gradually build up knowledge about their environment and the other AT-MAS agents within it.

Although the AT-MAS agents owe more to the Deliberative architecture than to the Reactive architectures their reasoning abilities are limited in comparison as they possess no symbolic reasoning abilities. However, they are significantly more intelligent than Reactive agents.

2.6 Chapter Summary

This chapter describes a number of different definitions of agents. Although each are valid for different viewpoints, none are truly effective in explaining the concept. This is due to the fact that the more general descriptions are too vague to be of more than limited use, but the more specific definitions are too limiting.

As a result, many researchers limit their definitions to listing the attributes that agents may possess. Of these, the only attribute that all researchers could agree was essential was *autonomy.* This, and other attributes and features that should be considered when designing agents are also described.

The final section describes Agent Architectures. Firstly, Deliberative architectures are described. These are based around the idea of viewing agents in terms of mentalistic attitudes such as Beliefs, Desires and Intentions.

The second architecture, developed to address the shortcoming of the Deliberative architecture, is the Reactive architecture. This is used for creating simple agents which must respond quickly to the effects of their environment. However, the simple hard-wired programming of these agents and their *action* : *response* and their lack of internal representations means that long term planning is not possible.

The third type of architecture integrates the previous two architectures - Hybrid, or Layered architectures usually consist of three layers; a low level reactive layer which provides the instant responses to dynamically occurring events, a top level layer to provide the higher level processes such as planning and knowledge representation, and a middle layer to coordinate the two other layers.

The final section of this chapter relates the AT-MAS agents to the information presented in the chapter. In this, the agents are discussed in terms of the the general definition which is most suited, the attributes that their possess and their architecture.

Chapter 3

Multi-Agent Systems

3.1 Definition of a MAS

Much of the work of agent theory involves applying the aspects of humanity that we see as important, to the field of computing. Intelligence, knowledge manipulation, reasoning, independence and autonomy are all programmed with a view to creating an effective artificial entity.

As with people, many of the tasks that an agent need to carry out require the assistance of others. These may be tasks that are too large (either geographically, or conceptually), too complex or require knowledge and/or skills that the agent does not possess. In some cases, the nature of the tasks may require cooperation of more than one agent. Another reason that multiple agents may be used is that it is just more efficient than using only a single agent.

3.1.1 Societies of Agents

In human society it is common for groups consisting of people with similar or complementary interests and skills to be formed. These may be sports clubs, book clubs, amateur dramatics societies, debating societies, unions, project teams, etc. Other groups may be formed by individuals through necessity - for example, children in a class at school, or the jury in a trial. In each case, the members of the group adopt the the social norms and roles required by the group or risk exclusion from the group. This is also true for agents.

> A group of agents can form a small society in which they play different roles. The group defines the roles and the roles define the commitments associated with them. When an agent joins a group, it joins in one or more roles and acquires the commitments of that role. Agents join a group autonomously but are then constrained by the commitments for the roles they adopt. The group defines the *social context* in which the agents interact.[83, page 19]

Some groups are more transitory than others. For example, a group consisting a class of school children will have a longer existence than the audience of a play. However, while the class may have a longer lifespan, it can be more dynamic than the audience of the play[1].

While the society dictates the possible roles that an agent can play (depending, of course, on the agents abilities to carry out the actions required by the role), the role played by each agent dictates its position within the organisation and its communication with other agents.

When using the clichd example of contacting a travel agent to arrange a holiday, a client (also referred to as a consumer) would contact the travel agent (broker) passing on the details of the holiday that they required. The travel agent would check with the tour operators (producers) to find out if there was a package deal that could be offered. If there was no package deal, then the travel agent would contact the airline companies to enquire about flights. They would contact

[1]I am basing this observation on my own experiences: since I grew up in Germany and attended a British Forces school, it was a regular occurrence to find new pupils joining my class, and others leaving. Similarly my experience of classical concerts and operatic performances is that people are not generally allowed to enter or leave auditorium while the concert or play is in progress

the tourist board in the area to find a list of hotels in the area, and then contact the hotel staff to find a list of rooms and their prices. Then they would contact the local car hire companies to find out the cost of car hire. The travel agent would then return with a price for the holiday and some form of negotiation would take place between the client and the travel agent. If the client and the travel agent could agree on the price and the other terms then the money would be exchanged and all of the contingent parts of the holiday would be booked.

At each stage of the process there is communication which may lead to cooperation and competition. Cooperation results in the common goal being achieved (in this case, the completion of the transaction), and the competition between the agents is to receive the best possible result - i.e. the lowest price for purchasing, and the highest price for selling.

In the example above, the travel agent acts as a broker - by purchasing the individual parts of the holiday from the producer, and the selling them to the consumer for a slightly higher price. However, brokers cannot charge too high a commission as they are in competition with other brokers to provide the best prices for the same or equivalent products and services. An alternative to completely brokered transactions is the use of matchmakers which recommend producers to the consumer, allowing the consumers to contact the producer directly. Alternatives to the brokering process are possible and these are described later in this chapter.

One important point that should be taken from the above example is that most agents are self-interested and therefore required to be motivated to help each other - either by the promise of assistance with its own tasks, monetary reward, and/or the desire for a common goal to be achieved. In the case of eCommerce systems although there is the common goal of the transaction being completed, each agent will be competing to achieve the best results for itself.

Even in altruistic systems such as the *Optimal Aircraft Sequencing using Intelligent Scheduling* (OASIS)[106] air traffic control system, where cooperation between the agents is essential to the success of the system, the agents still have to negotiate to achieve the best solutions, both for themselves and for the overall good of the system. In the case of OASIS, the agents (representing aircraft) will compete with each other for the use of resources such as runways, and air space.

In all MASs, the individual agents have an incomplete knowledge of their environment[169, page 80]. This may be due to its size or complexity. Even if the environment is small enough for a single agent to manage, the presence of other agents will still add uncertainty to the system. This is because an agent cannot view the internal state of another agent - it is only possible for an agent to reason about the other agents possible actions based on the agents previous actions or the possible rational choices within the environment. Similarly, the agents own motivations remain hidden unless it makes them known through its actions or by communication.

In open systems such as AgentCities[185, 184, 197] which allow unknown agents to enter and leave the system as required, it is not possible to reason about the possible behaviour of different agents as there is no way of judging an agents abilities and motivations and attitudes (is the agent benevolent, hostile, or neutral?). Usually, the agents in these systems are self-interested - they only perform actions which benefit themselves, unless they have a reason to assist other agents. However agents provided by the system are benevolent; operating to ensure the smooth running of the system.

> In particular, the personal assistants do not act benevolently unless it's in their interest to do so. The do not necessarily share information, they do not necessarily do things that other agents ask them to do unless they have a good reason for doing so.[146, page 354]

In contrast, closed systems such as the OASIS air traffic control system, which do not allow agents from unknown sources to enter the system will not suffer from this problem. The agents in these systems are mostly benevolent[187, page 190] and cooperate in order to achieve both their own goals and the overall goals of the system. If unknown agents were allowed access to the system they cause disruption or waste a valuable resources with potentially disastrous results.

One of the key characteristics of a MAS is the level and type of communication available to the agents. It is possible for agents in an MAS to co-exist, and in some cases cooperate, without being able to communicate. However, for the system to be classed as a MAS, the presence and actions of the agents must be able to influence the other agents. This may be directly - through the agents actions and communications with other agents, or indirectly by altering the environment in a way that influences the behaviour of other agents.

This is especially true in the case in systems of animat systems (artificial life simulations) such as the one created by Toby Tyrrell and John Meyhew[175] in which a predators and prey are simulated. Neither will cooperate, but each is aware of the presence of the other and will modify its behaviour accordingly. Therefore there is interaction between the entities, and by definition this is also a MAS.

If there is no interaction - either directly, or indirectly between the agents it is not accurate to call the system a Multi-Agent System. This point can be illustrated by referring to the internet. Although the internet hosts a number of both single agent systems and Multi-Agent Systems, they do not interact and do not affect each other except by co-incidence. Therefore it would be wrong to class the internet as a MAS and all agents within it (even single agents) as part of a Multi-Agent System.

Simply having a number of single agents operating on the same system is not enough. The agents must be aware of some aspect of the other agents presence and allow it to influence them. This may seem a vague definition as it does not require the agents in the system to cooperate, but as the definition of an agent requires that it is an independent entity, there is no necessity that the agents do communicate or cooperate even if they are able to.

3.2 Types of MAS

Multi-Agent Systems can be divided into two distinct categories: open and closed systems. Open systems, often based around the FIPA[2] standards for agent interaction, are designed with the intention that agents from any source can enter and leave at any time without having a detrimental effect on the running of the system. The biggest challenge in developing open systems is in the design of the agent middleware. This is a set of system agents and services that allow agents to find and contact other agents and services. These link agents requesting the services and the suppliers of the services - providing assistance in the form of *brokering* and *mediation* as required.

Closed systems are systems which have been developed for specific applications where allowing access to un-trusted agents could have an adverse affect on operation. They are often used for specific application and use propriety protocols and standards since there is no need for the system to cope with agents from other sources. For example, agents from the Tileworld[140] system created by Pollack and Riguette would be unable to function in another system such as the Touring machines[58, 57] system created by Ferguson.

3.2.1 Open Systems

Open Agent systems are systems in which agents can enter and leave at any time. This means that it is difficult to predict what agents will be present and what their capabilities will be.

> The characteristics of such as system are that its components are not known in advance, can change over time, and may be highly heterogeneous (in that they are implemented by different people, at different people, at different times, using different software tools and techniques).[88, page 6]

[2](Foundation for Intelligent Physical Agents)

> Ideally, Multi-Agent Systems are highly dynamic open systems, with an ever-changing population of agents: new agents emerge (or are created), existing agents die, move, learn/forget etc. The dynamics of such systems are hard to predict. The number of agents in large scale distributed applications such as eBusiness applications (virtual shopping malls and auctions), Internet-wide data warehouses, and navigation systems, can vary considerably over time.[25]

Another factor which can cause complications is that different agents may have been created by different developers and/or different companies. Despite this, they must be able to work together.

> For example, you have a personal digital assistant, you might have one that was built by IBM, but the next person over might have one built by Apple. They don't necessarily have a notion of global utility. Each personal digital assistant or each agent operating from your machine is interested in what your idea of utility is and in how to further your notion of goodness.[146, page 354]

In these systems, there tends to be no specific organisation imposed on agents visiting the system. This is because agents can enter and leave the system at any time. If there was a complex structure, then new agents entering the system would need to be allocated places in the structure. Similarly, as agents leave the system, gaps would appear in the structure which might cause disruption to the system.

The Open Agent System[47, 41, 115] uses a complex hierarchy of blackboards (see section 3.3.3) controlled by a server agent.

> The server is responsible both for storing data that is global to the agents, for identifying agents that can achieve various goals, and for scheduling and maintaining the

flow of communication during distributed computation. All communication between client agents must pass through the blackboard.[47]

More commonly, open MASs such as AgentCities[185, 184, 197], RETSINA[167, 168] and Kasbah[38, 110] consist of a number of agent platform (or agent server) installed on a number of hosts. Each agent platform can host a number of mobile agents (see section 2.3.7) simultaneously. They also provide both a mechanism which allows the agents to migrate from one host to another, and services and/or Middle Agents which agents on the platform may interact[193].

> Agents can exist only by virtue some kind of *agent platform*. Such a platform runs on a relatively small collection of machines and provides basic facilities such as creating and running an agent, searching for an agent, migrating agents to other platforms, and enabling the basic communications with other platforms that host an agent.[182, page 6]

In addition to the platform specific services and agents, almost all Multi-Agent Systems rely on the continuous functioning of centralised middleware. This may be in the form of databases, active directories or Middle Agents. which any agent entering or leaving the system must notify. This allows agents to be contacted whichever platform they are on. Without these directories the system will not function. Figure 3.1, taken from the paper *The Agentcities Network Architecture* by Wilmott et al[185] shows the three central directories which are required by the AgentCities system.

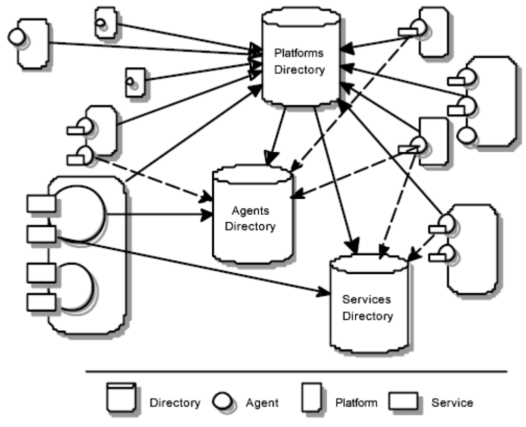


Figure 3.1: The AgentCities Architecture

However, this reliance on a small number of well-known agents and directories means that the system is vulnerable to denial of service attacks and overload caused by the volume of traffic.

> The current Network-support services are very simple and rely on centralized or star topologies with a single point of failure and no means for distribution of authority. The security and robustness of both Network-support services and individual agent platforms are also rudimentary. Each of these areas poses major challenges to be addressed as the Network grows.[185]

> Naming and capability-mediation infrastructure services are indispensable for finding agents and they constitute a fundamental functionality of MASs. The scalability of naming and capability-mediation services is therefore crucial for the overall scalability of MASs [130]

The Anthill[8] system distributes these global directories across each of the platforms in an attempt to improve the robustness and scalability of the system. Anthill is described in section 4.5.5 - Agent Based P2P Systems.

Since MASs can be very dynamic, it can often be a problem for an agent moving to the platform to locate agents which provide specific services. For example, an agent sent by a client to purchase a specific item such as coffee, may not know which agents are able to sell the product. In order to locate suppliers capable of providing assistance, the agent contacts one of the system agents.

> Middle Agents . . . assist in the discovery of service providers based upon a desired service capability description. For example, Middle Agents may help service requesters locate agents that provide stock purchasing services or those that return the ticker price for a given stock. Middle Agents may mediate communication between providers and requesters, . . . and support service discovery.[132]

These middle agents, known in various systems as facilitators, mediators, matchmakers and brokers are able to access the various system directories, and at the very least, return a list of agents capable of supplying the requested services.

> The solution that have been proposed rely instead on well-known agents and some basic interactions with them matchmaking and brokering. Standard agent communication languages (i.e., KQML) even define specific 'performatives' RECRUIT, BROKER, FORWARD) for these behaviors. These behaviors are also common in human open systems as well.[51, page 2]

> Cooperation among the agents of an OAA system is . . . normally structured around a 3-part approach: providers of services register capabilities specifications with a facilitator; requesters of services construct goals and relay them to a facilitator, and facilitators coordinate the efforts of the appropriate service providers in satisfying these goals.[115, page 7]

Matchmakers, Mediators and various types of Facilitator agent all perform different but related functions within a system. In different systems, the same terms are used in different ways.

> As variations on a general theme, matchmaking can follow many different specific modes. For example, the consumer might simply ask the matchmaker to *recommend* a provider that can likely satisfy the request. The actual queries then take place directly between the provider and the consumer. The consumer might ask the matchmaker to forward the request to a capable provider with the stipulation that subsequent replies are to be sent directly to the consumer (called *recruiting*). Or, the consumer might ask the matchmaker to act as an intermediary, forwarding the request to the producer and forwarding the reply to the consumer (called *brokering*).[101, page 93]

> Agent brokers provide addresses of agent servers and support mechanisms for uniquely naming agents and agent servers.[193]

InfoSleuth[13, 61, 179] is another open system which contains a number of different static system agents. Users communicate with their User Agents via a web browser, which in turn communicate with a number of different server agents.

When a user agent receives a request, it passes it to an Ontology Agent which returns the appropriate ontology. Having received the ontology, the user agent then makes a request to a Broker Agent for a Task Execution Agent which is capable of processing the request. The Broker Agent returns the identity of a Task Execution Agent to the User Agent and the user agent submits the User's request to the Task Execution Agent.

When the Task Execution Agent generates a plan, it asks the Broker Agent for a set of agents which can respond to the query. This is made possible since all agents advertise their capabilities to the Broker Agents when they are created by the system. If the query requires the results of more than one agent, then the Task Execution Agent will pass the complete query to a Multiresource Query Agent which will assign the sub-query parts to the resource agents that are capable of completing them. If the query was made by a Multiresource Agent, then it will receive the results from the individual Resource Agents and use them to create a single set of results which it then returns to the Task Execution Agent. The Task Execution agent takes the results and returns them to the User Agent which made the original request. Finally, the User Agent returns the data to the user.

However, this level of complexity is not always required in MASs.

> The overall design philosophy behind AgentScape is "less is more" and "one size does not fit all." The AgentScape middleware provides minimal but sufficient support for agent applications. In addition, the middleware is adaptive or reconfigurable

> such that it can be tailored to specific applications or operating systems/hardware platforms.[182]

As mentioned previously, open MASs must provide a minimum set of services which must remain operational for the MAS to work properly. If different MASs use/provide different services and different protocols for interaction then the agents from one system will not be able to move to another system. In order to allow MASs to support agents from a wide range of sources, different organisations such as FIPA have proposed standards.

> In AgentScape, interoperability between agent platforms can be realized . . . by conforming to standards like FIPA . . . or OMG MASIF . . . These agent platform standards define interfaces and protocols for interoperability between different agent platform implementations. For example, the OMG MASIF standard defines agent management, agent tracking (naming services), and agent transport amongst others. The FIPA standard is more comprehensive in that it defines also agent communication and agent message transport, and even defines an abstract architecture of the agent platform.[128]

3.2.2 Closed Systems

In closed systems, only the agents created specifically for the system are supported. No other agents are allowed. Closed systems are mainly used for specialized applications; such as the Intelligent Manufacturing-Simulation Agents Tool (IMSAT)[121], the OASIS Air Traffic Control System[106]. Other applications might be to control of a nuclear power station, to provide tactical support of army units[170], or to provide maintenance support for a transport network such as the European Railway Online Maintenance Project(EuROMain)[116].

Other closed systems are used for simulations: the SeSAm simulation[95] tool allows its users to create a number of simulations such as ant colonies[3], bee hives and forest fires. Another simulation - Pyrosim[149] - also simulates the spread of forest fires and their control through the use of agents.

> The Pyrosim platform simulates a forest-fire scenario in which a team of Agents (firemen) cooperates to control and extinguish the fire, while simultaneously trying to minimize the overall damage and losses. In Pyrosim, Agents have to deal with dynamic fire-fronts, terrain constraints and their own physical and logistic limitations. Factors such as wind intensity, vegetation density and terrain slopes, are taken into account in simulating the progression of fire-fronts.[149]

By preventing agents from entering the system, it is possible to create agents with specific capabilities. The MASSIVE[96] system was created to model the battle scenes in The Two Towers - the second film in the "Lord of the Rings" trilogy directed by Peter Jackson. Each combatant is modelled as an agent with a different set of attitudes, skills and reactions, making both the individual fights and the overall battles look realistic on-screen. As the system is a closed one, the designers were able to alter the parameters for agents on each side and ensure that the battles went as planned - ie, the correct side won.

For other applications, disallowing external agents allows the designers to impose a structure on the agents. The IMSAT system is one system which has been organized as a hierarchy with the decision making agents positioned at the top of the hierarchy and a number of monitoring agents further down the hierarchy. In other systems, middle agents are also used.

> The system consists of multiple organization levels. The decision-making functions residing at different levels interact with each other during the operation of the system.

[3]These systems, known as swarm systems are discussed later in this section

> Typically, reports flow from lower-levels agents to higher-level agents. The directives flow in the reverse direction.[121, page45]

There are other closed systems which have complex structures, although it is important that the structure relates to the domain and application of the system. It is important to note that middle agents are also to coordinate the actions and interactions of the agents in some closed systems.

Swarm Intelligence

One type of closed Multi-Agent System which is worthy of comment are Swarm Intelligence systems. These are also known as Complex Adaptive Systems (CAS) and are MASs which are populated by reactive agents (see section 2.4.2) with very limited abilities. Instead, it is the agents interaction with other agents through the use of the environment which produce the complex behaviour. In general, these systems are modelled on social insects such as ants, wasps or bees which cooperate altruistically for the good of their society.

> Swarm Intelligence is a property of systems of unintelligent agents of limited individual capabilities exhibiting collectively intelligent behaviour. An agent in this definition represents an entity capable of sensing its environment and undertaking simple processing of environmental observations in order to perform an action chosen from those available to it. These actions include modification of the environment in which the agent operates. Intelligent behaviour frequently arises through indirect communication between the agents, this being the principle of stigmergy. It should be stressed, however, that the individual agents have no explicit problem solving knowledge and intelligent behavior arises (or emerges) because of the actions of societies of such agents.[180]

This system[180] uses a number of digital ants for network routing. When a packet is to be dispatched, an 'ant' is created and send out into the network. As it passes from node to node, the ant will lay a trail of pheromones. As subsequent ants follow the trail, the level of pheromones will be increased. Since the level of network congestion at a particular network node will affect the time that the ant will take to visit and return from its destination, the routes with the least congestion will allow the most ants to travel, and as a result will have a higher level of pheromones deposited. The packets will then be routed along the links that have been marked with the most pheromones.

> Individual ants are behaviourally very unsophisticated insects. They have a very limited memory and exhibit individual behaviour that appears to have a large random component. Acting as a collective however, ants manage to perform a variety of complicated tasks with great reliability and consistency.[151, page 5]

In other ant-based systems, the ants are dispatched to find resources. As they look, the ants wander aimlessly within the environment. When an ant locates the required resource, it will return to its nest laying a pheromone trail. When another ant encounters the trail it will follow it to the resource. As more ants travel from the resource back to the nest, the pheromone trail increases in strength. When the resource is depleted, then the ants, will be unable to retrieve the resource and return to random wandering within the environment. When this happens, the pheromone trail will gradually fade until it disappears completely.

> Reactive agents do not have representations of their environment and act using a stimulus-response type of behavior; they respond to the present state of the environment in which they are situated. They do not take history into account or plan for the future. Through simple interactions with other agents, complex global behavior can emerge.[169]

Other similar systems use swarms of 'social insects' such as ant, wasps or bees to navigate over rough terrain, or perform explorations of unknown environments, or other forms or agent coordination. Since these are simple agents with no means of direct communication, the actions of the agents must be synchronized in other ways. In these systems, the insects lay pheromone trails which can be detected by the other agents.

> Stigemergy is a form of indirect communication through the environment. Like other insects, ants typically produce specific actions in response to specific local stimuli, rather than as part of the execution of some central plan. If an ant's action changes the local environment in a way that affects one of these specific stimuli, this will influence the subsequent actions of the ants at that location.[151, page 5]

It is the interaction of the simple insects through the environment that produces the complex behaviour. As a result, when the various definitions of an agent are considered (see section 2.1), there appears to be a good case for arguing that the agents in these types of system are not really agents. This is because the complex and intelligent behaviour is a product of the system as a whole, and not by the individual entities within the system.

However the individual entities within the system are autonomous, and react independently to their environment, which is one of the few attributes which the various researchers can agree on when defining agents. From this we can say that since the system as a whole generates a form of intelligent, but limited, behaviour and the system contains a number of individual autonomous software entities which interact, then the system is a Multi-Agent System.

3.3 MAS Communication

This section of the thesis deals with cooperation and communication between agents within a MAS. In order to provide examples, a hypothetical Multi-Agent version of the Tileworld[140] simulation created at SRI is used as an example when discussing the need for communication as a requirement for effective cooperation. Jeffrey Rosenschein and Gilad Zlotkin describe such a system:

> A *multiagent* version of the *tile world*, originally introduced by Martha Pollack, is an example of a worth-oriented domain. We have agents operating on a grid, and there are tiles that need to be pushed into holes. The holes have value to one or both of the agents, there are obstacles, and agents move around the grid and push tiles into holes.[146, page 360]

3.3.1 Cooperation without Communication

In this scenario, the agents work independently of each other. There is no communication between them and the only cooperation is due to the fact that they are working towards the same goals. There is great potential for duplication of effort and also for agents getting in the way of each other. However there are advantages if the agents are able to coordinate their actions without the need for communication.

In the system created by Sandip Sen, et al[152] two agents are given the task of pushing a block along a predefined route. However, the agents are unable to communicate with each other.

> These agents can therefore act independently and autonomously, without being affected by communication delays (due to other agents being busy) or failure of a key agent (who controls information exchange or who has more information), and do not

> have to worry about the reliability of the information received (Do I believe the information received? Is the communicating agent an accomplice or an adversary?). The resultant systems are therefore robust and general-purpose.[152, page 510]

Each agent is continuously monitoring the trajectory of the block along the target path. By altering the angle and force which the agents are able to coordinate their effort based of the other agents' influence on the block.

> The most surprising result of this paper is that agents can learn coordinated actions without even being aware of each other.[4] [152, page 514]

The agents have a number of possible goal outcomes, which may possibly conflict. For example, if there were a number of possible goal locations and as a result number of paths that the block could be pushed along, there is the likelihood that agents actions may conflict.

> To converge on the optimal policy, agents must repeatedly perform the same task. This aspect of the current approach to agent coordination limits its applicability. Without an appropriate choice of system parameters, the system may take considerable time to converge, or may not converge at all.[152, page 514]

Using the example of the Tileworld system in which there is no predefined path, it is possible that the agents could cause deadlock by both attempting to push the same tile from opposite directions, and neither of them giving up. Alternatively, the agents may each realise that another agent was also attempting to move the tile and abandon its own attempt. If this were to happen then the tile would remain unmoved.

[4]While this seemingly contradicts the definition of a Multi-Agent System provided earlier (see section 3.1) in that the agents are not directly aware of each other, the agents are affected by each others presence.

In a simple mobile robot simulation, a number of robots may be placed in the same environment with the same goals. If there was no communication between the robots, then their interaction would mainly consist of avoiding each other.

A different system where multiple agents were able to cooperate towards a common goal without the need for communication is in retrieving the pages for an internet based search engine. Each agent would retrieve the contents of a number of URLs not caring whether or not the results had already been retrieved by another agent. Since the URL on the remote server is not changed, and the only problem with updating the same record in the database more than once is the time wasted, then it may actually be more effective to allow this occasional duplication rather than add communicative abilities to the agents.

3.3.2 Nonverbal Communication

As with humans, one of the key features in the design of any agent is that no other agents in the system have the ability to view its internal state directly. As a result, its motivations, beliefs, desires and intentions remain hidden unless made known through its actions or by stating them. However, humans have emotions - expressed through the use of facial expressions - which allow other humans an insight into our current state of mind.

In her paper, "Agents that Reduce Work and Information Overload", Pattie Maes, describes an intelligent assistant which communicates via icons which change as the agents emotions change.

> The agent communicates its internal working state to the user via its facial expressions. These appear in a small window on the user's screen. The faces have a functional purpose: they make it possible for the user to get an update on what the agent is doing "in the blink of an eye". . . . The agents have deliberately all been drawn as simple cartoon faces, in order not to encourage unwarranted attribution of

human-level intelligence.[109, page 154]

Current work in this field is more aimed towards Human-Agent interaction rather than Agent-Agent interaction[135, 11, 160, 191, 109]. Instead, in the Multi-Agent Systems which contain agents with emotions, emphasis is placed on the interaction purely as a way of making the characters in the virtual world appear more believable.

If the agents within an information retrieval or resource coordinating Multi-Agent System, of which there are numerous examples, such as InfoSleuth[13, 124, 123], were able to display very simple emotions, then the other agents would be able to learn which agents were busy (looking stressed out or confused), available for work (looking calm), or hostile and use this information when assigning tasks and making requests of the agents.

Gestures are another form of nonverbal communication - the wave of a hand may be a greeting, agreement, disagreement, an insult, any one of a number of different messages depending upon the gesture, its context and the existing shared knowledge of the sender and receiver of the message[5]. In the book, Embodied Conversational Agents[31, pages 6–11], Justine Cassell considers such gestures and splits them into three separate groups;

Emblems These are gestures with cultural significance - a *V for victory* signal, or its reverse as an insult; *Thumbs up* and *Thumbs down* signals; the nod or shake of the head. All of these gestures are based on a shared culture, and often don't translate between cultures. For example, in Britain and America, the nod of the head indicates agreement, whereas in other countries, such as Greece and Albania, agreement is signalled by a shake of the head.

Propositional Gestures This class of gesture is the nonverbal equivalent of a speech-act. It is a planned gesture - for example, pointing at a location - usually as a supplement to a spoken command such as "put the box there".

[5]For the purposes of this work, Sign Language is not classed as simple signalling as it allows people to communicate effectively using a complete vocabulary

Spontaneous Gestures The third class of gesture is the most common. These are performed involuntarily by the speaker in order to emphasis certain aspects of the conversation. They tend not to be noticed consciously by the speaker or the listener, but they do affect the listeners perception of the speaker and what he/she is saying. In the book, Cassell uses the example of a colleague, Mike Hawley:

> As is his wont, in the picture [not shown] Mike's hands are in motion, and his face is lively. As is also his wont, Mike has no memory of having used his hands when giving this talk... Mike's interlocutors are no more likely to remember his nonverbal behavior than he is. But they do register those behaviors at some level and use them to form an opinion about what he has said... [31, page 6]

Facial expressions and gestures aren't the only form of nonverbal communication that it is possible for an agent to observe. Any action that has an effect on the environment which can be observed may be considered as a form of communication. For example, when a homeowner builds a fence or a wall they are marking their property as having an owner.

If an agent locks a resource for its own use, it is making an observable change to its environment, and is by its action communicating the fact that it requires the resource. In other words, if a resource is not available, then it is currently in use by another agent.

Again using the Multi-Agent Tileworld example, an agent would know not to push a tile that another agent was pushing. However, since the agents would not be able to communicate directly, both agents may form similar plans to push the same tile. However, only one agent would be able to complete its plan and the other agent would have to abandon its plan when it saw the other agent pushing the tile. If both agents, reasoned that the other agent was pushing the tile and abandoned their plans to move that particular tile then the tile would remain where it was.

The use of some form of communication would allow the agents to more effectively coordinate their actions so that they were working together rather than independently. By using a commonly agreed signal the agents coordinate their actions. One way of doing this would be to force an agent to mark the tiles that it was intending to move. An agent would not be able to mark a tile which had already been marked and would be forced to move any tiles that it had marked before marking any more.

One real world example of signalling occurs when driving. The use of brake lights, indicators and reversing lights, combined with a limited number of possible options - speed up, slow down, stop, follow the road ahead, turn off the road - mean that it is possible to predict with a high level of accuracy, the actions and intentions of a competent driver. However, as with any autonomous entity, the signal of intention from a driver, does not always guarantee that the action will be performed (One could signal left and turn right, for example). Therefore, road users have to be alert for unpredictable events.

In general, a signal is a simple action which implies a longer message. However, signalling is a very impoverished method of communication - very little information is passed between the agents, and as a result, this is not always the most appropriate method. For simple systems such as *Swarm Intelligence* (see section 3.2.2) systems, the limited communication is enough to direct the agents. However, while the pheromone trails provides a message which means "There may be food at the end of this trail.", it is not possible to provide any detailed information about the location or any hazards that the agent may encounter on the way.

Another feature of signalling is that it is a broadcast method of communicating. This may be either an advantage or a disadvantage depending upon the situation. For example, when driving a car, it is important that all road users know a drivers intentions and actions. But when an agent is negotiating for the use of a resource, it may be detrimental for an agent if its competitors were to learn of its offer.

For nonverbal communication to be a viable alternative, or even to supplement conventional language, the act of viewing and identifying the facial expression or gesture would have to be as efficient as querying the agent and receiving the response.

3.3.3 Cooperation with full Communication

While acknowledging that indirect communication does exist and is used effectively in a number of systems, it is important to realise the majority of communication in a MAS is direct communication. As a result, the remainder of this chapter will concentrate on the most common form of direct communication: language.

If the agents in the Tileworld example were able to communicate, they would be able to tell each other that they were going after a particular tile. By communicating, the agents would be able to coordinate their actions and ensure that they do not get in the way of each other. The communication would have to be short and succinct since the various holes into which the tiles are pushed open and close at random, but this would allow the agents to coordinate their efforts and avoid duplication of plans.

Additionally, there are a number of Multi-Agent Systems where direct and complex communication is not just a desired feature, but a prerequisite. An example of this would be a system where each of the agents controlled a resource - such as the multi-media channels for a particular user. For the users to contact each other, the agents would have to communicate and cooperate to establish a communications link. See [104] for an example of a Multi-Agent based conferencing system.

Coordination Between Communicating Agents

Having located groups of agents which may exist - possibly through the use of a mediator or facilitator (see 3.2.1), the agent must negotiate with them in order to purchase their services and/or products. However, in order to do this, a standard *conversation* (see section 3.3.3) is initiated.

The most common type of transaction conversation is the *Contract Net* protocol[187, pages 194–196].

> The activity modelled by this protocol, termed Contract-Nets, is task sharing, where agents help each other by sharing the computation involved in the subtasks. More precisely, an agent who needs help, decomposes a large problem into subproblems, announces the subproblems to the group, collects bids from the agents, and awards the subproblems to the most suitable bidders. In fact, this protocol gives us the best negotiation process for dynamically decomposing problems because it was designed to support task allocation. It is also a way of providing dynamically opportunistic control for the coherence and the coordination of agents.[37, pages 49–50]

In some cases, the *most suitable bidder* is not the agent which agrees to complete the problem for the lowest price. Other considerations are also relevant such a the timescale and the level of trust between two agents (see section 3.3.3). Often when a bid is made, it is not always accepted. Negotiation may take place with offers being exchanged between the agents until agreement is reached, or until either agent ends the negotiation by declining to make a further offer. Tuomas Sandholm and Victor Lesser describe a variation of the Contract Net protocol which allows for these occurrences in their paper *Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework*[148]

Other alternatives to the Contract Net are also possible. In both the HEARSAY II[143] and IMSAT[121] systems, the coordination is performed through the use of *blackboards*. These are shared data areas which act as a communications area for the agents. IMSAT agents towards the lower end of the hierarchy place information about onto the blackboard where it will be used by agents further up the hierarchy.The Open Agent Architecture (OAA) is another system which uses blackboards to coordinate the actions of agents.

> Individual agents can respond to requests for information, perform actions for the user or another agent, and can install triggers to monitor whether a condition is satisfied. Triggers may make reference to blackboard messages (e.g. when a remote computation is completed), blackboard data, or agent-specific test conditions (e.g. "when mail arrives...").[47]

The Open Agent Architecture (OAA) extends this concept by creating a hierarch of blackboards which allow requests to be passed between the blackboards.[47] This is similar to the Super-peer/Reflector node concepts used in P2P networks (see 4.5.3 for details).

Speech Acts

Most, if not all communicating agents make use of a communication concept known as speech acts.

> The main approach to agent level communication in the literature like KQML or ACL of FIPA are based on speech act theory. One of the basic insights of speech act theory is that the communication of a message not only consists in making a statement, but in a more broader sense constitutes a communicative action: making a promise, begging, informing, etc. are all different kinds of communicative actions.[73, page 77]

In practical terms, this means that the main agent languages are based around a relatively small number of *preformatives* - verbs which define the actions which are to be carried out: inform, promise, request. As a result, the agent's conversations tend to be very direct.

Agent Communication Languages

KQML A first attempt to produce an ACL that was both standard and general came out of the DARPA knowledge sharing initiative. The Knowledge Query and Manipulation Language (KQML) was originally devised as a means for exchanging information between heterogeneous knowledge systems. However, because of the generality of its high level primitives and its message orientated structure, KQML also functions well as language for agent communication.[52]

Due to the fact that KQML has been used in a wide number of different systems, many developers have used the ability to add their own performatives. This is allowed by the language providing that any existing performatives that are implemented are implemented to the common standard. Another important point is that an agent need not implement all of the core performatives, so it is possible to have a group of KQML-speaking agents which have an almost completely non-standard set of performatives.

> Although KQML, has a predefined set of reserved performatives, it is neither a minimal required set nor a closed one. A KQML agent might choose to handle only a few (perhaps one or two) performatives. The set is extensible; a community of agents might choose to use additional performatives if they agree on their interpretation and the protocol associated with each. However, an implementation that chooses to implement one of the reserved performatives must implement it in the standard way.[102]

This diversity of application is important in that it allows a large number of different problems to be considered. This means that the language is continuously being developed.

> The assumptions made about the required behaviour of KQML agents were very weak, and the resultant semantics of KQML messages were much more permissive than FIPA-ACL. As is now well know, this permissiveness allowed wide latitude

> in KQML implementations, and contributed to the proliferation of different and incompatible KQML dialects.[53, page 5]

An unfortunate consequence of this many dialects of the language existing is that few KQML-speaking agents will be able to communicate with agents outside their immediate group. However, this has been noticed and is not considered to be a problem:

> Although existing KQML implementations tend not to interoperate, this is mainly due to the lack of a real motivation. Agent-based systems research is still at an early stage, and there has been no benefit to individual research groups in focusing on interoperability issues.[102, page 47]

The other main drawback of a language that is this flexible is in that it cannot be formally specified in the same way that the other main ACL can be. Relatively recently, in an attempt to address these concerns, a number of researchers looked at re-writing KQML, and came up with a second version of the language which can be specified in a similar way to the FIPA-ACL.

FIPA-ACL The Foundation for Intelligent Physical Agents (FIPA) is the controlling body that created and now defines standard for the FIPA-ACL. This language was created after KQML and was intended to address the main concerns of developers.

One of the results of this is that the language has been formally specified and as a result is much stricter about what is allowable and what is not. However, the FIPA specification covers more than just the language and as a result, a developer wanting to create a FIPA compatible agent must implement a full BDI (Belief, Desire, Intention) Architecture - which is a non trivial task.

> As of spring of 1998, there were no published, deployed systems claiming to use FIPA ACL.[102, page 50]

Since this time, FIPA has gained widespread acceptance within the MAS community and a number of important systems such as RETSINA[167, 168] and AgentCities[185, 184, 197] have been developed which use it.

The designers of one agent system describe one of the features of KQML which they used in their system. It should be noted, that the FIPA-ACL also allows new performatives to be defined, however this is can only be achieved by combining existing performatives since FIPA-ACL has been formally proved.

> In FIPA-ACL, the set of primitives is smaller than in KQML (but new performatives can be defined by formally combining primitives) and this set also includes assertives or directives as in KQML.[36]

Other Languages In addition to KQML and FIPA-ACL, many developers have written their own languages; these include DAISY[29, pages 190–192], sACL[137, page 164], AGENT-0[155], and KAoS[66, 74]. However the agents within these systems will be limited to communicating with other agents from the same type of system.

Conversations, Conversation Policies

Humans by nature are communicative, and it is fair to say that a large percentage of what we, as individuals say, is aimed at emphatic rather than information transfer. We talk of the weather, our opinions, and any other things that we think of. Agent communication, due to the comparative simplicity of agents when compared to humans, tends to be limited in scope but more directed. In contrast, conversations between agents are short and very focused. The domain is task oriented and the goal of the communication is to exchange as much information as required, or reach agreement in the shortest possible time.

A conversation among agents is an exchange of messages made towards the accomplishment of some task or the achievement of some goal. In its simplest form it is a sequence of messages in which, after the initial message, each is a direct response to the previous one. More complicated structures occur when subdialogs are needed[119, page 147]

> . . . an agent will perform a certain communicative act with a certain expectation of getting a reaction of the receiving agent. For instance, a request for information is send with the expectation that the other agent will give that information or tell that it does not have that information (or does not want to give that information).[52, page 10]

For agents, these conversations need to be as short as possible while still allowing all of the relevant information to be shared. For many agent systems, due to the number of communication acts - questions/statements and responses - that are possible between agents and the fact that statements may refer to data obtained from the previous responses, the sequence of messages are in many cases, too complex to be chosen on an Ad-hoc basis.

> . . . for powerful ACLs, there is a many-to-many mapping between externally visible messages an agent produces and the possible internal states of an agent that would result in the production of the message. This would be a significant but manageable problem, except that agent interaction does not consist of agents lobbing isolated and context-free directives to one another in the dark. Rather, the fact that problems of high communicational complexity may be delegate to agents dictates that those agents must participate in extended interactions.[67, page 119]

Instead, a number of researchers have developed Conversational Policies in order to try and alleviate this problem. A conversation policy is a way of reducing the number of message options, ensuring that the agents remain focussed on the conversation and goal currently being pursued.

> Any public declaratively-specified principle that constrains the nature and exchange of semantically coherent ACL messages between agents can be considered a conversation policy. A given agent conversation will typically be governed by several such policies simultaneously. Each policy constrains the conversation in different ways, but there is no requirement that every policy be relevant or active in every interaction.[67, page 123]

> When a set of communication acts among two or more agents is specified as a unit, the set is called a *conversation*. Agents that intend to have a conversation require internal information structures that contain the results of deliberation about which communication acts to use, when to use them, whom the communications should address, what responses to expect, and what to do upon receiving the expected responses. We call these structures *conversation specifications*, or specifications for short.[136, page 133]

The KQML specification already has support for response-type messages through its use of the *in-reply-to* field. However Mihai Barbuceanu and Mark S. Fox[9]propose that a *conversation-id* field is added to allow a conversation policy to be included with the message. This idea is echoed by Marian Nodine and Damith Chandrasekara[123] who explain one of the potential problems in identifying the current conversation.

> Conversation Identification is an issue that is usually ignored in ACLs and often causes problems. For example KQML supports the chaining of messages using the `:reply-with` and `:in-reply-to` fields, which are predicated on a request-response model. Therefore one would have trouble chaining together messages in conversations that do not follow that model. Suppose an agent were to submit a query involving a large computation. At some point, the agent wishes to cancel the query or possibly

amend the query to provide additional restrictions because it is taking too long. However, since no response has been received there is no incoming message to chain the **discard** message to. Therefore it is evident that it is necessary to be able to specify which conversation a message belongs to.[123, page 9]

Ontologies

The command 'Open a window' a window causes a different action to be performed depending upon whether the context is a room or a computer system. In some cases, the context can be deduced, from the environment in which the conversation is taking place, but in other cases the terms in the request may be ambiguous or unclear to the other agent.

In human conversations, new words can be added to a language and words can have their meaning changed - just by people using them in a certain way. It is the way that language evolves, and it the reason that dictionaries are revised and updated on a regular basis. If a new word is used in human conversation and queried by one of the participants, the conversation may be paused while the word is explained. However agents are not usually sophisticated enough to be able to learn new words and concepts during a conversation. In addition, since the agent conversations are carried out in part fulfilment of a task, the agents are unlikely to have time to learn new words.

In some applications, it is important to define constant values such as the current rates of tax which must be applied to commercial transactions. For other agents, scales such as units of measurement - whether calculations need to be imperial or metric. If one agent was using one scale, and another agent was using a different scale then the effects could be disastrous. Although the systems involved were not agent systems, the disastrous 1999 NASA Climate Orbiter mission to Mars, illustrates what may happen when two different scales of measurement are used.[87]

So to provide a common point of reference, many designers have equipped their agents with the ability to understand ontologies. By referring to the same ontology, agents can guarantee that they are using the same units of measurement, the same currencies, and the same timescales. But an ontology is more than this as it defines everything that may be different between agents within a domain.

> Generally, A specification of the objects, concepts, classes, functions and relationships in the area of interest. For a given area, the ontology may be explicitly represented or implicitly encoded in an agent. More specifically, to support the sharing and reuse of formally represented knowledge among AI systems, it is useful to define the common vocabulary in which shared knowledge is represented; a specification of such a common vocabulary for a shared domain of discourse is called an ontology.[181, page 598]

EDEN (Environmental Data Exchange Network) is one system that makes use of ontologies;

> The ontology used in the EDEN pilot project focuses principally on the relationship between contaminated sites, the wastes that cause the contamination, and technologies used to remediate specific kinds of contamination in specific media at each site.[61]

Sincerity, Deception and Trust

One of the key concepts of Speech Acts is the notion that the speaker is sincere. In other words, if the speaker makes promise, then he/she is under an obligation to fulfil the promise. So an agent should not make commitments that it does not intend, or is not able to keep. Unfortunately, there is no definite way for one agent to tell if another is attempting to deceive it.

In general, in open environments, agents cannot safely determine whether or not another agent is sincere.[158]

The designers of the GOLEM[32] system have pointed out that there were three types of deception possible from one agent to another.

1. Deception about capabilities
2. Deception about personality
3. Deception about goals and plans

In general, an agent has very little to lose from making false claims, but it is possible for it to benefit from deceiving other agents. Sometimes, deception is not only possible, but for the smooth running of various processes, is required.

> There are certain circumstances where sincerity is certainly desirable, for example commercial activity, but there are others where it is not, e.g. certain role playing games. Examples of the latter include the board-game Diplomacy, where players connive their way to European conquest, and in an electronic setting, Multi-User Dungeons, where some players act by tricking their way into the confidence of another, before assassinating them. Even in the former case, though, while desirable, sincerity is routinely flouted (only we do not call it lying, we call it 'marketing' or 'advertising').
>
> More seriously, many Multi-Agent Systems are being developed for some kind of electronic commercial activity and underpinning these applications is the concept of negotiation. The idea of sincerity is somewhat compromised under these circumstances. ... For example, in negotiation an agent may bid some price as its opening

bid, but be willing to accept some other price to conclude a deal. To be more precise, much human social activity requires the freedom to be 'economical with the truth'.[138]

3.4 AT-MAS and Multi-Agent Systems

The AT-MAS system is a Multi-Agent System in which the agents communicate directly without the need for Brokers, Facilitators or Mediators. In addition, the agents are benevolent and cooperate as required eliminating the need for negotiation. As a result, the AT-MAS language used is simple in comparison to KQML and FIPA-ACL.

3.5 Chapter Summary

This chapter details Multi-Agent Systems (MAS). As in the previous chapter, it begins with a definition. Here they are is defined as a systems in which the actions performed by each agent can affect the other agents in the group. This interaction may be direct or indirect. In these systems, each agent adopts a role within the group.

Following this, the types of MAS are described. Firstly, open systems are described. These are systems where agents are allowed to enter and leave at will. They often consist of an agent platform and a number of system agents and services which are provided by the platform. Mobile agents enter the system, move between the platforms as required, and then leave. In order to enter the system, the agent must complete a registration process. This information is stored in a global database or directory (this varies from system to system). However, the use of global datastore introduces a single point of failure into the system. Other open MASs such as InfoSleuth[13, 61, 179] contain static agents which may appear and disappear. However, as with mobile agent MASs, these agents also rely on system agents in order to locate other agents.

In contrast, closed systems do not allow agents to enter or leave. They tend to be developed for specific applications, such as simulations and process control. One specific type of system - swarm intelligence - is described.

Next the types of MAS communication and cooperation are discussed. These vary from *cooperation with no communication* through to *cooperation with complete communication.*

Finally the AT-MAS system discussed in context of the information presented in the chapter.

Chapter 4

The Information Environment

As highlighted in section 2.3.6, the environment in which an agent operates has a direct effect on its success or failure.

With over 11.5 billion web pages[70] and rapidly growing, the internet can accurately be described as the largest and most significant repository of knowledge and information on the planet. However, like the fictional book *The Hitch-hikers Guide to the Galaxy* described in the eponymous novel[1], the Internet has "many omissions and contains much that is apocryphal, or at least wildly inaccurate".

In the prefix to his book "An Introduction to MultiAgent Systems"[187], Michael Wooldridge explains the problem of providing reliable references to internet based documents.

> It would be hard to write a book about Web-related issues without giving URLs as references. In many cases, the best possible reference to a subject is a web site, and given the speed with which the computing field evolves, many important topics are only documented in the 'conventional' literature very late in the day. But citing Web pages as authorities can create big problems for the reader. Companies go bust, sites

> go dead, people move, research projects finish, and when these things happen, Web references become useless. For these reasons, I have therefore attempted to keep Web references to a minimum.[187]

As a result of the unregulated ad-hoc organisation of this resource, locating specific information can be problematic. This problem is made worse by the ever-changing nature of the internet; the web pages represented as links appear and disappear without warning. In other cases, links may remain current, but it is the information that becomes out of date.

Some kinds of information, such as scientific facts remain relevant for a very long time. It is very rare that the laws of physics become obsolete, but occasionally a theory is disproved and a new theory replaces it. Other types of information such as share prices and news stories have a very limited lifetime. Seconds can mean that a share price is outdated - often with disastrous results for stockbrokers and the people concerned with share trading.

Unfortunately, there are no guarantees that information retrieved is the latest, most up to date version. This problem is made worse by the fact that there is no obligation to label information as either current or historical, and no obligation to remove out of date information either. In 1999, researchers investigated the problem of retrieving accurate information from the internet and found the following:

> People searching the internet for information are more likely to find the correct answer than a wrong one. The catch is that you're most likely not to find an answer at all. Researchers at Ohio State University in Columbus *(Reference and User Services Quarterly, vol 38, p360)* used search engines to carry out keyword searches on 60 simple questions such as "What is the population of Columbus, Ohio?", and found that 64 per cent of Web pages either didn't contain the answer or no longer existed. Of the remainder, 27 per cent had the correct information. The others were wrong.[122]

This problem is due to worsen since the number of online publications is continuously increasing. This trend looks set to continue.

> Soon the volume of data in scientific data archives will 'vastly' exceed the information - journal articles etc - in current commercial databases. Even now, 'printed' information constitutes only 0.003 per cent of the total information content stored. What is more, future journal articles will contain references - and links - to particular data sets within some of the vast repositories that already exist.[72]

Further problems are caused by both the wide range of languages [1] and the wide range of formats that the information is published in. As well as being in either HTML or one of its many variants (shtml, sgml, xml, dhtml) the information may be in a completely different format and either embedded into the page, or available as a downloadable file. Possible file formats include; pdf; mp3; mpeg; gif; jpg; avi; wav; png; xls; doc; ps; tar.Z and zip to name just a few.

> ...even though Google, at the time of writing , runs a cluster of 10,000 machines to provide its services, it only searches a subset of available Web pages (about 1.3 x 10^8 to create its database. Furthermore, the world produces two exabytes (2 x 10^{18}) bytes each year but only publishes about 300 terabytes (3 x 10^{12} bytes) i.e. for every megabyte of information produced, one byte gets published. Therefore, finding useful information in real-time is becoming increasingly difficult.[171, page 30-31]

Although search engines have been developed in an effort to make searching easier, they are limited, not only by the sheer scale of the problem, but also by the way that the bulk of the online information is structured. This information, often referred to as either the *Deep Web* or the *Invisible Web*[18, 164], is stored in databases and the web pages containing it are created

[1]A study carried out in 1999 found that English is the most popular language for Internet publishing and is used for 80% of the published documents. However 43% of the online population do not understand English.[131]

dynamically from a template when the page is accessed. As a result, a small number of templates can be used to generate a massive number of pages.

The use of dynamic pages also allows the site to be tailored to the individual user. The disadvantage of this, is that the majority of pages cannot be indexed by external search engines and therefore searches must be conducted from within the site using whatever facilities are provided.

> Sixty of the largest Deep-Web sites collectively contain about 750 terabytes of information – sufficient by themselves to exceed the size of the surface Web forty times.[18]

Within recent years the Internet has become home to a diverse range of applications which exist outside the realm of search engines. The increasing popularity of these alternative applications and the information that they manage, means that the World Wide Web is becoming increasingly less important in comparison. However, web browsers frequently provide the interface to these new technologies.

The World Wide Web, Web Services, Grid Computing, Peer-to-Peer networks and open Multi-Agent Systems all have the potential to interact so gradually the interfaces between them are becoming blurred. One key development that assisted this, was the creation of XML (Extensible Markup Language) which includes meta-data in the form of an XML Schema that allows the information to be more easily processed by other applications.

4.1 Client/Server

The Internet is the most well known example of the client/server topology. In it, the client application (in this case referred to as an Internet/Web Client/Browser) connects to a Web Server and requests an HTML page from one of the web sites that it is hosting. Multiple clients can connect to the same server/web site simultaneously. However, there are practical limits to the number of clients that a server can support at any one time. This means that during periods of high demand it is possible for the server to become overloaded and the web sites that it hosts to become swamped.

When a server becomes unavailable due to the deliberate actions of others, it is known as a Denial of Service (DOS) attack. The same effect may also be caused by a surge in the number of genuine visitors to a web site. This is known as the Slashdot effect[112] an is often caused when a web site on a small server or reachable from a low bandwidth connection is linked via a more popular web site. One example of this occurred when slashdot.org ran an article about the music sharing program ”Mojo Nation” which caused an increase in download requests from 300 copies per day to 10,000 per day and completely overloaded the server.[183]

For sites which regularly receive a high volume of requests, clusters of servers may be used in order to reduce the chance of overload during periods of very high demand. The popular search engine Google[76] uses a cluster of 10,000 computers in order to run its service.[171, page 30]

Clients may also request data from a number of different servers at the same time. This occurs most often when a web page contains links to images or program code which resides on a different server from the one which is hosting the page which references it. While it is possible for a client to connect to many servers and for a server to process the requests of many clients, it is not possible for a component to communicate directly with a component of the same type. I.e., Clients cannot connect to other clients, and servers cannot connect to other servers.

It is possible for a number of different clients connected to the same server to communicate via the server providing the server software allows it. This technique is used in numerous Internet applications such as chatrooms and online games.

In the past a client request resulted in the download of a simple HTML page - possibly with images. More recent versions of both Internet client and server software mean that the that the data returned by the server is more rich - often containing multimedia files, XML data or program code such as javascript code, java applets and/or ActiveX controls to be executed by the client. However, client applications are not restricted to Web Browsers; early internet agents such as Bullseye[85, 86] connect directly to a wide variety of servers and download data as though they were browser based clients. This becoming increasingly common since most modern languages contain libraries which allow programmers to access the HTTP protocol engine.

Similarly a web page may contain a call to another type of server-based program such as a *Web Service* or Common Gateway Interface (CGI) Script. These are not downloaded to the client application, but are instead, activated when data is submitted from the client. When this happens, the results are generated by the program on the server and then returned to the client.

4.2 The Semantic Web

The Semantic Web, proposed by Tim Berners-Lee (credited with creating the original World Wide Web) is an attempt to bring structure to the data stored online through the use of meta-data. The purpose of this is to allow applications - such as intelligent agents (see chapter 2) to extract content information easily from a page.

> Most of the Web's content today is designed for humans to read, not for computer programs to manipulate meaningfully. Computers can adeptly parse Web pages for layout and routine processing – here a header, there a link to another page – but in general, computers have no reliable way to process the semantics ...
>
> The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users... all this without needing artificial intelligence on the scale of 2001's Hal or Star Wars's C-3PO.[19]

In order to do this, the author of the page includes hidden tags within the page which define key data elements. For example, the text of a page may include the following:

> . . . my postcode is<Postcode>AB12 3CD</Postcode> and my age is <Age>36</Age>. . .

However, in addition to the tags highlighting the appropriate data fields, the author must create a RDF (Resource Description Framework) which describes the relationship of key elements by using a triple consisting of *element association element*, e.g.

> <age><is an><integer>
> <M. Gill><is a><student>
> <M. Gill's email><is an><email address>
> <M. Gill's email><belongs to><M. Gill>

By using these two sets of tags, the designers intend to provide structure to the Web - a somewhat ambitious task, considering the amount of data currently in existence. However, by converting key repositories of data, it will be possible to convert a useable subset of the Web to this format. Though some doubts have been expressed as to whether encoding data with the content of a web page is the best route to take.

> If the aim is really to make databases more accessible via the web then perhaps we should start by having more portals that point to databases and simply give them better web based front ends. It is certainly more cost effective to load data into a database than into web pages and it is certainly not beyond the wit of man to devise a graphical query format tied to the conceptual model which defines the database structure.[162]

It seems likely that a large amount of the pages created for the Semantic Web will be generated dynamically using fields from a database and a standard template. As a result, there will be little cost to the author of the system since the template can be re-used for each database record.

However, there may be some pages in which the data is uniquely structured and cannot be stored easily in a database. In some cases, the amount of meta-data to be included may either be too great to manage easily, or too trivial for people to be concerned with. Therefore, the tools must be available to ease the process of including the meta-data into future pages to a point where little or no additional effort is required. Otherwise, meta-data will simply not be included.

Since there will be the same lack of regulation as for the existing Web, then it is very likely that the Semantic Web will be plagued by the same problems of accuracy and omissions that the World Wide Web is currently suffering from. The eventual success of the Semantic Web will be dependent upon how well these gaps in the data can be dealt with.

4.3 Web Services

In the same way that the Semantic Web is intended to add structure to web pages, *Web Services* provide a standard for server-based programs such as the databases and other applications called by a client application.

As with any other server based program, when the client connects and makes a request to the server, it will return the results. However two fundamental differences exist; firstly, all communication between the client application and the web service must be performed using XML conforming to service interface, i.e. the WSDL (Web Services Description Language), and contained within a message conforming to the SOAP (Simple Object Access Protocol) standard.

Secondly all public Web Service must be registered with one of a number of central repositories such as the UDDI (Universal Description, Discovery, and Integration) Repository which can be searched by either a developer, or a program. Each public UDDI is linked, and all use the same protocol so that searches can span a number of repositories. Alternatively it is possible for a company or organisation to create its own private UDDI.[159, pages 96–100]

These two requirement combine to produce a framework which allows clients to easily locate and access independently created Web Services. As a result, Web Services are often created to support other technologies from some Multi-Agent Systems (see section 3.2.1) through to a number of Grid Applications (see section 4.6).

However, for some, Web Services are too complex for the objective that they are trying to achieve. When speaking at the *Free and Open Source Software Developer's meeting* in Brussels, Andy Oram, an editor at O'Reilly & Associates made the following comments:

> Web services are hot right now, but their infrastructure has turned out big and scary, which is not what the web was meant to be.... SOAP tries to cover every eventuality. You hardly get a chance to say what you were going to say by the time you've said everything you have to say *about* what you're going to say. And in trying to solve the presence and discovery problems, Microsoft and IBM and others have created an enormous superstructure resembling CORBA or COM, all implemented between angle brackets.[127]

4.4 Client/Peer Oriented

One of the distributed processing systems which has gained popularity in recent years is the seti@home[82] system which is used to process data from radio telescopes in an attempt to identify signals which may indicate the presence of extraterrestrial life. The system is implemented as a screen saver which, when active, downloads data from a central server. The data is then processed during the times when the user's computer is idle and then the results are uploaded. The process is then repeated for new data.

Due to the success of the concept, numerous other systems have been created such as the "Fight

Aids at Home" system[78][2]. Often these systems are confused with peer to peer (P2P) systems (see section 4.5) such as Napster and Gnutella. Like P2P systems, these harness the power of the client computers - in this case to provide the computing power for a number of high profile processor-intensive projects.

Although these systems came to prominence at the same time as the early P2P systems and are sufficiently different in concept from traditional web applications in terms of design to be associated with the P2P systems in the minds of many people, they cannot be classed as P2P. The reason for this is that in a P2P system, the clients are able to communicate directly with each other. In the seti@home system, and all other systems of this type, the clients have no direct contact. Instead, seti@home uses client/server technology: a number of separate clients contact a single server even though the bulk of the processing carried out is performed by the client rather than by the server.

One phrase that has been used to describe these systems is *peer oriented*[166], however this phrase does not seem completely accurate since the clients in the system do not have any contact with each other as peers as in a P2P system do. Therefore I would suggest that the terms *client oriented* or *client-based processing* systems are more appropriate.

4.5 Peer to peer (P2P)

Peer to peer (P2P) computing is a concept that the majority of its users have probably never heard of, but mention *music downloads*, *file sharing*, *Napster[81]* or, more recently, *Gnutella[79]* and the picture is likely to be very different.

[2]In addition to these systems, readers wishing to participate in many of the various distributed computing projects available are advised to visit the distributed.net web-site[75] which provides access to a number of other projects as well.

P2P computing isn't all that new. The term P2P is, of course, a new invention, but basic P2P technology has been around at least as long as USENET and FidoNet - two very successful, completely decentralized networks of peers. P2P computing may be even older . . . The bottom line is that many of the people using P2P applications today weren't even using computers when the first P2P applications appeared.[166]

While these applications have been beneficial, they haven't been as popular or as well known as the current of P2P systems. The launch of Napster in 1999[4, page 103] generated massive interest both from ordinary users of the system and also researchers.

. . . network traffic measurements at the University of Wisconsin suggest that in the period of April 2000, Napster-related traffic represented the 23% of their total network traffic, while at the same time web-related traffic only accounted for 20% . . . Although Napster traffic has been reduced . . . the percentage of peer-to-peer traffic (in total) has actually increased. For example, [more] recent measurement[s] from the University of Wisconsin suggest that in October 2001, peer-to-peer traffic reached more than 30% of the total traffic while at the same time, web-related traffic was little more than 19%[113]

While there is no indication as to the volume of traffic generated by either the P2P or web-based systems, these figures ". . . represent a significant and continually increasing percentage of the overall network traffic. . . "[113] This rapid increase in the popularity of P2P applications and resulting network traffic means that any discussion concerning networks must consider P2P. However, in order to evaluate the types of P2P system available, it is important to define them.

As with Agents (see chapter 2), there are a number of different definitions for P2P systems. In the simplest definitions, a peer to peer system consists of a network of nodes which communicate on an equal basis. In other words, all of the nodes in the network can initiate a connection, and all nodes can make and respond to requests. This is in contrast to a client/server system in which a central server responds to the requests made by a number of clients. The group of researchers from the European Grid of Solar Observations state:

> A P2P application is different from the traditional client/server model because the applications involved act as both clients and servers. That is to say, while they are able to request information from other servers, they also have the ability to act as a server and respond to requests for information from other clients at the same time.[42]

This has lead to the term *servent* - a concatenation of SERVer and cliENT - which is often used to describe the nodes in a P2P network.

For one researcher, Clay Shirky, the traditional definitions of P2P systems do not explain the shift in computer usage caused by P2P.

> This literal approach to peer-to-peer isn't plainly not helping us understand what makes P2P important. Merely having computers act as peers on the Internet is hardly novel, so the fact of peer-to-peer architecture can't be the explanation for the recent changes in Internet use.[153]

Instead, he provides an alternative definition about what makes P2P unique:

> If you're looking for a litmus test for P2P, this is it: 1) Does it treat variable connectivity and temporary network addresses as the norm, and 2) does it give nodes at the edge of the network significant autonomy?
>
> If the answer to both these questions is yes, the application is P2P. If the answer to either question is no, its not P2P.[153]

The first criteria of the definition relates to the fact that there are a limited number of IP addresses which means that the majority of devices connected to the Internet operate using dynamically generated IP addresses. While this is not a problem for applications acting as clients, traditional servers require fixed addresses so that they can be located by the clients. Since most P2P servents have dynamic IP addresses which may change for each session, nodes in the network must be able to cope with this.

There does appear to be some disagreement about the length of time that nodes remain connected to a P2P network. Matei Ripeanu[144, page 5] produced results from traces taken during November 2001, February/March 2001 and May 2001, which show that 40% of nodes are connected to the Gnutella network for less than 4 hours. In contrast, results by Stefan Saroiu et al[150] taken during May 2001 show that both for Gnutella and Napster, user sessions lasted approximately 1 hour. Similar work done using the "Mojo Nation" P2P system show that 80% users of user sessions last for less than an hour.[183]

The variations in the different sets of figures generated may be attributed to a number of factors; the types of software used to capture the data, the dates and times that the data was acquired, and the way that the raw data was converted into actual statistics, etc. However one thing is clear - P2P networks are transitory and a successful P2P network must be able to deal effectively with nodes appearing and disappearing without prior warning. The frequency at which nodes appear and disappear is referred to as the rate of churn.[39].

Although it could be argued that all devices on a network should be able to deal with the failure of connected devices, this is frequent occurrence in a P2P network. While it is possible for web servers to be unavailable, it is rarely the case as they almost always use fixed IP addresses and permanent connections.

While P2P networks are generally resilient to the arrival and departure of nodes, this volatile and unpredictable nature is not without consequence. As nodes are added and removed at different points, the network structure becomes uneven; gaps appear and some nodes become more important to the structure of the network than others. If a key node is removed, the network splits into different fragments which cannot be reconnected since a new node can only join one fragment of the network, and not re-unite disparate fragments.

> It has been shown that if 4% of the most highly connected nodes are removed from Gnutella, the network will severely fragment, rendering it useless. ... Gnutella's robustness to random failure and vulnerability to malicious attack is not unique. Indeed, the Internet has similar characteristics; an attack on 5% of nodes would result in a total collapse of the internet.[94]

The second requirement of the definition distinguishes P2P networks from the more traditional client/server model. In the client/server model, it is the clients which are at the edge of the network. They can make requests, but cannot process requests. The servers can receive requests, but cannot make requests. In P2P networks, all nodes can both make and receive requests.

It is important to note that the definition uses the words *significant autonomy*. If the definition required that all nodes required equal autonomy then both hybrid (see section 4.5.1) and super-peer (see section 4.5.3) systems would have to be excluded from the definition. However, since all of the nodes can perform a minimal set of operations; requesting a file and preforming a file transfer, the definition is still valid for these systems.

However, Shirky's definition does not exclude Client/Peer Oriented (see section 4.4) systems despite the fact that they rely on a central server and there is no communication between the peers. While this is intended, it does contradict one of the basic characteristics of P2P architectures - that the peers in the network can communicate directly. Instead he suggests an alternative view of P2P based on:

> Another way to examine this distinction is to think about ownership. It is less about

> "Can the nodes speak to one another?" and more about "Who owns the hardware that the service runs on?" The huge preponderance of the hardware that makes Yahoo work is owned by Yahoo and managed in Santa Clara. The huge proponderance of the hardware that makes Napster work is owned by Napster users and managed on tens of millions of individual desktops. P2P is a way of decentralizing not just features, but costs and administration as well.[153]

While Shirky's definition does highlight two important characteristics of modern P2P system, the requirement for peers to communicate directly cannot easily be ignored. After all, it distinguishes P2P systems from the client/server model.

> The name "peer-to-peer" suggests a egalitarian relationship between peers and, more importantly, suggests direct interaction between peers.. . .
>
> But not all distributed computing is P2P computing. Distributed applications like SETI@home and the various distributed.net projects exhibit little interesting peer-to-peer interaction, and are therefore not really P2P according to the definition above.[166]

4.5.1 Hybrid P2P Systems

In a *hybrid* P2P system, all of the nodes in the network must connect to the server. If for any reason, a server is not present, then the system cannot operate. How much assistance the server provides is dependent on the system. In some, it simply acts as a name server - merely providing a list of the currently logged on peers. In other systems, the server could provide a list of the content that is available for download as well. Napster[81] opted for this second approach.

When a user searches using Napster, the search is passed to the server which returns a list of results - files, their locations, and other relevant information. The users selects a file from the list returned, and the node initiates a direct connection to the node containing the requested file and downloads it.

More recently, BitTorrent[14, 22] was developed to use a hybrid approach. However, rather than incorporate the search facility into the P2P servent as in other systems, Bit Torrent requires users to download a data file from the web site.[3] This file can then be loaded into the Bit Torrent application which will retrieve the data requested. One distinctive feature of Bit Torrent is that the files are split into chunks and multiple copies spread around the network. This allows the node downloading to retrieve data from a number of locations before rebuilding the files when the download is complete. Since the system is mainly used for large media files, such as films, this allows downloads to continue even when a node leaves the network.

4.5.2 Pure Peer-to-peer

In contrast, *pure* P2P systems such as Gnutella[79], Freenet[141] do not rely on access to any server or web site. Instead, the nodes within the network provide all of the services required. The two most notable P2P systems are; Gnutella[79] and Freenet[45, 44, 141] although other systems such as OceanStore[99] and DISCWorld[156] also exist.

Since there is no central server, when a node joins a pure P2P network, it must be given the name of one operational node. In general, the process by which a new node joins the network is similar with the main difference being that the first time a Freenet node is activated, it creates a unique key to identify itself.

> The node joins the network by connecting to one Gnutella node, which can be any node on the network making it generally easy to join in a decentralized fashion.
>
> Once it has joined the node discovers other nodes through the first node by issuing

[3]The most popular web site for Bit Torrent download files is http://www.supernova.org

> *ping* and receiving *pong* descriptors from peers accepting connections. [171, page 40]

When the Freenet node joins the network, the messages passed between the node include the keys used to identify the nodes. These keys are stored by the nodes in a routing table which is used when searching.

By using the *ping* and *pong* messages, a new node is able to discover and connect to existing nodes without the need for a central server. However, this process can generate significant traffic.

> The traces showed that Pings and Pongs messages were just over 50% of the network traffic seem on the network, and that Queries were another 40% of packets seen in the traces.[5, page 5]

More recent analysis[144, page 5–6] in June 2001, shows that *ping* and *pong* messages only accounted for approximately 8% of the traffic on the network even though the number of node had grown from 2,063 in November 2000 to 48,195 nodes in May 2001.

This is due to a redesign of the Gnutella servent which lead to a decrease in the number of Ping and pong messages required, and introduced Super Peers (see section 4.5.3).

> . . . careful engineering led to significant overhead traffic decreases over the last six months. Second, the network connectivity of Gnutella participating machines improved significantly . . . Finally, the efforts made to better use available networking resources by sending nodes with low bandwidth at the edges of the network eventually paid off. . .
>
> Apparently, by June 2001 these engineering problems were solved with the arrival of newer Gnutella implementations: generated traffic contained 92% QUERY messages, 8% PING messages and insignificant levels of other message types.[144, page 5-6]

While the continuous development of Gnutella has reduced much of the bandwidth consumed by the node discovery process, concerns have are frequently raised about the scalability of the system. This is due to the *broadcast* method of searching for files which is costly in terms of bandwidth consumed.

> Gnutella nodes typically connect to three nodes and then search by broadcasting their search request to all connected neighbours.. . . Each neighbour repeats this search request to his/her neighbours and so on, which is known as *flooding the network.*[171, page 40]

In order to prevent the query messages from being passed around the network indefinitely, Gnutella packets include a *time-to-live (TTL)* parameter - usually with a default value of 7. As the message is received by a node the TTL is decremented. If the value for the TTL is positive, then the message is forwarded to all of the connected nodes except for the node that it was received from. In addition, each message also contains a unique ID which is used to prevent messages from being sent to nodes which have already received them.

> The "standard" TTL is 7 hops, so, how far is that? A 7-hop radius combined with network conditions (i.e. 4 connections) means that around 10,000 nodes are reachable within a fully connected network.[171, page 104]

The range of the search, as dictated by the TTL, is referred to as the search horizon, and although it is an artificial limit, it is a necessary one. Without it a query would pass throughout the network until the whole network was processing the query. While in theory, flooding the network with requests to find as much information as possible is a good idea, the network becomes swamped very quickly.

Free Riding[2] is a problem highlighted by researchers at Xerox Palo Alto Research Center. This occurs when users downloads files from the network but does not make any files available for others to download.

> In our analysis we consider two types of free riding. In the first type, peers that free ride on Gnutella are those that only download files for themselves without ever providing files for download by others. The second definition of free riding considers not only the amount of the downloadable content a producer has, but how much of that content is actually desirable content. This is essentially a quantity verses quality argument that also poses a social dilemma when there is a cost to the provider to make desirable files available to others. In the "old days" of the modem-based bulletin board services (BBS), users were required to upload files to the bulletin board before they were able to download. In response to this requirement users would upload their own bad artwork or randomly generated text files and would be able to download high quality content generated by others . . . [2]

As a result of this activity, it was found that a large percentage of the files were provided by a very small number of users.

> Specifically, we found that nearly 70% of Gnutella users share no files, and nearly 50% of all responses are returned by the top 1% of sharing hosts.[2]

This has a number of effects on the network.

Firstly, the uneven distribution of the files means that the nodes with large amounts of data are acting as servers. This means that these nodes perform the bulk of the data transfers, and may become overloaded. Since they are more critical to the operation of the network than nodes with no data, their absence from the network reduces the amount of data that can be located. This, in essence, creates the same dependencies as in a client/server network - something which a well balanced P2P network avoids.

Secondly, the number of *empty* nodes dramatically reduces the search horizon. This occurs because even nodes with no data still affect the TTL. Rather than a message passing between, for example, 7 nodes with data; it may only pass between 2 nodes with data, and 5 empty nodes. An extreme possibility is that the network includes nodes which are out of range of any significant stores of data.

Despite the use of the TTL to create a search horizon, the search requests which flood the network account for a high percentage of the traffic leading Jordan Ritter, one of the creators of the original Napster system, to write a paper titled "Why Gnutella Can't Scale. No, Really." which was published during February 2001. In this document, he calculated that the sending a single simple search message - *"grateful dead live"* - consisting of an 83 byte data packet across the network, generated 800Mb of search and response data for the single query.

> On a slow day, a GnutellaNet would have to move 2.4 gigabytes per second in order to support numbers of users comparable to Napster. On a heavy day, 8 gigabytes per second.
>
> ...it should also be noted that only search query and response traffic was accounted for, omitting various other types of Gnutella traffic such as PING, PONG, and most importantly, the bandwidth costs incurred by actual file transfers. 2.4GBps is just search and response traffic, but what about the obnoxiously large amount of bandwidth necessary to transfer files between clients?[145]

Later versions of the Gnutella protocol have incorporated mechanisms which are intended to improve the scalability. These include the use of caching to reduce the cost of a new node joining the network, and the introduction of *super-peers* (see section 4.5.3) which are described later in this chapter.

Rather than use a *network flooding* approach, Freenet uses a more sophisticated search mechanism. When a users searches for a specific file, the search is used to generate a coded key which is 128 bits long. The Freenet servent contains a routing table with a list of other nodes and the keys that they contain.

When a node receives a request for a file, it checks its own storage space. If the file is not found, the node selects the remote node with the key closest to that of the file to be retrieved. The request is forwarded to that node. If the response from the remote node is that the request has failed, then it chooses the next best node from its list of known nodes. The process is repeated until there are no more nodes to contact, or the file is found.

In order to prevent loops from forming in the chain, each request contains a GUID - a unique ID - which is checked by the receiving node. If the GUID is recognised, then the message has been received previously and so a 'request failed' message is returned. The other condition which causes a 'request failed' message to be generated is that of the limit of the search horizon.

When a file is located, it is passed back along the chain of nodes to the source of the request. Each node of the chain stores a copy of the file. This results in the more popular documents being distributed more widely throughout the network.

> Such caching services form the basic building blocks of the Freenet network since each peer contains a routing table, similar in principle to the Gnutella super-peers or Napster indexes. The key difference is that Freenet peers do not store locations of the files at all, rather they contain file keys that indicate the direction in the key space where the file is *likely* to be stored.[171, page 156]

As the space available at each node is finite, when new files are to be stored, the older files are deleted. This means that the files which are requested least gradually expire.

More recently, techniques have been developed which improve the usability of Freenet.[43] The key feature of the new routing mechanism is that nodes in the network collect statistical information about the other nodes in the network, "including response times for requesting particular keys, the proportion of requests which succeed in finding information and the time required to establish a connection in the first place"[43]. The second enhancement to Freenet involves the sharing of this data between nodes in order to improve searching when a new node joins the network.

4.5.3 Super-peer

In 2001, the Gnutella network was changed to include *super-peers* (also known as Ultra-peers, Hubs, & Reflector Hubs) in order to improve its scalability.

The following quotation, taken from a technical report by Yang and Garcia-Molina describes one of the limitations of pure P2P systems in general, but the early versions of Gnutella:

> Another source of inefficiency is bottlenecks caused by the very limited capabilities of some peers. For example, the Gnutella network experienced deteriorated performance - e.g. slower response time, fewer available resources - when the size of the network surged in August 2000. One study . . . found these problems were caused by peers connected by dial-up modems becoming saturated by the increasing load, dying, and fragmenting the network by their departure. However studies . . . have shown considerable hetrogeneity (e.g. up to 3 orders of magnitude difference in bandwidth among the capabilities of participating peers. The obvious conclusion is that an efficient system should take advantage of this hetrogeneity, assigning greater responsibility to those who are more capable of handling it.[192]

By re-designing the Gnutella servent to allow for the use of super-peers, the problems with scalability were greatly reduced.

> Super-nodes act as caching servers to connected clients and perform a similar operation to Napster servers. So, rather than propagating the query across the entire set of nodes, the super-peer will check its own database to see if it knows the whereabouts of the requested file and if so, it returns the address to the client, just like Napster. If not, it performs a Gnutella-type broadcast across the decentralized set of super-peers to propagate this across the network. This means that a client can search an entire network without consuming vast quantities of bandwidth.[171, page 126]

By granting super-peer status to some nodes in the network, designers have created a hierarchy based on resources. Theoretically there is no limit to the level of hierarchy in a super-peer network; super-peers could, quite easily, become part of a group controlled by a *super-super-peer.*

Comparisons have been noted between the newer versions of Napster with its cluster of servers, and the Super-peer version of Gnutella.

> It is interesting to note that both Gnutella and Napster converged towards a centralized/decentralized topology, even though they came from completely different sides of the coin. Gnutella started life as a decentralized system and Napster started life as a centralized search architecture, with brokered communications. However, Gnutella inserted super-peers and Napster duplicated its centralized search engine for scalability, both resulting in a similar design topology. . . [171, page 127]

4.5.4 Overlay Systems

However, *Super-peers* are not the only alternative to *hybrid* and *pure* P2P systems. Researchers have experimented with the idea of overlay systems. These are systems where the designers have imposed a topology on top of the P2P network. One system; Chord[163, 7], uses a ring topology. Pastry[35, 34], Skipnet[71] are other overlay systems which also use ring topologies but differ from Chord in the way that the routing information is structured and used. Other overlay systems such as CAN[7] and Tapestry[7, 196] are based around different topologies.

All overlay systems function in similar ways, with a key being generated and stored in a specific location dependant upon its value, in order to improve the efficiency of searches. The strategy for dividing the address space between the nodes is very much dependent upon the topology of the overlay, although different algorithms may be used with the same basic topology depending upon the properties that are desired.

> In these systems. . . files are associated with a key (produced, for instance, by hashing the file name) and each node in the system is responsible for storing a certain range of keys. There is one basic operation in these . . . systems, `lookup (key)`, which returns the identity (e.g., the IP address) of the node storing the object with that key.[142]

In order to improve the searching, each Chord node maintains an index referred to as a *finger table*[4] which contains routing information. Each entry in the table points to the successor of the nodes spaced exponentially around the ring. This allows queries to be directed to the approximate location of the key rather than for every node between the source of the request and the node containing the key needing to be contacted.

> Each chord node needs routing information for only a few other nodes (only 0(log N) for an N-node system in the steady state), and resolves all lookups via 0(log N) messages to other nodes. Performance degrades gracefully when routing information becomes out of date due to nodes joining and leaving the system; only one piece of information per node need be correct in order for Chord to guarantee correct(though slow) routing of queries.[7, page 18]

[4]This is known as a Distributed Hash Table (DHT) or Routing Index (RI) in other systems

However, due to the way the overlay systems work, with keys being distributed between the nodes, maintenance needs to be performed whenever a node enters or leaves the network.[33] The minimum work involved required redistributing the keys between the current set of nodes in the network.

> To maintain a mapping when a node n joins, certain keys previously assigned to n's successors are reassigned to n. When n leaves the network, all of its keys are reassigned to n's successor. No other changes in assignment need to occur.[105]

While this is true for the very basic operation of the chord network, it does not take into account the updating of the finger tables of the nodes that referenced the departing node. As a result the number of messages may be significantly more;

> In contrast [to pure P2P systems], churn does cause significant overhead for DHTs. In order to preserve the efficiency and correctness of routing, most DHTs require 0(log n) repair operations after each failure. *Graceless* failures, where a node fails without beforehand informing its neighbors and transferring the relevant state, require more time and work in DHT's to (a) discover the failure and (b) re-replicate the lost data or pointers. If the churn rate is too high, the overhead caused by these repair operations can become substantial and could easily overwhelm nodes with low-bandwidth dial-up connections.[39]

4.5.5 Agent Based P2P Systems

Considering the current popularity of both agents and P2P system it is in some ways surprising that there are not more systems that combine the two. However, there are researchers that consider many existing MASs such as RETSINA[167, 168] and AgentCities[185, 184, 197] to be

P2P systems due to the fact that the agents communicate on an equal basis. Instead, this section concentrates on systems which are structurally designed to take specific advantage of the P2P topology, i.e. systems where instead of global system-directories, each node is self-contained with its own directories.

Most commonly Agent-based P2P systems are created with each node functioning as a complete MAS.

> ...we propose an architecture where each participant/partner (i.e., a peer of the P2P network) has its own MAS...[129]

Anthill[8] is one system of this type. It consists of a series of *nests* (the P2P nodes) and dynamically created *ants* (mobile agents) which can move between them. Each nest contains its own set of middleware.

> Ants are generated in response to user requests; each ant tries to satisfy the request for which it has been generated. An ant will move from nest to nest until it fulfills its task, after which (if the task requires this) it may return back to the originating nest. Ants that cannot satisfy their task within a *time-to-live* (TTL) parameter are terminated.[8]

The Squirrel MAS[30] is based on the foraging and storage behaviour of squirrels. Its purpose is to ensure that documents are distributed evenly across the disk space available in the network. As with the Anthill system, the peers in the Squirrel environment are self contained and do not rely on global directories and centralized middleware. However, as with Anthill, the agents (squirrels) are able to travel between the peers (locations).

> The Squirrel MAS system consists of a P2P environment of locations. Each location has one or more caches where squirrels hoard acorns. Squirrels live in these locations

> in small groups. When they have acorns, they go through the locations "sniffing" to find a cache suitable for the acorn. . . [30]

JEAP (Java Environment for Agent Platform)[129, 133] and PeerGroups[21], were created using on the Jxta[171, page163–179,199-216] framework. This is a set of protocols which allow peer to peer services to be developed for a wide range of devices including PDAs, servers and desktop computers.

JEAP contains three levels; Wrappers which provide an agent based interface to the resources in the system, Mediators perform the pre-processing of the query - converting it into a set of actions which are passed to the various information sources via the Wrapper agents. The final layer of the structure contains the facilitator. This allows agents to locate other agents with the services they require.

In order to provide for situations where the local knowledge is not sufficient to answer a query, the mediators and the facilitators are able to communicate with their corresponding levels on other nodes. This allows node and services information to be exchanged between the different nodes.

In contrast to these systems, NeuroGrid[89, 90, 91, 92, 93] is a P2P system in which each node is a single entity. Each node maintains its own knowledge base which is used to determine which agents to contact for assistance.

> Each NeuroGrid node facilitates search of the network by forwarding queries to a subset of nodes that it believes may possess matches to the search query. . . . Each node maintains a knowledge base of keywords-node associations that are based on the nodes belief about the contents of remote nodes. So, for example, given that a node receives an incoming search consisting of keywords A, B & C, the node will consult its knowledge base and retrieve any remote nodes that are associated with

> these keywords. The nodes retrieved from the knowledge base are ranked depending upon the degree of match to the search query...
>
> NeuroGrid nodes utilize the results of searches in order to update their knowledge bases and add new connections to the nodes that provide results to search queries. The best analogy is to think of the nodes as humans, that know something about what their friends know about, and when asked can put you in touch with a friend, who may well be able to put you in touch with a friend who ... and so on.[89]

Despite this learning ability and the maintenance of a knowledge base, the designers prefer not to consider the nodes as agents even though the nodes are as powerful as many reactive agents.

> One could go further, and suggest that all the nodes in a p2p network such as NeuroGrid are themselves agents, in as much as they learn from experience, communicate with one another, even behave "autonomously" ... However, there does not seem much to be gained from applying such a label, so let us consider the elements of the p2p network simply as nodes, leaving any relevance to agents down to the general needs of any "agent" that must operate in a distributed environment.[89]

The Information Retrieval (IR) system created by Haizheng Zhang et al[195] is similar to NeuroGrid in that nodes contain a single entity - in this case, the nodes are recognised as agents. The system is described as a "mediator-free information retrieval system for P2P networks".

> This search involves locating and retrieving relevant documents distributed among one or more databases. We assume each data-base is associated with an intelligent agent that is cooperating with other agents in the distributed search process.
>
> ... each agent maintains an independent index and IR search engine for its local document collection. However, we do not introduce any further restrictions on the

> local search engines and thus the network can be populated by agents with very different local search engines.[195]

The agents in this system build up their knowledge of other agents in the network by exchanging *agent-views*.

> The agent-view structure, also called the local view of each agent, contains information about the existence and structure of other agents in the network and thus defines the underlying topology of the agent society. The functionality of an agent-view is analogous to the routing table of the network router. In practice, the agent-view structure contains the collection model of the collections and other related information about these agents...
>
> agents exchange their local agent-view s to expand the scope of their local agent-view so that each agent is more informed about thre content distribution over the entire network.[195]

The agent is able to use this information on which agents are available, and their information content. As a result, queries can be routed to the correct agents.

> In the absence of a mediator, agents must cooperate to forward the queries among themselves so as to locate appropriate agents, rank the collection , and finally return and merge the results in order to fulfill the information retrieval task in a distributed environment.[195]

In the discussion section of their paper, they propose a new design which includes a mediator as a way of reducing the communications overhead of the system. However, from the brief description given, the mediator will act in a similar way to a super-peer node (see section 4.5.3 for further details).

4.6 Computational Grids

The Grid is an attempt to provide an global network of high performance computers which can be accessed as a single resource without the need for separate passwords and access protocols. By removing the barriers that prevent the separate computer systems from working effectively, the grid provides a way of pooling resources so that larger and more complex computing problems can be tackled effectively. Although the name suggests that the grid is a single infrastructure, numerous grids exists, and interact.

> Science is driving the creation of grids because it already has problems that push the limits of supercomputers, such as analyzing supercollider data, simulating weather and creating a virtual observatory. Three major grid projects (TerraGrid in the U.S., the National Grid in the U.K. and a Dutch grid interconnected through SURFnet) were recently announced despite existing technical challenges. Engineering and biotech firms are likely to follow because of the complexity of the problems they face.[84, page 1]

Legion is another grid project with this same goal:

> Our vision of Legion is a system consisting of millions of hosts and billions of objects co-existing in a loose confederation united through high-speed links . . . Users will have the illusion of a very powerful desktop computer through which they can manipulate objects.[68, page 40]

With Legion, as with other computational grids, the intention is to create the high-performance equivalent of a national electricity grid.

> The Grid *dream* is to allow users to tap into resources off the Internet as easily as electrical power can be drawn from a wall socket. . . . For example, imagine when you

> plug in your kettle, your only concern is, have you filled it with water. You should not have to worry about where the electricity comes from, whether it is brought from other countries or generated from coal, windfarms, etc. You should simply take for granted that when your appliance is plugged in it will get the power it needs.[171, page 57]

Merely providing a system which allows users the ease of use of the power grid is not enough. For the Grid to be successful, separate and often diverse resources must be managed and integrated to provide a system which is both powerful and flexible. In order to provide these services, systems such as Legion must perform a number of complex tasks in order to simplify the view of the system which is presented to the user.

> Legion is responsible for supporting the abstraction presented to the user, transparently scheduling application components on processors; managing data migration, caching, transfer, and coercion, detecting and managing faults; and ensuring that the users' data and physical resources are adequately protected.[68, page 40]

While for many grid researchers, the idea of a global network is the ultimate goal, there are currently numerous incompatible grids. This is similar to another physical grid: the transport system, which consists of a number of grids which overlap and in some places intersect (airports, and harbours are the most common locations for this to occur). Often users of the transportation system must use a number of different grids in order to achieve their goals. For example, a businessman starting in Edinburgh and travelling to New York will need to use both road and air transport to reach his destination.

> In reality, however, there is not one single "Grid", rather there are many different types: some are evolving, some private, some public, some regional, some global,

some specific (e.g. dedicated to one specific application) and some generic. Such Grids have realistic goals but do not attempt to solve the whole Grid problem. It will be some time before the power grid analogy becomes reality (if ever).[171, page 58]

4.6.1 Virtual Organisations

In order to make the management and use of these separate grids easier, Virtual Organisations (VOs) are created. These are flexible groupings of (possibly geographically distributed) users and resources which may possibly span several organisations. In this model, it is possible for both participants and resources to be part of a number of different organisations at the same time.

One example of a VO is that of a group of researchers working on a complex problem. Although based in the different laboratories, they are able to share the total resources of the group - data, processor cycles and the results without problems. Similarly, it is possible for researchers to be members of a number of different VOs at the same time.

> Virtual Organisations may enable scientists from numerous countries and backgrounds to work together to analyse and interpret a new discovery from a deep space radio telescope. Organisations in Australia and the UK may pool their processing resources into a Virtual Organisation so that while one country sleeps, the other country, on the opposite side of the world, may use the processing power of its peer's idle computers.[3]

4.6.2 Grid Services

In the earlier days of Grid Computing, systems were always developed independently. However, this changed with the creation of the Globus Toolkit. The Globus Toolkit was based around the I-WAY environment and "allowed the assembly of unique capabilities that could not otherwise be created in a cost-effective manner".[171, page 60] As a result, Globus became the basis of many of the modern Grid systems.

> Grid technologies have evolved through at least three distinct generations: early ad hoc solutions, de facto standards based on the Globus Toolkit (GT), and the current emergence of more formal Web services (WS)-based standards within the context of the Open Grid Services Architecture (OGSA).[60]

Although the OGSA services have received criticism from the Web services community and as a consequence have been redesigned, they provide a significant improvement in the development of Grid technologies.

> Open Grid Services Architecture (OGSA) is an open standard at the base of all of these future grid enhancements. OGSA will standardize the grid interfaces that will be used by the new schedulers, autonomic computing agents, and any number of other services yet to be developed for the grid. It will make it easier to assemble the best products from various vendors, increasing the overall value of grid computing. More information about OGSA can be obtained at http://www.globus.org/ogsa.[20]

> OGSA services, or Grid services, extend Web services ... to add features that are often needed within distributed applications. Specifically, OGSA adds *state* to Web services in order to control the remote services during its lifetime. Whereas Web services are *stateless*, ... OGSA services are *stateful*.[171, page241]

> OGSA is likely to become one of the most commonly used Grid architectures, opening the possibility of standardisation, widespread InterGrid communication, and the emergence of the Grid itself.[3]

However concern was expressed that Grid Services did not conform fully to the Web services standards. As a result, further work was carried out to develop the Web Services Resource Framework (WSRF) to replace OGSA. While the functionality of OGSA and WSRF services are the same, the difference is in the interface. This interface, referred to as the OGSI specification "defines a component model by using extended WSDL and XML schema definitions."[171, page 246].

> The essential difference here is that OSGI uses the same construct to represent a Web service and a stateful resource, whereas WSRF uses different constructs for both.[171, page 251]

4.7 AT-MAS and the Information Environment

The AT-MAS network of agents functions mainly as a pure P2P network such as Gnutella or Freenet. However, each AT-MAS agent is also able to act as a server to a simple AT-MAS client. This client/server interaction is important as it allows users without data to use the system without the need to run an agent with either no data or worthless data. As a result of this, the problem of Free Riding is almost eliminated completely.

Free riding is further reduced since the AT-MAS agents will only sends messages to the agents most likely to be able to assist. When a request is sent, the AT-MAS agent will record the result of the request and use it to influence the decision of whether to contact that agent in the future. This means that the routing of requests is based on the agent's experience of the making similar requests in the past. In other words: successful interactions improve the chances of future interactions taking place, whereas unsuccessful requests will cause the agent to request assistance elsewhere.

However, the main difference between AT-MAS and other P2P systems is that it allows single data elements to be transferred around the network, whereas other P2P systems are concerned with the movement of complete files.

4.8 Chapter Summary

This chapter begins with a description of the internet as a environment for agents. The internet can accurately be described as a vast unregulated, unstructured source of information. Since there are over 11.5 billion web pages, finding information is a difficult; often impossible task. This is further complicated by the fact that the available search engines such as Google are only able to index a fraction of the content due to its size.

Another of the problems of retrieving information from the internet is that most of the information is only accessible through dynamically generated web pages which act as an interface to database systems and other data stores. This information is referred to as the *Deep Web* and is estimated to be at least 40 times the size of the existing surface web. However, as this information is contained within databases, it has structure and as a result is potentially more accessible to applications other that web browsers.

This is followed be a description of the different technologies that can be used to access the internet. This include; Client/Server, the Semantic Web, Web Services, and Client/Peer Oriented, Peer to Peer, and Grid systems which are described. Of particular interest in connection with the AT-MAS system is the section describing Agent Based P2P Systems (see section 4.5.5).

The final section briefly compares the AT-MAS system with P2P systems.

Chapter 5

The Agent Trees Multi-Agent System (AT-MAS)

5.1 An Overview of the AT-MAS Network

The purpose of the AT-MAS system is to provide distributed information retrieval, filtering and processing facilities across a network of intelligent agents. The system combines the conceptual simplicity of a P2P network with the power and flexibility of intelligent agents. It was originally inspired by the desire to find an alternative to mobile agents which posed fewer security risks while allowing remote processing to take place.

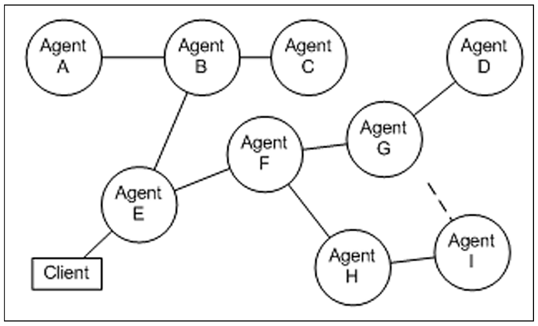


Figure 5.1: An Agent Tree

AT-MAS is an open system - any agent can join the network providing that it adheres to the simple communications protocol and conversation format that has been defined. Similarly, a simple client application can be created which can request information and services from any agent. As a result of this, the range of applications are not limited to those provided by the existing AT-MAS agents.

5.1.1 The AT-MAS system in operation

When the user connects to an agent, using a simple java applet, they will typically submit a task consisting of a number of goals. Some these goals may be carried out by the agent to which the client is connected to - referred to as the *root agent* - other goals may need to be passed to other agents. In order to decide how the goals are to be dealt with, the agent will form a set of plans - each consisting of a number of actions - for each of the goals.

When all of the plans for a goal have been created, the first one is attempted. If a plan is completed successfully then the goal has been achieved and the next goal is attempted.

If any of the actions in the plan fail then the plan is considered to have failed and next one is tried. This process is repeated until one of the plans is completed successfully or all of the plans have been failed. If all of the plans for a goal fail then the goal has failed and the next goal is attempted. Frequently the success of one goal is dependent upon the successful completion of a previous goal. If this is the case, and the earlier goal fails, then the latter goal will also fail. The planning process is described in section 5.3.3

5.1.2 Building the Agent Tree

Due to the scale and diversity of the information on the internet, it is likely that a significant percentage of requests for information cannot be satisfied by the local agent alone. Instead, the assistance of other agents may be required. When this happens, the root agent will send out messages to other agents in a process that mirrors the process of the client connecting to the root agent. These agents may in turn contact other agents which may also contact other agents. It is this expanding tree pattern of connections that gives the system it name (see figure 5.1).

It should be noted that agents are prevented from contacting a node that is already part of the tree. In figure 5.1, agent I is refused a connection to agent G because agent F has already successfully contacted it.

This rule can easily be enforced since clients and remote agents are not able to contact other agents directly. Instead, they must contact a front-end server to obtain an authorization code known as a *Task ID* and the address of the port that the agent will use to communicate. For further details of the server, see section 5.2.2).

As well as this, receiving a task ID from the server, if the request is a new one i.e. from a client rather than from an agent, the server will create a *Tree ID* which uniquely identifies the query. This is then included with every request made to a server or agent. If the server receives a request with the same Tree ID of one which it has already received, then the new request will be refused. As a result repeated requests; whether accidental, or deliberate, are refused and agents will not become bogged-down performing the single task.

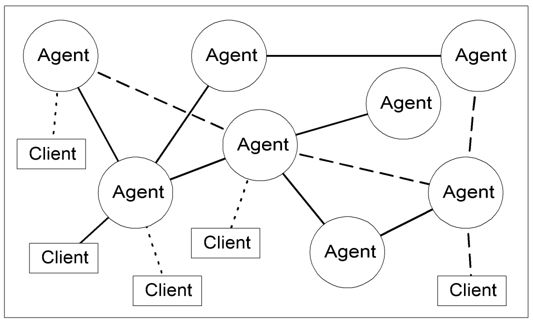


Figure 5.2: Snapshot of an Agent Tree Intranet

Although the figure 5.1 gives an image of a single Agent Tree, in reality, the agents in the system are able to handle more than one task at a time, and as a result a system of agents is most likely to look as follows (see figure 5.2);

5.1.3 Discovering other agents in the Network

During the design of the agent trees system, there were two key goals concerning the way that discovering new agents should be handled. Firstly agents should know as many other agents as possible - so that the chances of network fragmentation would be reduced. Secondly, the mechanisms used to obtain this knowledge should have as little effect on the network load as possible, unlike Gnutella. Additionally, these goals should not conflict with the any of the other existing design goals - such as keeping each node independent.

Part of this independence requires that each node maintains its own knowledge base, listing all of the other agents that a particular agent knows. As each agent maintains its own knowledge base, the list of other agents known will differ from agent to agent. Under normal circumstances, this can lead to fragmentation of the network. This is most likely to occur when all of the nodes of the network end up communicating via a single node. This is a problem because it means that if the *key* node fails or is removed from the system then the network is split with no easy way of re-combining the nodes[156].

The most obvious way of preventing network fragmentation is to ensure that all of the agents know about all of the other agents, although in a massively distributed system this is not possible unless a central server is used. Instead the agents in the Agent Trees will attempt to learn about as many other agents as possible. This is so that there will always be a large number of links to other agents , thereby the possibility of all queries passing through a single node are greatly reduced.

In the current design, this is achieved by using a <Contacted></Contacted> tag within the *new task* and *task complete* messages. When the root agent (for example Agent A) requests the assistance of another agent (Agent B), it adds its address to the contacted field. This process is continued until the final agent of the current branch of the tree is called. This means that the agents towards the end of the query will learn about the agents that assisted earlier in the query.

Agent	A	B	C	D
initial KB	B	C	D	-
outward	-	A	AB	ABC
return	ABCD	BCD	CD	D
final KB	ABCD	ABCD	ABCD	ABCD

Table 5.1: 'Contacted Field' Results for a Chain of Agents

Agent	A	B	C	D
initial KB	BC	-	D	-
outward	-	A	A	AC
return	ABCD	B	CD	D
final KB	ABCD	AB	ACD	ACD

Table 5.2: 'Contacted Field' Results for a Tree of Agents

When the final agent has completed its processing, it will pass its name to the calling agent via the contacted field of the *task complete*. This process is repeated all the way along the branch of the tree until the root agent receives the *task complete* from the agent that it had called.

This is illustrated by table 5.1 which shows the results from a small network[1].

However when a complete tree is created, as is the norm, the results are slightly less effective. Table 5.2 was generated for a tree in which the root agent A called two agents B and C. From this, agent C called agent D. However, the contacted field does not give as good results for a tree which has more than one branch.

From the tables, it can be seen that each node will learn of any new nodes between itself and the

[1]Since the tests were carried out using University computers with long names, the names have been changed to improve the readability.

root node - referred to as the *path to root* - and any new nodes in the subtree that it creates. In the case of nodes B there is no subtree so only the root node is added to Agent B's Knowledge Base. Similarly, the subtree for node D is empty but its path to root consists of nodes A and C. For node C the path to root also consists of node A but the subtree of C consists of nodes D.

Each time the agent receives a message containing the contacted field, it will extract the names of agents that it was not previously aware of and add them to its knowledge base for future use. This allows an agent to learn of new agents without the need for a central name server, or repeated 'Ping' messages as required by some other P2P systems. All that is required for this method to work is for each agent to be given the name of at least one other agent when it is installed.

In addition to this technique, the capability exists for the agent to request information from other agents, and also respond to such requests. One proposed enhancement to the system is to allow agents to query other agents during quieter times. This would enable an under-utilized or newly installed agent to improve its own knowledge of the network. However, it is not intended as the primary method for an agent to gain knowledge, as it would lead to increased network load. This is discussed further in section 7.4.

5.1.4 Expanding the Search Horizon

One of the key goals of any distributed system is to locate information efficiently and the AT-MAS system is identical in this respect.

Like many P2P systems (see section 4.5), Agent Trees has to restrict the number of computers that are searched during a query in order to prevent the network from getting swamped. Unlike P2P systems such as Gnutella and Freenet which just attempt to limit the search horizon, Agent Trees uses two variables instead of a single one.

The *Tree Depth* variable is the equivalent to the *Time To Live* and *Hops To Live* variables found in Gnutella and Freenet. It is an integer value which is included in the messages sent by the Client to root agents. At each stage of the query, the *Tree Depth* is decremented until its value is 0. At this point, the tree is considered complete and the results are returned. If an agent has an initialization file value for the *Tree Depth* which is less than the current value, then the lowest value is used.

The second variable used to define the tree is similar to the first. The *Tree Spread* determines the maximum number of other agents that can be contacted by the agent in order to assist with a query. As with the *Tree Depth*, the *Tree Spread* is included in messages by the client, and an agent acting as a client. As well as being included by the client, the value is also specified in the agents initialization file. However unlike the *Tree Depth*, this value is not decremented for each node contacted.

Initially this value was set at 4, as this is the default value used by Gnutella. However is is possible to adjust this value to allow more or fewer agents to be contacted as required. As with the previous variable, if the *Tree Spread* value in the agent's initialization file value is less than the current *Tree Spread* value, then the lowest value is used. If a *Tree Spread* of 1 is used then a *chain* of agents is produced rather than a tree.

The default values for both variables are set in the agents initialization file and can therefore be changed to suit the conditions of the host on which the agent resides. However, the agent includes an internal limit of 6 for each variable in order to prevent deliberate flooding of the network with queries.

Instead, by maintaining Knowledge Bases, the agents in the Agent Trees system are able to direct searches in an effort to find information efficiently without the need for flooding. In practice this means that the agents which are most likely to be able to provide the data will be contacted first. Each result, or lack of result, is noted and contributes both to a score for the reliability of

Depth	1	2	3
Query 1	A	B	C!
Query 2	A	C!	D!

Table 5.3: Bypassing Agents without Data

the agent and also the knowledge of the subject/application that the agent possesses.

By using a directed search (also known as a *Semantic Routing*), it is possible for the agents to by-pass agents that have either no data or irrelevant data. Instead searches are directed to agents which are most likely to be able to assist effectively. This means that the search horizon is extended each time a query involving the same subject is run. This is the same technique as used by the NeuroGrid P2P system, which was described in section 4.5.5.

If, for example, the first time that a query is run, a list of random agents is chosen. If any of the agents in the tree that is created produces a result, then their score in the Knowledge Base of any agents between the source of the data and the root agent will be increased so that the likelihood of the agent being chosen for the next similar query is high compared to the other agents.

In the table 5.3, the Tree Depth is set to a value of 3. Agent A contacts agent B, which in turn contacts agent C. In the first query, only agent C is able to provide results - this is signified by the *!* symbol next to the agent name. During the second run, agent C is contacted by agent A and this results in agent D being included in the search as well as agent C. Since Agent B has no relevant data, it is excluded from the searches - thereby extending the search horizon.

5.1.5 Returning Results

One of the original design rules required that when results were returned, duplicate results were removed at each stage of the tree in order to reduce the network load. This meant that if three result messages with the same value arrived at a node then the result value would be kept but the sources would be merged - reducing the three messages to one;

```
<result>3.14</result><source>B</source>
<result>3.14</result><source>C</source>
<result>3.14</result><source>D</source>
```

would become;

```
<result>3.14</result><source>B,C,D</source>
```

While this reduces the network load, it does increase the time taken since the agent must wait until all of the results have been received before it can combine the messages and forward them to its requesting agent.

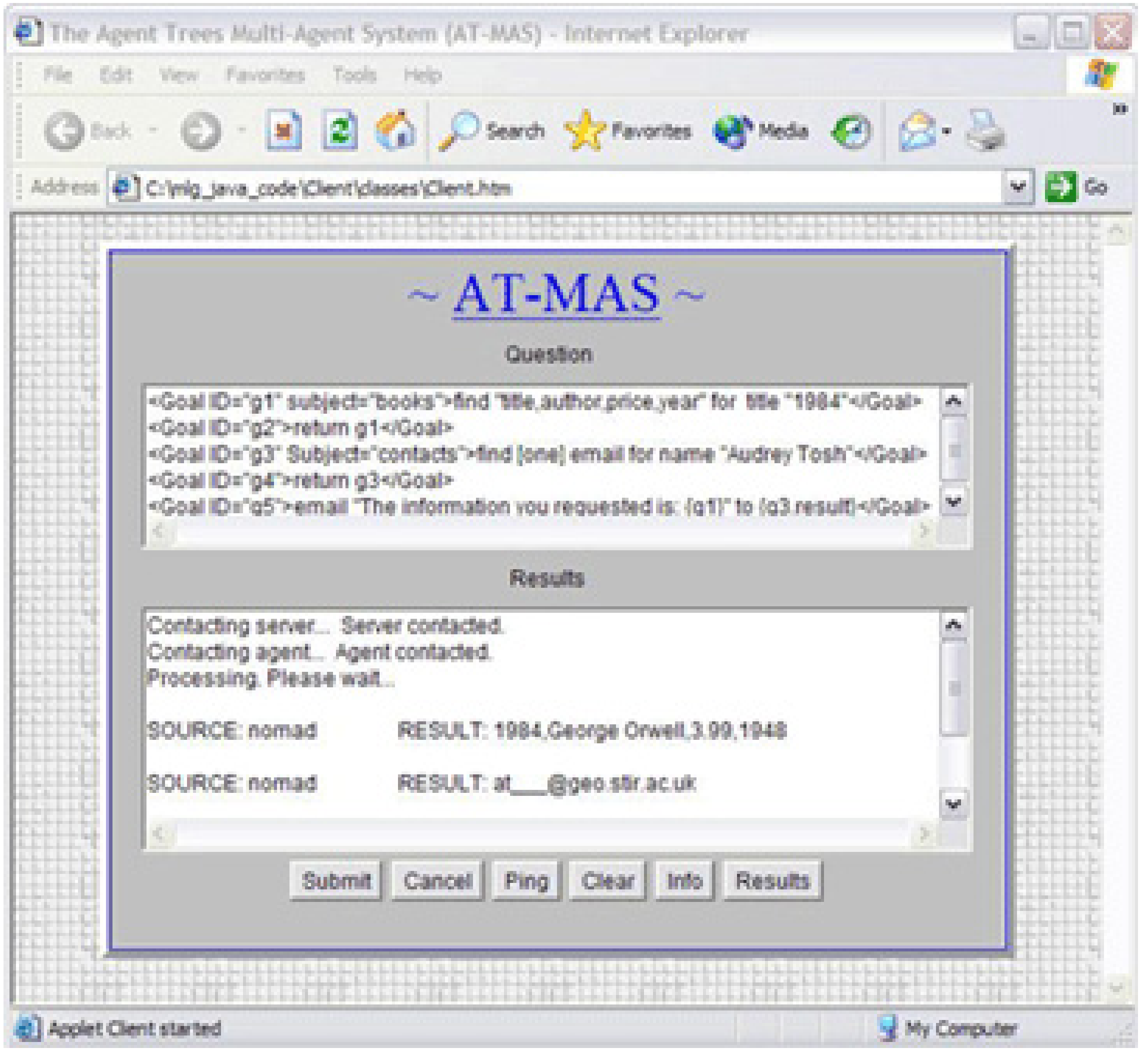
5.2 Components of an AT-MAS Node

All of the code of the AT-MAS system was written in Java[80] for the simple reason that it is a portable, multi-threaded modern language with a rich set of libraries.

Each node of the system contains the minimum of an Agent and a Server and a number of datasources. Two addition applications a Logger and an Admin application are optional. These components are described below.

5.2.1 The Client Applet

Although it is possible for any application to connect to the AT-MAS network providing it adheres to the simple communications protocol required (see section 5.4) and make requests, a simple java applet (see figure 5.4) has been developed to demonstrate the system. To use it, the user types in the list of goals into the top window and presses the submit button. The applet


The Agent Trees Multi-Agent System (AT-MAS) - Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\mlg_java_code\Client\classes\Client.htm
Go
~ AT-MAS ~
Question
Results
Submit Cancel Ping Clear Info Results
Applet Client started
My Computer

```
<Goal ID="g1" subject="books">find "title,author,price,year" for  title "1984"</Goal>
<Goal ID="g2">return g1</Goal>
<Goal ID="g3" Subject="contacts">find [one] email for name "Audrey Tosh"</Goal>
<Goal ID="g4">return g3</Goal>
<Goal ID="g5">email "The information you requested is: {g1}" to {g3.result}</Goal>
```

```
Contacting server...  Server contacted.
Contacting agent...  Agent contacted.
Processing. Please wait...

SOURCE: nomad            RESULT: 1984,George Orwell,3.99,1948

SOURCE: nomad            RESULT: at___@geo.stir.ac.uk
```

Figure 5.3: Screenshot of the AT-MAS Client

will contact the server to obtain the Task ID, the Tree ID, and the full Agent Address which consists of the name of the server and the port number on which the agent will listen for the client connection. The client applet will then contact the agent directly.

In many respects, Agent Trees is very similar to existing P2P systems. However there is one important difference; the nodes in a P2P system are usually run by individual users. As a result, many nodes may appear and disappear frequently as users connect and disconnect from the internet. The situation is becoming more stable with the increasing popularity of ADSL and similar technologies but it will be a long time before the majority of users machines will be permanently connected to the internet. Even then, few users will be willing and able to provide the amounts and kinds of data that will justify running an agent. In general users tend to be *information consumers* rather than *information providers*.

As a result, it was decided to separate the client connection from the individual agents themselves. By doing this, the need for *free riding* (see section 4.5.2) is removed. Only users/organisations that have data, and wish to make it publicly available, need to. As a result of this, there are fewer 'empty nodes' in the network.

The additional advantage of separating the clients from the agent nodes is that it becomes more feasible to create interfaces for mobile devices due to the small client requirements. It is possible to have the client connections as agents - in fact, any application can connect to the network providing it adheres to the correct protocol. This could eventually lead to AT-MAS clients being integrated with numerous other application such as operating systems, word processors, etc. In the current version of Agent Trees, clients are only able to connect to an agent and submit their requests via a java-enabled web-browser.

5.2.2 The Front-End Server

Since the system has been designed to be open, it is more vulnerable to malicious attacks or accidental damage than a closed system. Therefore, the front end server was added in order to provide an additional level of security and robustness. If the server suffers from a denial of service attack or any similar overload then the agent will still be able to continue working and complete the existing tasks.

One concern about the server that was expressed was that it would complicate the system to little advantage. However, this has not been that case since the server is only contacted at the start and the end of the agents interaction.

As the agents in the network learn about other agents and the information that they control, there is the possibility that the load on some nodes would be too high. This occurs because the information distribution across the network is not consistent and therefore some information could only be accessible by contacting certain agents.

Although the agents are not directly subject to the Slash-dot effect[112] in which a web-site receives popularity and is swamped as a direct result of people visiting it, a similar effect may occur when a large number of users request the same information and the agents with access to the information are overloaded. Rather than allowing the agents to simply become swamped with requests, the system has been designed so that each agent is protected by a small server.

Initially, a client will contact this server and if the agent is able to process the request, then the address of the agent is passed to the client along with a task ID. The task ID is also passed to the agent and used to ensure that only valid messages are responded to. When a request is completed by the agent, it sends a message to the server. The server then removes the task from its list of current tasks.

If an agent becomes overloaded with requests, the server will refuse to accept any more until some of the existing requests have been completed by the agent. This refusal will be reflected in the remote agents knowledge base scores for the agent. As a result, agents which are consistently overloaded will be relied on less. This, in turn will cause the loading on the agent to decrease and the network load will spread between other nodes.

Although this functionality could be performed by the agent itself, it was decided to implement a separate server. The reason behind this was so that if the server was subject to attack then the agent would still be able to function independently - it would just not be able to begin any new tasks.

Any agent which is refused a connection by another remote agent will make a note of this failure and it will be used to influence its choice for future requests in the short-term.

Similarly, if a server is the victim of a denial of service (DOS) attack, this will be limited to the single node of the network, and the remaining nodes in the network will route their messages to other servers and agents. This means that the network is more robust than other MASs because there is no single point of failure. There is no dedicated name server and no node relies completely on any other node. If a node fails then the rest of the network will continue unaffected except for occasionally checking to see if the node has been restored. In the current implementation of the system, the agent must reside on the same computer as the server. However with only minor changes to the code, it will be possible to allow a single server to protect a number of agents on different computers.

5.2.3 The Logger Application

Although not required during the normal day to day operation of the AT-MAS system, the Logger is a useful means of obtaining and managing the timings for the client, server and agent. These are sent to the logger by the various components and when a complete set of results are obtained, the logger writes a line to the log file. Use of the Logger requires that its address is sent by the front-end server to the client (or remote agent) as part of the *Request Accepted* message. This allows the client to send a message to the Logger when the task complete message is received from the agent. Use of the logger is determined by settings in the agent initialization file and was necessary to obtain the results in chapter 6.

5.2.4 The Admin Application

A simple admin application has been created in order to allow system commands to be sent to the other components in the system. For reasons of security, the Admin application can only affect the components on the same host machine. When this component is activated, it reads the initialization file and retrieves the Admin port number and the security key which are used in all of the messages that are sent. This guarantees that the messages are genuine.

Although the Admin application has been designed so it is possible to send a number of different messages (by providing a different command-line parameter when the component is run), only one message has been implemented. This message allows all of the AT-MAS applications on a single host to be closed down at the one time. It then sends shutdown messages to the server, agent and, if it is active, the logger. Upon receipt of a valid message (i.e. a message received from the the correct address and containing a valid security key), each system component will end its execution. After sending the messages, the admin application will close.

5.2.5 The AT-MAS Agent

The agent has been designed as a number of multi-threaded components which operate in parallel so that multiple tasks can be processed at the one time. In addition, extensive use is made of thread pools which allow a group of threads to monitor a message queue. When the agent is waiting for input for the task - for example results from another agent - the task is placed in a *paused Queue* until either the inputs arrive, or a timeout occurs. While this means that the tasks may take longer to process, it also means that a task will not be stalled while the running task waits for results to be returned from another agent and as a result the agent works more efficiently.

One of the critical decisions made was to restrict each node to a single agent rather that creating each node as an MAS. In a very early design, the agents were transitory - they were created as needed and shared information via a blackboard system. This design was abandoned as it became clear that a single persistent agent with its own knowledge base would be preferable for a number of reasons.

Firstly, having a number of transitory agents means that there is a higher maintenance overhead. This increased time would be taken up with the creation of new agents, and their destruction after a task has been completed which would waste resources and reduce the number of tasks that an agent can carry out at the one time. Since the single agent is designed to work on a number of different tasks at the one time, there is no real advantage in having a number of different agents active in each node at the one time.

Secondly, if each node was contained within an MAS there would be two distinct levels of communication; the inter-node communication and the in-node communication. With a each node containing a single agent, only the inter-node communication is required, reducing the overall communication significantly. One solution to this would have been to use a blackboard approach (see section 3.3.3) but it was felt that this would add unnecessary complexity to the system.

Additionally, multiple agents introduce data consistency problems since each agent would have its own knowledge, potentially causing conflicts. Whilst it is acceptable for the agents in different nodes to contain different information, different agents within the same nodes containing inconsistent information may lead to some queries being handled differently.

It is important to note that although AT-MAS was created as a network of single agents for the reasons given, it is possible to create re-code the system so that each node contains a complete MAS. Providing that the agent conforms to the communications protocol, there is no requirement for an agent/agent system to be implemented in a certain way. Therefore it is possible for other AT-MAS Agent nodes to be implemented in different ways.

5.3 The AT-MAS Agent dissected

5.3.1 The Knowledge Base

When an agent creates and sends messages requesting assistance with a task, it doesn't send them out to all of the other agents that it knows. Instead, the agent searches its own knowledge base for the best set of agents to contact for assistance.

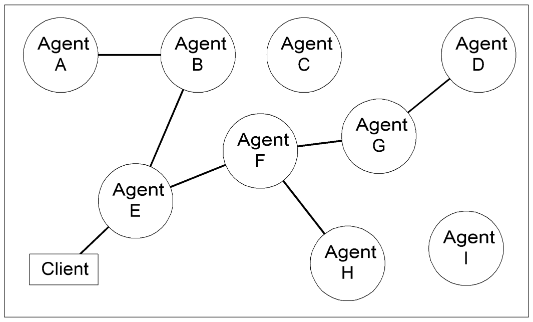


Figure 5.4: Pattern of Connections 1

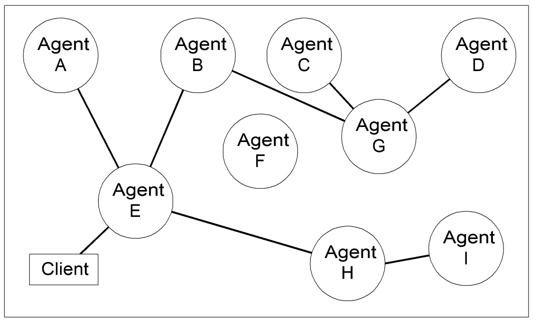


Figure 5.5: Pattern of Connections 2

Since there is no central server or directory providing information about the agents and the various services that they can provide, the decision about what constitutes a best set of agents for a particular task is based solely on the agents previous experience. In order to build up this information, the agent keeps track of the results of any requests that it makes: every successful result, and every failure is noted, and these are used to influence future queries.

Due to the wide range of tasks that an agent may be required to assist with, it is important that it gathers as much information about the other agents as possible - rather than relying on a single, or small number of values. The reason for this is that the distribution of information within the network is not consistent. This means that some agents may be more suited to answering questions about books, others may have access to personal data; names, addresses and telephone numbers, etc. In other words, the most suitable set of agents will be different for every subject, and this must be taken into consideration at the appropriate time.

For example, in a trained[2] network of agents, a request for the subject of Astronomy may produce the pattern in shown in the figure 5.4, whereas a request for a specific email address may produce the pattern in the second figure 5.5. As a result of this monitoring process, an agent is able to learn the strengths and weaknesses of the other agents in the network. This in turn provides the agent with the information about which agents to contact for assistance.

Even after the optimal set of servers has been found, the agent will still attempt to contact some agents outwith the set. This exploration is needed so that any new agents with knowledge of a particular subject or any existing agents that have acquired new knowledge can be identified. This is important since removing the dependence of external directories means that there is no effective way for an agent to advertise its new capabilities, without having to resort to sending information request messages.

Additionally, each agent stores a server index for every remote agent that it is aware of. This is used to give an indication of the agents reliability and level of cooperation. Every time that an agent assists with a query - even if the query is ultimately unsuccessful, its index value is incremented. A refusal to assist means that the index is decremented. However, for agents which start a transaction, but do not finish it, the penalty is more severe - with a larger decrement. The reasoning is that this is more costly to the agent making the request as it has committed resources to contacting the remote agent.

5.3.2 Communications Component

One key design decision when creating the AT-MAS was to maintain the connections to other clients, servers, and agents simply for the length of time that they would be needed. This

[2]A trained network is one in which the agents have been functioning for some time and have gained knowledge about the other agent in the network

is in contrast to P2P systems such as Gnutella which maintain a small number of permanent connections.[171, page 129] The reasoning behind this decision was that it would reduce the load on the network and allow the agent to connect easily to any currently active agent in the network. How much load this is depends upon the connection but certainly HTTP (for example) use short-lived connections and this is seen to help with efficiency.

Many of the problems caused when creating the AT-MAS agents were as a direct result of the communications. This is due to the complexity of the Communications component which must be able to function in a number of different ways depending on the type of connection:

1. Connections must be maintained until explicitly closed by the initiator.
2. The agent must be able to receive connections from clients, the local server, remote servers and remote agents.
3. The agent must be able to initiate connections to remote servers and remote agents
4. Communications are asynchronous with messages being sent and received at any stage in the process.

A brief summary of the agent communications process is as follows;

1. The agent receives a message from the server with a port number, client information and task information.
2. The agent opens the port specified by the server and waits for a message from the client,
3. The client connects to the agent, which must keep this connection open for further messages[3],

[3]see section 7.4 - Future Work for information about changing the code to add support for transitory connections

4. Finally, the agent breaks the connection and closes the port after all of the data has been sent.

In addition, due to the nature of the system, the agent must also be able to behave like a client.

1. The agent must contact a remote server, as though it were a client.
2. The server will provide the task information and port number of the remote agent.
3. The agent will close the connection to the server and contact the remote agent on the port specified.
4. The agent sends a "New Task" request to the remote agent and when it receives an "Request Accepted", it sends the question (consisting of a number of goals) to the remote agent.
5. Finally the agent waits for a number of results messages - if appropriate to the question, followed by a "Task Complete" message which provides details such as the number of goals completed and the number of results messages that should be received.

5.3.3 The Planner Component

The first part of the planning process involves the agent checking its own rule base to find a goal condition associated with the request. If the agent finds a command which satisfied this condition it will link together actions until it finds the combination that will produce the required end result.

Numerous combinations of actions may exist so it is important to find the *best* plan in the situation. In order to do this, when a plan is created, it is assigned a value which is based on both the chance of the plan succeeding and the cost of carrying it out. This allows the plans to be stored in order of effectiveness so that the least costly plans with the highest chance of success will be attempted first. In order to do this, the following calculation is used.

$$\text{plan fitness } (\mathrm{F}_{plan}) = \frac{CF_{A1} * CF_{A2} ... * CF_{AN}}{COST_{A1} + COST_{A2} ... + COST_{AN}}$$

Each action in the plan Ai has an associated confidence factor *(CF_{Ai})*. This figure represents the probability that the action can be completed successfully. Since all of the actions in the plan must be completed for the plan to be completed, the product of these values is the chance that the plan will be completed successfully.

Similarly, each action has an associated cost value Cost_{Ai}). This represents the cost in terms of processing resources and time that completing the action uses. By taking the sum of the costs for the actions in the plan,it is possible to obtain a value for the total cost of the resources required for the plan. Both the confidence factor and the cost values are stored in a system file and may be altered to suit the local system.

It is possible for a more complex evaluation to be made concerning the success of the plans, but it was felt that this may impact on the efficiency of the planning process, and reduce the effectiveness of the agent.

5.3.4 The Status of Actions

When the actions in a plan are executed, the agent checks the result before attempting any further actions. This is necessary since a failed action results in the failure of a plan, and forces its abandonment. For some actions, such as searching one of the locally stored databases, the result status is available immediately.

However for other actions, such as contacting a remote agent, the agent must wait until the result status of the action is known. In order to allow other tasks to be processed during this time, the task for which the result is not yet known is paused until the result is available.

Any action which produces a delayed result, generates a *pending status message* which is sent to the ProcessStatus object. This message is stored in a message queue until the result of the action is known. At this point in time, the task is paused while a separate process completes the action. When the action is completed, a *status* message is generated which is also sent to the ProcessStatus object. The two messages are matched and processed.

Depending upon the nature of the action, different numbers of status messages may be required to generate a pass result. For some, only a single successful result is required, but for others, all of the results must be successful for an overall pass result.

When the required number of passes have been received, the *statusPending* field of the *pending status* message is parsed. This contains information that allows variables to be set, updated, deleted, or to have values appended. For example, when a *new task* request message is sent to another server, the *status pending* field contains the following information; *goal|SERVERS_ASKED|[receiver]|append*

When processed, this information causes the name of the remote server to be appended to the list associated with the SERVERS_ASKED variable. Multiple variables and updates are possible depending upon the different circumstances. This allows the Post-conditions of the action to be set and ensures that the plan is up to date.

Another set of variables which can be changed are those which are used to determine when the task is removed from its paused status, ready for the next action or next plan depending upon the result. When a number of messages are sent to remote servers, the task remains paused even when a pass result is known. This allows the agent time to receive and process the result messages from the remote agents before continuing with the plan execution. In this case, the task is restarted by the *ProcessMessages* object which processes all of the incoming messages received by the agent.

5.4 AT-MAS Communications

The initial idea was to use a dialect of KQML for the communications. This was in part due to the fact that FIPA standards were being developed and at the time of the initial design of the system.

In addition, KQML was chosen in preference to FIPA-ACL since it allowed for the language to be extended providing that none of the existing performatives were redefined. This is important as the agents in the system needed to be able to request the services of other agents without the need to Register, and Advertise their services with other agents. Instead, it was felt that a version of an Ask performative would be more suited to this task.

As the structure of the conversations between the client (local agent),the remote server, and the remote agent were defined, it became clear that the existing languages were not suited to the conversations/interaction planned for the AT-MAS agents, but were instead more appropriate for implementing the advertisement based systems commonly found. The most obvious difference is that both KQML and FIPA-ACL include performatives that allow an agent to subscribe to a system and advertise its services via a Middle Agent (see section 3.2.1). In AT-MAS, this capability is redundant since there are no Middle Agents.

Further differences become apparent when considering the AT-MAS requirement for the Task ID and Tree ID fields which neither KQML or FIPA-ACL currently possess. While both languages included a MessageID and an In-reply-to field, these were not really suitable. The reason for this is that the Tree ID refers to the complete query and therefore is used by a number of different agents, but the Task ID is assigned by the server and is used both as a reference and as a security key. Therefore it is preferable that the Task ID is not known outside the conversation between the two agents.

5.4.1 Message Format

Another important design decision concerning the communications was the choice of language to use. Although lisp was the original choice for KQML, and subsequently FIPA-ACL, some work has been done to update the messages to use XML[69]. However, it has always been the intention to implement the language using a simple form of XML for a number of reasons.

The first reason is that it is possible to extend the message format easily. New fields can be added as the language is developed, and unused fields can be omitted. Similarly, it is easy for an agent to simply ignore the fields that it is not expecting, or does not require. For example, if the AT-MAS protocol was used as a container for a different language, then an XML tag would need to be included to specify this. In cases where the language used is the AT-MAS ACL, then this tag is redundant and is omitted.

```
<RECEIVER>nomad:agent:1234</RECEIVER>
<SENDER>magellan:agent:1237</SENDER>
<ACTION>New Task</ACTION>
<LANGUAGE>FIPA-ACL</LANGUAGE>
<QUESTION> ... </QUESTION>
```

Secondly, the XML allows complex nesting to be used when required. This means that a message can include XML data fields if required. The following example is the EOD field of the task complete message from the agent.

```
<EOD>
    <GOAL ID = ”g1” MESSAGES = ”4”>
    <GOAL ID = ”g2” MESSAGES = ”1”>
</EOD>
```

Field Name	Description
Sender	The address of the agent to receive the message
Receiver	The address of the agent sending the message
Action	The action/request to be carried out or message type
Tree ID	The query identification code
Task ID	The authorisation code from the server
Security Key	Authenticate messages passed between the agent & server

Table 5.4: The AT-MAS ACL Required Message Fields

5.4.2 Message Fields

The message fields in the AT-MAS ACL can be divided into two categories. The first category contain the fields which are required as part of every query made to the agent. These include the address of the message sender; the address of the intended recipient; the action field, which either contains the action which the message is intended to perform; e.g. Ask, Tell, Accept, Refuse, etc; or the type of the message e.g. Result or Task Complete. In addition, the Task ID and the Tree ID are also required. The Security Key is a required field in messages that are exchanged between the server and the agent. This is used to ensure that the messages are genuine. The required messages are summarized in table 5.4.

The second set of fields are only required in certain messages. These are summarized in table 5.5.

5.4.3 Conversation Format

Conversations in the AT-MAS system are based on a simple protocol. As mentioned in sections 5.1.2 and 5.2.2, the client (or an agent acting as a client) must contact the server in order to

Field Name	Description
Question	This is the list of goals submitted to the agent
Result	This is the result of a goal
Source	This is the name of the agents which supplied the result
EOD	This contains a list of the goals and the number of results sent
Data	This is a container tag for miscellaneous information.

Table 5.5: The AT-MAS ACL Context Specific Message Fields

obtain a Task ID or Tree ID (if required) and the address of the Agent. Following this, the client can send either a simple or a complex task to the agent.

Complex tasks consist of a number of goals to be completed which involve searches and processing of information which the agent must either retrieve from its local databases or from other agents. The results of these tasks may require any number of results messages to be returned to the client.

Simple tasks are formed as simple *Ask* commands which can be answered with a single *TELL* message. The most common type of simple query is a basic request for information, this returns two lists; the first is a list of remote servers that the agent knows, and the second list is of the subjects that the agent knows.

The format of the agents conversations is summarized in the table 5.6. However for the sake of clarity, *refusal* messages from both the server and agent have been omitted. These messages can be sent at any point during the conversation and effectively terminate it.

5.4.4 The AT-MAS Language

The AT-MAS language consists of a four simple commands, although a further three commands are planned for future versions - see section 7.4.5 for more details.

Contacting the Server

Source	Destination	Message	Description
Client	Server	New Task	Requests access to the agent
Server	Agent	New Task	Task ID + Tree ID + Client Address
Server	Client	Accept	Task ID + Tree ID + Agent Address

Contacting the Agent - Complex Request

Source	Destination	Message	Description
Client	Agent	New Task	Task ID + Tree ID + Question
Agent	Client	Processing Please wait...	
Agent	Client	Result(s)	Repeated for each unique result
Agent	Client	Task Complete	Includes the No. of Messages
Agent	Server	Task Complete	Allows the server to update its task count

Contacting the Agent - Simple Request

Source	Destination	Message	Description
Client	Agent	Ask	Task ID + Tree ID + Question
Agent	Client	Tell	Answer

Table 5.6: The AT-MAS ACL Conversation Format

Find This is the main command that is used to retrieve data from the system. It requires four variables:

subject Although each agent may implement its data storage, and consequently, its retrieval methods differently, this variable contains that subject of the query. In the basic system, the value of this field corresponds to the name of the XML data file that is to be searched for the answer to the query.

Again, future versions of the system will allow an initialization file switch that will specify the behaviour of the system if the subject has been left blank. For systems with small amounts of data, it may be feasible to search all of the data repositories if the subject has been left blank. However, for agents controlling large amounts of data this may not be practical.

Value to Find This may consist of either a single value, or a list of comma separated values. This is the name of the field containing the value(s) which are to be returned to the user. If more than one value is requested, then the values are returned in the order that the field names appear in the query. For example, the results returned from a request with a *value to find* list of 'name,age,email' will be different from the results returned when the *value to find* list consists of 'name,email,age'.

Value to Query This is the field name that the search criteria is to be matched against.

Query Criteria The criteria can either consist of a full string, or a string containing the '*' wildcard. This will be compared to the data and if a match is made, then the data will be processed to obtain the results.

It should be noted that this command has different effects depending upon whether the command is being executed by a *root* agent or not. If the agent is not a root agent then the data will be returned to the agent that requested the search. If, however, the agent was the root agent, then that results of the query are not returned to the client but are instead stored until they are required in the completion of another goal.

<GOAL ID="g1" SUBJECT="music">find 'title,artist' for year "20*"</GOAL>

The above example is of a query which would return the title, artist and price for any music that was released since, and including, the year 2000.

Return As mentioned in the previous description, the results of a query are only returned if the agent is not a root agent. This is to allow the client the option of not receiving the results from a particular query. If the client requires the results to be returned, then this must be done by using the *return* command. This command only takes a single parameter which is the identifier of the goal which has the results to be returned. A typical example of this command in use may be as follows:

<GOAL ID="g2">return g1</GOAL>

Since the parameter is solely the name of the goal, it does not need to be enclosed in curly braces " and " like the *email* command.

Email This command is similar to the previous command (Return) in that it is only executed by the *root* agent. The command takes two variables which are the *Text String* to be sent and the *email address* of the person that the data is to be sent to. Due to the way that the agent is able to locate information and use it in later goals, it is possible that both the Text String and the Email Address be obtained through the completion of previous goals. If goal data is to be included in this way then the name of the goal must be included in curly braces.

<GOAL ID="g5">email "Hello {g1.result}. How are you?" to {g2.result}</GOAL>

This is in contrast to the *return* statement which does not need braces around the goal reference.

Message The message command allows an agent to *TELL* the calling agent/client an item of information. This is similar to the return command except that all agents can use it, and that the Text String returned may consist of more than just results. As this is the case, results from a goal must be enclosed in the curly braces.

Although these simple commands are adequate to test the system, it will be possible to add commands at a later date. These new commands are described in section 7.4.5 as part of the future work to be carried out on the system.

5.4.5 Support for Other ACLs

Although this aspect of the system has not been extensively tested, the message format allows for messages in other languages to be passed on by the agent. When this happens, the agent acts as a proxy. The messages are passed on to other agents which can process them, and the results are returned to the calling agent of the client. This is similar to the process which occurs when the agent is not able to provide information on a specific subject. A further similarity is that the agent will store information about the other agents which can use different languages and can direct the requests to the correct agents in future.

```
<RECEIVER>nomad:agent:1234</RECEIVER>
<SENDER>magellan:agent:1237</SENDER>
<ACTION>New Task</ACTION>
<LANGUAGE>FIPA-ACL</LANGUAGE>
<QUESTION> ...</QUESTION>
```

5.5 Chapter Summary

This chapter describes the Agent Trees Multi-Agent System (AT-MAS). Firstly the an overview and the basic operation is described. This includes a description of the how the basic tree is formed and how the agents are able to discover new agents during the normal running of the system.

The second main section of this chapter details the separate components of the AT-MAS system. Of these, the most complicated component is the AT-MAS Agent. This is described in detail, with many of its internal components described.

Finally, the AT-MAS communications are described.

Chapter 6

Evaluating the AT-MAS System

6.1 Evaluation by Results

6.1.1 Obtaining the Results

In order to test the system a small dedicated network of 4 computers was used, each of which were running Microsoft Windows XP Professional with Service Pack 2. The computers were all running Java Runtime Environment 1.4.2_03. and were connected by a 100Mbps network.

Name	Processor	Memory	Free disk space
A	Intel Pentium III, 648MHz	384MB	2.64GB
B	Intel Celeron, 299MHz	128MB	401MB
C	Intel Pentium III, 548MHz	128MB	6.25GB
D	Intel Celeron, 299MHz	128MB	637MB

Table 6.1: The computers used for testing

In addition to the Agent and the Server, the Logger (see section 5.2.3) application was activated. This waits for timing summary messages from the Server, Agent and the Client. When it receives them, the logger writes a summary line to a log file. Although running the Logger application has an effect on the timings of the system, it is not possible to obtain accurate timings without using the Logger application. This means that the exact difference in timings cannot be determined. Therefore, for the purpose of the testing, the effect is assumed to be consistent across the whole range of computers. This is acceptable for the tests carried out since they are based around the relative timings of the system.

In each case, the same set of 5 goals are used.

```
<GOAL ID="g1" Subject="books">find 'price' for title "1984"</GOAL>

<GOAL ID="g2">return g1</GOAL>

<GOAL ID="g3" Subject="contacts">find [one] email for name "Audrey Tosh"</GOAL>

<GOAL ID="g4">return g3</GOAL>

<GOAL ID="g5">email "The information you requested is {q1}" to {q3.result}</GOAL>
```

In the tests, each set of results is made up of the average timings for 50 runs. This is intended to reduce the effect of factors such as network load and delays due to multi-tasking or paging - however, these inconsistencies cannot be eliminated completely. In tests which take a shorter length of time, such as those involving a single agent, the effects are more noticeable.

The graph in figure 6.1 shows a complete set of results for the first of the tests to determine the scalability of a chain of agents. It is assumed that the high values for the first run are due to the computers paging the agent code into memory.

This is further illustrated by figure 6.2 which shows the separate frequency distribution graphs for each of the agents.

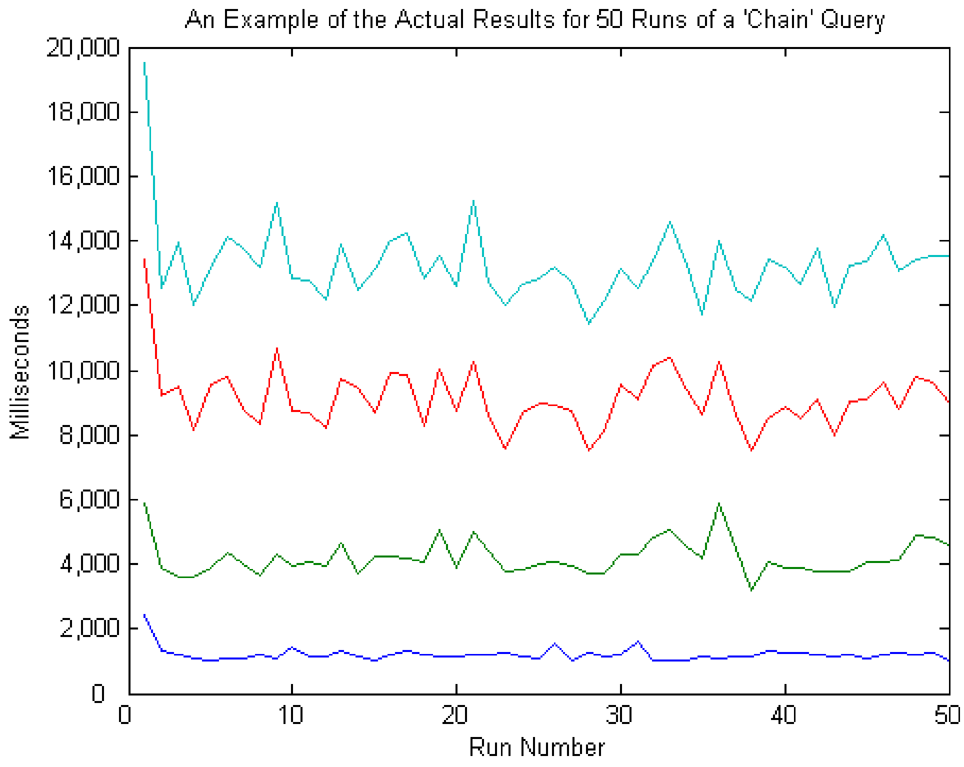


Figure 6.1: The actual results for the chain of agents A-B-C-D. The top line shows the total time taken for the sequence, the next line shows the time taken for agent B, the third for agent C, and bottom line for agent D.

6.1.2 Scalability Testing

The first set of tests were carried out to determine whether the system has the potential to scale. Figure 6.3 shows the increasing run times when contacting longer chains of agents. The linear, rather than exponential increases in timings, while not proving conclusively that the system can scale, supports the claim that it can. An exponential graph would have strongly suggested that the system was not scalable.

Figure 6.4 shows the results of a number of tests in which different agents were contacted at the same time. One important feature of this graph is that some the data lines are of an unusual shape. This is due to the fact that when an agent contacts a number of different agents, there

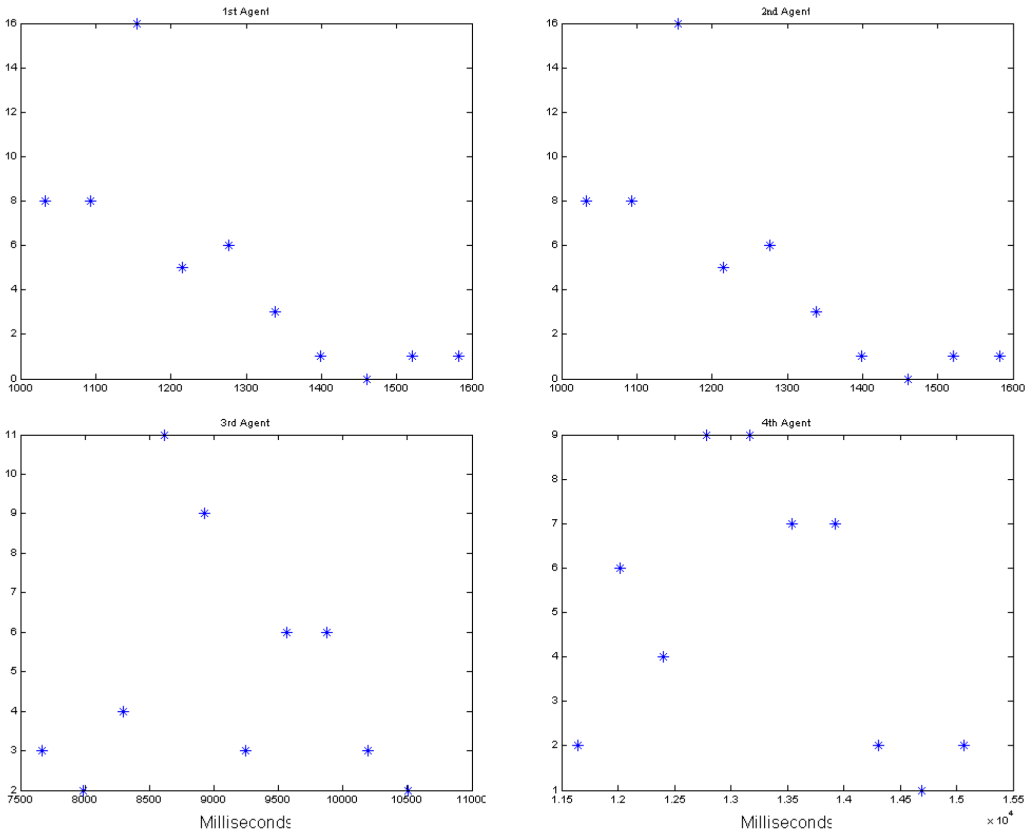


Figure 6.2: The distribution graphs for the 4 agents shown in figure 6.1. The first run has been omitted. Plotted points show the histogram frequency counts (sum is 49).

is no way of determining the order in which the results messages are received from the remote agents. However, the graph shows that the final (endpoint) timings are linear.

In graphs 6.3 and 6.4, each data line represents the timings for a different sequence of agents and as a result it can be seen that the order in which the agents are contacted does not matter to the timings.

Having shown that the AT-MAS system scales in a linear fashion for both chains (see Figure 6.3) and trees (see Figure 6.4) of agents, tests were carried out to see if the number of queries that an agent performed had a significant impact of the timings. Previously, the agent had been tested using the same query containing 5 goals. In the following test, used to determine the

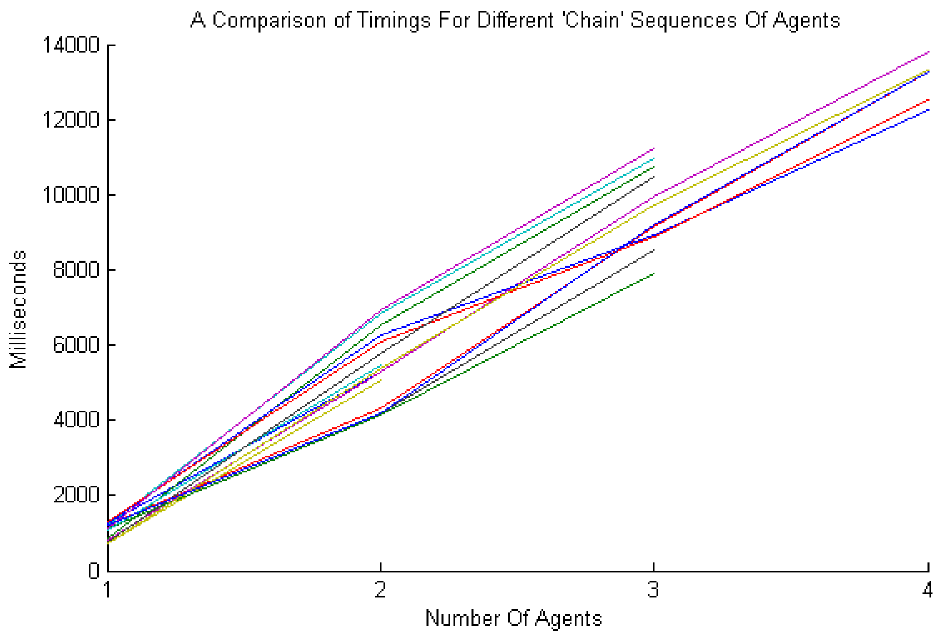


Figure 6.3: A Comparison of Timings for Different Chain Sequences. The 15 results are each from an average of 50 runs.

scalability for different numbers of goals, a single agent was given between 1 and 10 identical goals to complete as part of the same task.

Although the results in figure 6.5 are not completely linear, they show a general trend towards linearity. However, since the length of the query is very short with the queries taking between 1.2 and 1.8 seconds, any randomness introduced by the operating system (for example paging or process switching) will have a considerable effect.

6.1.3 Knowledge Base Updates

Under normal circumstances, the knowledge levels for each subject in the Knowledge Base are based on percentage values which represent the agents knowledge of a subject. As the network is run, the values are incremented and decremented depending upon whether a query was completed

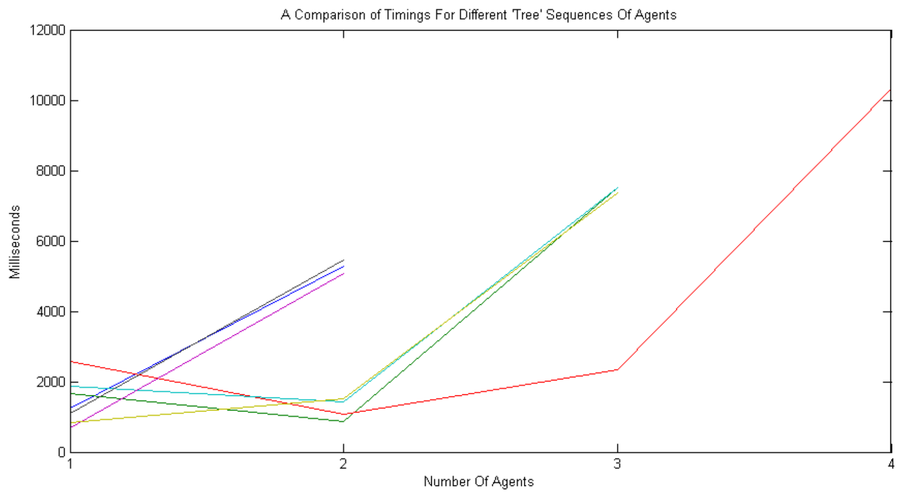


Figure 6.4: A Comparison of Timings for Different Tree Sequences. The 7 results are each from an average of 50 runs. The X axis shows the number of agents contacted. Each line shows timings for contacting each agent. The Endpoint shows the total time taken. The leftmost lines show A contacting B, A contacting C, and A contacting D. The middle lines shows A contacting B and C, A contacting B and D, A contacting C and D. The rightmost lines show A contacting B, C and D.

successfully or not.

Figure 6.6 shows how the levels for 8 subjects in an agent's Knowledge Base vary when 75 separate goals were run in a random order until 2,00 queries had been completed. In this test, the agent has access to data which allows it to answer some, but not all of the queries. As the number of runs increases, the level of information in the Knowledge Base begins to reflect the actual level of information that the agent has access to.

The results of these tests were inconclusive. While random testing seemed to be the best way of performing the experiments since it most closely mirrors the normal operation of the system, the fact that there were an uneven number of queries meant that some were being run more often than others. As a result, the levels in the Knowledge Base changed unevenly and never accurately reflected the levels of knowledge in the agents knowledge base.

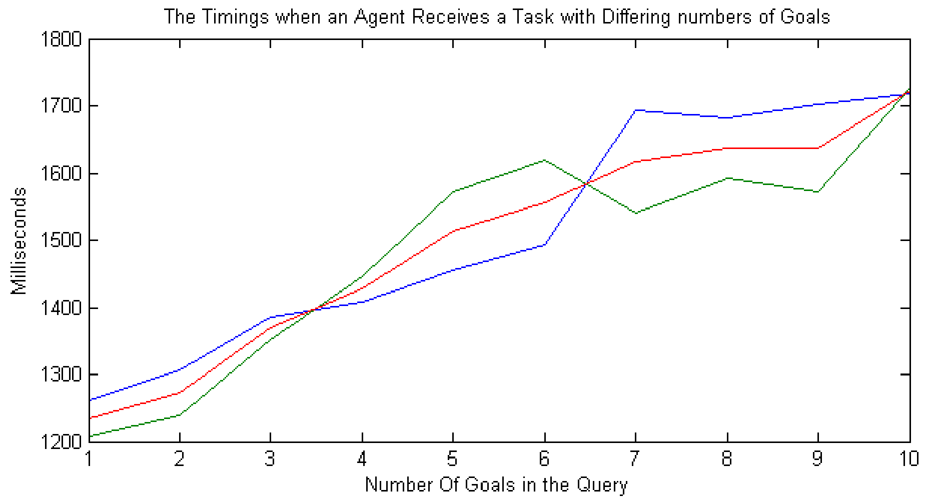


Figure 6.5: A Comparison of Timings for Different Numbers of Goals

For example, from the distribution of queries, it is most likely that a query for the subject of music will be run with a 15% chance - compared to queries on the subject of films with a 3% chance. In addition, successful queries had a higher chance of affecting the queries, with 31 out of 52 queries being passable.

This is summarized in table 6.2.

Since the tests were performed, the algorithms have been re-examined. One feature of the algorithm which was used to update the knowledge base is that all of the increments and decrements are for the same amount:

$$increment = (1^{+}_{-}(50 - subject)/50))$$

with the value being added to/subtracted from to the subject depending upon whether the update represents a success or a failure. As a result, the probability is that the queries for subjects which have a greater than 50% chance of succeeding will eventually force the knowledge level to 100% and the queries with less than a 50% chance will be forced to 0%. Instead, if the increments and decrements were set as follows:

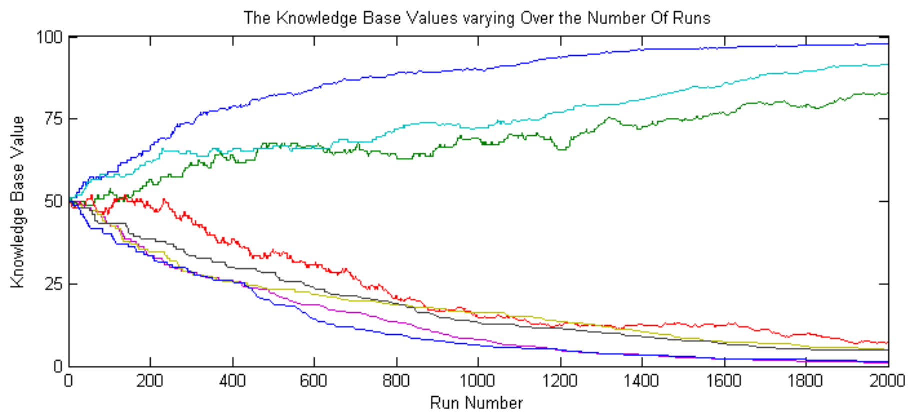


Figure 6.6: Knowledge Base values varying as random queries are processed

$$increment = k.(1 - x)$$

$$decrement - k(x)$$

then the values should stabilize.

6.2 Evaluation by Definition

There are a number of criteria by which both the system and the entities within it must be compared if the project is to be classed as a success. Since the AT-MAS system is the result of a combination of different technologies - specifically Intelligent Agents, Multi-Agent Systems (MAS), Mobile Agents and Peer to peer (P2P) systems - it is important to evaluate it in comparison to them.

6.2.1 Is the AT-MAS agent a true agent?

By the more simple definitions, the AT-MAS agents are true agents as they are able to complete a set of tasks for the user without the need of every set of the process explicitly defined by the

Subject	Passable Queries	Percentage Passable	Actual Results
Books	9/13	69%	97.89%
Contacts	8/12	66%	83.01%
Music	9/20	45%	7.28%
Bibliography	5/7	71%	91.70%
Astronomy	0/7	0%	1.08%
Films	0/4	0%	4.90%
Vehicles	0/5	0%	4.70%
Computer games	0/7	0%	1.27%

Table 6.2: A Summary of the Knowledge Base Levels after 2,000 runs with initial Knowledge Base value of 50%

user. When the user makes a request like -

<GOAL ID="abc">Find "album,artist,price" for artist "REM"</GOAL>

- a plan is formed and the actions in the plan executed. If the agent has direct access to the data, then the goal is easily completed. It is slightly more complicated than if the user were to type an SQL statement and send it to a local database. Instead, the agents abilities become apparent when the data cannot be found locally. The ability of the agent to make judgements about the probable location of the data indicates a level of intelligence - however limited - and the ability of the agent to try a number of different techniques to complete the goals set indicates a level of autonomy.

A further indication of autonomy is that the agents are persistent. They will continue to function even when there are no tasks to be completed for the users.

One factor that counts against the claim for agency is that the agents are cooperative. That is, they will automatically attempt any task that is requested rather than deliberate about whether the task is in the best interests of itself or its owner. This lack of self-interest may be seen as a drawback, but there are numerous MASs in which automatic cooperation does occur.

A second argument against claim for agency is that the agents are not pro-active. The agents react to user requests but they do not pro-actively search for information on their own. In section 2.3.1, it was suggested that *pro-activeness*, while not being a defining criteria in the same way that *autonomy* was, still counted added weight to the claim of agency.

In distributed applications, pro-active behaviour which increases network load should be kept to a minimum whenever possible. Consequently, such behaviour has been considered as a future enhancement - see section 7.4.4 for further information. It is possible to implement pro-active behaviour by altering the code so that when there are no outstanding tasks, a *system* task would be generated by the agent that would query other agents to enquire about their capabilities. It was subsequently decided that this would be undesirable due to the additional network traffic generated.

Each AT-MAS agent contains its own Knowledge Base storing its current *beliefs*. Their *goals* are provided by the users - and the agents have an implicit *intention* of attempting to complete them. Furthermore, the agent has commitments in that it continues to attempt a goal until it discovers that the goal is no longer possible.

In spite of this, these agents cannot realistically be considered as *strong* as their have only very limited knowledge base both in terms of form and structure - as the knowledge only consists of the other agents known, and the local agents knowledge of the subjects that the other agent can assist with. There is no capacity within the agent for storing more general knowledge. In addition, there is no capability for the agents to perform reasoning about the knowledge that they posses.

6.2.2 Is AT-MAS a Multi-Agent System?

Having established that the AT-MAS agents can be accurately classed as agents, the classification of AT-MAS as a MAS is relatively easy. By the definition provided in chapter 3, a MAS is a group of agents which interact - the actions of each agent must be able to affect the other agents in the system, either directly or indirectly.

The basic operation of the AT-MAS system involves this kind of interaction; agents request information from other agents in the system and, in turn, provide information to other agents (or clients) in the system. Therefore it is possible to accurately describe AT-MAS as a MAS.

More completely, it is accurate to describe AT-MAS as an open MAS since agents from unknown sources can participate.

6.2.3 Is the AT-MAS system a viable alternative to existing MASs

AT-MAS was intended to be a very simple MAS. It achieved this by reducing the reliance on middle agents and supporting services. Similarly, some of the features; such as support for mobile agents, ontologies and brokered/mediated transactions are not present in the system. However, the success of the AT-MAS system proves that these are not always required.

6.2.4 Is the AT-MAS system a viable alternative to Mobile Agents

As stated before, mobile agents are a way of allowing data to be filtered/processed on remote computers so that only the required information is transmitted across the network. However, in order to do this, the code of the mobile agent must be transported to the remote computer.

In order to allow this, the administrator/owner of the system must install an agent framework consisting of an agent platform which hosts a number of system agents and/or services which the remote agent can interact with.

A number of different agent frameworks exist; each with different levels of support and interfaces. In spite of all of the safeguards present, it still remains the fact that the agents are often created by an unknown third-party. As such, trust is always an issue - trust that the agent is well behaved, trust that the MAS and support services are robust enough to prevent any malicious actions from disrupting the system. In AT-MAS, the administrator/owner has complete control of all of the code and data on his/her system. All that enters the system are the requests from the other agents.

The cost of moving mobile agent code from one computer to another is significant when compared to the cost of sending messages. The disadvantage of the AT-MAS system when compared to mobile agents is that currently, the system does not take account of transitory network connections - for example, connections to mobile devices which may be broken. By this, it is meant that a when a query is executed, the connection must be maintained for the duration of the execution.The effects of this can be reduced by implementing a caching system which allows the messages to be stored and sent when a connection is available. This is discussed in section 7.4.3. In addition, the ability to have the results emailed to a person instead, does partially compensate for this deficiency.

However, the AT-MAS network topology is not fixed and as a result can cope with agents joining and leaving the system by bypassing the agents which are no longer present and utilizing the ones which are.

6.2.5 Is AT-MAS a P2P System?

Although the core of the network - the agent to agent communication is P2P, the client connections to the root agent are client/server. However it is possible for any client written in the form of an application rather than as an applet, to connect to the agent and communicate as effectively as any other agent. Since this is that case, it is possible to class the AT-MAS system as a P2P system which has the additional benefit of being able to support clients in a client/server fashion.

The main criteria for excluding AT-MAS from the category of P2P is that it makes no provision for agents which do not have a valid IP address or hostname. Whilst it is possible for an agent to use an IP assigned by NAT instead of having a fixed IP, this causes problems as it means that the agents address can change. Since there is no mechanism which allows for the other agents to learn of such changes and update their knowledge bases, the old information about the agent becomes worthless, and new information must be collected. This limitation is addressed in section 7.4.2

In spite of this limitation, AT-MAS still provides an "egalitarian relationship between peers and, more importantly, suggests direct interaction between peers."[166] and by these terms it is a P2P system.

6.2.6 Is the AT-MAS system a viable alternative to P2P

The key advantage of AT-MAS compared with P2P systems is that its basic unit of operation is not the file or document, but can be as small as a single fact; a number, name, or email address, etc. This increases the range of data processing that can be carried out, and as a result, the range of possible applications has also been increased. This also reduces the network traffic as the files are analysed locally and only the data required is transferred.

However, the increased range of applications means that instead of a compact and simple protocol for searching, with single bits used to specify the action to be performed, AT-MAS uses comparatively large messages. Another consequence of this flexibility is that rather than simply forwarding queries as existing P2P systems, the agents in AT-MAS form plans before carrying out any actions.

As a result the searching within AT-MAS will never be as fast as a P2P system such as Freenet or Gnutella, which are optimised for a small range of tasks, specifically locating and downloading files. However, while AT-MAS is not intended to compete directly with these systems, it can provide the services[1] that P2P systems can, and also those of many others.

Another factor in AT-MASs favour is that users can connect to a node without having to make data available to the system. As a result, free riding is not a problem since it is only the agents with data that are contacted during a search.

6.3 Chapter Summary

This chapter evaluates the AT-MAS system. It begins with a short description of the network used to perform the testing and the method used to obtain the results. An example of the complete timing data of one the tests is given along with the frequency distribution graphs for that data. This is used to support the decision to use the average of 50 query runs to obtain the results for a single data line.

This is followed by a number of tests designed to prove that the AT-MAS system is scalable. In each of the tests the results are linear, suggesting that the system will scale consistently. Following this, a number of tests were conducted on the agents knowledge base. These tests were not as conclusive as the scalability tests.

[1]with the current exception of the file download - see section 7.4.5 for details of how this may be implemented.

The second part of this chapter discusses the AT-MAS system and compares it to the technologies such as Multi-Agent Systems (MASs) and Peer to Peer (P2P) systems which have influenced it.

Chapter 7

Conclusions

7.1 Evaluation of the work undertaken

In their paper, "Pitfalls of Agent Oriented Development"[188], Wooldridge and Jennings argue that designers should always develop using one of the many existing agent architectures already developed. They continue by stating that if a system is developed without using an existing architecture then any existing *de facto* standards should be followed.

From that viewpoint, the AT-MAS system is flawed: it was developed "from scratch" with a proprietary communications protocol. However, in terms of flexibility, this is one of its advantages.

As mentioned in chapter 1, the decision was made not to use any agent based development tools since it was felt that they would unduly influence the development of the AT-MAS system.

1. The AT-MAS is an unconventional MAS, owing much of its inspiration to the development of P2P systems. As such, it was important to make a conscious departure from the traditional MAS design. This is especially true since the AT-MAS system has been influenced both positively and negatively by existing MASs.

2. The whole purpose of the agent frameworks and development is to make the process of building systems easier. In doing so, they provide a pathway - a simple route to creating the agent - unless the final destination is different from that envisaged by the framework designers. However, by simply providing tools, modules, interfaces and off the shelf components to assist with creating the agents and the infrastructure to support them, these systems provide a tempting *easy solution* which may cause the original idea to become diluted.

3. The *"one size fits all"* nature of agent frameworks, means that simpler agent systems can be easier to develop without them. This principle extends to systems such as AT-MAS which a large amount of *non-agent* (i.e. P2P) code.

7.2 Evaluation of the AT-MAS system for different users

Due to its flexibility, the AT-MAS system is of interest to a number of different groups of users. This section gives a brief description of how the system may be of use to them.

7.2.1 Research - General

As a general research tool, AT-MAS allows users to search a wide variety of data sources. However, as with any search tool, its effectiveness is ultimately limited by the data available. Currently it is possible for searches to be carried out on XML and BibTEX (LATEX bibliography) files. As the system is developed, other structured datasources, whether static files, web services, etc. may be integrated, improving the range of data available to the agents.

One important feature of the system is that the data comes from a known source. This allows facts to be checked, copyrights to be enforced and sources to be verified through other means.

In doing so, it provides some slight regulation to the internet that P2P systems do not.

7.2.2 Research - Specific

This work provides no specific technical breakthroughs for the research community. Instead AT-MAS bridges the gap between between two distinct, but related, disciplines. As such, it is of use to researchers from both fields.

To P2P researchers, it shows that by expanding the range of data types handled, it is possible to vastly increase the range of applications possible.

To MAS researchers, the use of static agents in an open system shows that it is possible to create a MAS without the need for complex naming and brokerage services.

Furthermore, the simple protocol may allow researchers from both fields to further collaborate; spawning a network of both P2P and MAS nodes cooperating effectively. Although no research has been done on this possibility, it may be of use to the Grid Computing community which specializes in the integration of disparate online resources.

7.2.3 Commercial

Although some work needs to be done to prepare the AT-MAS system for widespread use, it has the potential to be usable in a wide range of situations such as client to business eCommerce. However, commercial systems of this type have had a mixed reception since their users are no longer required to visit retailers websites.

> . . . a third of the online CD merchants accessed by BargainFinder blocked all of its requests. One reason was that many merchants don't want to compete on price

> alone. Value-added services offered on merchants' Web sites were being bypassed by BargainFinder and therefore not likely considered in the consumer's buying decision. However, Andersen Consulting also received requests from an equal number of smaller merchants who wanted to be included in BargainFinder's price comparison. In short, companies competing on price and welcoming exposure wanted to be included; the others didn't.[110]

The only problem arising from the use of AT-MAS is the possible inclusion of adverts by the remote agents. The effects of this can be reduced through the use of a reputation system as described in the section 7.4.1 later in this chapter.

7.2.4 Leisure

Depending upon the information available, AT-MAS can potentially provide a wide variety of information based services. Its is anticipated that a wide range of data will be available such as; from genealogy information for people tracing their family histories, recipes, computer game cheats, through to profiles for online dating sites. Since XML is used, almost any data can be represented and distributed. In addition, enhancements are proposed to the system (see section 7.4.5) which will allow users to download files, and make online purchases.

7.3 Problems and Limitations of AT-MAS

7.3.1 Attack From Within

One disadvantage of AT-MAS is that it is vulnerable to attack from within. Since AT-MAS is intended to be open to all agents, this is a serious limitation. However, P2P systems also suffer from the same complaint - namely that there are no security mechanisms that prevent a corrupt or badly behaved node from causing disruption to the network.

> Locating a willing resource provider does not guarantee the user will be satisfied with its service. Selfish peers may offer resources to maintain the impression of cooperation, but not put in the necessary effort to provide the service. Worse, certain nodes may join the network, not to use other peers ' resources, but to propagate false files or information for their own benefits...Accessing invalid or falsified resources can be expensive in terms of time and money.[114, page 92]

This can be shown by the use of altered nodes to collect statistical information from the network. Although this is a benign change, it could have just as easily been a malevolent one. Either way, the common P2P protocols do not prevent or detect this behaviour. Instead, provided that a client conforms to the protocol, it is accepted as part of the network.

There are a number of different ways that a corrupt AT-MAS agent could cause disruption;

Information Overload: While the Front-End Server (see section 5.2.2) prevents the agent from becoming overloaded by too many tasks, the agent may become overloaded due to the amount of data that another agent can send to it. In theory, a malicious agent could send gigabytes of data to a different agent in the system as the reply to valid requests with possibly disastrous consequences.

One way of preventing this, is for a requesting agent to specify the maximum number of results and/or the maximum amount of data that it is prepared to receive in response to a request. When this amount has been reached, the receiving agent would close the connection in order to prevent overload. However, a malicious agent would still be able to send invalid data up to the maximum amount specified.

Although not specifically malicious, it would be possible for an agent to send additional, but irrelevant data along with the requested results. This may be as part of requested data received from commercial web sites. However, if the message is structured correctly, this information will be discarded during either the removal of duplicate results, or the final root agent processing. As a result, there is little incentive for companies to send spurious data except as part of the results.

Privacy of Requests and Data: In the current implementation, there is no way of protecting the privacy of the results as they are passed back along the tree. In fact, this is contradictory to the philosophy of the system as the agents are designed to filter and process the results as they are received.

In some specific situations, the privacy is essential. In order to allow this, a simple encryption scheme can be implemented. In this, if a request contained a <Public_Key></Public_Key> tag, the agent receiving it would know that it needed to return the results of the request within an <Encrypted></Encrypted> tag.

However, this would only be possible for direct transactions such as online purchases (see section 7.4.5) since a malicious or deceptive agent contacted could substitute its own public key in place of the one provided when contacting other agents. When it received the results, it would be able to re-encrypt them using the original agents public key and return them. This invasion of privacy would be undetectable without some form of cross-referencing amongst the agents in the network.

Guaranteed Validity of the Data Similarly, in the current system, there is no way of preventing an agent from maliciously altering information that it receives before passing it back along the tree. Again, this problem could be tackled through the use of encryption to ensure that the data could not be altered, but as before, this would only be possible for direct communications.

One solution that was proposed was to allow a proxy to be specified as part of a request.

If this were to happen, the agents that followed the proxy agent in the tree would return their results directly to it, instead of to any of the intermediate agents. Unfortunately this idea suffers from the same problem as the encryption. A malicious agent could substitute its own address instead of the proxy address - unless the proxy address is restricted to that of the root agent. However this would also be open to abuse since a malicious agent could forward the query as its own - in essence, becoming a root of a new query.

As a result of these deficiencies, AT-MAS cannot currently be recommended for use with any form of sensitive data. However, as a general data retrieval tool, for retrieving publicly available information from trusted sources, the AT-MAS is a valuable tool

7.4 Future Work

Although the core of the system has been implemented, a number of enhancements have been devised during the life-cycle of the project. However, these have not yet been implemented for a number of reasons. This has mainly been due to timescale, but also due to the fact that they are not strictly relevant to the system functionality.

As with any alteration to a system, it is important to consider the effect on the system before it is implemented. In particular with the AT-MAS system, it is important to take into account the effect on the network bandwidth. For example, tests may prove that it is impractical for agents to request copies of other agents knowledge bases on a regular basis. Similarly, the ability to download files may be to the detriment of the network.

7.4.1 Identifying Malicious Agents

As mentioned in the previous section, the main disadvantage of the AT-MAS system is its lack of security and its lack of resilience to attack from malicious agents within the system. Since AT-MAS is designed to be open, this is a serious concern.

In addition to the simple solutions proposed (download limits and encryption) it would be a valuable enhancement to the system if a reputation system[114] was implemented.

This could be implemented as part of the Knowledge Base by extending the *Index* attribute (see 5.3.1) which provides a very general indication of whether a remote agent is likely to respond to a request. If information about *how* an agent was likely to respond to a request was stored as well, potentially malicious agents could be identified, and avoided.

These techniques are already in use with other systems. In the NeuroGrid P2P system, the user is able to provide his/her own feedback.

> As the user receives potential matches from other nodes the local node monitors whether the user ignores them, or performs some feedback activity, either implicit - bookmarking the match, or explicit - clicking a ”spam” button. Depending upon the feedback the local node adjusts the relation between the query keywords and the remote node that provided the recommendation. Thus node that consistently provide results unsatisfactory to uses will not be queried in future.[93]

However, in order to provide a user feedback service, changes would need to be made to the client applet, the AT-MAS agent, and also the AT-MAS ACL.

7.4.2 Use of a GUID instead of an address

One of the limitations of AT-MAS compared to P2P systems is that it relies on agents with fixed addresses. If however, the agents were given a GUID (Globally Unique Identifier) in addition, this would allow the address of the agent to frequently change without consequence as the Knowledge Base would contain both the GUID and address of the other agents and would refer to the GUID rather than the address.

7.4.3 Caching to allow for Transient Connections

The increasing use of mobile devices which are not permanently connected to the internet currently provides a problem for the AT-MAS system. This is due to the fact that the AT-MAS network requires a permanent connection. However, with slight changes to the message structure system, it is possible to cater for these short term connections.

The first part of the agent conversation would remain the same, however, after the agent sends its ”Processing: please wait” message, the client would break the connection. As the agent received the results, it would store them. The client would then reconnect later and request the results which the agent would send. In order to ensure that the correct results were given to the correct client, the agent would require that the client address, the Task ID and the Tree ID would match the values that were assigned when the client made the original request.

It would not be possible for a client to make a new request until the existing data had been retrieved. This would prevent clients from making many requests and overloading the agent. If the client does not reconnect within a certain timescale - set in the agents initialization file - then the results would be discarded.

7.4.4 Requesting Knowledge Base Information

Although it is not required by the design of the system, one simple enhancement would be to provide the facility for a new agent to contact an existing agent and request a copy of its Knowledge Base. Currently, the agent's Knowledge Base is populated from a *servers* file which is loaded during the agents initialization process. However, since this is a static file, the values it contains will very quickly become out of date.

This improvement is similar to those made to the routing mechanism used by the latest versions of Freenet (see section 4.5.2) and is currently used in the IR system (see section 4.5.5). Further work must be carried out to determine the affect of this increased network load. If it is found that the load on the network is not detrimental, the idea can be extended. Rather than only requesting initialization information, agents will be able to request Knowledge Base data from other agents during times when they are not processing any other tasks. However, the balance between the agents being too pro-active and not proactive enough must be carefully considered along with the implications for network bandwidth.

Either in addition to or as an alternative to the solution proposed above, it would be a useful enhancement for the agent to create backups of its Knowledge Base data at regular intervals.

7.4.5 Enhancements to the AT-MAS ACL

While the AT-MAS system can provide a number of basic information retrieval services, it is limited in that it is currently unable to allow users to download files, or make online purchases. In order to remedy this, three new commands will have to be implemented

choice: This command is different from the existing AT-MAS commands since it is a client based. If the users applet supports it, the command will cause a list of options to be displayed based on the results of a previous goal. The first parameter - hidden from the user is the value returned as the result of the command when the user selects one of the options.

download: Since one of the main influences in the design of the system was P2P systems, it seems appropriate that the AT-MAS network is able to download files. It is anticipated that the command would require a two parameters; the local file or directory name, and the remote filename. It is intended that remote filename parameter would allow wildcards to be specified so that groups of files can be downloaded at the one time. This command would be a root-only command so that files can be downloaded directly to the client. However, safeguards would have to implemented to limit the number of files downloaded or the network would quickly become swamped. The following example shows how the *find*, *choice* and *download* commands can be combined to provide a simple file download system.

<GOAL ID="g1" SUBJECT="music">find 'title,price,url' for artist "Rush"</GOAL>
<GOAL ID="g2">return g1</GOAL>
<GOAL ID="g3">choice g2.url "g2.title,g2.artist,g2.price"</GOAL>
<GOAL ID="g4">download g3.result</GOAL>

purchase: The inclusion of a choice command, when combined with a purchase command will make it possible for a user to make online purchases from participating suppliers. This example shows how the AT-MAS commands, can be used to allow purchases to be made. As before, the *purchase* command uses the *find* and *choice* commands, however a new variable - *authorization* is required as well. This could either be an encrypted payment token, or an account number that would the purchaser to make a payment for the specific item.

<GOAL ID="g1" SUBJECT="music">find 'title,price,code' for artist "Rush"</GOAL>
<GOAL ID="g2">return g1</GOAL>
<GOAL ID="g3">choice g2.code "g2.title,g2.price"</GOAL>
<GOAL ID="g4">purchase g3.result authorization</GOAL>

7.4.6 Increasing the Parsing Abilities of the AT-MAS Agents

Currently the AT-MAS agents are able to extract information from simple XML files and BibTEX files. However, despite the increasing popularity of XML, there are many more structured data formats in existence and therefore it would be an important upgrade to increase the number of data formats that the AT-MAS agents can parse. At this stage of the project, no decision has been made as the which additional formats will be supported.

7.4.7 An Intelligent Interface Agent

A final long term goal for development of the system would be to provide an agent driven front end instead of the basic applet. The interface agent would be able to communicate effectively with the user and translate his/her requests into goals that can be distributed across the network. As well as this, by using an agent, rather than an applet it is possible to store information about the user. This allows the agent to adapt to the users style of working (see section 2.3.4).

In addition, the agent would be able use AT-MAS to assist the user in other ways such as;

- Ensuring the drivers required by the user's computer are kept up to date by locating and downloading drivers as required. This would be done either during the installation process, or more commonly when new hardware is installed or new driver versions become available.
- Locating documents and files of possible interest to the user. The agent would maintain a list of the users preferences and periodically search the network for information relevant to these preferences. This may include the creation of a daily newspaper based in information retrieved from news website agents.
- Scheduling appointments. To do this, the users agent would first of all locate the remote agent representing the person with whom the meeting was to be requested. It would then

request a list of times that the remote user was free from his/her agent - this information would be stored in a calendar/schedule file which could be searched by the remote agent. When the list of times were returned, a list would be presented to the local user. who would be able to choose the one or more suitable times. The agent would then contact the remote agent which would request confirmation or refusal from the remote user. If a time was agreed, both of the agents would update their calendar/schedule files as required.

7.5 Final Thoughts

In creating AT-MAS, I set out to build a system which would provide a powerful, robust and flexible network of agents which did not rely on agent middleware in order to function correctly. It would provide the remote processing capabilities of mobile agents without the security risks and control issues. The system would be open; allowing any agents to connect to/disconnect from the network at any time.

The simple communications protocol and language, influenced by both KQML and FIPA-Acl, was developed to both support the basic operation of the system, and allow for expansion of the language as the system is developed. No support was included for the use of mediators, facilitator, and brokering as it is not required.

By these criteria, the system is a success.

Bibliography

[1] Douglas Adams. *The Hitch-hikers Guide to the Galaxy.* Pan Books, 1979. ISBN 0-330-25864-8.

[2] Eytan Adar and Bernardo A. Huberman. Free riding on gnutella. Technical report, Xerox PARC, August 2000. URL http://citeseer.ist.psu.edu/adar00free.html.

[3] Simon Adcock. How does the grid extend the internet, and what is the future vision for this development?, December 2002. URL http://citeseer.ist.psu.edu/554105.html.

[4] John Alderman. *SonicBoom.* Fourth Estate, 77-85 Fulham Palace Road, London W6 8JB, 2001. ISBN 1-84115-513-6. URL http://www.the4thestate.com.

[5] Kelsey Anderson. Analysis of the traffic on the gnutella network. Technical Report CSE222 Final Project, University of California, San Diego, March 2001.

[6] Tracy L. Anderson and Max Donath. Animal behavior as a paradigm for developing robot autonomy. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 145–168. MIT/Elsevier, 1990. ISBN 0-262-63135-0.

[7] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer file sharing technologies. Technical report, Electronic Trading Research Unit (ELTRUN), Athens University for Economics and Business, 2002. URL http://www.eltrun.gr/whitepapers/p2p_2002.pdf. White paper.

[8] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems, July 2002. URL http://citeseer.ist.psu.edu/babaoglu02anthill.html.

[9] Mihai Barbuceanu and Mark S. Fox. Cool: A language for describing coordination in multiagent systems. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference oil Multi-Agent Systems (ICMAS-95)*, pages 17–24, San Francisco, CA, USA, 1995. AAAI Press. URL http://citeseer.ist.psu.edu/barbuceanu95cool.html.

[10] Don Barker. Microsoft's new animated agent technology. URL http://www.botspot.com/pcai/article1.html.

[11] J. Bates. The role of emotion in believable agents. *Communications of the ACM*, 37(7): 122–125, 1997. URL http://citeseer.ist.psu.edu/bates94role.html.

[12] J. Bates, A. B. Loyall, and W. S. Reilly. Broad agents. *Sigart Bulletin*, 2(4):38–40, 1991. URL http://citeseer.ist.psu.edu/bates91broad.html.

[13] R. J. Bayardo, Jr, W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M Rusinkiewicz, R. Shen, C. Unnikrishnan, A. Unruh, and D. Woelk. Infosleuth: Agent-based semantic integration of information in open and dynamic environments. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 205–216. Morgan Kaufmann, San Francisco, CA, USA, 1997. URL http://www.mcc.com/projects/infosleuth.

[14] Doug Bedell. Bittorrent snaps up hollywood bit by bit, August 2004. URL http://www.freep.com/money/business/moore26e_20040826.htm.

[15] Randall D. Beer, Hillel J. Chiel, and Leon S. Sterling. A biological perspective on autonomous agent design. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 169–186. MIT/Elsevier, 1990. ISBN 0-262-63135-0.

[16] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa-compliant agent framework. In *In Proceedings of Practical Application of Intelligent Agents and MultiAgents (PAAM '99)*, pages 97–108, April 1999.

[17] F. Bergenti and A. Poggi. Leap: a fipa platform for handheld and mobile devices, 2001. URL http://citeseer.ist.psu.edu/bergenfi01leap.html.

[18] Michael K. Bergman. The deep web: Surfacing hidden value, 2000. URL http://www.brightplanet.com.

[19] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web, May 2001. URL http://www.scientificamerican.com/2001/0501issue0501bernerslee.html.

[20] Viktors Berstis. Fundamentals of grid computing. Technical report, IBM, November 2002. 28 pp. URL http://www.ibm.com/redbooks.

[21] D. Bertolini, P. Busetta, M. Nori, and A. Perini. Peer-to-peer and multi-agent systems technologies for knowledge management applications. an agent-oriented analysis.

[22] Bit torrent, September 2003. URL http://bitconjurer.org/BitTorrent/index.html.

[23] R. Peter Bonasso, James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2/3):237–256, April 1997. URL http://citeseer.ist.psu.edu/article/bonasso97experiences.html.

[24] Jeffrey M. Bradshaw, Stewart Dutfiled, Pete Benoit, and John D. Woolley. Kaos: Towards an industrial-strength open agent architecture. In Jeffrey M. Bradshaw, editor, *Software Agents*, pages 375–418. AAAI Press / The MIT Press, 1997. ISBN 0-262-52234-9.

[25] F. Brazier, M. van Steen, and N. Wijngaards. On mas scalability. In T. Wagner and O. Rana, editors, *Proceedings of Second International Workshop on Infrastructure for*

Agents, MAS, and Scalable MAS, pages 121–126, Montreal, Canada, May 2001. URL http://citeseer.ist.psu.edu/article/brazier01mas.html.

[26] Rodney A. Brooks. Elephants don't play chess. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 3–15. MIT/Elsevier, 1990. ISBN 0-262-63135-0.

[27] Rodney A. Brooks. Intelligence without reason. In *Proceedings of the Twelveth International Joint Conference on Artificial Intelligence*, pages 569–595, Sam Matoo, California, 1991.

[28] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–160, 1991.

[29] Alper K. Caglayan and Colin G. Harrison. *Agent Sourcebook*. Wiley Computer Publishing, 1997. ISBN 0-471-15327-3. 349 pp.

[30] Sergio Camorlinga, Ken Barker, and John Anderson. Multiagent systems for resource allocation in peer-to-peer systems.

[31] Justine Cassell. Nudge nudge wink wink: Elements of face-to-face conversation for embodied conversational agents. In Scott Prevost Justine Cassell, Joseph Sullivan and Elizabeth Churchill, editors, *Embodied Conversational Agents*. MIT Press, 2000.

[32] C. Castelfranchi, R. Falcone, and F. de Rosis. Deceiving in golem: How to strategically pilfer help. In *Autonomous Agent '98: Working notes of the Workshop on Deception, Fraud and Trust in Agent Societies*, 1998. URL http://citeseer.nj.nec.com/article/castelfranchi98deceiving.html.

[33] Miguel Castro, Manuel Costa, and Antony Rowstron. Should we build a gnutella on a structured overlay?

[34] Miguel Castro, Peter Druschel, Y. Charlie Hu, and Anthony Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays, 2003.

[35] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them al: Service discovery and binding in structured peer-to-peer overlay networks.

[36] B. Chaib-Draa and F. Dignum. Trends in agent communication language. *Computational Intelligence*, 18(2), 2002.

[37] B. Chaib-Draa, B. Moulin, R. Mandiau, and P. Millot. Trends in distributed artificial intelligence. *Artificial Intelligence Review*, pages 35–66, 1992.

[38] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, London, UK, 1996. Practical Application Company. URL http://citeseer.ist.psu.edu/chavez96kasbah.html.

[39] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable, August 2003.

[40] David Chess, Colin Harrison, and Aaron Kershenbaum. Mobile agents: Are they a godd idea? Technical Report RC 19887, IBM Research Division, December 1994.

[41] Adam Cheyer and David Martin. The open agent architecture.

[42] Luigi Ciminiera, Andrea Sanna, and Claudio Zunino. Survey on grid and peer-to-peer network technologies. *European Grid of Solar Observations*, October 2002.

[43] Ian Clarke. Freenet's next generation routing protocol, July 2003. URL http://freenet.sourceforge.net/index.php?page=ngrouting.

[44] Ian Clarke, Scott G. Miller, Theodore W. Hong, Oskar Sandberg, and Brandon Wiley. Protecting free expression online with freenet. *IEEE Internet Computing*, January/February: 40–49, 2002. URL http://computer.org/Internet/.

[45] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system, 2000. URL http://www.freenetproject.org/index.php?page=papers.

[46] David Cliff. Computational neurothology: A provisional manifesto. In *Animals to Animates. First Int'l Conf. Simulation of Adaptive Behavior*, pages 29–38, 1991.

[47] Philip R. Cohen, Adam Cheyer, Michelle Wang, and Soon Cheol Baeg. An open agent architecture. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 197–204. Morgan Kaufmann, San Francisco, CA, USA, 1997. URL http://citeseer.ist.psu.edu/article/cohen94open.html.

[48] Philip R. Cohen and Hector J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[49] The collins paperback english dictionary, 1991.

[50] M. Dastani, F. Dignum, and J. Meyer. Autonomy and agent deliberation, 2003. URL http://citeseer.ist.psu.edu/dastani03autonomy.html.

[51] Keith Decker, Mike Williamson, and Karia Sycara. Matchmaking and brokering, December 1996. URL http://citeseer.ist.psu.edu/decker96matchmaking.html.

[52] Frank Dignum. Agent communication and cooperative information agents. In *Cooperative Information Agents*, pages 191–207, 2000. URL http://citeseer.ist.psu.edu/article/dignum00agent.html.

[53] Frank Dignum and Mark Greaves. Issues in agent communication: An introduction. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 1–16. Springer, 2000.

[54] Thomas Erickson. Designing agents as if people mattered. In Jeffrey M. Bradshaw, editor, *Software Agents*, pages 79–96. AAAI Press / The MIT Press, 1997. ISBN 0-262-52234-9.

[55] Oren Etzioni. Intelligence without brooks (a reply to brooks), 1994.

[56] The Jade FAQ. http://jade.tilab.com/community-faq.htm.

[57] I. A. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, University of Cambridge, Cambridge, UK, 1992.

[58] I. A. Ferguson. Touringmachines: Autonomous agents with attitudes. *IEEE Computer*, 25:51–55, May 1992. URL http://citeseer.ist.psu.edu/article/ferguson92touringmachines.html.

[59] Leonard N. Foner. What's an agent, anyway? a sociological case study, 1993.

[60] I. Foster, N. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other, 2004. URL http://citeseer.ist.psu.edu/article/foster04brain.html.

[61] Jerry Fowler, Marian Nodine, Brad Perry, and Bruce Bargmeyer. Agent based interoperability in infosleuth. Technical Report mcc-insl-006-99, MCC, 1999. URL http://www.mcc.com/projects/infosleuth.

[62] John Fox, Martin Beveridge, and David Glasspool. Understanding intelligent agents: analysis and synthesis.

[63] Stan Franklin and Art Graesser. Is it an agent or a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag, 1996.

[64] Erann Gat. Integrating reaction and planning in a heterogeneous asynchronous architecture for mobile robot navigation.

[65] M. P. Georgeff. Situated reasoning and rational behavior. Technical Report Technical Note 21, Australian Artificial Intelligence Institute, 1991.

[66] M. Greaves, H. Holmback, and J. Bradshaw. Cdt: A tool for agent conversation design, 1998. URL http://citeseer.ist.psu.edu/greaves98cdt.html.

[67] Mark Greaves, Heather Holmback, and Jeffrey Bradshaw. What is a conversation policy? In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 118–131. Springer, 2000.

[68] Andrew S. Grimshaw, Wm. A. Wulf, and the Legion team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.

[69] Benjamin Grosof and Yannis Labrou. An approach to using xml and a rule-based content language with an agent communication language. Technical Report RC 21491, IBM Research Division, May 1999.

[70] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages, 2005. URL http://www.cs.uiowa.edu/ asignori/web-size/size-indexable-web.pdf.

[71] N. Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties, March 2003. URL http://citeseer.ist.psu.edu/harvey03skipnet.html.

[72] Tony Hey. Why engage in e-science? *Library + Information Update*, 3(3):25–27, March 2004.

[73] Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Semantics of communicating agents based on deduction and abduction. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 63–79. Springer, 2000.

[74] Heather Holmback, Mark Greaves, and Jeffrey M. Bradshaw. Agent a, can you pass the salt? - the role of pragmatics in agent communication, October 1998. URL http://citeseer.ist.psu.edu/holmback98agent.html.

[75] Distributed.net Homepage. http://www.distributed.net.

[76] Google Homepage. http://www.google.com.

[77] Microsoft Agents Homepage. http://www.microsoft.com/msagent.

[78] The 'Fight Aids At Home' Homepage. http://www.fightaidsathome.org.

[79] The Gnutella Homepage. http://www.gnutella.com.

[80] The Java Homepage. http://www.java.sun.com/.

[81] The Napster Homepage. http://www.napster.com.

[82] The Seti@home Homepage. http://setiathome.ssl.berkeley.edu/.

[83] Michael N. Huhns and Munindar P. Singh. Chapter 1: Agents and multiagent systems: Themes, approaches, and challenges. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*. Morgan Kaufmann Publishers, Inc., 1997.

[84] Grid computing: Distributed advantage, November 2001.

[85] Official intelliseek website, 2000. URL http://www.intelliseek.com/.

[86] Will web search ever catch up to web content?, 1999. URL http://www.intelliseek.com/prod/iw_whitepaper.htm. Copyright 1999-2000, IntelliSeek Inc.

[87] Douglas Isbell and Don Savage. Mars climate orbiter failure board releases reports, 1999. URL http://solarsystem.jpl.nasa.gov/whatsnew/pr/991110A.html.

[88] Nicholas R. Jennings and Michael J. Wooldridge. Applications of intelligent agents. In Nicholas R. Jennings and Michael J. Wooldridge, editors, *Agent Technology: Foundations, Applications, and Markets*, pages 3–28. Springer-Verlag: Heidelberg, Germany, 1998. URL http://citeseer.ist.psu.edu/jennings98applications.html.

[89] Sam Joseph. Adaptive routing in distributed decentralized systems: Neurogrid, gnutella & freenet. URL http://www.neurogrid.net/publications/publications.html.

[90] Sam Joseph. An extendible open source p2p simulator.

[91] Sam Joseph. P2p metadata search layers. URL http://citeseer.ist.psu.edu/567253.html.

[92] Sam Joseph. NeuroGrid: Semantically routing queries in peer–to–peer networks. In *International Workshop on Peer-to-Peer Computing*, 2002. URL http://citeseer.ist.psu.edu/joseph02neurogrid.html.

[93] Sam Joseph and Takashige Hoshiai. Decentralized meta-data strategies: Effective peer-to-peer search, June 2003.

[94] Pedram Keyani, Brian Larson, and Muthukumar Senthil. Peer pressure: Distributed recovery from attacks in peer-to-peer systems.

[95] Franziska Klgl and Frank Puppe. The multi-agent simulation environment sesam.

[96] Dan Koeppel. Massive attack, 2004. URL http://www.popsci.com/popsci/print0,21553,390918,00.htm

[97] D. Kotz and R. S. Gray. Mobile agents and the future of the internet. *ACM Operating Systems Review*, 33(3):7–13, August 1999. URL http://www.cs.dartmouth.edu/ dfk/papers/kotz:future2.pdf.

[98] D. Kotz, R. S. Gray, and D. Rus. Future directions for mobile agent research. *IEEE Distributed Systems Online*, 2002. URL http://cmc.cs.dartmouth.edu/cmc/papers/kotz:dwta.pdf.

[99] John Kubiatowicz, David Bindel, Yan Chen, and Steven Czerwinski. Oceanstore: An architecture for global-scale persistent storage, November 2000. URL http://oceanstore.cs.berkeley.edu.

[100] Deepak Kumar and Stuart C. Shapiro. Architecture of an intelligent agent in sneps. *SIGART Bulletin*, 2(4):89–92, 1991. URL http://doi.acm.org/10.1145/122344.122362.

[101] Daniel Kuokka and Larry Harada. Matchmaking for information agents. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 91–97. Morgan Kaufmann, San Francisco, CA, USA, 1997.

[102] Yannis Labrou, Tim Finin, and Yun Peng. Agent communication languages: The current landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999. URL http://citeseer.nj.nec.com/labrou99agent.html.

[103] Michail G. Lagoudakis. Planning and intelligent sysytems: An introductory overview, 1996.

[104] Kuo-Chu Lee, William H. Mansfield, Jr., and Amit P. Sheth. A framework for controlling cooperative agents. *COMPUTER*, pages 8–15, July 1993.

[105] David Liben-Nowell, Hari Balakrishnan, and David Karger. Analysis of the evolution of peer-to-peer systems.

[106] Magnus Ljungberg and Andrew Lucas. The OASIS air-traffic management system. In *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence (PRICAI '92)*, Seoul, Korea, 1992. URL http://citeseer.ist.psu.edu/ljungberg92oasis.html.

[107] Robert Logie, Jon H. Hall, and Kevin G. Waugh. Beliefs, desires and intentions in a hybrid coached agent architecture.

[108] Pattie Maes. The agent network architecture (ana). *SIGART*, 2(4):115–120, 1991.

[109] Pattie Maes. Agents that reduce work and information overload. In Jeffrey M. Bradshaw, editor, *Software Agents*, pages 146–164. AAAI Press, 1997.

[110] Pattie Maes, Robert H. Guttman, and Alexandros G. Moukas. Agents that buy and sell. *Communications of the ACM*, 42(3):81–91, 1999. URL http://citeseer.ist.psu.edu/article/maes99agents.html.

[111] Chris Malcolm and Tim Smithers. Symbol grounding via a hybrid architecture in an autonomous assembly. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 123–144. MIT/Elsevier, 1990. ISBN 0-262-63135-0.

[112] Rob Malda. What is the 'slashdot effect?', June 2000. URL http://slashdot.org/faq/index.shtml.

[113] Evangelos P. Markatos. Tracing a large-scale peer to peer system: an hour in the life of gnutella. In *2nd IEEE/ACM Int Symp. On Cluster Computing and the Grid*, 2002.

[114] Sergio Marti and Hector Garcia-Molina. Limited reputation sharing in p2p systems, May 2003.

[115] David L. Martin, Adam J. Cheyer, and Douglas B. Moran. Building distributed software systems with the open agent architecture.

[116] Jerg Meller. Business applications for agent technology. In *6th European Agent Systems Summer School (EASSS 06)*, July 2004. URL http://www.agentlink.org.

[117] Jean-Arcady Meyer and Agnes Guillot. Simulation of adaptive behavior in animats: Review and prospect.

[118] Microsoft agent user interface, 2003. URL http://www.microsoft.com/msagent/using/userinterface.as

[119] S. A. Moore. On conversational policies and the need for exceptions. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 144–159. Springer, 2000.

[120] Antonio Moreno, Ada Valls, and Alexandre Viejo. Using jade-leap to implement agents in mobile devices, 2002. URL http://jade.tilab.com/papers/EXP/02Moreno.pdf.

[121] Gajanana Nadoli and John E. Biegel. Intelligent manufacturing-simulation tool (imsat). *ACM Transactions on Modelling and Computer Simulation*, 3:42–65, January 1993.

[122] Newswire. Web of mystery. *New Scientist*, 164(2215):17, December 1999. URL http://www.newscientist.com.

[123] Marian H. Nodine and Damith Chandrasekara. Agent communication languages for information-centric agent communities. *HICSS*, 1999. URL http://citeseer.nj.nec.com/article/nodine99agent.html.

[124] Marian H. Nodine, Brad Perry, and Amy Unruh. Experiences with the infosleuth agent architecture.

[125] H. S. Nwana and D. T. Ndumu. An introduction to agent technology. *BT Technol J.*, 14, October 1996.

[126] Hyacinth S. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 1996.

[127] Andy Oram. A free software agenda for peer-to-peer, February 2002. URL http://www.openp2p.com/lpt/a/1596.

[128] Benno J. Overeinder, Etienne Posthumus, and Frances M. T. Brazier. Integrating peer-to-peer networking and computing in the agentscape framework. In *Proceedings of the 2nd IEEE International Conference on Peer-to-Peer Computing*, pages 96–103, Linköping, Sweden, September 2002. URL http://citeseer.ist.psu.edu/overeinder02integrating.html.

[129] Mauizio Panti, Louis Penserini, Luca Spalazzi, and Simone Tacconi. A multi-agent system based on the p2p model to information integrration.

[130] M. Paolucci and K. Sycara. An exploration in mas scalability. URL http://www.cs.cmu.edu/People/softagents/papers/paolucci.pdf. Tracking Number 794.

[131] Penelope Patsuris. Lost in the translation, August 1999. URL http://www.forbes.com/tool/99/aug/0827/feat.htm.

[132] Terry R. Payne, Massimo Paolucci, Rahul Singh, and Katia Sycara. Communicating agents in open multi agent systems.

[133] Louis Penserini, Mauizio Panti, and Luca Spalazzi. Agent-based transactions into decentralised p2p (preliminary report).

[134] Lynellen D. S. Perry. Emotionware, 1996. URL http://www.acm.org/crossroads/xrds3-1/emotware.html.

[135] R. Pfeifer. Artificial intelligence models of emotion, 1988. URL http://citeseer.ist.psu.edu/pfeifer88artificial.html.

[136] Laurence R. Phillips and Hamilton E. Link. The role of conversation policy in carrying out agent conversations. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*. Springer, 2000.

[137] Jeremy Pitt and Abe Mambani. Communication protocols in multi-agent systems: A development method and reference architecture. In *Issues in Agent Communication*. Springer, 2000.

[138] Jeremy Pitt and Abe Mambani. Some legal aspects of inter-agent communication: From the sincerity condition to 'ethical' agents. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*. Springer, 2000.

[139] Martha Pollack and John F. Horty. There's more to life than making plans. *The AI Magazine*, 20(4):71–84, 1999. URL http://citeseer.ist.psu.edu/pollack99theres.html.

[140] Martha Pollack and Marc Ringuette. Introducing the tileworld: experimentally evaluating agent architectures. In Thomas Dietterich and William Swartout, editors, *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189. AAAI Press, Menlo Park, CA., 1990. URL http://citeseer.nj.nec.com/pollack90introducing.html.

[141] The Free Network Project. http://freenetproject.org/index.php?page=index.

[142] Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. Routing algorithms for dhts: Some open questions, 2002. URL http://citeseer.ist.psu.edu/article/ratnasamy02routing.html.

[143] M. Reddy and G. M. P. O'Hare. The blackboard model: a survey of its application, 1991.

[144] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network, 2001.

[145] Jordan Ritter. Why gnutella can't scale. no, really., February 2001.

[146] Jeffrey S. Rosenschein and Gilad Zlotkin. Designing conventions for automated negotiation. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 353–370. Kaufmann Publishing, Inc., San Fransisco, California, 1997.

[147] Stanley J. Rosenschein. Distributed intelligent agents. In F. H. Vogt, editor, *Personal Computers and Intelligent Systems*, pages 61–63. Elsevier Science Publishers B. V. (North-Holland), 1992.

[148] Tuomas Sandholm and Victor Lesser. Issues in automated negotiation and electronic commerce: Extendin the contract net framework. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 66–73. Morgan Kaufmann, San Francisco, CA, USA, 1997. URL http://www.mcc.com/projects/infosleuth.

[149] Lus Morais Sarmento. An emotion-based agent architecture, March 2004.

[150] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002. URL http://citeseer.ist.psu.edu/article/saroiu02measurement.html.

[151] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, 1996. URL http://citeseer.ist.psu.edu/schoonderwoerd96antbased.html.

[152] Sandip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages p353–370. Kaufmann Publishing, Inc., San Fransisco, California, 1997.

[153] Clay Shirky. What is p2p and what isn't, 2000. URL http://www.oreillynet.com/pub/a/p2p/2000/11/24/shirky1-what isp2p.html.

[154] Ben Shneiderman. Direct manipulation verses agents: Paths to predictable controllable and comprehensible interfaces. In Jeffrey M. Bradshaw, editor, *Software Agents*, pages 97–106. AAAI Press, 1997.

[155] Yoav Shoham. Agent-oriented programming. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 329–349. Kaufmann Publishing, Inc. San Fransisco, California, 1997.

[156] Andrew Silis and K. A. Hawick. The discworld peer-to-peer architecture. Technical Report DHPC-028, University of Adelaide, January 1998.

[157] V. Silva, A. Garcia, A. Brandão, C. Chavez, C. Lucena, and P. Alencar. *Taming Agents and Objects in Software Engineering*, pages 1–26. Springer-Verlag, 2003. URL http://twiki.im.ufba.br/pub/Aside/NossasPublicacoes/TAO.pdf.

[158] Munindar P. Singh. A social semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*. Springer, 2000.

[159] James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP*. O'Reilly & Associates, 1005 Gravenstein Highway North, Sebastopol, CA 95472, January 2002. ISBN 0-596-00095-2.

[160] Daniel Thalmann Soraia. Virtual humans' behaviour: Individuals, groups, and crowds. URL http://citeseer.nj.nec.com/580141.html.

[161] Luc. Steels. Exploiting analogical representations. In Pattie Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 71–88. MIT/Elsevier, 1990. ISBN 0-262-63135-0.

[162] G. Stephenson. The semantic web - will it work?, May 2001.

[163] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM 2001*, August 2001.

[164] Danny Sullivan. Invisible web gets deeper, August 2000. URL http://www.searchenginewatch.com/sereport/index.html.

[165] Todd Sundsted. An introduction to agents. *How-To Java*, June 1998. URL http://javaworls.com/jw-06-1998/jw-06-howto.html.

[166] Todd Sundsted. The practice of peer-to-peer computing: Introduction and history. *developerWorks*, 2001. URL http://www-106.ibm.com/developerworks/library/j-p2p/index.html.

[167] Katia Sycara and Joseph Giampapa. The retsina mas infrastructure, January 2001.

[168] Katia Sycara, Joseph Giampapa, Brent Langley, and Massimo Paolucci. The retsina mas, a case study, 2003.

[169] Katia P. Sycara. Multiagent systems. *AI Magazine*, pages 79–89, 1998.

[170] Austin Tate, John Levine, Peter Jarvis, and Jeff Dalton. Using ai planning technology for army small unit operations. URL http://www.aiai.ed.ac.uk/ oplan.

[171] Ian Taylor. *From P2P to Web Services and Grids*. Springer-Verlag, 2005. ISBN 1-85233-869-5.

[172] Mario Tokoro. The society of objects. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 421–429. Kaufmann Publishing, Inc., San Fransisco, California, 1997.

[173] Predrag T. Tosic and Gul A. Agha. Towards a hierarchical taxonomy of autonomous agents, 2004. URL http://citeseer.ist.psu.edu/710000.html.

[174] Stuart J. Tuck. Software agents; a general guide to agent-oriented project development, 1998. URL http://www.spiralnebula.demon.co.uk/Agent/agent.htm.

[175] Toby Tyrrell and John E. W. Mayhew. Computer simulation of an animal environment, 1991.

[176] Robert Valdes. In the mind of the enemy: The artificial intelligence of halo 2. *How Stuff Works*, November 2004. URL http://stuffo.howstuffworks.com/halo2-ai.htm/printable.

[177] Steven. A. Vere. Organization of the basic agent. *SIGART*, 2(4):164–168, 1991.

[178] Steven. A. Vere and Timothy Bickmore. A basic agent. *Computational Intelligence 6*, pages 41–60, 1990.

[179] The Infosleuth Website. http://www.mcc.com/infosleuth/.

[180] T. White and B. Pagurek. Towards multi-swarm problem solving in networks. In Y. Demazeau, editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, Paris, France, 1998. IEEE Press. URL http://citeseer.ist.psu.edu/white98towards.html.

[181] Gerhard Wiess. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* Massachusetts Institute of Technology, 2001.

[182] Nick Wijngaards, Maarten van Steen, Benno Overeinder, and Frances Brazier. Supporting internet scale multi-agent systems.

[183] Bryce Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proceedings First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA, March 2002. URL http://www.cs.rice.edu/Conferences/IPTPS02/188.pdf.

[184] Steven Willmott, Jonathan Dale, Bernard Burg, Patricia Charlton, and Paul O'Brien. Agentcities: A worldwide network open agent architecture.

[185] Steven Willmott, Matteo Somacher, Ion Constantinescu, Jonathan Dale, Stefan Poslad, David Bonnefoy, Jerome Picault, and Juan Jim Tan. The agentcities network architecture.

[186] Michael Wooldridge. Agent-based software engineering. *IEE Proceedings Software Engineering*, 144(1):26–37, 1997. URL http://citeseer.ist.psu.edu/article/wooldridge97agentbased.html.

[187] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, Ltd, Baffins Lane, Chichester, West Sussex,PO19 1UD, England, 2001. ISBN 0-471-49691-X.

[188] Michael Wooldridge and Nicholas R. Jennings. Pitfalls of agent-oriented development.

[189] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. HTTP://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h (Hypertext version of Knowledge Engineering Review paper), 1994. URL http://citeseer.ist.psu.edu/article/wooldridge95intelligent.html.

[190] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3): 285–312, 2000. URL http://citeseer.ist.psu.edu/article/wooldridge00gaia.html.

[191] I. Wright. Emotional agents. Technical report, University of Birmingham, 1997. URL http://citeseer.nj.nec.com/wright97emotional.html.

[192] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network.

[193] J. Yang, V. Honavar, L. Miller, and J. Wong. Intelligent mobile agents for information retrieval and knowledge discovery from distributed data and knowledge sources, 1999.

[194] Sule Yildrum, Turhan Tunah, and Pavel Petrovic. A hybrid task planner architecture for pick and place sequencing, 2003.

[195] Haizheng Zhang, W. Bruce Croft, Brian Levine, and Brian Lesser. A multi-agent approach for peer-to-peer based information retrieval systems, July 2004.

[196] Ben Y. Zhao, Kubiatowicz, and Antony D. Joseph. Tapastry: An infrastructure for fault-tolerant wide-area location and routing, April 2001.

[197] Y. Zou, T. Finin, L. Ding, H. Chen, and R. Pan. Taga: Trading agent competition in agentcities, 2003. URL http://sherry.ifi.unizh.ch/zou03taga.html.