Coordination of FIPA compliant software agents using utility function assignment

Steven James Lynden

A dissertation submitted in partial fulfilment of the requirements of Cardiff University for the degree of Doctor of Philosophy

School of Computer Science
Cardiff University

June, 2004

ABSTRACT

A Multiagent System (MAS) consisting of interacting autonomous agents motivated by individual objectives may be utilised to achieve global objectives in scenarios where centralised control is very difficult because of the distributed nature, complexity, or other restrictions present in a problem domain. When deploying MAS to achieve global objectives, a degree of coordination is usually required in order to ensure that the behaviour of the system is desirable with respect to these objectives. A sub-optimisation problem occurs when individual agent objectives are inconsistent with global objectives. Current approaches towards this problem within the context of learning based agents include configuring the utility functions possessed by individual agents so that the maximisation of individual utility functions results in a desired global behaviour. Interoperability is of critical importance in Internet-scale MAS, and a promising standard for this is currently evolving in the form of the "Foundation for Intelligent Physical Agents" (FIPA) specifications. This thesis focuses on the integration of techniques based on the assignment of utility functions in order to coordinate MAS within the domain of FIPA compliant agent systems. The notion of utility is extended to form two separate types: performance and functional utilities. Whereas functional utility is based on abstract application specific objectives, performance utility concentrates on performance engineering related issues. The benefit of this approach is demonstrated by a software toolkit supporting the development of learning based FIPA compliant MAS, which is applied within two domains, an application based on market based buyer agents in artificial markets, and a computational Grid based application.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Agents, multiagent systems, and coordination

The study of Multiagent Systems (MAS) is a sub-field of Artificial Intelligence (AI) concerned with interacting autonomous processes known as agents. Although there is no universally accepted definition of an agent [29], the following is consistent with the majority of definitions:

Agent: "An agent is an encapsulated computer system that is situated in some environment and is capable of flexible, autonomous action in that environment in order to meet its design objectives" [76]

The frame of reference stated by Jennings [39], is that agents can be viewed as a logical extension of object-oriented [60] programming techniques and may therefore be considered as the next logical step in the evolution of software development methodologies.

MAS allow problems to be decomposed into sub-problems, each of which can be approached by an agent with local goals and an autonomous behaviour. Where MAS are used to solve problems that require collective effort between agents, a means of coordination is required in order to maximise efficiency and prevent agents from working at cross-purposes. Various coordination techniques exist, as discussed further in chapter 2, however this thesis focuses on the roles of learning and adaptation in multiagent coordination. The motivation for this is that MAS based on learning, adaptive agents have the potential for emergent behaviours, which are of great interest within many application domains [36].

Agents using machine learning techniques are able to adapt their behaviour in order to learn to optimise rewards, or payoffs, which give an indication of each agent's success with respect to their associated behaviour. The term *utility* is often used to refer to this concept, where an agent's utility may depend on its problem solving ability, success in terms of the decisions it has made, or other criteria by which agent behaviour may be quantified. Where a global utility function exists, which quantifies the behaviour of an entire MAS, the coordination problem is one of aligning local utility functions with global utility functions, and engineering a scenario in which the maximisation of local utility by agents results in global utility being maximised. This means that local utility functions are configured so that positive contributions to local utility by agents are also positive contributions to global utility, in effect constraining agent behaviours so that they contribute to the global objectives of the MAS, which are encapsulated by the global utility function. Additionally, a further objective is to provide local utility functions with high signal-to-noise ratios. A utility function with a high signal-to-noise ratio is one which enables an agent to clearly determine the effect that an action performed by the agent has upon its local utility, limiting the noise that may occur due to the actions of other agents in the system and their impact upon the utility function. Utility functions with high signal-to-noise ratios enable agents to learn effectively and are very important when the number of agents in a system is large.

Utility functions with high signal-to-noise ratios can be developed using the Collective Intelligence (COIN) [66, 67, 71, 72, 73, 74, 75] framework, where local utility functions are derived from global utility functions in order to coordinate agent behaviour. Significantly, COIN local utility functions are aligned with global utility, meaning that an agent can determine the effect of its behaviour with respect to the objectives of the system via its local utility function. In order to achieve this, local utility functions may be parameterised by information that may not necessarily be available to an agent locally. This non-local information may consist of the partial states of other agents, or the environment in which they exist, and may be a communicated via a globally broadcast signal which allows local utility functions to be computed, or via personalised communication channels aimed at individual agents. The COIN framework does not specify how this communication should be implemented.

1.2 Novel contributions of this thesis

This thesis is motivated by the desire to develop a framework for coordinating machine learning based software agents. Additionally, agent interoperability is considered to be of critical importance, and for this reason we concentrate on agent systems conforming to the agent standards specified by the Foundation for Intelligent Physical Agents (FIPA) [1]. This is achieved by LEAF, the "Learning Agent based FIPA Compliant Community Toolkit". LEAF consists of both a framework, and a software toolkit which implements this framework, for developing learning based MAS utilising utility function assignment as a means of coordination. The LEAF framework is novel in the sense that it integrates utility function assignment (based on COIN techniques) with FIPA compliant software agents (agents that adhere to the FIPA specifications). An additional novel contribution of LEAF is that two classes of utility are defined: *functional utility* and *performance utility*. Functional utility is based on the notion of utility used in COIN [72, 66, 74], and other work related to multiagent learning [64, 55, 37], where utility is generally defined as a function parameterised by a set of variables considered by an agent, which may include variables manipulated locally by an agent, and variables dependent on other agents in a MAS. A utility function in this context equates to a payoff, or reward function, based on application specific criteria typically related to an agent's problem solving ability, game theoretic payoff table, or the mean-squared-error on a local data set. In this context, utility is used by the agent to evaluate and update its strategy, with the objective of the agent being to maximise utility.

Traditionally, utility functions have been restricted to what is defined above as functional utility. We concentrate on learning based MAS within FIPA compliant agent systems, and the implementation of utility function assignment within software agents that are independent of the platforms on which they are deployed. Restricting the definition of utility to functional utility, which may be defined at a high level of abstraction, potentially results in low-level data relating to an agent's implementation dependent behaviour being discarded. This data may be of significance in analysing the behaviours of both individual agents and a MAS as a whole. LEAF therefore defines an additional class of utility, performance utility, which is highly dependent on the speed of execution and system resource usage of an agent, which relate to the implementation architecture of the agent. When considering software agents, the performance utility of an agent, or MAS, may impact the functional utility of agents/MAS, and provide an improved means of analysing MAS behaviour, specifically with respect to:

- *Implementation decisions:* the MAS developer may be able to evaluate design/implementation choices offline, and make improvements before deploying agents.
- *Learning processes:* Performance utility may be of use in online agent learning, in which automated optimisation processes may be utilised to attempt to improve performance utility. Optimising functional utility may also be costly to an agent's performance utility, and vice versa, therefore necessitating a trade-off: optimisation of functional utility versus optimising performance utility in order to maximise an agent's positive effect on a community of agents.

It is proposed that the use of performance utility, in conjunction with the more traditional notion of functional utility, can be of use both in the development of MAS, and in particular, we hypothesise that many multiagent application domains may possess an important relationship between performance and functional utility which can be exploited to gain a better understanding of MAS performance.

1.2.1 Hypothesis

The hypothesis therefore consists of the following two parts:

1. We propose that a general framework for utility function assignment as a coordination technique can be integrated with FIPA compliant software agents.

2. We propose that when using software agents, it should be advantageous to define utility as two separate types, performance utility and functional utility. Specifically, we hypothesise that analysing the evolution of performance and functional utility functions in a MAS can lead to the discovery of specific properties of the system's behaviour that can be utilised by the MAS designer in order to improve the system's performance.

In order to attempt to demonstrate that this hypothesis holds true, this thesis presents the following:

1. The LEAF framework, which integrates utility function assignment as a means of coordination with FIPA compliant agent systems.

2. The LEAF toolkit, which allows developers to rapidly construct MAS conforming to the LEAF framework. The LEAF toolkit extends the FIPA-Open Source (FIPA-OS) agent construction toolkit [52] to achieve this.

3. The application of LEAF within two application domains motivated by the objective of evaluating the developed framework. The first application involves the coordination of a community of buyer agents, which exist in separate markets. The buyer agents are coordinated to make purchases that benefit the community as a whole, and the effects of utility function assignment within this context are analysed. The second application involves an application based on a computational Grid [28], and in this context the potential benefits of our approach when defining utility as two separate types are explored.

1.3 Structure of this thesis

The structure of this dissertation is as follows (figure 1.1 illustrates this structure visually):

- **Chapter 2:** Chapter 2 surveys the background areas of research related to the contents of this thesis. MAS coordination approaches, and in particular the concepts behind COIN, are introduced. Standards for agent interoperability are described and evaluated. Implementation approaches, including various MAS construction toolkits, are also reviewed.

- **Chapter 3:** Chapter 3 describes the LEAF framework, and how it integrates with FIPA compliant agent systems.

- **Chapter 4:** In chapter 4, the LEAF toolkit is described. In particular, an overview of the application programmer interface (API) provided by the toolkit is given.

- **Chapter 5:** Chapter 5 provides evidence to support the first part of the hypothesis, by applying LEAF to a coordination problem involving a community of market based buyer agents. Results from this investigation are presented and discussed.

- **Chapter 6:** An application based on a computational Grid [53] is presented, which provides empirical results supporting the definition of performance and functional utility in LEAF.

- **Chapter 7:** Chapter 7 investigates scalability issues with respect to the application introduced in chapter 6.

- **Chapter 8:** The lessons learnt from the development of LEAF, and its application to the problems described in chapters 5,6 and 7 are discussed. The conclusions reached in relation to the hypothesis declared are presented.

- **Chapter 9:** A discussion of the potential avenues of further work in this area that have emerged as a result of our investigations.

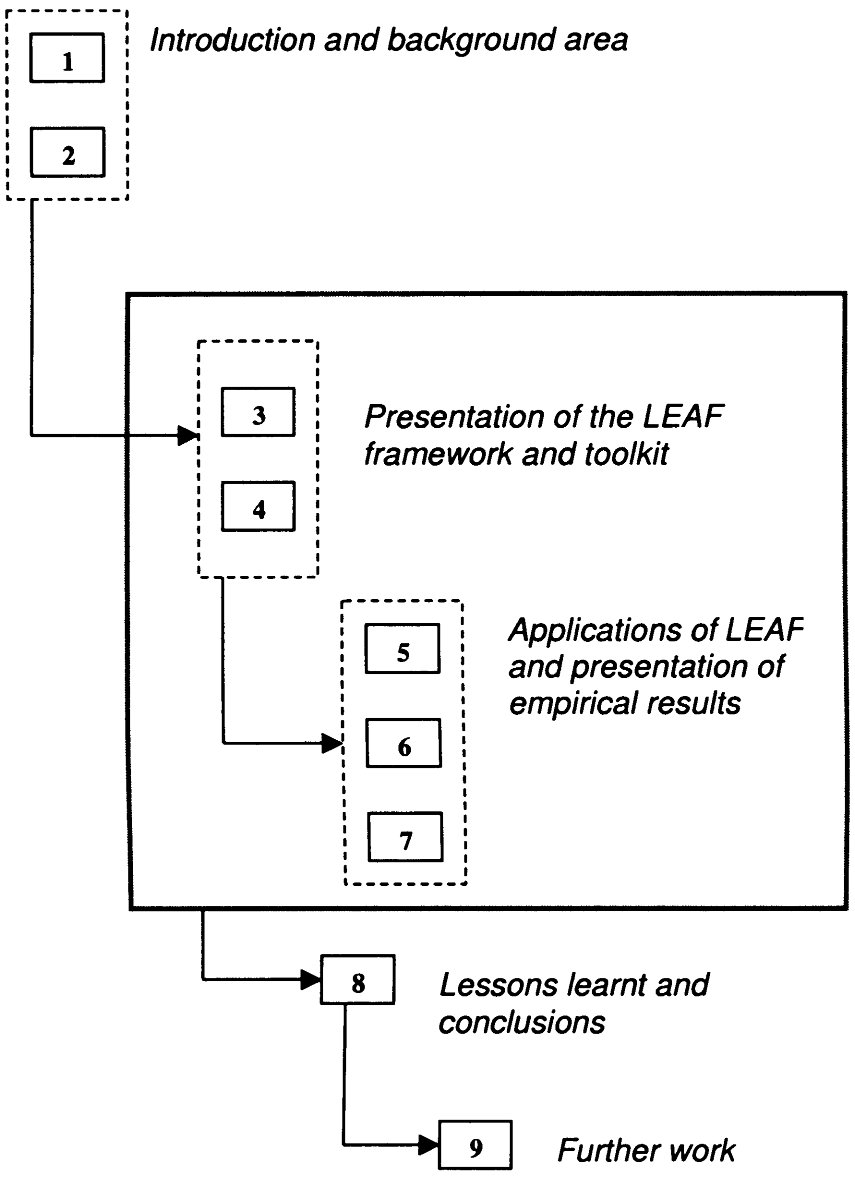

1
2
Introduction and background area
3
4
Presentation of the LEAF framework and toolkit
5
6
7
Applications of LEAF and presentation of empirical results
8
Lessons learnt and conclusions
9
Further work

Figure 1.1: The structure of this thesis

As illustrated, the main contributions of the thesis are presented in chapters (3-7). Chapters 3 and 4 introduce LEAF, and chapters 5,6 and 7 present results from the application domains. Chapters 8 and 9 conclude by presenting the lessons learnt from this work, and the potential for further work. The arrows illustrate the conceptual order in which the thesis is presented.

CHAPTER 2

Multiagent system development techniques

2.1 Introduction

In this chapter, the background areas related to the work presented in this thesis are discussed. Firstly, multiagent coordination techniques are introduced and the COIN framework is described. Following this, techniques and standards for agent interoperability are reviewed. Finally, technologies which provide support for the development of MAS are discussed, with emphasis on FIPA compliant and learning based approaches.

2.2 Multiagent coordination

MAS allow autonomous components to interact with one another to achieve their objectives. Generally, a MAS can be categorised by one of the two following scenarios:

1. A MAS exists due to the presence in a common environment of a number of self-interested agents. These agents are working individually, and there is no global objective of the MAS. This is the case in agent based electronic commerce systems, where agents have been deployed by multiple individuals/organisations. In such scenarios, agents compete on behalf of their owners

to achieve certain objectives.

2. A MAS has been deployed in order to achieve a defined set of objectives. Usually this will involve the deployment of agents in the system by a single individual/organisation, or agents may be deployed by collaborating individuals/organisations. The reason for the deployment of the MAS in this scenario is to divide global objectives into specific roles to be undertaken by different agents, as this is in some way necessary or beneficial. Reasons for the division of objectives among agents include:

    - A problem is inherently distributed, and a MAS is the natural method for approaching the problem.
    - A complex problem exists, embedded in a dynamic environment, and no centralised method or global algorithm exists to solve the problem. The problem must be decomposed into less complex units that can be handled by agents, where each agent may undertake a different role.
    - Computational resources are distributed, and to take advantage of this, agents are assigned objectives that they can achieve locally, which are subsequently combined to achieve global objectives.
    - The control of agents centrally is difficult or impossible due to scale or communication restrictions.

In the first scenario, coordination is not required by the system. The agents are inherently competitive, and deployed with local objectives only. This thesis focuses on cooperative systems, categorised by the second scenario, an important aspect of which is coordination, required in cooperative MAS for the following reasons:

- In most large MAS, no single agent will have knowledge of how to achieve global objectives. A coordination scheme is therefore required to enable agents to behave in ways that contribute toward global objectives.
- An individual agent may be unable to accomplish a task locally, and may be required to collaborate with other agents.

- Un-coordinated agents can work at cross purposes in a multiagent environment.

Coordinating MAS often proves to be a difficult task, especially when large numbers of agents are involved, and dependencies exist between the tasks performed by the agents – referred to as interdependence by Jennings [38]:

Interdependence "Interdependence occurs when goals undertaken by individual agents are related - either because local decisions made by one agent have an impact on the decisions of community members or because of the possibility of harmful interactions amongst agents."

The vast majority of MAS deployed to achieve non-trivial objectives will involve interdependent relationships between agents, where inefficient coordination can lead to situations in which agents work at cross purposes or attempt to perform tasks inefficiently. Avoiding these pitfalls and maximising the efficiency of agent behaviours is the objective of coordination. Most notably, work in distributed AI (DAI) [11], computational economics [68] and game theory [13] is strongly related to research in multiagent coordination. Approaches to multiagent coordination include:

Organisational structuring: Organisational structuring involves a process in which the objectives that each agent should aim to achieve within a MAS are identified, and the subsequent achievement of these objectives results in the achievement of system-wide objectives.

Tuple spaces: Tuple spaces [48], similar to the AI notion of blackboards [46], allow agents unaware of one another's existence to interact by publishing data in a shared tuple space. Agents use tuple spaces to collaborate between tasks, and perform computation concurrently.

Heuristic methods and intelligent optimisation techniques: Heuristics exist for optimising distributed processes towards global objectives, including simulated annealing [50] and genetic algorithms [35], however these techniques are usually centralised and can violate the requirement of autonomy with respect to MAS.

Norms and social laws: Norms and social laws [25], inspired by the observation of human social interaction, can be used to promote coordinated behaviour, for example, simple social laws/norms, such as "drive on the left side of the road" can reduce traffic collisions. Social laws can be defined prior to system deployment (static), or can emerge at runtime (dynamic), as discussed in [58].

Contracts, commitments and conventions: Commitments and conventions [38] allow agents to commit to performing certain tasks, and use conventions to reason about breaking or re-evaluating commitments. Contracting is used to encourage agents to participate in collaborative engagements by designing competitive processes for securing contracts, the most famous example of which is the contract net [59] task allocation protocol.

Distributed planning: Decentralised distributed planning [23] involves the decentralised construction and execution of coordinated plans in multiagent domains.

Mechanism design, game theory, and market oriented programming: Mechanism design is the process of designing self-interested agent interactions in a way that a desired collective global behaviour emerges. Generally, mechanism design approaches to coordination utilise economic models [68] to govern agent interactions.

Swarm intelligence: Some biological systems, despite the low intelligence of individuals, are able to perform apparently complicated tasks collectively. Swarm intelligence [15], inspired by such systems, involves the development of MAS where behaviour is comprised of very simple agents, usually large in number, that can perform tasks of varying degrees of complexity.

Agent systems vary to a large extent in the nature of the tasks they are designed to perform, the complexity of the environments in which agents exist, the ability of agents to perceive their environments, the degree of intelligence possessed by the agents, the number of agents involved, and the amount of communication present between agents. For these reasons, no one coordination technique can be labelled as being superior to others, as the choice of a coordination scheme depends on the application domain. In many of the previously discussed coordination approaches, a large amount of pre-execution coordination is required in order to induce coordinated behaviour. This is true in distributed planning, where organisational structuring and plan libraries must be developed offline, as well as with mechanism design and swarm intelligence, which involve a large degree of pre-execution design, where protocols for behaviour are defined. Designing commitments, conventions, social laws, negotiation/auction processes, and so forth may all require excessive pre-execution design, and the use of market based coordination techniques require the coordination problem to be cast as a resource allocation problem, which may not always be easy.

2.2.1 Learning and coordination

Designing agents with the ability to learn, and therefore adapt their behaviour to maximise utility, brings with it the promise of MAS that can learn how to coordinate, and reduce the amount of pre-execution coordination and hand-tailoring required in designing coordination strategies. Learning agents exploit machine learning techniques in order to learn a behaviour, or strategy, for the environment in which they are situated. A variety of techniques can be implemented to endow agents with the ability to learn [9], the majority of which stem from previous work in artificial intelligence and machine learning research. Multiagent learning techniques related to coordination generally concentrate on discovery based learning, where new knowledge is generated at runtime, as this provides greater adaptability and less static/design/pre-execution hand-tailoring than other learning techniques [57]. Currently, the vast majority of research into coordinated learning in MAS involves reinforcement learning (RL) [61, 40], a machine learning technique which endows an agent with the ability to learn, by trial and error, how to behave in an environment in order to maximise reward signals given to the agent as feedback for its behaviour. Reward signals are generated using payoff functions, which determine the reward that an agent receives following the execution of an action by the agent.

2.2.2 Collective Intelligence

One approach to coordinating a learning based MAS is to use team-game payoff functions, where agents share a common payoff function [71]. Team-game payoff functions are utilised in scenarios where it is desirable for a team of agents to work towards a common goal, and all agents are rewarded equally based on the success with which this goal is achieved, regardless of the degree with which agents contribute to the goal. While team-game payoff functions reflect accurately the performance of a team of agents with respect to a system's objectives, the agents may suffer from a problematic signal-to-noise ratio in terms of associating their actions with subsequent payoffs. This is due to the fact that each agent's payoff can be affected by the actions of any agent in the team, and each agent is attempting to optimise its local behaviour from the feedback received for a team's aggregate behaviour only. Despite the signal-to-noise problem inherent in using team-game payoff functions, success has been achieved using this approach in MAS where agents are small in number. In [22], Crites and Barto apply team-game learning techniques with success to multiagent RL, however the work is limited to MAS consisting of four agents. The lack of scalability of team-game approaches is highlighted in [72], where team-game payoff functions are compared to more learnable payoff functions developed using the COIN framework.

The drawback of team-game reward functions is apparent when a large number of agents exist in a single team, and the signal-to-noise ratio of an agent's actions and their effect on the reward shared by the team results in agents being unable to learn effectively. Collective Intelligence (COIN), is a framework that addresses this problem.

COIN is a framework for coordinating large-scale learning based systems, where centralised control of a system is extremely difficult or impossible due to scale. A COIN is defined as a "collection of interacting goal driven processes" [73], essentially a MAS. The assumption is made that with a COIN, a world utility function exists, which rates the global state of the system, and that each agent works to maximise a local payoff (or utility) function using a local learning technique. The COIN framework places no restrictions on whether independent or social learning techniques are used, however some trial and error based learning technique, the most suitable of which is RL, needs to be implemented by agents. The problem addressed in the COIN framework, stated in [73], is:

"How should one initialise/update the payoff functions of the individual processes (agents) so that the ensuing behaviour of the entire collective achieves large values of the provided world utility?"

The assumption is made that the modelling of large distributed systems is impossible, and therefore the COIN framework is an attempt to avoid such modelling. The COIN framework is based on the premise that individual learning agents are able to maximise their payoff functions, and the approach taken is one of configuring agent payoff functions to that they will maximise world utility. The world utility function encapsulates the overall objectives that the system was designed to achieve, and therefore maximising world utility should achieve these objectives. Individual payoff functions are aligned with world utility (world utility is in effect a team-game payoff function), so that the maximisation of local payoff functions results in beneficial contributions to world utility.

In [72], the "Wonderful Life Utility" (WLU) utility function is introduced, which has the important characteristics of being aligned with world utility and learnable by agents due to its high signal-to-noise ratio. The COIN framework defines a vector space Z, the elements of which, ζ, define the joint states/actions of all agents in a MAS. The desired outcome is to find a configuration of ζ that maximises world utility, $G(\zeta)$. Using WLU, an agent, η, is assigned a payoff function, WLU_η, which is of the form:

$$WLU_\eta = G(\zeta) - G(\zeta_{\hat{\eta}}, CL_\eta) \tag{2.1}$$

where the WLU function is affected by a clamping parameter; $G(\zeta_{\hat{\eta}}, CL_\eta)$ is the world utility, computed with the elements of ζ perturbed by η made equal to the clamping parameter, as illustrated in figure 2.1. The COIN framework defines WLU

$$\begin{array}{cc} & \zeta \\ \begin{array}{c}\eta_1\\ \eta_2\\ \eta_3\end{array} & \begin{bmatrix} 1\,0\,0\,0\,2 \\ 0\,1\,1\,0\,0 \\ 1\,0\,3\,4\,5 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} (\zeta_{\wedge\eta_2}, \vec{0}) \\ \begin{bmatrix} 1\,0\,0\,0\,2 \\ 0\,0\,0\,0\,0 \\ 1\,0\,3\,4\,5 \end{bmatrix} \end{array}$$

Figure 2.1: Clamping parameters in WLU utility functions.

The figure shows an example clamping operation performed on a MAS consisting of three agents, η_1, η_2, η_3. The joint state space of the MAS is represented by the matrix labelled ζ. Each agent is associated with a vector representing its state, for example agent η_2 is represented by the state (01100). World utility is a function of the combined states of the agents. Using a clamping parameter $\vec{0}$ for agent η_2, the matrix labelled $(\zeta_{\hat{\eta}_2}, \vec{0})$ shows the effects of clamping the state of this agent to a "null" vector, having the effect of removing it from the MAS. Regardless of whether $(\zeta_{\hat{\eta}_2}, \vec{0})$ is actually a physically possible configuration of the MAS or not, this new joint state space can be evaluated to obtain a new world utility, which can be used to compute a WLU value for agent η_2. This figure is adapted from a similar example given in [67].

as an adaptation of difference utilities, which are of the form:

$$U(\zeta) = G(\zeta) - \Gamma(f(\zeta)) \tag{2.2}$$

where $\Gamma(f(\zeta))$ is independent of the agent/agents for which utility is being computed, and for WLU, $f(\zeta) = G(\zeta)$. The theme central to the COIN framework is one of assigning difference utilities to influence the behaviour of learning agents in such a way that world utility is maximised. Agents are selfish with respect to the fact that they are learning to optimise their local utility, and the desired outcome is a Nash equilibrium [13]. The aim of assigning utility functions is one of 'lining up' global optima with Nash equilibria. Essentially, when the clamping parameter is equal to $\vec{0}$, WLU defines a payoff function that rewards an agent based on the agent's contribution to world utility. This is achieved by 'subtracting' an agent's presence from the collective, and computing world utility as if the agent never existed. This derived world utility is then subtracted from the actual world utility to obtain the agent's WLU value.

In [72], WLU is shown to significantly out-perform team-game utility functions with respect to the Bar problem [10]. Additionally, [66] demonstrates the ability of WLU utility functions to avoid Braess's Paradox [32] in a network routing scenario, and [67] demonstrates the benefits of WLU within a classical multiagent "Grid World" [61] problem domain. In particular the results presented by the COIN researchers show that COIN is a scalable coordination technique (large numbers of agents are involved in many of their experiments), and that unsophisticated individual learning methods (simple RL algorithms) can be used by agents to local utilities.

2.3 Agent interoperability

There is currently an increasing amount of interest in techniques based on multiagent technology within both commercial and research communities which are producing numerous agent based applications. Multiagent systems are being developed for a wide range of commercial information systems including banking [20], investment portfolio management [62] and auctioning systems [42]. Agents may play an important role in the emerging Grid infrastructure and Peer to Peer (P2P) [45] systems. As a result of the emergence of agent technology in varied application areas there are different architectures and communication protocols used to implement agents, and this can cause problems when agents developed by different organisations need to interact with each other. There is therefore a need to standardise agent technologies in order to allow agents to interoperate on a large scale in the way that standards such as TCP/IP [34] allow computers to interact via the Internet.

Agent standards exist in order to enforce consistency between developers and allow agents developed by different individuals or organisations to interact successfully. An important aspect of agent interoperability is the exchange of information, and successful interoperability occurs when agents can reliably communicate using a common standard. The utilisation of an agent communication language (ACL) is necessary to enable successful information exchange between agents. An ACL specifies a protocol for interactions between agents at a level of abstraction that is independent of the content of the information exchanged between two agents. Generally, the messages that comprise an ACL are based on speech-acts [56], which formalise the illocutionary and perlocutionary effects of human communication. Specific ACLs are discussed and compared later in this chapter. In addition to supporting communication between agents, agent interoperability may also be promoted by the standardisation of other areas of agent technology, including:

- Security. The provision of standard security mechanisms for agent environments and agent communications.

- Agent management. This includes the provision of various support services, such as directory or lookup services which enable agents to discover one another, and naming services used to assign unique identifiers to agents.

- Agent-software integration. Standards may be defined which specify how various software components and legacy systems can be exposed as agents.

- Ontologies. An ontology consists of a conceptual classification scheme which allows agents to unambiguously interpret the semantics of specific concepts in a given domain. An ontology is essential in order for agents to communicate effectively. Standards can ensure that ontologies are shared by agents developed by different organisations.

- Mobility. Mobile agents are capable of altering their physical location in order to perform various functions. Standards can facilitate the migration of agents between different hardware and software platforms.

We will now examine the three significant standards supporting agent interoperability in multiagent systems, FIPA [47], Knowledge Query and Manipulation Language (KQML) [17, 24, 41, 21] and the Object Management Group's (OMG) Mobile Agent Systems Interoperability Facility (MASIF) [44].

2.3.1 OMG MASIF

MASIF is a standard aimed at facilitating the development of mobile agents by defining a set of interfaces which implemented agents must adhere to. The MASIF standard is based on the following concepts.

- Place: The physical location in which an agent is executed.
- Agent system: A platform which is able to create and manage agents. An agency may be associated with a number of places.
- Region: A set of agent systems belonging to the same organisation.

MASIF defines two interfaces to which a developer must adhere in order to construct MASIF compliant systems. The *MAFAgentSystem* interface supports agent management and the transfer of agents between places. The *MAFFinder* interface is associated with each region, and defines agent naming and lookup specifications. Although MASIF provides extensive support for agent mobility, there is no support provided for agent communication.

2.3.2 KQML

KQML is an ACL developed by the Knowledge Sharing Effort [24] and implemented by a number of research groups. KQML supports interaction between agents using a model of communication which is based on sequences of message performatives, where each performative is based on a specific speech-act. KQML assumes that each agent has a mental state, which encapsulates an agents beliefs about its environment. The performatives provided by KQML allow agents to communicate and directly manipulate the mental states of other agents. Each KQML message performative reveals information about the message at a high level of abstraction which is independent of the actual content and content language of the message. KQML does not specify the physical transport mechanisms by which messages should be delivered, but does address some agent management related requirements by introducing a facilitator agent class. Facilitators play a special role by providing the following services to KQML compliant agents:

- A registry of agents and the services offered by these agents.
- Matchmaking, which facilitates service discovery by connecting agents requesting specific services to agents that can adequately supply the requested services.
- Advanced communication services, including message forwarding and broadcasting.

Due to the fact that different organisations were involved in the initial development of early KQML implementations, a number of a different KQML versions were developed [17, 41, 24]. As a result of this, KQML has failed to evolve into a universally accepted standard, and the different KQML versions share inconsistencies. Different KQML implementations may utilise different message transport mechanisms in order to deliver messages and are therefore incompatible with each other. The use of KQML can therefore be problematic when agents developed using different variants of KQML need to interact with one another [21].

2.3.3 FIPA

In contrast to the different KQML variants, FIPA specifications provide a single set of standards promoted by one organisation. As is the case with KQML, the FIPA specifications concentrate on supporting information exchange between agents by specifying an ACL, however FIPA also publishes specifications supporting agent message transport, agent management, agent-software integration and other areas [47]. The FIPA specifications consist of normative specifications, which FIPA compliant agent systems must adhere to, and informative specifications, which are intended as a guide for developers. Normative specifications include agent management, agent communication, and agent software integration specifications (which define how non-agent services may be wrapped as agents). More information on agent software integration and informative specifications can be found by referring to the various specification documents published by FIPA [1]. Among the FIPA specifications, the agent communication and agent management specifications are of the most significance with respect to promoting agent interoperability.

Agent communication: FIPA ACL consists of conversations, each of which is a sequence of messages following a specific protocol. Each message has an ontology, content language, performative (also referred to as a communicative act), content, and various information required for message delivery such as the sender and receiver agents. Message content is encoded using the content language, and messages are independent of the language, allowing any content language to be used regardless of the message performative. A message ontology attribute indicates which ontology should be used to interpret the message content. The structure of a FIPA ACL message is illustrated in figure 2.2. In comparison with KQML performatives, FIPA ACL performatives do not allow the direct manipulation of the mental state of an agent, but otherwise KQML and FIPA ACL performatives are in effect very similar and are both based on speech-acts. Conversation protocols govern the messages that can be sent in a conversation between agents, and the sequence in which they must be sent, as illustrated by the FIPA-Request protocol in figure 2.3. FIPA defined communication protocols exist range from simple agent interactions such

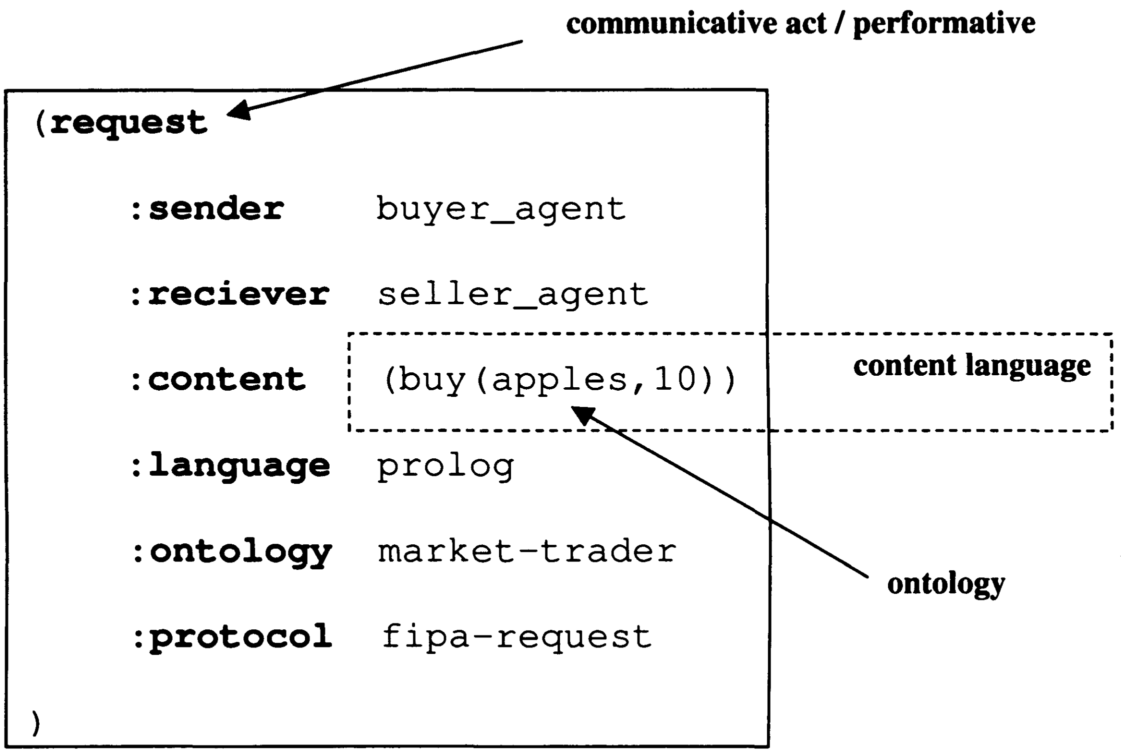


Figure 2.2: The structure of a FIPA ACL message.

The figure, adapted from [47], illustrates the structure of a FIPA ACL message with an example in which the `buyer_agent` *sends a request message to the* `seller_agent`. *Prolog is used as a content language and the conversation follows the FIPA-Request protocol.*

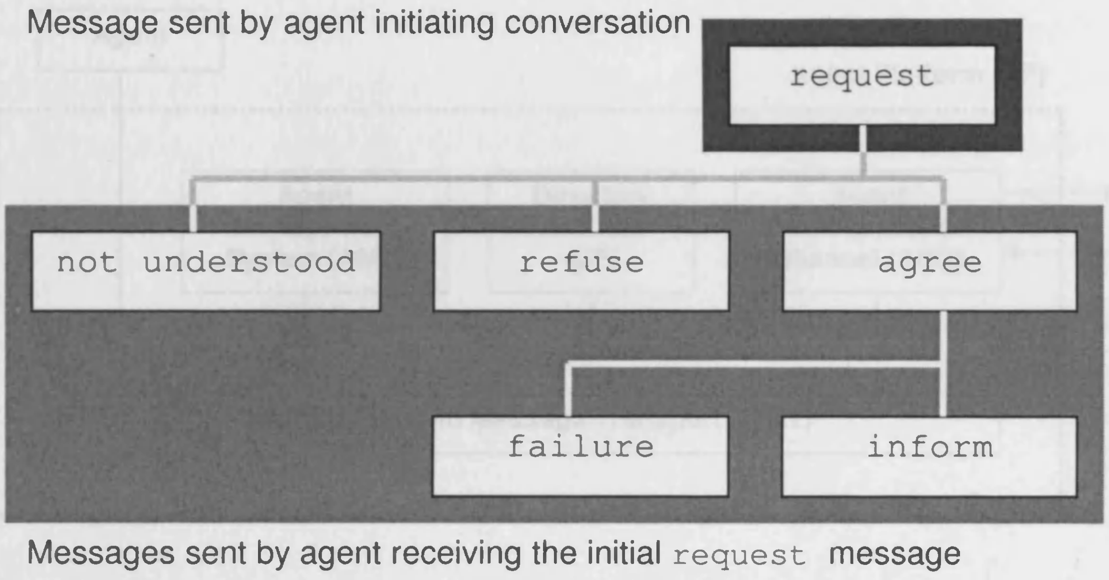

Message sent by agent initiating conversation
request
not understood
refuse
agree
failure
inform
Messages sent by agent receiving the initial request message

Figure 2.3: The FIPA-Request protocol.

The figure, adapted from [47], illustrates the sequence of messages which must take place in a conversation conforming to the FIPA-Request protocol. For example, if two agents, a and b, engage in a conversation following the FIPA-request protocol, an example of a valid conversation would be (1) agent a sends an request performative, (2) agent b replies with an agree performative, and (3) agent b follows this with an inform performative.

as requesting and querying to more complex protocols for brokering and auctions. Additionally, new communication protocols may be defined by developers.

Agent management: Agent management consists of providing a message transport service (MTS), security, agent naming services used to uniquely identify agents, and yellow pages services for discovering agents. These services are provided by the agent platform, which consists of three agents: a Directory Facilitator (DF) agent, an Agent Management System (AMS) agent, and an Agent Communication Channel (ACC) agent. An internal platform message transport (IPMT) protocol is used by agents to interact with the agent platform and with other agents managed by the same platform, as illustrated in figure 2.4. The AMS provides naming services for agents, and controls access to services. The DF allows an agent to register certain properties, including the services it offers (using a number of service descriptions), the ontologies and interaction protocols that can be used in messages sent to the

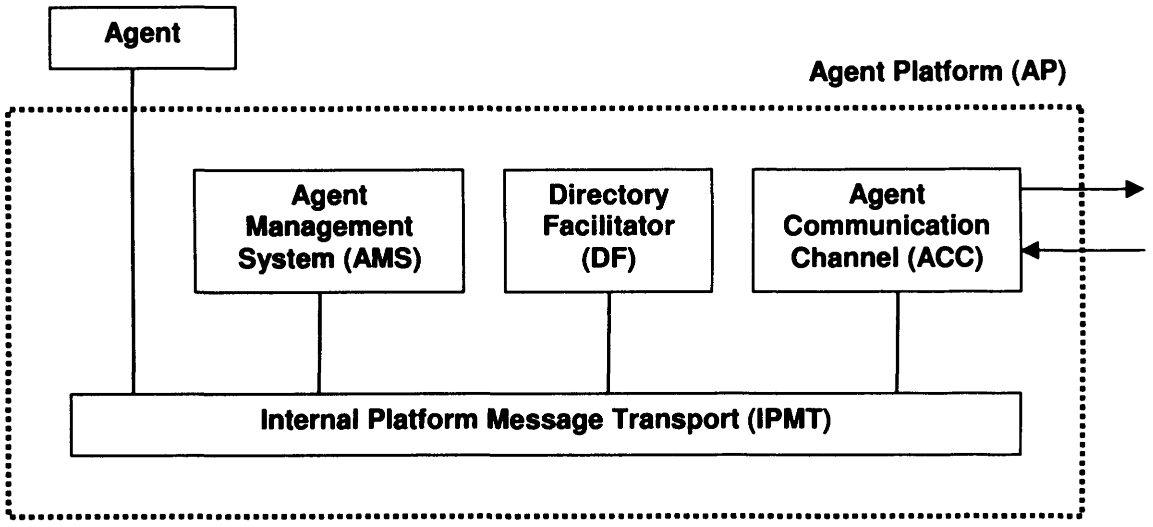

Agent
Agent Platform (AP)
Agent Management System (AMS)
Directory Facilitator (DF)
Agent Communication Channel (ACC)
Internal Platform Message Transport (IPMT)

Figure 2.4: The FIPA agent platform.

The figure, adapted from [47], illustrates the elements comprising the FIPA agent platform.

agent, and the agent's name, which can be used to uniquely identify the agent. Agents are able to query a DF agent and discover information about registered agents. DF agents may register with remote DF agents belonging to other agent platforms, which allows federated queries to be performed. The functionality of the FIPA DF agent provides a greater degree of support than is specified by the KQML communication facilitator agent, which plays a similar role in KQML agent systems. Another special FIPA agent, the ACC agent, handles messages between agent platforms, and must support the Internet Inter-Orb Protocol (IIOP) [34] as a means of inter-platform communication. When an agent sends a FIPA ACL message to an agent residing on a different platform, the message is forwarded to the ACC, which then forwards the message to the remote platform ACC, via which the message reaches the receiver agent.

FIPA specifications are constantly evolving, with meetings held several times each year at which various contributing companies and organisations participate in open discussions about future directions. The FIPA specification life cycle involves the development of preliminary and experimental specifications, which must be extensively tested before being given a standard status. The aspects of FIPA compliant agents discussed here all belong to standard specifications. Over 60 organisations participate as active FIPA members, working on the development of the FIPA specifications, including NASA, IBM, Intel, British Telecom, Fujitsu, Hewlett Packard, Sun Microsystems, and various academic institutions. Additionally, the Agentcities [70] network, a world wide collaborative initiative aiming to construct a global network of interacting agents promotes the use of FIPA as an agent standard.

2.3.4 Evaluation of agent interoperability standards

Of the interoperability standards described, FIPA is the most promising standard and is currently widely used. It should be noted that MASIF and FIPA are not necessarily competing standards, as they concentrate on different aspects of agent interoperability. For instance, the Grasshopper [18] agent development toolkit implements both the MASIF specifications and partially supports the FIPA specifications. Crucially, FIPA supports agent communication, which is essential for the construction of AI based agents which must interact with each other in order to perform their objectives. Such agents are the focus of this thesis, and therefore the adaptation of an agent standard which supports communication is required.

The KQML language variants utilise the same speech-act based model as FIPA ACL, but regardless of which version of KQML is being considered, the FIPA specifications have two important advantages in comparison:

- FIPA is a single standard whereas KQML exists as various de facto standards each of which has failed to become universally accepted. Whereas FIPA compliance ensures that agents developed by different organisations will be able to interact, KQML compliance does not offer the same guarantee.

- FIPA specifications cover a wide range of areas including agent management and agent-software integration whereas KQML focuses on agent communication only.

For these reasons, the FIPA specifications provide by the most promising effort towards standardising interoperability in agent systems.

2.4 Implementing multiagent systems

Developing agents can be a tedious and time consuming experience. The implementation of specifications such as those published by FIPA can require a high level of development expertise and the distributed nature of agent based systems can make the design and deployment of agent systems challenging tasks. As agent based applications become more widespread, there is a need for application domain independent development tools and mechanisms that support the task of designing, implementing, debugging and coordinating MAS.

2.4.1 Agent development toolkits

A range of software toolkits have emerged to simplify the MAS development process. Agent development toolkits focus on providing support for the design, implementation and deployment processes involved in developing MAS. As the focus of this thesis is learning technologies and FIPA compliant agent systems, the significant agent development toolkits that provide either learning technologies or support the FIPA standards are listed in table 2.1.

The most extensive support for FIPA compliant agent development is provided by FIPA-OS and the Java Agent Development Framework (JADE) [12] toolkit. FIPA-OS is an open source software toolkit which assists developers in constructing FIPA compliant agent systems by providing a Java API which enables the construction of application specific agents. An implementation of the FIPA agent platform agents (the DF, AMS, and ACC agents) is also provided. FIPA-OS encourages developers to contribute to bug fixes and further development of the platform which has resulted in a reliable, well documented, freely available toolkit. Full support for the FIPA specifications is provided by FIPA-OS. JADE also provides FIPA compliance and agent development support through a Java API similar to FIPA-OS. Another common feature of both FIPA-OS and JADE is the implementation of the FIPA platform agents to support agent management. JADE concentrates more on providing graphical tools for debugging, managing and monitoring the deployment of agents than FIPA-OS, and also supports agent mobility, which FIPA-OS currently

Toolkit	Comments	Support for FIPA standards	Learning support	Coordination support
FIPA-OS (FIPA open source) http://fipa-os.sourceforge.net	Open source toolkit written in Java. Provides an API for agent development.	YES	NO	Support for contract net and auctions
JADE (Java Agent Development Framework) http://jade.tilab.com	Open source toolkit written in Java providing an API and visual development tools.	YES	NO	Support for contract net and auctions
ZEUS http://more.btexact.com/projects/agents/zeus	Open source toolkit written in Java. Provides visual development tools and an agent component library.	YES	NO	Coordination engine for managing interaction protocols.
Grasshopper (Also known as "enago Mobile") http://www.grasshopper.de/	Concentrates on supporting mobile agent development. Provides a FIPA 'add-on' for communication using FIPA ACL.	YES	NO	NO
DirectIA (Direct Intelligent Adaptation) www.directia.com	Provides a development kit aimed primarily at implementing AI requirements within video games.	NO	YES	NO
ABLE (Agent building and learning environment) http://www.alphaworks.ibm.com/tech/able	Enables the construction of agent behaviours using a software component library which includes machine learning components. Provides visual tools for developing and deploying agents.	YES	YES	Direct hierarchical control through the "Autotune" agent.

Table 2.1: Agent construction toolkits.

The table surveys significant toolkits that support FIPA or learning technologies.

does not.

The only toolkit which supports both the development of learning agents and the FIPA standards is the Agent Building and Learning Environment (ABLE) [14]. ABLE supports an agent based approach to the development of autonomic [4] systems, which are capable of learning, adaptation and reasoning about their internal system state. ABLE provides a set of agent components implemented using Java, which may be combined together to develop agent systems organised into hierarchical structures. Components are provided that support various optimisation, data management and machine learning techniques. A hierarchical model of coordination is employed by ABLE in which a specialised "Autotune" agent component is able to receive goals from higher levels of the hierarchy and control agents at lower levels of the hierarchy. The Autotune agent is able to directly adjust the parameters of the agents it controls in order to optimise their performance. The support for the FIPA specifications is limited, and ABLE does not support the FIPA specifications to the same extent as FIPA-OS or JADE. ABLE provides a FIPA agent component, which is able to communicate using FIPA ACL, but support is not provided for implementing elements of the FIPA agent platform or areas of the FIPA specifications other than communication.

2.5 Conclusions

There is a clear potential advantage in the utilisation of learning techniques within MAS. Clearly, non-trivial MAS application domains will involve scenarios where it is impossible to deploy each agent with a predefined behaviour which decides an action which should be taken given every situation which each agent may encounter. RL has been shown to be particularly applicable in such scenarios, where agents learn via repeatedly interacting with their environment. A key aspect of RL, on which the degree of success obtainable using RL agents depends heavily, is that of choosing payoff functions for agents. In particular, the COIN framework addresses this problem, and it has been concluded that COIN provides a scalable technique for deriving individual payoff functions within MAS consisting of RL agents. The

COIN framework has been applied to domains that encompass difficult coordination problems, and empirical results obtained to show the benefit of the technique.

The survey of current multiagent construction toolkits in this chapter has highlighted the need for an investigation into how support can be provided for implementing coordinated learning based agent systems. The COIN framework has been identified as a scalable technique for coordinating learning agents, and the advantages that the FIPA specifications provide with respect to promoting agent interoperability have been discussed. The focus of this thesis is therefore to develop a programming toolkit for learning agents, based on the COIN framework, which complies with the FIPA specifications.

CHAPTER 3

Utility function assignment

3.1 Introduction

COIN researchers have concentrated on how local utility functions can be derived from world utility functions to coordinate the behaviours of learning agents. The COIN literature, however, does not define a process by which utility functions can be assigned to agents, or how to support such a process. Concentrating on software agents, we focus on developing a framework where utility functions can be assigned at runtime in order to coordinate agents towards objectives which may be specified dynamically. Additionally, in order to compute utility functions such as WLU, information is required that would usually be unavailable to an agent locally, requiring some form of "broadcast signal" which partially parameterises utility functions, as discussed in [71]. This information is required in order to compute utilities where an agent assesses its impact on a society of agents rather than its achievement of local objectives only. LEAF formalises an approach for supporting utility function assignment, and enables the assignment of utility functions that may require remote information in order to be computed. Additionally, LEAF enables interoperability on a large scale by conforming to the FIPA agent system specifications. Utility, standards for agent systems, and the FIPA specifications are now discussed before proceeding to describe the LEAF framework in detail.

3.2 Utility

Utility relates to a concept which has its origins in the areas of economics and decision theory:

- In economics, utility refers to the ability of goods/services to satisfy the requirements of consumers.
- In decision theory, utility is introduced to formalise the rational decision making processes employed by people, where the objective of the decision maker is to choose actions which maximise the expected utility when faced with a set of alternatives, the outcomes of which may be probabilistically determined.

When referring to agent based systems, the various definitions of utility are based on one, or both of these definitions, and are often used to encapsulate one or more of the following:

A payoff function: The notion of utility as a payoff function relates to use of a reward signal to assess the benefit that a specific behaviour has to a learning agent. The goal of the agent's learning process becomes one of maximising utility.

An analysis tool: Utility may be utilised to assess the behaviour of an agent or MAS with regard to various criteria ranging from physical properties to abstract concepts. Of particular interest in the analysis of MAS are the relationships between the utilities of different agents, and correlations between agent utilities and MAS utility.

A cost function: Utility may define a cost function quantifying the success of an agent or MAS with respect to a task.

A measure of trust: The utility of an agent may represent the degree of trust a different agent or user has in that agent, based on previous interactions, or other information.

A quality of service (QoS) measure: Utility may encapsulate the QoS associated with services provided by an agent or set of agents.

In this thesis, the concept of utility extends traditional approaches by defining two distinct utility classes: an abstract, "functional" utility, and an implementation based "performance" utility.

3.3 LEAF – The Learning Agent FIPA Compliant Community Toolkit

The aims of LEAF are to provide a domain independent mechanism for supporting the coordination of agent systems with the following characteristics:

- Global objectives exist, which can be solved by the collective effort of agents.
- It is difficult or impossible to achieve global objectives without coordinating the actions of the agents in the system.
- It is difficult of impossible to achieve objectives by implementing predefined agent behaviours, implying that there is a potential benefit in the utilisation of learning behaviours.
- Agents are able to communicate via the Internet.

As discussed previously, COIN is an efficient coordination technique for agents learning to adapt their behaviours. It is our belief that approaches based on COIN can provide powerful techniques supporting the development of large scale interoperable agent systems by providing domain independent support for coordination. The FIPA specifications offer a widely adopted set of standards for agent technology, and for this reason, we approach the problem of adapting COIN for FIPA compliant agent systems. COIN researchers have so far concentrated on how local utility functions are derived from world utility functions, and the performance of RL agents learning to optimise these functions in a variety of domains. In considering the adaptation of techniques based on COIN within FIPA compliant software agents, it is of course possible to assume that agents can be deployed with predefined utility functions, where learning to optimise utility functions will in turn have the desired global behaviour. There are two difficulties associated with this assumption:

1. COIN utility functions require information (in order to compute utility) that would not normally be available locally to an agent. Obtaining this information by communicating with other agents may be difficult, and may require that agents are configured in such a way that they supply this information to each other. Designing agents in such a way that this is achieved may be time consuming, and may require excessive hand tailoring for individual applications.

2. If, following the deployment of an agent system, global objectives change in some way, the world utility function may have to be redefined, which requires agents to be re-deployed with new local utility functions.

It should be noted that the COIN researchers concentrate on a wide range of potential applications whereas the LEAF framework is developed for software agents only. The COIN researchers cite in their literature various potential applications including coordinating teams of robotic rovers with respect to future missions to Mars [67], and coordinating constellations of communication satellites [71]. Applications such as these would involve an extremely high cost, and an extensive analysis and design process. It would be difficult to imagine a scenario in which projects such as this would require world utility functions to be altered following deployment, and extensive analysis would take place to efficiently configure and supply remote information to utility functions. In contrast, we are approaching the problem of designing a framework for coordination in software agent systems, where much less analysis and design would take place, global objectives may change at runtime, and a general framework for assigning utility functions may be used in multiple applications. Additionally, the performance of software agents may be dependent on implementation architectures and the platforms on which agents are deployed, in addition to the more abstract behaviour optimised by RL algorithms. The LEAF framework is intended to be applicable by an individual or team designing a MAS capable of achieving a set of objectives cooperatively. The individual/team engaged in this process is referred to in this thesis as the MAS designer. The key areas of functionality which LEAF aims to provide are as follows:

- To provide a means of utility function assignment, allowing utility functions to be defined by the MAS designer and assigned to agents dynamically, that is, utility functions can be assigned during the course of an agent's runtime execution, and not just at deployment/creation time. It should also be possible to re-assign utility functions, replacing previously assigned functions.
- It should be possible to assign utility functions that require information not available locally in order to be computed, allowing the assignment of utilities such as WLU. The LEAF framework should provide support for supplying this information to assigned local utility functions.
- To approach the notion of utility from two different perspectives, performance utility and functional utility. Whereas functional utility is associated with abstract agent/MAS behaviour, performance utility focuses on performance engineering related aspects of behaviours.

The LEAF framework integrates utility function assignment with FIPA compliant agent systems in order to provide a means of coordination for learning agents based on the assignment of utility functions to influence agent behaviour. The LEAF approach to coordination is based on the notion of an agent community, where a community has a defined set of objectives (encapsulated by global performance and global functional utility functions), and a set of local utility functions which may be assigned to agents in order to coordinate their behaviours. LEAF specifies no particular rules for defining local utility functions, they may by derived from global utility functions (as are COIN utility functions), or they may be specified arbitrarily by the MAS designer.

An agent is able to optimise its utility functions by using local learning techniques, and may belong to multiple communities (see figure 3.1), in which case multiple utility functions may be assigned where an aggregate function used to combine

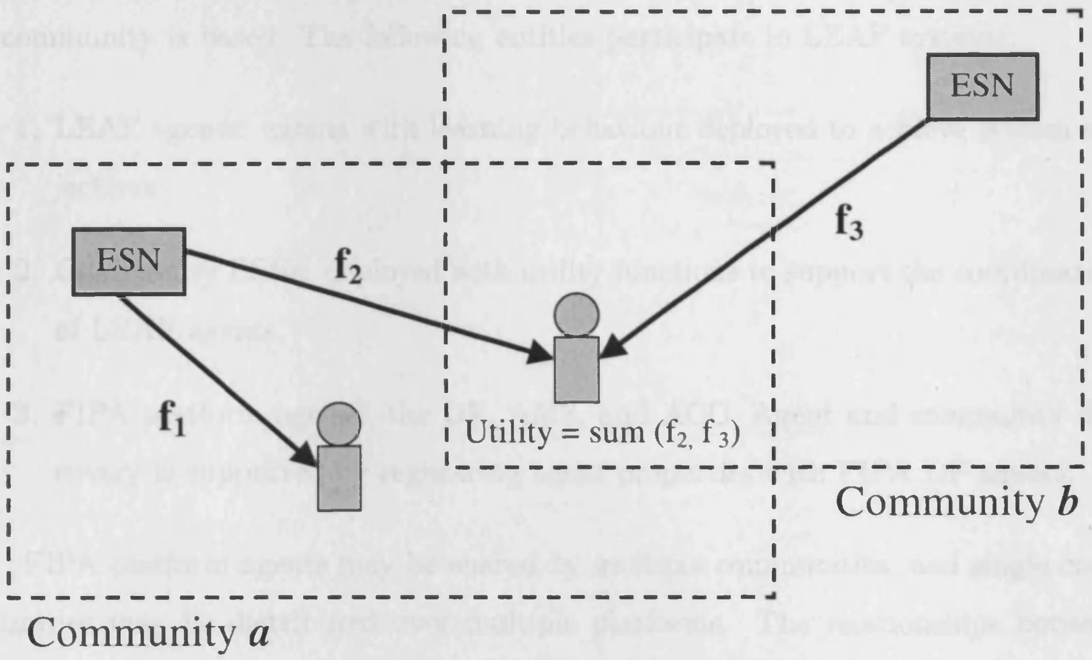


Figure 3.1: Assigning multiple utility functions.

Agents belonging to multiple communities maintain multiple utility functions. The resulting utility of the agent will be by default a summation of the assigned utility functions. Other aggregation schemes for combining the output of multiple utility functions may also be defined.

these functions is subsequently optimised by the agent. Similarly, performance and functional utility functions may also be combined by an aggregate function, allowing an agent to optimise its behaviour with respect to a combination of the aspects of agent behaviour that performance and functional utilities represent. Aggregate utility functions may be used to weight the importance of maximising functional versus performance utilities in the agent's learning process. Utility functions are assigned by a community environment service node (ESN), which is the entity around which a community is based. The following entities participate in LEAF systems:

1. LEAF agents: agents with learning behaviour deployed to achieve system objectives.
2. Community ESNs: deployed with utility functions to support the coordination of LEAF agents.
3. FIPA platform agents: the DF, AMS, and ACC. Agent and community discovery is supported by registering agent properties with FIPA DF agents.

FIPA platform agents may be shared by multiple communities, and single communities may be distributed over multiple platforms. The relationships between these entities are illustrated in figure 3.2.

LEAF agents

When registering with the platform DF, an agent may expose a number of service descriptions, which describe certain properties of the services it offers. Consequently, agents are able to discover certain properties of other agents when they utilise the lookup services provided by FIPA platform DF agents. Associated with a FIPA service description are a range of attributes, including a service type, service name, ontologies, and other properties. A FIPA service description is utilised to provide a means of detecting LEAF agents and community ESNs. All LEAF agents possess a service description with the following properties:

1. Service type = leaf.agent.leaf-agent

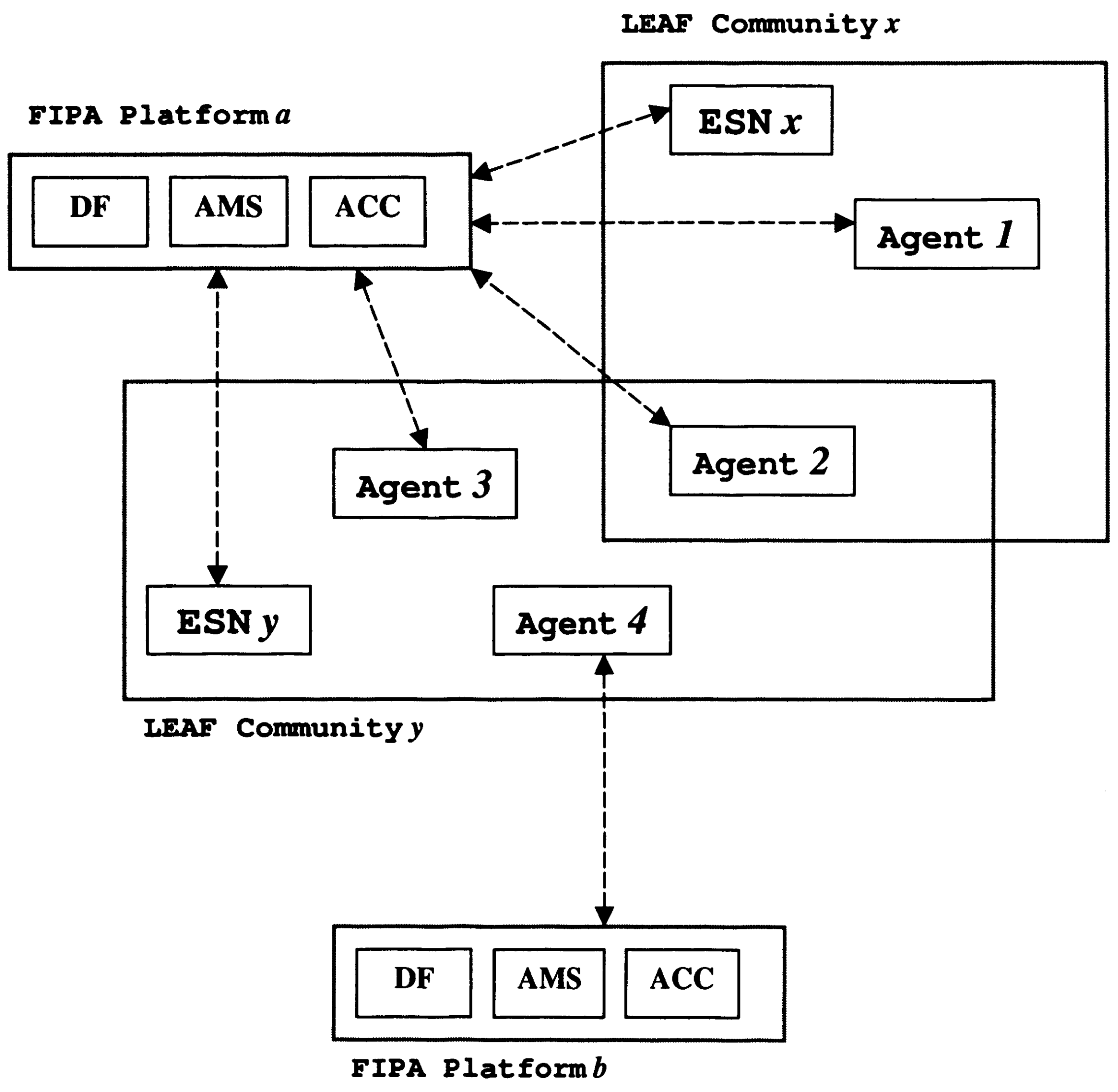


Figure 3.2: The architecture of a LEAF system.

Illustration of a possible configuration of two LEAF communities, x and y. Each community is represented by one ESN, and each ESN is a FIPA compliant agent registered with a FIPA agent platform. Multiple ESNs may reside on the same agent platform. Agents join communities by contacting the relevant ESN, and do not have to belong to the same agent platform (as is the case with agent 4) in order to do this. Agents may also belong to multiple communities, as illustrated by agent 2.

2. Service ontology = leaf.ontology.basic-ontology

These elements of the LEAF agent service description allow LEAF agents to be identified. Additionally, every LEAF agent is associated with a specific *agent type*, and this information is also made available via a service description. The set of LEAF agent types are used to categorise agents, and a valid agent must implement a behaviour conforming to its associated type, for example, the agent type **`leaf.data.storage`** may define a specific type for agents managing the storage of data. A LEAF agent's type is identified by the service-name field of the LEAF agent service description. The MAS designer defines the properties/behaviours associated with agent types, and this information is assumed to be available to all ESNs/agents. Retrieving an agent's type by reading the service name field of a LEAF agent service description therefore allows certain assumptions to be made about the properties and behaviour of an agent. Multiple service descriptions may be specified for a LEAF agent, however only one LEAF service description, of the form specified, may be defined for a specific agent.

Community ESNs

A community ESN is a FIPA compliant agent, possessing a service description with the following properties:

1. Service type = leaf.community.community-esn
2. Service ontology = leaf.ontology.basic-ontology

Each community is associated with a specific *community type*, which defines certain application specific properties of the community. The properties of each community type are defined by the MAS designer, and the mapping of community types to application specific properties is assumed to be available to all LEAF agents. The community type of an ESN is defined in the service-name field of the ESN's service description, and only one service description is defined for each community ESN. Specific communities are identified by a community ESN's agent identifier (FIPA identifiers (AID)s are globally unique identifiers possessed by all FIPA compliant agents).

3.3.1 Global utility functions

An ESN representing a LEAF community, c, is deployed with a global functional utility function and a global performance utility function, which compute the community's functional utility, G_c^F, and performance utility, G_c^P, respectively. Referring to [72], global utility in COIN is defined as:

$$G = f(\underline{\zeta}) \tag{3.1}$$

where the system's state evolves over a set of discrete time steps, $t \in \{0, 1, ...\}$. $\underline{\zeta}_{\mu,t}$ is defined in [75] as follows:

$\underline{\zeta}_{\mu,t}$: "We let all relevant characteristics of an agent (μ) at time t - including its internal parameters at the time as well as its externally visible actions - be encapsulated by a Euclidean vector $\underline{\zeta}_{\mu,t}$".

$\underline{\zeta}$ is defined as the combined state of all agents in the system over all time steps. The size of $\underline{\zeta}$ for a community therefore depends on what is considered to be the relevant characteristics of each agent in the community and what the externally visible actions of each agent are. In LEAF, the assumption is made that a MAS designer can identify, prior to the deployment of a MAS, the relevant characteristics of each agent, and also the relevant externally visible actions of agents (some externally visible actions may not feature in the computation of global utility).

3.3.2 Local utility functions

Each LEAF community is deployed with a set of local utility functions, specified by the MAS designer. For each LEAF agent type permitted to join a community, a functional utility function may be defined, capable of computing a functional utility value for agents of each type, and similarly, a performance utility functions may be specified also. One of the principle aims of the LEAF framework is that it should support the assignment of local utility functions that give an indication of the impact of an agent's behaviour on a LEAF community, and not just an indication of the impact of an agent's behaviour with respect to local self-interested objectives.

Utility functions with this property essentially have a degree of *social context*, which means that an individual agent is able to assess the effects of its behaviour upon the community to which it belongs. This is the approach taken in COIN, where utilities such as WLU are defined.

3.4 Observable properties, remote parameters, and utility function assignment

Following the previous discussion, which introduced LEAF agents, community ESNs and utility functions, it is now necessary to describe in detail the process of utility function assignment, and how this is used to support the development of coordinated MAS. Broadly speaking, the process of utility function assignment is supported by *remote parameters*, which provide local utility functions with information from the community in which they exist, and *observable properties*, which encapsulate various aspects of an agent's behaviour relevant to the computation of local utility. An observable property can be defined as any part of an application specific agent's state at a given time. Global utility functions are parameterised by the observable properties of the agents in a community, and community ESNs must manage the supply of remote parameters to the agent's belonging to their communities. Observable properties, remote parameters, and utility function assignment are now described in more detail.

3.4.1 Observable properties

Specifically, when applying the LEAF framework, the MAS designer must identify a set of observable properties for each LEAF agent type, which are equivalent to the relevant characteristics and visible actions of the agent with respect to the computation of global utility.

The global utility functions maintained by ESNs are parameterised by the observable properties of the agents belonging to a community. Therefore, a mapping exists, which maps each LEAF agent type, t, to the set of observable properties, O_t

required of that agent type for the computation of global utility. Each observable property consists of a property name and a property value, where values can range in complexity from single numeric values to more complex data structures. It is the responsibility of the MAS designer that:

- The set of observable properties maintained by each LEAF agent type that can belong to a given community adequately encapsulate all the information required from an agent of that type in order for the community's global utility functions to be computed.
- Observable properties can be represented using character strings.

3.4.2 Remote parameters

Utility functions with social context will generally be parameterised by information that is not observable locally by an agent. The LEAF framework allows for this by introducing remote parameters, which supply remote information to local utility functions as illustrated in figure 3.3. Associated with a local utility function for an agent type, t, are a set of remote parameters, R_t, each of which consists of a parameter name and a parameter value. When specifying a local utility function for an agent of type t, the MAS designer may assume the availability of R_t, which may be used to parameterise the utility function in order to supply social context.

Remote parameters originate from the community ESN, and are based on the observable properties of the agents belonging to a community. Generally, a remote parameter is a function of a set of observable properties (as are the functional and performance global utility functions, which may themselves by defined as remote parameters), and the ESN dynamically computes remote parameter values and manages the supply of these values to the utility functions assigned to LEAF agents, enabling the computation of utility functions with social context. It is the responsibility of the MAS designer to ensure that:

- The set of observable properties maintained by each LEAF agent type allowed to belong to a given community adequately encapsulate all the information

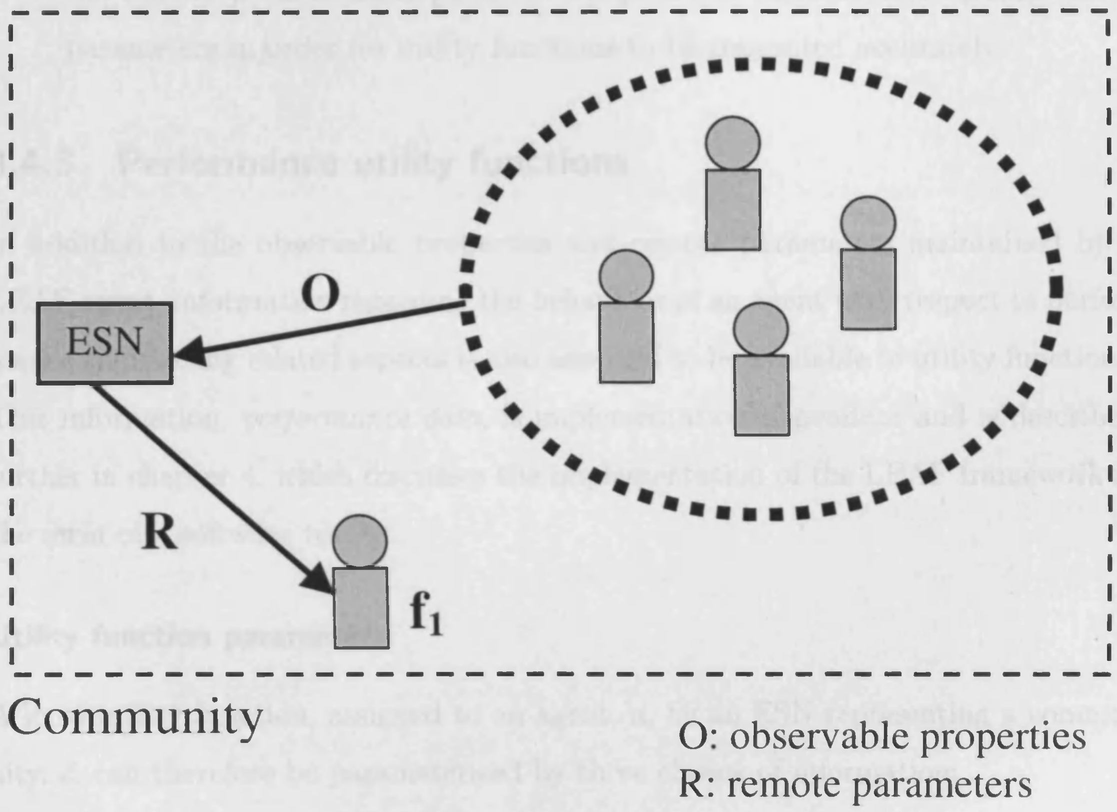


Figure 3.3: Utility function assignment.
Agents joining a community are assigned a local utility function by that community. Utility functions are assigned based on the agent's LEAF agent type. The ESN extracts observable properties from the LEAF agents in its community in order to maintain a global utility function, and also to update the remote parameters of LEAF agents in the community. Remote parameters are used to engineer social context into utility functions.

that may be required in order to compute the remote parameters required by any utility function that may be assigned to an agent belonging to the community.

- When a utility function is assigned, the ESN supplies the utility function with the remote parameters required to compute the function and updates these parameters in order for utility functions to be computed accurately.

3.4.3 Performance utility functions

In addition to the observable properties and remote parameters maintained by a LEAF agent, information regarding the behaviour of an agent with respect to performance engineering related aspects is also assumed to be available to utility functions. This information, *performance data*, is implementation dependent and is described further in chapter 4, which discusses the implementation of the LEAF framework in the form of a software toolkit.

Utility function parameters

A local utility function, assigned to an agent, a, by an ESN representing a community, c, can therefore be parameterised by three classes of information:

1. The set of observable properties maintained by a.
2. The set of remote parameters supplied by the ESN representing c.
3. The performance data maintained by a.

3.5 Utility function assignment and computation

3.5.1 ESN behaviour

An ESN representing a community, c, is deployed with the following:

- A set of agent types, M_c, allowed to be members of a community c. This set is used to form a policy dictating which LEAF agents are allowed to join the community which the ESN represents.
- A set of local functional utility functions, L_F. A mapping, $f : M_c \mapsto L_F$, dictates which functional utility functions should be assigned to members of community c.
- A set of local performance utility functions, L_P. The mapping $p : M_c \mapsto L_P$ dictates which performance utility functions should be assigned to members of community c.

The purposes of the ESN are to assign utility functions to agents, support assigned utility function by supplying remote parameters and to compute global utility functions. Therefore, ESNs require certain observable properties from members of the communities they represent, firstly as they may be needed to compute global utility functions, and secondly because remote parameters are derived from observable properties.

In order to support the assignment of the utility functions $L_F \cup L_P$ within a community c, each function requires a set of remote parameters in order to be computed. Given an agent type t, which is a member of a community, c, a set of observable properties, $O_{t,c}$ are relevant in the derivation of the remote parameters required by $L_F \cup L_P$.

The behaviour of an ESN representing community c can therefore be summarised as:

Communication with LEAF agent requesting to join/leave community c: the ESN must only allow agents of a type belonging to the set M_c to join the community.

Assignment of utility functions: a LEAF agent of type t joining community c will be assigned utility functions defined by the mappings f and p. The ESN will also specify the set of observable properties $O_{t,c}$, which are required from the joining agent in order to derive remote parameters for the utility functions $L_F \cup L_P$. It is the responsibility of the agent to inform the ESN of the values of these parameters, as described in section 3.5.3.

Update of remote parameters: Each utility function assigned to an agent may require remote parameters in order to be computed accurately, and the ESN must supply these parameters by deriving them from the observable properties of the agents in the community.

Removal of utility functions: when an agent leaves a community, any utility functions that have been assigned to that agent should be removed by the ESN.

3.5.2 Aggregating utility functions

In order to aggregate the utility functions assigned to an agent, an agent is deployed with three functions, f_1, f_2 and f_3, which combine the output of assigned utility functions. Firstly, the performance and functional utility functions assigned to an agent by each community ESN are combined into single performance and functional utility values. Where an agent is a member of a set of communities $\{c_1, c_2, ..., c_n\}$, local performance utility is defined by $L^P = f_1(L^P_{c_1}, L^P_{c_2}, ..., L^P_{c_n})$, and local functional utility by $L^F = f_2(L^F_{c_1}, L^F_{c_2}, ..., L^F_{c_n})$, where $L^P_{c_i}$ is the output of the local performance utility function assigned by the ESN representing community c_i, and $L^F_{c_i}$ is the output of the local functional utility function assigned by the ESN representing community c_i. $f_3(L^P, L^F)$ combines the values of L^P and L^F into a single local utility value for an agent. An agent has access to this single local utility value, in addition to L^P and L^F, therefore enabling it to obtain its performance utility, functional utility, and aggregate utility.

3.5.3 Agent behaviour

A LEAF agent is deployed with a learning behaviour which is based around the maximisation of local aggregate utility. There are two possible aspects of this learning process:

1. As a member of a community, or possibly multiple communities, learning a behaviour which maximises the local utility.
2. Learning which communities it is beneficial to be a member of, based on the utility the agent is able to obtain in specific communities.

The experimental work described in chapter 5 concentrates on demonstrating how (1) can be achieved. Demonstrating the learning process inherent in (2) is beyond the scope of this thesis, but is an exciting avenue for possible further research. In addition to learning which community an agent should join, it is also possible to deploy agents with simple rules which dictate which community they should join, which is the approach used in the experimental work in this thesis.

In addition to an agent's application specific learning behaviour, the following functionality is required of all LEAF agents:

Maintain observable properties: a LEAF agent of type t must maintain accurate values of the observable property set O_t.

Communicate with the ESN when joining/leaving communities: in order to join a community, a LEAF agent must first discover the ESN representing the community, which may be achieved by performing a search of the FIPA DF agent with which the ESN is registered. An agent must be able to communicate with the ESN in order to join/leave communities.

Accept assigned utility functions: an agent should respond to a request from an ESN to assign a utility function. The only aspect of utility function assignment that should be visible to an agent's application specific behaviour is the result of the computation of the aggregate performance, functional, or overall utility value of the agent.

Maintain remote parameters: a LEAF agent must maintain any remote parameters that are sent to the agent by ESNs representing communities that the agent is a member of. Assigned utility functions must be able to access these parameters.

Maintain performance data: a LEAF agent maintains various performance data, which is accessed by performance utility functions to compute utility. This is described in more detail in chapter 5.

Remove assigned utility functions: an agent must respond to a request from the ESN to remove a utility function that the ESN has previously assigned.

Update observable property values: during the process of utility function assignment, an ESN specifies a subset of observable properties (O_t for an agent of type t) which are required in order to compute global utility, or to derive remote parameters for the utility functions $L_F \cup L_P$. In addition to which observable properties are required, the ESN specifies a scheme for updating observable properties, which may be either (a) observable properties are updated each time their values change, or (b) observable properties are updated periodically with a specified time period. The use of schemes a or b depends of how often observable properties are expected to change, the degree of accuracy required when obtaining observable properties, and the time taken to transfer observable properties (which in turn depends partially on the potential size of the observable property values). The LEAF agent must implement the specified observable property updates specified by the ESN.

3.5.4 Communication mechanisms

Communication between the ESN and LEAF agents takes place in order to (a) allow agents to join/leave communities (b) assign utility functions, and (c) transport observable properties and remote parameters. FIPA ACL is used to achieve a and b, whereas a faster means of communication provided by TCP sockets [34] are utilised in order to achieve c. The implementation of these communication mechanisms is described in the following chapter.

3.6 Conclusion

This chapter has described the LEAF framework, which supports utility function assignment within FIPA compliant agent systems. LEAF uses communities as the basis for developing large scale MAS, where a community is defined by a set of objectives, specified by global performance and functional utility functions. The members of a community may change over time, and LEAF agents implement local learning techniques enabling behavioural adaptation to take place, where agents learn to maximise their utility functions, in turn maximising the global utility functions of the community. Additionally, agents may learn which community to join and possibly belong to multiple communities.

3.6.1 The benefits of utility function assignment

The key benefit of adopting the utility function assignment model presented here is that it provides a mechanism for specifying utility functions which reflect the dynamics of a social group of agents. This provides a learning value function which allows an agent to adapt its behaviour for the benefit of its associated social group, or groups. Depending on the number of agents in the social group(s) in which an agent participates, the most basic utility function which may be assigned to coordinate agent communities, a team-game utility function, may perform inadequately due to the signal to noise problem in determining the effects of local actions only on a value influenced by the actions of numerous agents. Significantly, if difference utilities, for instance the COIN WLU function, are assigned, this signal to noise problem may be overcome.

3.6.2 Assumptions concerning agent behaviours

The implications of utility function assignment are that the autonomy of a LEAF agent is impacted. In order for the mechanism to work, an agent must not be able to refuse to accept an assigned utility function, and must strive to maximise the aggregate utility value obtained from the utility functions that have been assigned to the agent. In the next chapter, an implementation of the LEAF framework is presented, and the mechanism via which this is achieved is described. In subsequent chapters, this implementation is used to apply the LEAF framework to problem domains, and evaluate it as a mechanism for the purposes of supporting the development of coordinated, learning based MAS.

3.6.3 Performance Utility

The use of performance utility is an extension to existing approaches to defining utility which allows efficiency of an agents utilisation of system resources to be categorised separately from an agents success with regard to the achievement of application level objectives. It may therefore be possible to design agents which utilise performance utility functions to optimise their effects on communities of agents which share common resources, and additionally, performance utility may demonstrate interesting relationships with functional utility and enhance the MAS designer's ability to analyse MAS behaviour, as demonstrated in chapters 6 and 7.

CHAPTER 4

Implementation of the LEAF toolkit

In this chapter, the implementation of a software toolkit based on the LEAF framework is presented. This toolkit is intended to be of use to an individual or team during the construction of learning based MAS conforming to the LEAF framework. In this thesis, the individual or team involved in the construction of a MAS is referred to as the MAS developer, the toolkit used by the MAS developer is referred to as the LEAF toolkit, and the resulting MAS is referred to as a LEAF MAS.

The structure of this chapter is as follows, firstly the requirements analysis and requirements definition of the toolkit are discussed. The implementation of the toolkit, including the choice of programming language and the decision to utilise the FIPA-OS agent toolkit is justified. An overview of the salient features of FIPA-OS is also given. The architecture of the toolkit is then presented, and finally the development process that must be undertaken to use the LEAF toolkit is discussed from the viewpoint of the MAS developer.

4.1 Requirements analysis

In order to explain how the concepts described in the previous chapter are implemented in the form of a software toolkit, the starting point for this discussion is an analysis of the requirements of the software.

4.1.1 Requirements definition

The software must provide a toolkit supporting the development of FIPA compliant learning based agents and the coordination of agents using utility function assignment based techniques.

4.1.2 Requirements specification

The requirements of the LEAF toolkit are to support the MAS developer in the process of constructing LEAF MAS via an API. A LEAF MAS consists of two types of agents, LEAF agents, and LEAF ESNs. LEAF agents and LEAF ESNs should be fully FIPA compliant agents, allowing interoperability with FIPA platform agents and other FIPA compliant agents. The required features of the LEAF API are now identified in detail.

Development of LEAF agents

The toolkit should provide a means of constructing LEAF agents, assisting the developer with the following tasks:

1. Interacting with FIPA platform agents.
2. Communicating with other FIPA compliant agents using FIPA ACL.
3. Discovering other LEAF agents and the services they offer.
4. Discovering LEAF communities.
5. Joining and leaving LEAF communities.
6. The declaration and modification of the observable properties of the agent.
7. Accessing local performance and functional utility values. This implies the handling of local utility function assignments from ESNs.

Specification of utility functions

The toolkit should provide a means of specifying utility functions, and must support the assignment of utility functions to LEAF agents by ESNs.

Development of LEAF ESNs

The toolkit should provide a means of constructing LEAF community ESNs as FIPA compliant agents, assisting the developer with the following tasks:

1. Interacting with FIPA platform agents.
2. Notifying an ESN when LEAF agents join/leave the community the ESN represents.
3. Assigning performance/functional utility functions to LEAF agents.
4. Accessing and updating observable properties and remote parameters within the community managed by an ESN.

4.2 Choice of programming language

It is possible to implement the toolkit using any programming language that provides support for developing distributed applications, such as C/C++ or Java. Java is chosen to implement the toolkit, due to the following advantages it provides over other programming languages in this domain:

- Java is a platform independent language, allowing developers to write software that can be compiled once for execution on different platforms.
- The current popularity of Java is a factor due to the fact that it is anticipated that many developers are familiar with the language and will therefore be able to use a Java based LEAF toolkit.
- Java allows distributed computing applications to be developed with ease when compared to many other programming languages [34].
- A number of agent construction toolkits, including JADE, Zeus and FIPA-OS, are implemented in Java, therefore enabling the integration of LEAF with these toolkits.

It is assumed that the reader is reasonably familiar with the Java programming language and object oriented development methodologies.

4.3 Decision to utilise the FIPA-OS agent toolkit

One of the requirements of the LEAF toolkit is that it must utilise FIPA compliant agents. Ensuring FIPA compliance can be difficult and time consuming, as FIPA standards are complicated and constantly evolving. In order to avoid "reinventing the wheel", the logical approach to implementing LEAF is to build on an existing FIPA compliant toolkit. At the time of development, FIPA-OS provides a freely available, well documented, reliable toolkit and is therefore chosen to implement the LEAF toolkit.

4.3.1 Developing agents using FIPA-OS

The utilisation of FIPA-OS requires the developer to have at least a basic knowledge of the Java programming language, and developers must be competent Java programmers to construct more complex agent applications. The FIPA-OS API provides a set of Java classes encapsulating the various components of FIPA-compliant agent systems required by the developer. The principle FIPA-OS classes relevant to the discussion presented in this chapter are described in table 4.1. It should be noted that all instances of FIPA-OS mentioned in this thesis refer to Standard FIPA-OS version 2.1.0.

Agents are encapsulated by the `FIPAOSAgent` class, which is extended by the developer in order to construct an application specific agent. The application specific code defining the agent's behaviour is then able to utilise the FIPA-OS API, which allows the developer to interact with the FIPA agent platform and communicate with other FIPA compliant agents using FIPA ACL.

An agent's application specific behaviour is programmed in Java, and while FIPA-OS places no constraints on its composition, support is provided for defining the behaviour of agents. An important methodology often utilised in agent based development methodologies, and supported by FIPA-OS, is the construction of agent behaviours based around the concept of a *task*. In an effort to simplify the development process, tasks are used to identify modular aspects of an agent's functionality and develop these modules as separate entities. In a similar way to

FIPA Specifications	FIPA-OS Implementation
FIPA-compliant agent	`FIPAOSAgent` class – Allows the developer to create application specific agents by extending this class.
FIPA-compliant agent with JESS interface	`JessAgent` class – Allows the developer to create application specific agents with an interface to JESS.
FIPA agent identifier (AID)	`AgentID` class – Allows developers to identify an agent by its associated AID.
ACL message	`ACL` class – Allows developers to create and handle FIPA ACL messages.
FIPA ACL Conversation	`Conversation` class – Allows developers to create and handle FIPA ACL conversations. A conversation contains a series of ACL messages.
Service Description	`ServiceDescription` class – Allows developers to construct, read, and compare the properties of FIPA agent service descriptions.

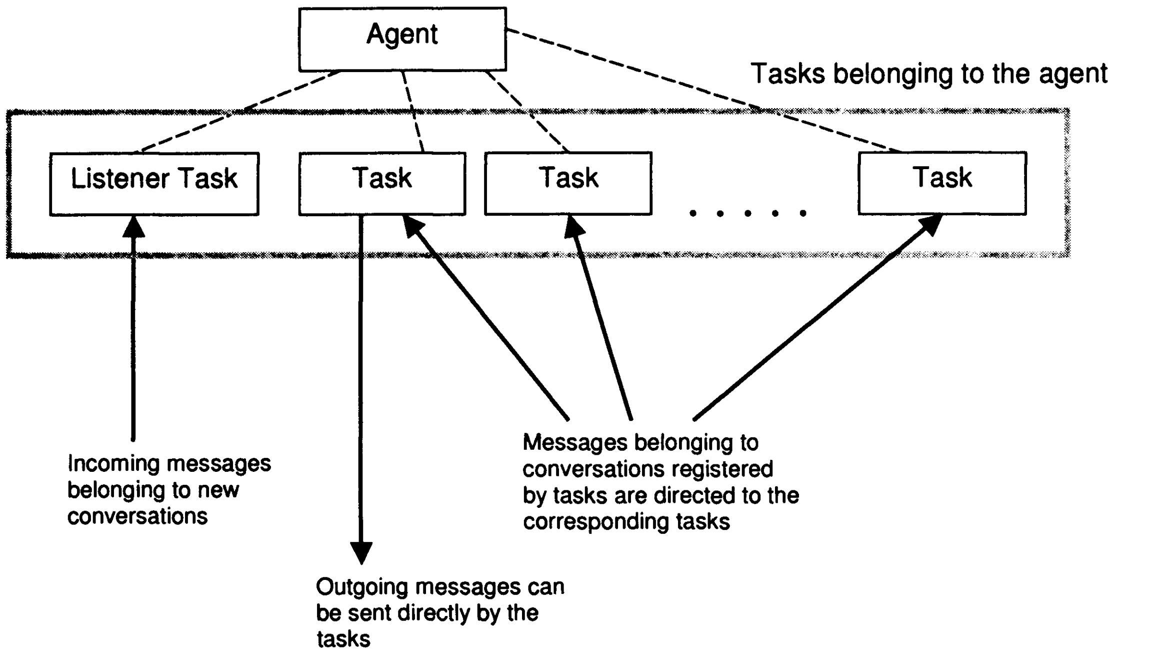
Table 4.1: Key components in application specific agent development using the FIPA-OS API.

object-oriented programming techniques, tasks promote the re-use of modules and simplify the design and programming processes. Support for tasks is provided by FIPA-OS via the `Task` class, which is extended to write application specific tasks. FIPA-OS allows tasks to register as owners of FIPA ACL conversations, and handles the routing of messages relating to registered conversations to the corresponding task. A strategy supported by FIPA-OS, illustrated in figure 4.1, is to use a listener task to accept new incoming conversations, which creates the relevant tasks needed to handle these conversations directly. This strategy can simplify the design and implementation of agents, as it alleviates the need for managing numerous conversations and their associated processes within the agent's main class.

Additionally, FIPA-OS provides an interface to the Java Expert System Shell (JESS) [5]. JESS is an expert system shell providing an environment for the construction of rule based expert systems, which may be used to support the development of intelligent agents. FIPA-OS allows developers to interface with JESS by extending the `JessAgent` class instead of the `FIPAOSAgent` class when developing agents.

4.4 LEAF architecture

The LEAF toolkit architecture consists of an API that builds on FIPA-OS to provide support to the developers of MAS conforming to the LEAF framework. The architectures of LEAF agents and LEAF ESNs are built directly on the architecture of the FIPA-OS agent. FIPA-OS provides two base classes used to implement agents, the `FIPAOSAgent` class and the `JessAgent` class. These classes are extended by the developer to construct application specific agents, and in a similar way, the developer extends the `Task` class to define application specific tasks. The LEAF agent is required to build on this functionality, satisfying the requirements outlined in section 4.1, and continuing to allow the use of the underlying FIPA-OS toolkit. In order to do this, the LEAF agent architecture is based on three classes which extend the FIPA-OS `FIPAOSAgent`, `JessAgent` and `Task` classes. The `LeafNode` class extends the `FIPAOSAgent` class, the `JessLeafNode` class extends the `JessAgent` class,


Agent
Tasks belonging to the agent
Listener Task
Task
Task
Task
Incoming messages belonging to new conversations
Messages belonging to conversations registered by tasks are directed to the corresponding tasks
Outgoing messages can be sent directly by the tasks

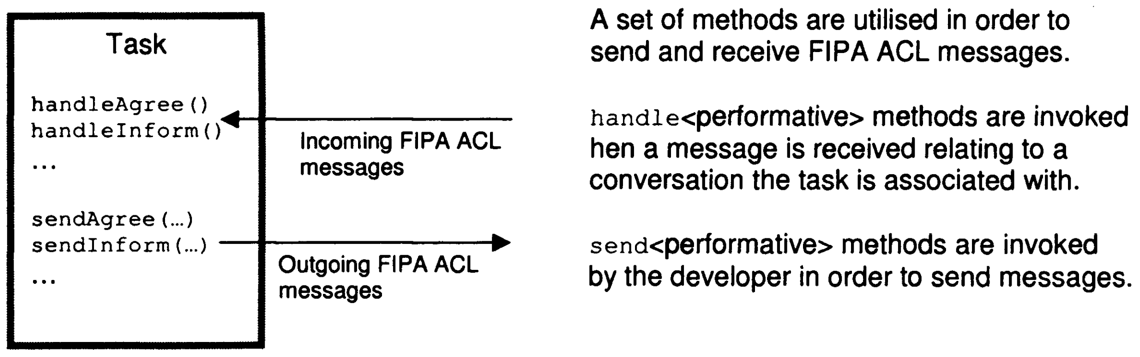

Task
handleAgree()
handleInform()
...
sendAgree(...)
sendInform(...)
...
Incoming FIPA ACL messages
Outgoing FIPA ACL messages
A set of methods are utilised in order to send and receive FIPA ACL messages.
handle<performative> methods are invoked hen a message is received relating to a conversation the task is associated with.
send<performative> methods are invoked by the developer in order to send messages.

Figure 4.1: The use of tasks to construct agents.

The figure illustrates how tasks can be utilised in order to build agents using a modular, or component oriented approach. The FIPA-OS software toolkit provides support for this development technique.

and the `LeafTask` class extends the `Task` class. The implementation of the LEAF ESN is based around the `ESN` class, which also extends the `FIPAOSAgent` class, as the ESN is also a FIPA compliant agent. The MAS developer using LEAF extends these new classes to develop application specific agents, tasks, and ESNs. Figure 4.2 illustrates how this development methodology relates to the FIPA-OS architecture.

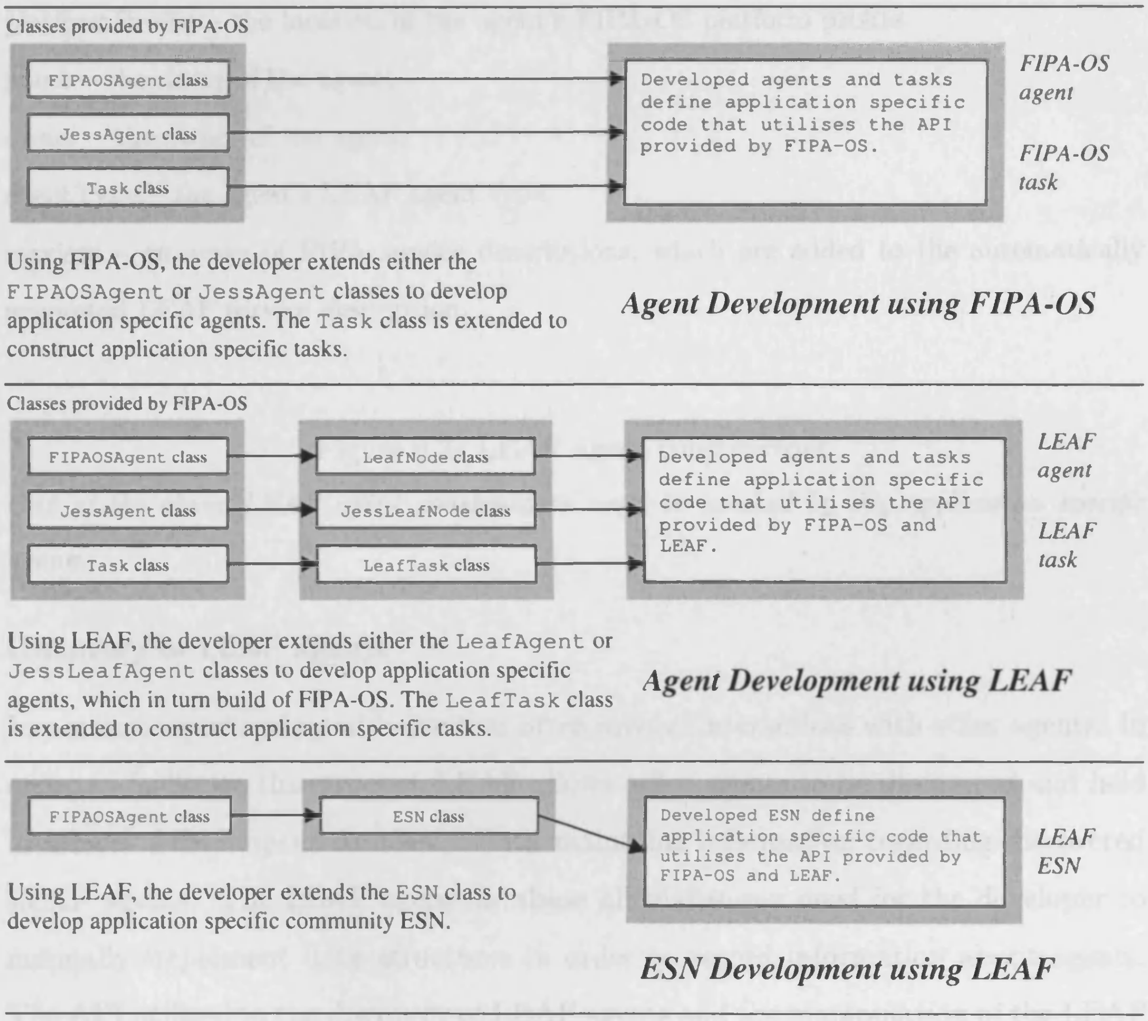
This section now proceeds to describe the four principle components that form the LEAF toolkit, LEAF agents, LEAF tasks, utility functions, and LEAF ESNs. This discussion is intended to benefit the reader with an understanding of the salient features of the LEAF toolkit architecture.

4.4.1 LEAF agents

The developer must utilise either the `LeafNode` class or the `JessLeafNode` class to construct a LEAF agent, depending on whether or not the agent requires the reasoning capabilities of JESS. The methods offered by the `LeafNode` and `JessLeafNode` classes are identical, and the use of JESS is supported by the underlying FIPA-OS API. In order to define the application specific behaviour of a LEAF agent, the developer must define a `startBehaviour()` method, which is invoked following registration of the LEAF agent with FIPA platform AMS and DF agents. In specifying the agent's behaviour, the developer has at their disposal the LEAF agent API, the features of which are now discussed.

Registration with the FIPA agent platform

LEAF agents automatically register with the FIPA platform DF and AMS agents when a constructor belonging to the `LeafNode` or `JessLeafNode` classes is invoked. The constructors made available by these classes are illustrated in figure 4.3. When a LEAF agent is registered, a service description is automatically generated, which can be used by other LEAF agents to detect when searching the DF that the registered agent is a LEAF agent, and also determine the LEAF agent's type. Additional service descriptions specifying in detail the services the agent provides may also be specified, in which case they will also be registered with the DF.


Classes provided by FIPA-OS
FIPAOSAgent class
JessAgent class
Task class
Developed agents and tasks define application specific code that utilises the API provided by FIPA-OS.
FIPA-OS agent
FIPA-OS task
Agent Development using FIPA-OS
Classes provided by FIPA-OS
FIPAOSAgent class
JessAgent class
Task class
LeafNode class
JessLeafNode class
LeafTask class
Developed agents and tasks define application specific code that utilises the API provided by FIPA-OS and LEAF.
LEAF agent
LEAF task
Agent Development using LEAF
FIPAOSAgent class
ESN class
Developed ESN define application specific code that utilises the API provided by FIPA-OS and LEAF.
LEAF ESN
ESN Development using LEAF

Using FIPA-OS, the developer extends either the `FIPAOSAgent` or `JessAgent` classes to develop application specific agents. The `Task` class is extended to construct application specific tasks.

Using LEAF, the developer extends either the `LeafAgent` or `JessLeafAgent` classes to develop application specific agents, which in turn build of FIPA-OS. The `LeafTask` class is extended to construct application specific tasks.

Using LEAF, the developer extends the `ESN` class to develop application specific community ESN.

Figure 4.2: Extension of FIPA-OS base classes by LEAF.

This figure illustrates the principle classes involved in the development of agents using FIPA-OS and how they are extended by LEAF.

- `LeafNode(String platformProfile, String name, String owner, String agentType)`
- `LeafNode(String platformProfile, String name, String owner, String agentType, ServiceDescription[] services)`

platformProfile – the location of the agent's FIPA-OS platform profile.
name – the name of the agent.
owner – the owner of the agent.
agentType – the agent's LEAF agent type.
services – an array of FIPA service descriptions, which are added to the automatically generated LEAF service description.

Figure 4.3: LEAF agent constructors.
One of the above LEAF agent constructors must be invoked by any application specific agent.

Discovery of LEAF agents

Important aspects of agent behaviour often involve interactions with other agents. In order to facilitate this process, LEAF allows other agents to be discovered and held in a local *LEAF agent database*, which maintains information regarding discovered LEAF agents. The LEAF agent database alleviates any need for the developer to manually implement data structures in order to record information about agents. The API utilised in the discovery of LEAF agents and the manipulation of the LEAF agent database is shown in figure 4.4. LEAF agents are discovered via FIPA platform DF agents, and periodic searches are required to keep the LEAF agent database up to date. Discovered LEAF agents are modeled using the `LeafAgent` class, illustrated in figure 4.5, which holds the agent's FIPA agent ID, service descriptions, and agent type.

- `void discoverAgents(String agentType, boolean federated)`

Searches the local platform DF for agents with the specified LEAF agent type. Results are added to the local LEAF agent database.

- `void discoverAgents(String agentType, ServiceDescription[] services, boolean federated)`

Searches the local platform DF for agents with the specified LEAF agent type and specified services. Results are added to the LEAF agent database.

- `LeafAgent[] searchLocalAgentDB(String type)`

Searches the LEAF agent database for agents with the specified LEAF agent type.

- `LeafAgent[] searchLocalAgentDB(String type, ServiceDescription[] services)`

Searches the LEAF agent database for agents with the specified LEAF agent type and services.

- `void deleteFromLocalAgentDB(AgentID agent)`

Deletes an agent from the LEAF agent database.

- `void deleteFromLocalAgentDB(String agentType)`

Deletes all agents with the specified LEAF agent type from the LEAF agent database.

- `void addToLocalAgentDB(LeafAgent agent)`

Adds an agent that has not been discovered by a DF search to the database.

- `void clearLocalAgentDB()`

Deletes all agents from the LEAF agent database.

Figure 4.4: API used to discover LEAF agents via the FIPA platform DF

- `AgentID getAgentID()`

Returns the FIPA agent ID (AID) of the agent.

- `String getAgentType()`

Returns the LEAF agent type of the agent.

- `DFAgentDescription getAgentDescription()`

Returns the FIPA DF description as defined by the FIPA agent management ontology. This is encapsulated by the FIPA-OS `DFAgentDescription` class.

- `boolean equals(Object other)`

Compares the LEAF agent to another for equivalence.

Figure 4.5: The `LeafAgent` class.

*The **LeafAgent** class is used to encapsulate discovered LEAF agents. The methods offered by the class allow retrieval of the agent's AID, agent type and services.*

Interaction with LEAF communities

LEAF agents are able to discover LEAF community ESNs via the LEAF API. A local *community ESN database* maintains information regarding discovered communities. The LEAF API supporting interaction with communities is illustrated in 4.6. When the API is used to discover communities, a search of the DF agent is performed, and any LEAF community ESNs discovered are added to the local LEAF community agent database, which can be queried without the need for searching the DF. Periodic searches are required to keep the local community ESN database up to date.

Once a community is discovered, the agent may join the community. This process involves FIPA ACL communication with the community agent, and once an agent has joined a community, the community is added to the agent's community membership data structure, which lists the communities an agent is currently a member of. The LEAF agent may leave any community it has joined, which also involves FIPA ACL communication. The process of joining/leaving communities is handled automatically by the LEAF agent when the corresponding API method is invoked (see figure 4.6).

- `void discoverCommunities(String type, boolean federated)`

Searches the local platform DF for community agents of the specified type. Results are added to the local community ESN database.

- `void joinCommunity(AgentID communityAID)`

Invoking this method engages the agent in a FIPA ACL conversation with the community ESN, which will result in the agent joining the community. Community ESNs are identified by their AID, which is globally unique.

- `void leaveCommunity(AgentID communityAID)`

Invoking this method engages the agent in a FIPA ACL conversation with the community ESN, which will result in the agent leaving the community.

- `AgentID[] getCommunityMembership()`

Used to obtain the set of community ESNs representing the communities that the agent is currently a member of.

- `String getCommunityType(AgentID communityAID)`

Used to obtain the community type of a community agent in the local community ESN database.

- `AgentID[] listLocalCommunityDB()`

Returns the AIDs of all agents in the local community ESN database.

- `void deleteFromLocalCommunityDB(AgentID communityAID)`

Deletes a community ESN from the local community ESN database.

- `void deleteFromLocalCommunityDB(String communityType)`

Deletes all community ESNs of the given type from the local community ESN database.

- `void clearLocalCommunityDB()`

Deletes all community ESNs from the local community ESN database.

Figure 4.6: API used to discover and interact with LEAF communities

Handling of utility function assignment

Once an agent has joined a community, the community ESN may assign both performance and functional utility functions to the agent. Utility functions are assigned via a FIPA ACL conversation following the FIPA-request protocol, initiated by the ESN. Local utility functions are defined by a Java class, written by the MAS designer, which implements an abstract `LocalUtilityFunction` class, and defines a single method, `compute()`, invoked to obtain the output of the function. Java objects may be communicated via FIPA ACL using Java's object serialization [34] mechanism, which provides a means of writing a Java object as a stream of bytes, and subsequently reconstructing an object by reading a stream of bytes. This allows utility functions to be serialized and used as the content in a FIPA ACL message.

An ESN may need to assign both performance and functional utility functions simultaneously, and also specify other information regarding the transfer of observable properties and remote parameters. The `UtilityFunctionAssignment` class, illustrated in figure 4.2, is used to encode all of this information within a single Java object. When an ESN assigns a utility function to an agent, a serialized instance of the `UtilityFunctionAssignment` class is sent as a content object within a FIPA ACL message to the agent receiving the utility function assignment.

Assigned utility functions require access to the remote parameters, observable properties and the performance data maintained by an agent. The `UtilityFunction-Assignment` class therefore contains a single field to which the agent is able to assign an implementation of the `UtilityFunctionDataSource` interface. The `Utility-FunctionDataSource` interface, illustrated in figure 4.7, allows the utility function to obtain remote parameters, observable properties and performance data via the methods it defines. A LEAF agent must assign an implementation of the `UtilityFunctionDataSource` interface following the assignment of a utility function.

Local utility function assignment is initiated by an ESN when a LEAF agent joins a community. LEAF agents may be assigned multiple local utility functions from different ESNs, the results of which are aggregated. The assignment of utility

Field	Purpose
`performanceUtilityFunction`	The performance utility function being assigned to an agent.
`functionalUtilityFunction`	The functional utility function being assigned to an agent.
`observableProperties`	A list of observable properties required by the ESN from the agent.
`updateTimePeriod`	The time period with which the agent should send observable property values to the ESN. If equal to 0, properties are sent each time their values change.
`hostName & portNumber`	Enables the agent to make a connection with the ESN's server socket.
`removePerformanceUtilityFunction`	A boolean field used to indicate that the performance utility function assigned by the ESN should be removed.
`removeFunctionalUtilityFunction`	A boolean field used to indicate that the functional utility function assigned by the ESN should be removed.

Table 4.2: The fields of the `UtilityFunctionAssignment` class.
An instance of this class is used to communicate information relating to the assignment of utility functions.

`getObservableProperty(String propertyName)` Retrieves the value of the specified observable property.
`getRemoteParameter(String parameterName)` Retrieves the value of the specified remote parameter.
`getPerformanceData(String key, String[] args)` Retrieves the value of the specified performance data, where `args` may be used to specify an array of implementation dependent arguments.

Figure 4.7: The methods of the `UtilityFunctionDataSource` interface. *A utility function is provided with an implementation of the methods in this interface by the agent to which the function is assigned. Subsequently, a utility function may invoke the methods defined by this interface in order to compute local utility values.*

- `double computeFunctionalUtility()`

Computes the aggregate of all currently assigned functional utility functions.

- `double computePerformanceUtility()`

Computes the aggregate of all currently assigned performance utility functions.

- `double computeUtility()`

Computes the aggregate of performance and functional utility functions.

- `String[] getFunctionalUtilityRequiredProperties()`

Returns the observable properties required to compute the agent's functional utility functions.

- `String[] getPerformanceUtilityRequiredProperties()`

Returns the observable properties required to compute the agent's performance utility functions.

Figure 4.8: API used to access assigned utility functions

functions is handled automatically, and the developer is able to compute the result of local utility functions. The methods used to access assigned utility functions are illustrated in figure 4.8.

Utility function aggregation

If multiple utility functions are assigned from different communities, the utility value is an aggregation of the assigned utility functions. The three functions used to combine utility functions must be defined by an implementation of the `UtilityAggregateFunction` interface, an instance of which may be specified as the parameter when invoking the `setUtilityAggregateFunction` method. The `UtilityAggregateFunction` interface specifies three methods:

- `double combinePerformanceFunctions(double[] output)`: this method is used to combine the outputs of multiple functional utility functions, which are passed to the method as an array of `double` values.

- `double combineFunctionalFunctions(double[] output)`: similarly, this method is used by LEAF to combine multiple performance utility functions.
- `double combineUtilityTypes(double functionalUtility, double performanceUtility)`: this method is used by LEAF to combine performance and functional utilities into a single utility value for the agent.

If the `setUtilityAggregateFunction` is not invoked in order to specify aggregation functions, the default aggregate functions combine assigned utility functions using summation.

Transfer of observable properties and remote parameters

The transfer of observable properties and remote parameters are achieved using TCP sockets as the slower, more verbose, conversation based form of communication provided by FIPA ACL is not required for the simple one-way transfer of observable properties and remote parameters. In order to establish a socket connection between a community ESN and an agent, the ESN acts as a server, specifying a hostname and port number to which the agent is able to connect. In addition to communicating utility functions, a `UtilityFunctionAssignment` object is also able to specify how observable properties and remote parameters are transferred. This information consists of:

- The hostname and port number to which the ESN accepts a connection from the agent.
- The set of observable properties required from the agent by the ESN.
- A time period, with which an agent must broadcast the specified observable property values, or alternatively, the ESN may specify that observable properties should be broadcast each time a property value changes.

Once this information is received by the agent, it makes the socket connection with the ESN, and sends the required observable properties according to the specified scheme.

Inter-agent communication

LEAF uses FIPA ACL for inter-agent communication, including the assignment of local utility functions and the joining/leaving of LEAF communities. In order to recognise messages that relate to these activities, which should be handled by the appropriate LEAF routines rather than be dealt with by application specific code, LEAF modifies the FIPA-OS communication model. The FIPA-OS communication model involves setting an application specific listener task, which receives new conversations from other agents (see figure 4.1). In LEAF, the listener task is not application specific and is not used directly by the developer. Instead, the listener task places conversations on a queue, which the developer can then use to process incoming conversations. The reason for this is that the listener task can examine incoming messages, and deal with messages involved in utility function assignment processes and interactions with LEAF communities. All messages not relating to these activities are placed on the queue, to be dealt with by application specific code. Incoming messages can be removed from the queue by invoking the `popIncomingConversation()` method, which returns the FIPA ACL conversation associated with the message, encapsulated by an instance of the FIPA-OS `Conversation` class.

Executing tasks

LEAF tasks are executed by invoking the `newTask(LeafTask task)` method, which utilises the FIPA-OS task manager. Tasks should not be passed directly to the task manager when using LEAF, as would be the case using FIPA-OS. A task is passed to the task manager by LEAF when the `newTask(LeafTask task)` method is invoked, which is available from both LEAF agents and LEAF tasks, allowing the creation of child tasks. This modification of the way in which tasks are created is made in order to enable the LEAF toolkit to record information about the tasks executed by each agent. Results are received from tasks using the same approach as provided by FIPA-OS, which involves the invocation of `done` methods on the parent (task or main agent class) of a task, as described in [2].

Learning techniques

It is possible to integrate any learning technique with a LEAF agent, as the developer defines an agent's application specific behaviour. In order to support the construction of LEAF agents, the LEAF toolkit provides an implementation of several learning approaches, including simple neural network and reinforcement learning techniques. In addition to these techniques, the developer also has access to an expert system in the form of JESS.

Terminating a LEAF agent

Invoking the `shutDown()` method will cause a LEAF agent to deregister from the platform DF and AMS agents, and leave all LEAF communities the agent has joined.

4.4.2 LEAF tasks

The developer constructs application specific LEAF tasks by extending the `LeafTask` class. The behaviour of the task must be specified in the `startTask()` method, which is invoked by FIPA-OS task manager. The methods of the LEAF task provide most of the LEAF agent functionality described above, with the exception of registering with the platform AMS/DF agents, and obtaining FIPA ACL conversations from the incoming message queue.

4.4.3 Utility functions

Local utility functions are defined by the abstract `LocalUtilityFunction` class. The developer creates application specific utility functions for use in LEAF MAS by extending the `LocalUtilityFunction` class and implementing the `compute()` method with application specific code capable of dynamically computing the output of the utility function.

Assigned utility functions must be able to access the observable properties and remote parameters maintained by an agent, and additionally have access to various data regarding the performance of the agent. This is achieved via access to the methods declared in the `UtilityFunctionDataSource`, described in section 4.4.1.

Access to performance data, which may be used to compute performance utility functions is also provided via the data source, including:

- The length of the agent's incoming message queue. This information should help to assess the speed with which the agent performs, as having messages accumulate in this queue would indicate a slow rate of performance, whereas agents performing at an adequate speed would usually have a very small number of messages waiting on the incoming message queue.
- Information on the agents tasks, including the number of completed tasks, the number of active tasks and the average completion time of completed tasks by task type.
- The number of messages sent by the agent.
- The number of active tasks belonging to the agent.
- The number of completed tasks.
- The CPU and memory usage of the agent.

Additionally, domain specific performance data may also be available depending upon the platform used to deploy an agent. When a local utility function is assigned, the LEAF agent makes an implementation of the `UtilityFunctionDataSource` interface available to the function via the `source` field in order to link the function to the agent, allowing the application specific code residing in the local utility function's `compute()` method to then retrieve this information. For example, in order for a utility function to retrieve the value of an observable property named "items_sold", the following code would be used:

```
String items_sold = source.getObservableProperty("items_sold")
```

This approach can be used to construct local utility functions, as illustrated in figure 4.9

```
public double compute() {
      int revenue = Integer.parseInt(
                     source.getObservableProperty("revenue") );
      int expenditure = Integer.parseInt(
                     source.getObservableProperty("expenditure") );
      return revenue - expenditure;.
}
```

Figure 4.9: The `compute()` method of an application specific utility function. *The figure illustrates a possible local functional utility function for a hypothetical market based agent, the goal of which is to make profit in its environment. The agent maintains two observable properties, "expenditure", and "revenue". These properties must be updated via the agent's application specific code. An assigned utility function has access to these properties, and the example function extracts these properties then subtracts the agent's expenditure from its revenue to determine the agent's local functional utility. All observable properties and remote parameters are represented as strings, and* ***`Integer.parseInt`*** *is used to convert a string to an integer.*

4.4.4 LEAF ESN

Implementation of the ESN is based on the `ESN` class, which extends the `FIPAOSAgent` class. The MAS developer must extend the `ESN` class, and define a `startBehaviour()` method in order to create an application specific ESN. An overview of the ESN implementation is now given based around the areas of functionality the LEAF ESN possesses.

Registration with the FIPA agent platform

When the `ESN` class constructor is invoked, the ESN registers itself with the DF and AMS agents automatically. The ESN class constructor requires 4 parameters:

- The location of the FIPA-OS platform profile.
- The name of the community that the ESN represents.
- The type of the community that the ESN represents.
- The owner of the ESN.

The ESN registers with the DF as a LEAF community ESN, allowing LEAF agents to discover it by searching the platform DF. Each community ESN has a community type, which is used to identify certain application specific properties an agent can expect of the community. There are two fields of the FIPA service description that are used to identify LEAF communities, the service type field, which is assigned the value "leaf.community.community_agent", and the service name field, which is made equal to the community ESN's community type. The ESN possesses only one service description, and is not permitted to change this once deployed.

Supporting a LEAF community

Once an ESN is registered with the platform DF as a community agent, LEAF agents may discover the community that the ESN represents and join it. Once a LEAF agent is a member, it may then leave the community at any time. The process of joining/leaving a community involves FIPA ACL communication and is initiated by the LEAF agent. Two abstract methods must be implemented, the `joiningLeafAgent(LeafAgent agent)` and `leavingLeafAgent(LeafAgent agent)` methods, which are used to inform the ESN when an agent joins or leaves the community. Application specific code provided by the developer must specify how the ESN handles the events generated by joining/leaving LEAF agents.

Utility function assignment

The API utilised to assign utility functions by the ESN is illustrated in figure 4.10. Once a utility function is assigned, the ESN is responsible for updating any remote parameters required by the function. The API used to accomplish this is illustrated in figure 4.11

4.5 Developing MAS using the LEAF toolkit

The development process involved when constructing a MAS using LEAF is now described step by step. The implementation of a LEAF MAS is examined in further detail in appendix D using a basic scenario as an illustrative example.

Step 1: Identify the MAS objectives decompose into communities

The first step the MAS developer takes in developing a MAS using LEAF is to identify what the objectives of the MAS are, and how these objectives can be achieved by the application specific agents the developer intends to deploy. The global objectives of the system are decomposed into one or more communities, around which agents belonging to a specific community will work towards the objectives of that community.

Step 2: Design utility functions

Each community will assign local utility functions to the agents that join it, and the maximisation of these functions should result in the behaviour of the agent being of use to the community. The observable properties and remote parameters required to

- `void assignPerformanceUtilityFunction(LeafAgent agent, LocalUtilityFunction performanceUtilityFunction, String[] subscribedProperties, long updateTimePeriod)`

Assigns a performance utility function to the specified LEAF agent. An array of observable properties is specified, which will be updated with the time period specified. If the time period is equal to 0, the observable properties will be updated whenever their values change.

- `void assignFunctionalUtilityFunction( LeafAgent agent, LocalUtilityFunction functionalUtilityFunction, String[] subscribedProperties, long updateTimePeriod )`

Assigns a functional utility function to the specified agent.

- `void assignUtilityFunctions( LeafAgent agent, LocalUtilityFunction functionalUtilityFunction, LocalUtilityFunction performanceUtilityFunction, String[] subscribedProperties, long updateTimePeriod )`

Assigns both performance and functional utility functions simultaneously.

- `void removeFunctionalUtilityFunction( LeafAgent agent )`

Removes a functional utility function from the specified LEAF agent.

- `void removePerformanceUtilityFunction( LeafAgent agent )`

Removes a performance utility function from the specified LEAF agent.

- `void removeUtilityFunctions( LeafAgent agent )`

Removes both performance and functional utility functions from the specified LEAF agent.

Figure 4.10: API used to assign local utility functions

- `void pushParameter(LeafAgent agent, String parameter, String value)`

Sends the parameter with the given parameter name and parameter value to the specified LEAF agent.

- `void pushParameters(LeafAgent agent, String[] parameters, String[] values)`

Sends an array of parameters and their associated values to the specified LEAF agent.

- `void changeSubscribedProperties(LeafAgent agent, String[] subscribedProperties, long updateTimePeriod)`

This method can be used to change the scheme used to update observable properties.

- `void getUpdatedProperties(LeafAgent agent)`

Retrieves any updated parameters from the socket connection with the given LEAF agent.

- `String getRemoteProperty(LeafAgent agent, String property)`

Retrieves the value of an observable property updated by a specified LEAF agent.

- `void disconnectAgent(LeafAgent agent)`

Terminates a socket connection with a specified agent. This will result in the agent removing any utility functions assigned by the ESN.

Figure 4.11: API used to retrieve observable properties and update remote parameters

compute local utility functions must be identified, and a scheme for updating these properties/parameters must be designed.

Step 3: Implement LEAF agents, utility functions and community ESNs

The application specific agents should now be constructed, including the learning techniques they utilise in order to maximise their local utility functions. Each agent must maintain the set of observable properties required by ESNs to compute local utility functions. The joining of a community is initiated by the LEAF agent, and each agent's scheme for discovering and joining communities must be specified within the agent's behaviour.

A set of local utility functions and ESNs must now be implemented. The ESN's application specific code must handle the transfer of observable properties and remote parameters throughout its community. Each ESN represents a community, and must be able to map a joining LEAF agent to its corresponding functional and performance utility functions by inspecting its LEAF agent type. The ESN must also maintain a global utility function, which quantifies the performance of the community according to the objectives that the community is pursuing. The global utility function may be parameterised by the observable properties of any LEAF agents belonging to the community.

Step 4: Deploy the system

In order to deploy the LEAF agents and LEAF ESN, the FIPA-OS platform must be started, as described in [2]. The LEAF agents and ESNs may then be deployed.

4.6 Conclusion

The implementation of the LEAF toolkit as an extension to the open source FIPA-OS implementation of the FIPA specifications has been described. LEAF is also to be made available as open source software, and will be available for download via the World Wide Web at the following location:

`http://www.cs.cardiff.ac.uk/User/S.J.Lynden/leaf`

CHAPTER 5

Agent learning in LEAF communities

5.1 Introduction

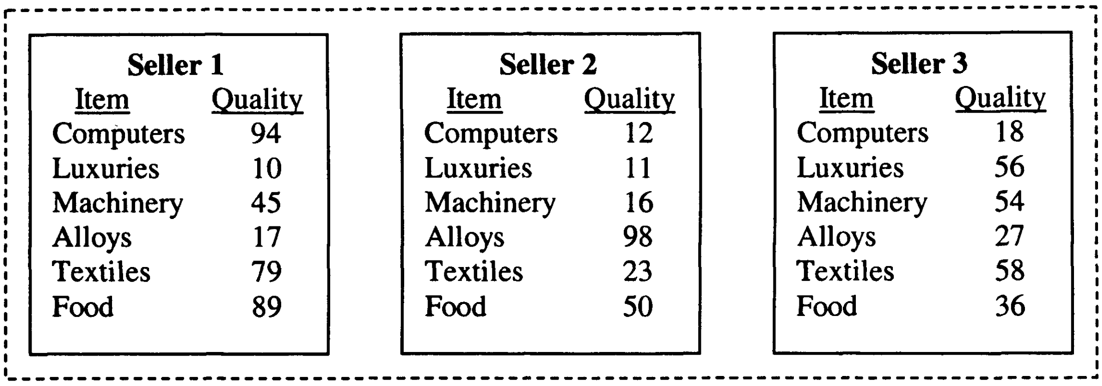
In this chapter, LEAF is applied to a coordination problem involving buyer agents residing in distributed markets, where the purchases made by the buyer agents are coordinated to achieve a globally defined objective. Here, utility is used by agents as a reward signal to guide their learning behaviours. Buyer agents learn to coordinate their actions using independent learning RL techniques within a cooperative single-stage repeated game. The translation of this problem into a market based scenario is motivated by the desire to demonstrate how LEAF communities may be deployed in a practical setting with relevance to e-Commerce [42] applications. In particular, the application shows how each agent is able to join a LEAF community and learn to maximise a local utility function that influences the agent's adaptive behaviour so that a contribution to the objectives of the community is made, therefore demonstrating a key aspect of the benefits provided by LEAF in supporting the development of MAS.

The focus in this chapter is on agent learning using functional utility, and performance utility is the focus of chapters 6 and 7. Although the problem outlined in this chapter can be solved by other approaches, such as simulated annealing [50], the purpose of this investigation is to explore utility function assignment as a means of multiagent coordination, where agents retain their autonomy under the assumption that agents learn to maximise local functional utilities. Although the ESN in our approach has a centralised role, we believe that the approach is more suitable for agent based applications than completely centralised optimisation techniques such as simulated annealing. Centralised optimisation techniques involve complete control over the actions performed by the agents in the system, whereas the approach we take involves the centralised assignment of utility functions and the distribution of observable properties only, which allows agents to learn individually and is consistent with the requirement of autonomy for agents. The fact that agents learn individually means that they are able to continue to learn even when the performance of the centralised entity (ESN) degrades, which is investigated in section 5.6.1.

A number of assumptions are made about agent behaviour in this application, however, the core problem is one of optimising the behaviours of a set of purchasing agents to maximise a globally defined objective and potential benefits of this approach exist with respect to more complex applications. The market based system used in this investigation is now described, followed by the presentation of results assessing the performance of LEAF in this application.

5.2 System architecture

The system consists of a number of buyer agents, each of which exists in a separate market (see figure 5.1). Agents interact within the same market only, and are unaware of any information regarding the states of other markets. The items sold by seller agents fall into 6 categories, and a quality metric is associated with each individual item, which can take any integer value in the range 1-100. Quality metrics apply universally, meaning that all agents will agree that a specific item has the same quality value. Buyer agents are able to make one purchase per day, where a day is a system-wide standard unit of time, of which all agents are aware. The assumption is made that all items cost the same arbitrary amount, which is chosen to be one dollar. The LEAF community consists of the buyer agents deployed in the distributed markets, as illustrated in figure 5.2, and buyer agents are coordinated



Seller 1

Item	Quality
Computers	94
Luxuries	10
Machinery	45
Alloys	17
Textiles	79
Food	89

Seller 2

Item	Quality
Computers	12
Luxuries	11
Machinery	16
Alloys	98
Textiles	23
Food	50

Seller 3

Item	Quality
Computers	18
Luxuries	56
Machinery	54
Alloys	27
Textiles	58
Food	36

Figure 5.1: An individual market

Each buyer agent is deployed in a market with a number of seller agents (3 in this illustrative example). Each seller agent offers six item categories, and the quality of the items held by each seller agent vary. The quality values of the items offered by seller agents are generated randomly when a seller agent is deployed, and the seller agents do not make these values available to buyer agents. Buyer agents must purchase items from seller agents in order to discover the quality of the items they sell.

using utility assignment, where utility functions are assigned by a LEAF "buyer" community ESN. Initially, the application is investigated in scenarios where 6 markets exist, equalling the number of item categories. The community objective in this setting is to maximise the combined value of the purchases made by the buyer agents, while ensuring that at least one purchase of each item category is made each day. Scenarios in which the number of markets is increased are investigated in section 5.5.

5.2.1 Seller agents

The behaviour of seller agents is very simple, and entirely reactive. A transaction between a buyer and seller agent is initiated by the buyer agent using a FIPA-request performative. When a seller agent receives a request, it returns an item of the requested category. The assumption is made that both money and items can be transferred using FIPA ACL, and once that a buyer agent receives a reply from a seller agent, it can assess the quality of the returned item. It is also assumed that

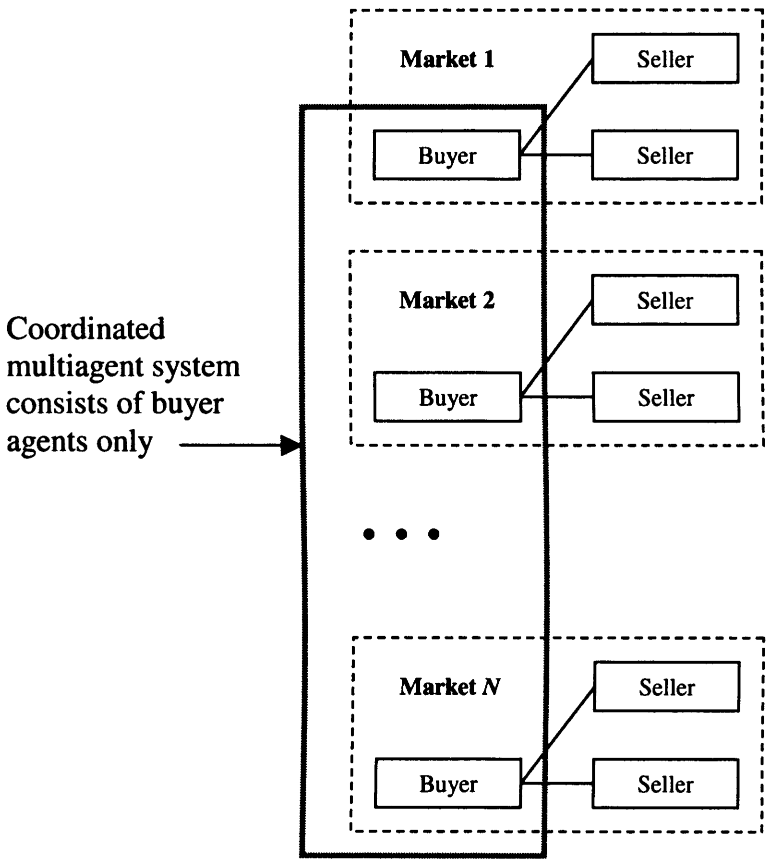


Figure 5.2: The LEAF buyer community.

The buyer community consists of the collective set of buyer agents existing in the distributed markets.

by making a request for an item, the buyer agent has also transferred one dollar to the seller agent. Seller agents have a large number of items, and are always able to comply when a buyer agents desires to purchase an item. The quality of each item sold by a seller agent is decided as a seller agent is deployed, when for each item category, a random quality value is generated and each subsequent item of the category sold will be of this quality. The generated quality values are not made public, and buyer agents must interact with seller agents by purchasing items in order to discover the quality of the items a seller is offering.

5.2.2 Buyer agents

Buyer agents repeatedly purchase items from the seller agents in their market using RL to optimise their local utility functions, which are described in section 5.3. When a buyer agent is deployed, it will attempt to join a LEAF buyer community, and will subsequently by assigned a local functional utility function by the ESN representing the community. The buyer agent's behaviour then consists of selecting one purchase to make each day. When deciding which transaction to make, a buyer agent is faced with two decisions, firstly, which item category to purchase, and secondly, which seller agent to purchase the item from. The buyer agent's approach to making these decisions is now described.

Learning which item to purchase

Buyer agents use a simple RL algorithm to decide which item category to purchase. It should be noted that the RL techniques used in the course of applying LEAF to this application domain are intended to be unsophisticated, as the investigation aims to assess the effects of utility function assignment, and not the performance of different RL techniques in this domain. More sophisticated RL techniques may be available that can provide better results and converge faster than the techniques used here, however, this comparison is not made.

A set of action-values are maintained by each agent, where actions correspond to the item categories the agent can purchase - food, textiles, machinery, computers, luxuries or alloys. Buyer agents make one purchase each day, and compute their local utility function at the end of a day. The action-value, $AV(i)$, for item i is the average value of local utility following the purchase of item i. Buyer agents decide which item categories to purchase using these action-values, and select an item using Boltzmann action selection [61] to control the exploration/exploitation of the possible actions that can be selected. Using this approach, the probability, $p(i)$ of selecting the purchase of item category i on any day is:

$$p(i) = \frac{e^{AV(i)/T}}{\sum_{j \in A} e^{AV(i)/T}} \tag{5.1}$$

where T is a temperature value in the range [0-1] used to control exploration. High values of T, close to 1, will result in almost random selection of item categories, whereas values close to 0 will result in the selection of the item category associated with the greatest action-value. Each buyer agent is initially deployed with a high value of T, and this value is steadily decreased until close to zero.

Learning which seller to interact with

Once a buyer agent has decided which item category to purchase, the agent assumes that it is desirable to maximise the quality of the item purchased. Due to the fact that the quality of items belonging to a specific category from a specific seller are static, the buyer agent is able to simply record the quality received in previous transactions, and use this information to choose a seller to buy from. Buyer agents explore their market by choosing seller agents randomly at first, but then exploration is restricted using a Boltzmann controller synchronised with the temperature parameter described previously (used in the RL process to select item categories). When the exploration rate is very small, a buyer agent purchasing an item category will choose to buy from the seller that has returned the highest quality of that item in previous interactions. In this way, buyer agents will converge to select both an item category to purchase and a seller agent to purchase from simultaneously.

5.3 Utility function assignment

5.3.1 Global utility function

As previously discussed, the global objective of the buyer community is to maximise the combined value of the purchases made by the buyer agents, whilst adhering to the constraint that at least one purchase of each item category is made each day. This objective is chosen due to the fact that it cannot be solved locally by agents without knowledge of the purchases made by other agents in the community. Although agents can choose to maximise the quality of the items they purchase, without precise knowledge of the item categories all other buyer agents in the community intend to purchase on each day, there is no guarantee that certain item categories will not be purchased more then once (or not at all), which does not achieve the globally defined objective. The approach presented here does not allow any communication between buyer agents, and agents use independent learning techniques. The global objective is represented as a global utility function, which is computed by the buyer community ESN each day as follows:

$$\begin{aligned} G_d^F \;=\; & MaxQ_d^{Food} + MaxQ_d^{Textiles} + \\ & MaxQ_d^{Luxuries} + MaxQ_d^{Computers} + \\ & MaxQ_d^{Machinery} + MaxQ_d^{Alloys} \end{aligned} \tag{5.2}$$

where $MaxQ_d^i$ is the maximum quality of item i purchased in any individual purchase made on day d by any of the buyer agents. The global utility function therefore only takes into account purchases if they represent the maximum quality value of all items of that category purchased on a specific day. As the number of item categories equals the number of markets, maximising the global utility function will result in each category of item being purchased once each day.

5.3.2 Local utility functions

Buyer agent local utility functions are based on COIN WLU functions. WLU functions have been previously discussed in section 2.2.2 or this thesis. The local utility function for an agent μ, is given by:

$$L^F_{\mu,d} = G^F_d - G^F_d(P_{\mu,d} = 0) \tag{5.3}$$

where $P_{\mu,d}$ is the quality of the item purchased on day d by agent μ. $G^F_d(P_{\mu,d} = 0)$ is the global utility function for day d computed without considering the purchases made by agent μ. Essentially, the local utility is a WLU function with a "null action" clamping parameter [67].

5.3.3 Observable properties and remote parameters

Each agent has one observable property, named "last purchase", which records the last item category the agent purchased. The community ESN is informed of each agent's last purchase property every time the value of this property changes. In order to compute local utility functions, agents are informed each day of the highest quality value of each item category purchased on that day. This information is encoded into one remote parameter, named "maximum quality array". The maximum quality array allows local utility to be obtained (as specified by the local utility function defined above) by subtracting the result of a global utility function computed with the agent's last purchase property set to its most recent value from the result of a global utility function computed with this property set to zero.

5.3.4 Summary of system behaviour

The system's behaviour can be divided into two parts, the initialisation phase, which follows the deployment of the buyer agents, and the iterative phase, in which buyer agents repeatedly purchase items from seller agents.

Initialisation phase

It is assumed that seller agents, and the buyer community ESN already exist prior to the deployment of the buyer agents. When deployed, buyer agents join the LEAF buyer community, and subsequently each buyer agent is assigned a local functional utility function. During the assignment of this function, the ESN registers as a listener to each agent's last purchase observable property. Once a buyer agent has joined the buyer community it waits for the start of the next day before beginning the iterative phase of its behaviour.

Iterative phase

Each day, the following events occur sequentially:

1. Each buyer agent chooses an item to purchase, and a seller agent to purchase from.
2. The selected transaction is made.
3. Following the completion of a transaction with a seller agent, each buyer agent updates its last purchase observable property, and the ESN is automatically notified of the new value of this property.
4. The ESN distributes the maximum quality array to each buyer agent.
5. Buyer agents wait until the end of the day before computing their local functional utility function. The resulting local utility value is used to update the RL action-values as described in section 5.2.2.

5.4 Results and observations

Experiments were performed with two seller agents in each market. Buyer agents are deployed with an initial learning temperature value of 1, and this value is reduced by 0.01 each day, until it is equal to 0.01 after 99 days. The length of each system day was made equal to 5 minutes for all the experiments performed. Global utility was measured as agent learning converged. Convergence is reached as the point at which each agent repeatedly makes the same purchase, and does not deviate from this action. The performance of agent learning using LEAF was compared to the optimal global utility obtainable with complete system information (the internal states of all seller agents in all markets) and centralised control (calculating the optimal global utility for the buyer community is possible using an exhaustive search algorithm when the number of markets is small). 20 experiments were performed, each with randomly generated seller agent item quality values.

The results of these experiments are included in appendix C section 1. An example of the explorative learning process performed by agents is illustrated in figure 5.3. The mean global utility value obtained using LEAF was 491.5 with 95% confidence interval bounds of 472.16 and 510.84. The mean optimal global utility in the 20 experiments was 532.1 with 95% confidence interval bounds of 521.78 and 542.42. The benefit of the sub-optimal results achieved using LEAF is that they were achieved without centralised control, using utility function assignment and observable property updates only.

5.5 Application scalability

Application scalability is now investigated with respect to an increase in the number of buyer agents within the LEAF buyer community. An increase in the number of buyer agents requires adjustments to the global utility function described previously, and the resulting new global utility function is now described.

5.5.1 Global utility function

Previously, the number of buyer agents equalled the number of item categories, and the global objective of the system was to maximise the combined purchase quality of all items purchased, with the constraint that each item category should be purchased only once each day. Without increasing the number of item categories, the number of markets is now increased, with one buyer agent in each market as before. Buyer agents therefore outnumber item categories, so optimising the global utility function used previously would result in a situation where some buyer agents are redundant, that is, their purchases would never contribute to the global utility function. The problem with this scenario is that the local utility function of a redundant agent would not be affected by the purchase made by the agent, which would prevent the agent from being able to learn from the purchase made. The result of this would be that the optimisation of local utility by redundant agents would be impossible and

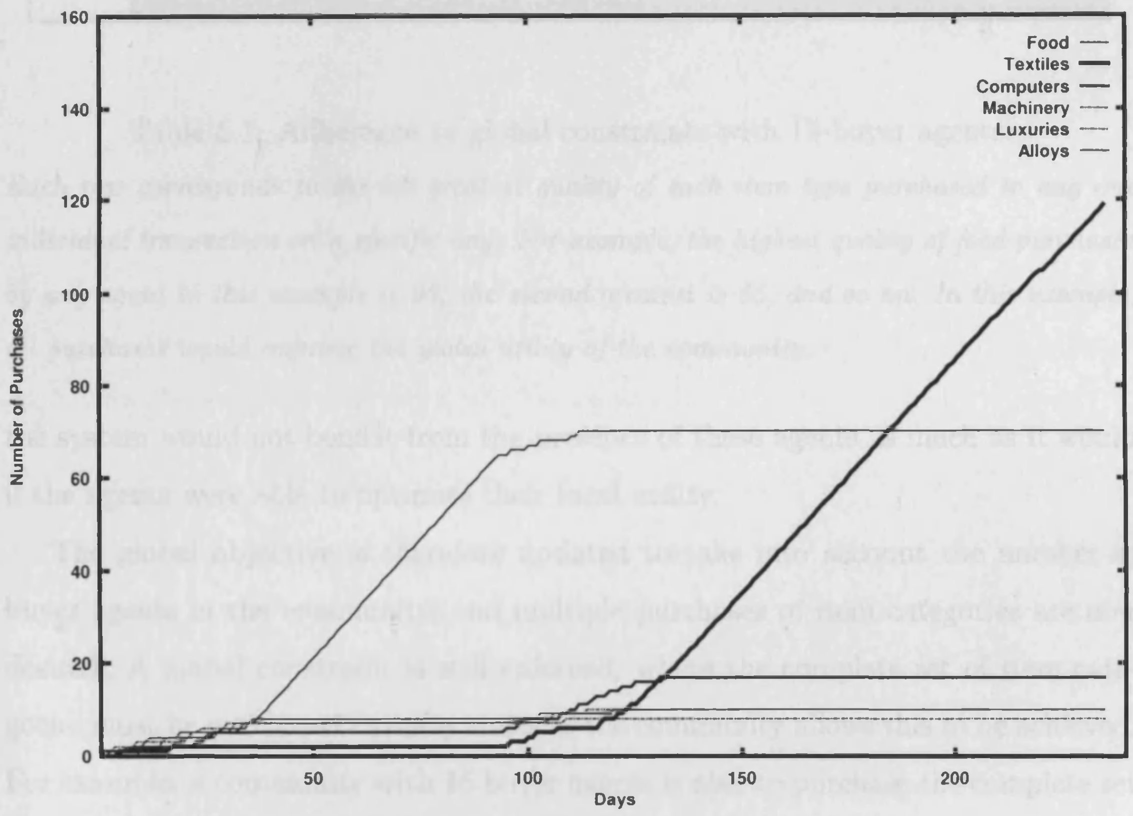


Figure 5.3: Transactions made by a buyer agent in a buyer community with 6 buyer agents in 6 markets, with 2 seller agents per market.

The number of purchases of each item category is plotted against the number of days elapsed since the agent was deployed. The trends shown in this figure were common to all buyer agents, specifically that agents initially explore their possible actions (purchases), and converge after approximately 100-150 days, after which one purchase (in the case illustrated, textiles) is repeatedly made.

	Food	Textiles	Luxuries	Computers	Machinery	Alloys
1	93	98	55	62	34	78
2	55	65	15	52	31	55
3	10	65	▨	▨	▨	22
4	▨	▨	▨	▨	▨	▨

Table 5.1: Adherence to global constraints with 15 buyer agents.
Each row corresponds to the ith greatest quality of each item type purchased in any one individual transaction on a specific day. For example, the highest quality of food purchased by any agent in this example is 93, the second greatest is 55, and so on. In this example, all purchases would improve the global utility of the community.

the system would not benefit from the presence of these agents as much as it would if the agents were able to optimise their local utility.

The global objective is therefore updated to take into account the number of buyer agents in the community, and multiple purchases of item categories are now desired. A global constraint is still enforced, where the complete set of item categories must be purchased as many times as the community allows this to be achieved. For example, a community with 15 buyer agents is able to purchase the complete set of item categories twice. In this situation, it is undesirable to purchase any item less than twice, as illustrated in table 5.2. Once the complete set of item categories has been purchased twice, the objective is for 3 additional purchases to be made, with no multiple purchases of the same item within these remaining purchases, as illustrated in table 5.1. Similarly, if the community contained 16 buyer agents, the objective would be to purchase every item twice and make 4 additional purchases. For 18 markets the objective would be to purchase every item 3 times. For 24 markets, every item should be purchased 4 times, and so on. Within these constraints, the objective of maximising the combined quality of all purchases continues to apply.

	Food	Textiles	Luxuries	Computers	Machinery	Alloys
1	93	98	55	62	34	78
2	55	65	15	52	▨	55
3	10	65	▨	▨	▨	22
4	8	▨	▨	▨	▨	▨

Table 5.2: Non-adherence to global constraints with 15 buyer agents.

Each row corresponds to the ith greatest quality of each item type purchased in any one individual transaction on a specific day. Non-adherence is illustrated by the community's failure to purchase the item category machinery twice. In this example, the 4th highest purchase of food, and the 3rd highest purchases of food, textiles and alloys would make no improvements to the global utility of the community.

Computation of the global utility function is specified by the following algorithm:

```
1     Initialise an integer variable, G.
2     Add to G the combined quality of the highest value
   purchases in each item category.
3     FOR I = 2 to (N/6)
4      Add to G the combined quality of the Ith highest
       value purchases in each item category.
5     END FOR
6     IF the complete set of item categories has been purchased
   N/6 times, add the remaining highest quality values of
   each item category to G.
7.    Return G as the global utility value.
```

The global utility function is designed to encourage a uniform distribution of item categories to be purchased by the buyer community.

5.5.2 Local utility functions

WLU functions are once again assigned to the buyer agents, and local utility is again obtained for each agent by subtracting the global utility as computed without the

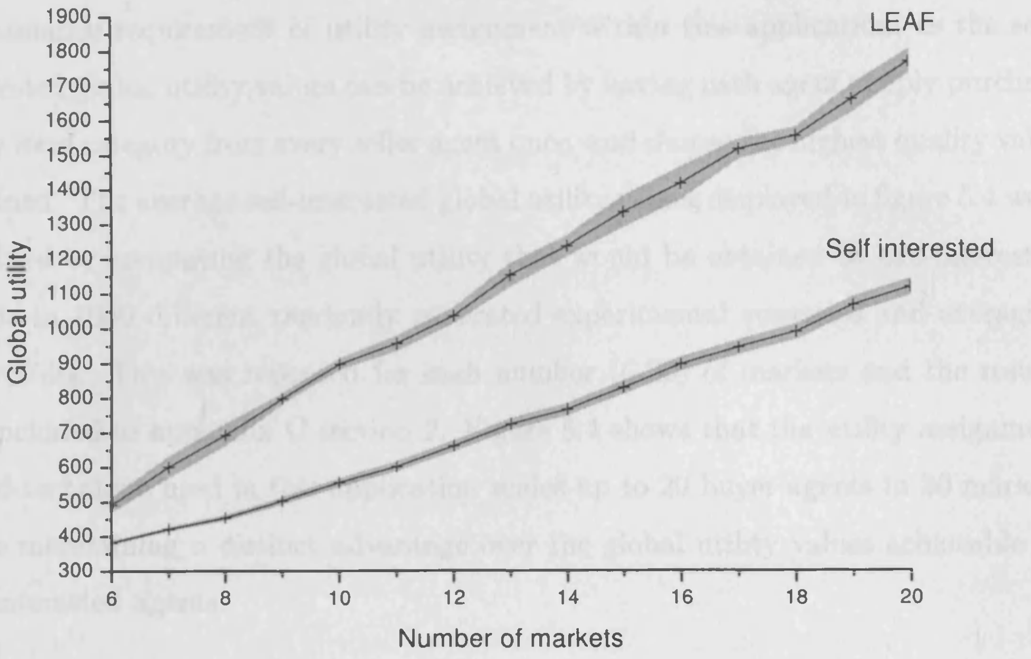


Figure 5.4: Global utility with 6-20 markets.

The mean global utility of the 20 experiments is compared to the mean global utility obtainable by self-interested agents. The shaded areas represent the 95% confidence intervals around each mean.

agent's contribution (purchase) from the global utility computed with the agent's contribution (purchase).

5.5.3 Results and observations

Experiments were performed with two seller agents in each market. As was the case in the previous experiments, buyer agents are deployed with an initial learning temperature value of 1, and this value is reduced by 0.01 each day, until it is equal to 0.01 after 99 days. The length of each day was made equal to 5 minutes. Experiments were performed with 6-20 markets (with one buyer agent per market), and in each case 20 experiments were performed, each with randomly generated seller agent item quality values. Results are included in appendix C section 2. The average global utility at convergence is illustrated in figure 5.4, which compares the performance of the system to the average global utility obtained if every buyer agent optimises its purchasing decisions locally based on item quality only. This establishes a minimum performance requirement of utility assignment within this application, as the self-interested global utility values can be achieved by having each agent simply purchase every item category from every seller agent once, and choose the highest quality value obtained. The average self-interested global utility values displayed in figure 5.4 were obtained by computing the global utility that would be obtained by self-interested agents in 1000 different randomly generated experimental scenarios and averaging the results. This was repeated for each number (6-20) of markets and the results are included in appendix C section 2. Figure 5.4 shows that the utility assignment based technique used in this application scales up to 20 buyer agents in 20 markets while maintaining a distinct advantage over the global utility values achievable by self-interested agents.

5.6 Investigating the effects of remote parameter updates on agent learning

The learning processes of the buyer agents in this application are supported by the maximum value array remote parameters, which allows an agent's local utility function to be computed. This remote parameter is updated each day in order to allow an accurate WLU value to be computed for each buyer agent. The effects of altering the update frequency of this remote parameter are now investigated, motivated by the desire to discover whether the current remote parameter update frequency is required for agents to learn adequately, or whether agent learning converges with less frequent remote parameter updates. Investigating a community's ability to continue learning with respect to different remote parameter updates has implications in evaluating the LEAF framework. The ESN possibly provides a single point-of-failure for a LEAF community, and therefore it is worthwhile to investigate the performance of LEAF agents when the functionality provided by the ESN degrades, in order to assess to what degree the ever-present availability of the ESN is critical to a LEAF community's performance in this domain.

5.6.1 Decreasing the reliability of remote parameter updates

The motivation for this investigation is to assess the ability of a LEAF buyer community to continue learning in the event of the unpredictable loss of remote parameter updates from the ESN. Previously, the ESN updated each agent's maximum value array once each day. A stochastic element to the update of remote parameters is now introduced, where instead of always being updated, remote parameters are updated on any given day with a probability U.

5.6.2 Results and observations

Experiments were performed with 2 seller agents per market, 1 buyer agent per market, a system day length equal to 5 minutes, and an initial learning temperature of 1, reduced by 0.01 each day. The following values of U were used: 0.1, 0.05 and 0.01. Note that U was equal to 1 in the experiments described in the previous sections. For each value of U, the system was executed 20 times for each different number of markets (6-20), and the results of each 20 executions averaged to obtain a global utility at convergence. Results are illustrated in figures 5.5, 5.6 and 5.7, which compare the performance of each experiment category to the global utility obtainable by self-interested agents, described previously in section 5.5. The results of these experiments are included in appendix C section 3.

It can be seen that utility function assignment achieves better results than self-interested agents when U is equal to 0.1. Therefore, the community is still capable of learning when utility functions are computed on average 10 times for each single remote parameter update. Values of U equal to 0.05 and 0.01 do not show any significant improvement over self-interested agents. Additionally, figure 5.8 shows that the number of days needed for the learning process to converge does not increase significantly for values of U equal to 0.1 or 0.05, and convergence only becomes erratic when U is made equal to 0.01. These results indicate that within reasonably stable environments, agents in LEAF communities may be able to compute utility functions at a faster rate than remote parameters are updated and still learn successfully.

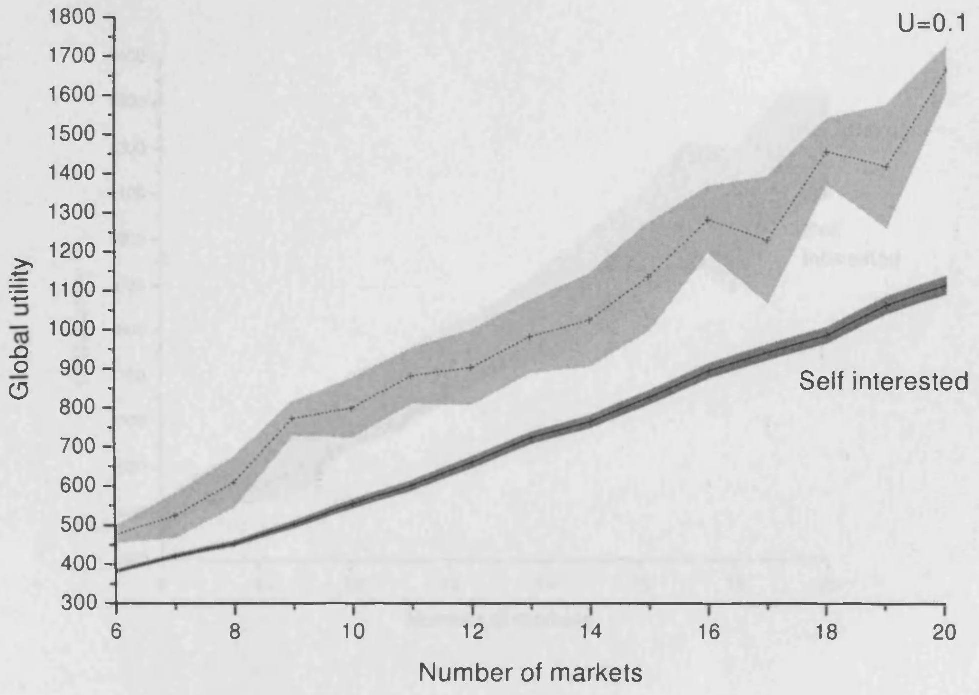


Figure 5.5: Global utility with U equal to 0.1.

The mean global utility of the 20 experiments is compared to the mean global utility obtainable by self-interested agents. The shaded areas represent the 95% confidence intervals around each mean.

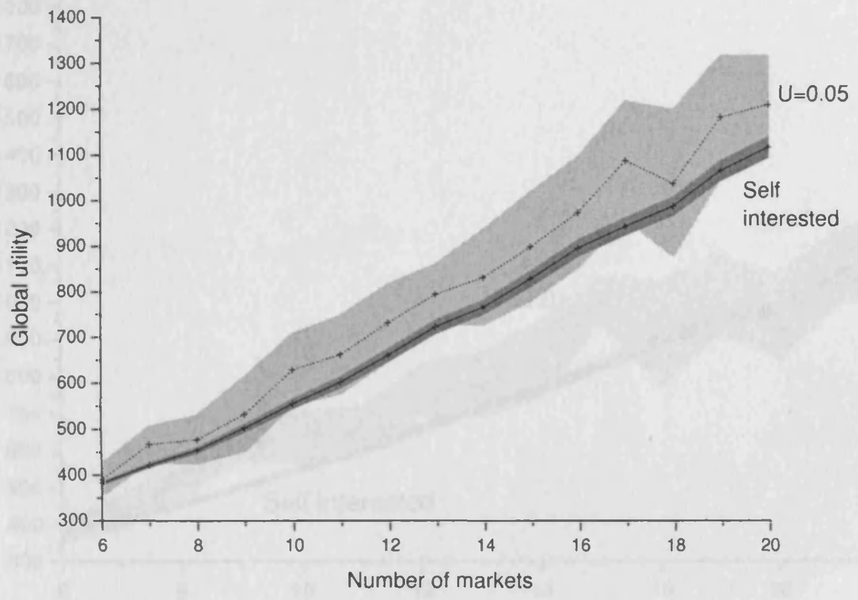


Figure 5.6: Global utility with U equal to 0.05.

The mean global utility of the 20 experiments is compared to the mean global utility obtainable by self-interested agents. The shaded areas represent the 95% confidence intervals around each mean.

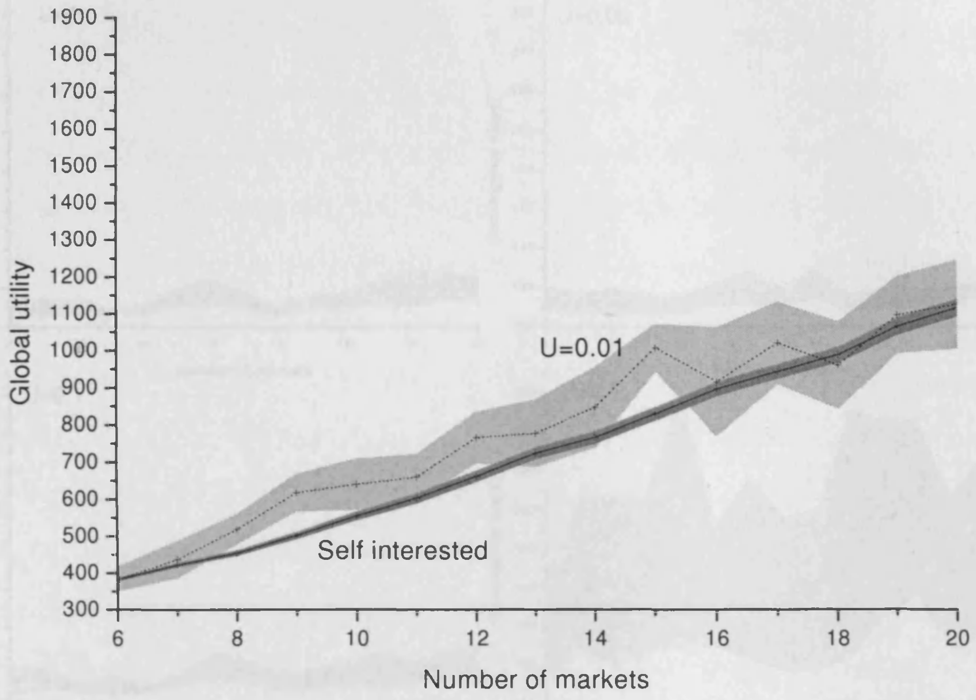


Figure 5.7: Global utility with U equal to 0.01.

The mean global utility of the 20 experiments is compared to the mean global utility obtainable by self-interested agents. The shaded areas represent the 95% confidence intervals around each mean.

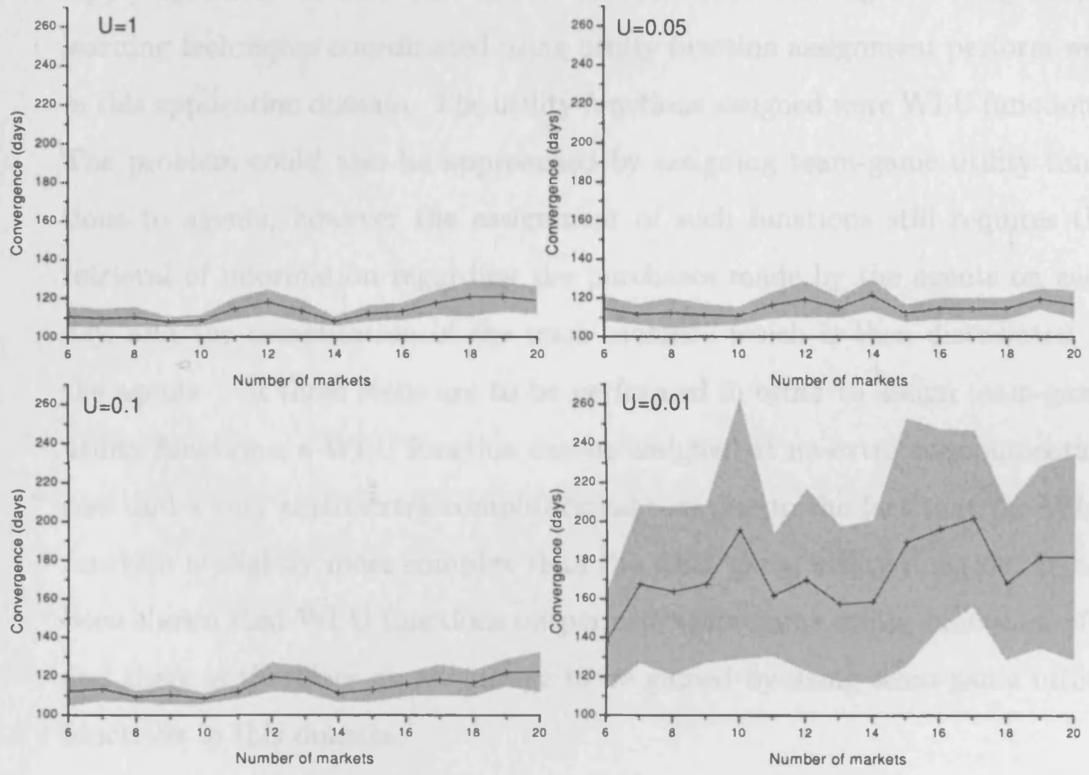


Figure 5.8: Learning convergence of buyer agents.

The mean number of days needed for learning to converge for different values of U. The shaded areas represent 95% confidence intervals around each mean.

5.7 Conclusion

This investigation has served as a demonstrator application illustrating the benefit of utility function assignment using LEAF. The benefits of LEAF have been explored utilising an application based on the coordination of market based buyer agents. The principle lessons learnt from this chapter can be summarised as follows:

- Applying LEAF in this domain has demonstrated that agents using simple learning techniques coordinated using utility function assignment perform well in this application domain. The utility functions assigned were WLU functions. The problem could also be approached by assigning team-game utility functions to agents, however the assignment of such functions still requires the retrieval of information regarding the purchases made by the agents on each day, and the computation of the team measure which is then distributed to the agents [1]. If these steps are to be performed in order to assign team-game utility functions, a WLU function can be assigned at no extra communication cost and a very small extra computational cost due to the fact that the WLU function is slightly more complex than the team-game utility function. It has been shown that WLU functions outperform team-game utility functions [72] and there is therefore no advantage to be gained by using team-game utility functions in this domain.

- Utilising LEAF in this domain has been shown to provide scalability in terms of its ability to effectively coordinate increasing numbers of agents.

- The performance of agent learning was investigated under conditions in which the frequency of remote parameters updates was decreased, demonstrating that it is possible for the buyer agents to continue to learn under these conditions.

[1] Alternatively the information regarding the purchases made by the agents may be distributed to each agent, which then computes the team utility measure.

CHAPTER 6

Performance utility in LEAF

6.1 Introduction

In this chapter, the benefits of LEAF are investigated using an application based on distributed computational task processing, of relevance to the emerging field of research involving Grid computing. Results are presented that illustrate, within this context, the potential benefits of the LEAF approach which defines utility as two separate types.

6.2 Agent-based computational Grids

The currently active field of research referred to as Grid computing involves an attempt to provide users with universal access to resources on a large scale via an infrastructure referred to as the "Grid". Foster et al. [27] identify the "real and specific problem" that underlies the realisation of a Grid computing infrastructure as "coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations". Computational Grids concentrate specifically on sharing computational and data resources, which may provide various services, including the ability to execute computationally intensive applications, access to data storage resources, and access to physical instruments such as sensors, telescopes etc. Facilitating the sharing of such resources on a large scale involves a management problem to which agent based computing is particularly applicable. In [49], Overeinder et al. identify the following Grid computing related problems that can benefit from the intelligent, autonomous behaviours provided by agents:

- Resource allocation and scheduling.
- Brokering services.
- Monitoring and diagnostic services.
- Workload management and collaboration frameworks.

The benefit of multiagent RL using the COIN framework has already been demonstrated within the context of job scheduling in computational Grid based environments [65]. In this chapter, we aim to demonstrate how the approach developed in LEAF is also of benefit in this area. In particular, our aims are firstly, to show how performance and functional utility functions can be defined for an agent based computational Grid community, secondly, to investigate the relationship between performance and functional utilities, and finally, to evaluate the significance of the results obtained, and discuss the potential benefits of the approach.

6.3 The multiagent system

We utilise a setting based on a computational Grid where the system consists of a community of resource agents. The utilised Grid model has many similarities with those presented by Tumer and Lawson [65], and by Rana et al. [54].

6.3.1 Computational resource agents

The system consists of a set of LEAF agents, each of which acts as a resource belonging to the computational Grid. Resources are heterogeneous, and are categorised by one of the following types:

- Computational (C): the resource allows computationally intensive applications to be submitted and executed.

- Visualisation (V): the resource provides a visualisation based service.
- Storage (S): the resource facilitates the storage of data.
- Instrumentation (I): the resource provides access to a physical instrument.

The four resource categories provide an abstraction of resources that may exist in a computational Grid. Each agent accepts computational tasks (CTs), from a set of users associated with the agent. CTs are categorised by a task type (C,V,S or I), and a CT can only be executed by a resource of the same type. CTs may enter the system at any point, and may not be executable by the agent to which the CT has initially been submitted. This requires the CTs to be forwarded to other agents, which subsequently execute them, and return the results.

Agent behaviour

A resource agent's reactive behaviour is triggered by the following events:

1. *A CT is submitted to the agent by a user:* A resource agent accepting a CT from a user processes the CT by executing it locally if the agent's resource is of the same type as the CT. If a CT is not executable locally, it is sent to a resource agent capable of executing the task.
2. *A CT is submitted to the agent by another resource agent:* Following this, the CT is executed, and the results returned to the resource agent from which the CT was submitted.

Learning behaviours are not implemented by resource agents, therefore, when a resource agent is unable to execute a CT, it selects another resource agent (capable of executing the CT) randomly. A possible extension to this mechanism is to incorporate a learning algorithm into this selection process, where agents aim to submit CTs to agents that have performed well when utilised previously. A number of simplifying assumptions are made concerning resource agents and CTs:

- CTs and resources are simulated. When a CT is executed by a resource agent, the resource agent is required to execute a Java code segment, following the

```
SET sum = 0
FOR i = 1 to 1000
        sum = sum + (randomly generated number)
        WAIT for 1 millisecond
END FOR
RETURN sum
```

Figure 6.1: CT algorithm pseudocode.
A Java algorithm is used to simulate a CT, and the execution time of this algorithm will vary depending on the agent's speed of performance. The purpose of this algorithm is to place a computational burden on the resource agent executing it.

algorithm illustrated in figure 6.1. This algorithm simluates the execution of a CT by requiring an agent to spend an amount of time "busy" and expend system resources on executing the code.

- Resource agents are able to identify the capabilities of other agents, including which CT types they are able to execute, by examining the FIPA service descriptions possessed by other resource agents.

- A CT of a given type can be executed by any resource agent of the same type. The properties of the resource, such as available memory and processor configurations, do not prevent CTs from being executed.

- Although a physical limit to the number of CTs that can be executed at any one time will exist for agents due to the fact that CPU cycles and available memory are limited, there is no predefined limit to the number of CTs that can be executed simultaneously by a resource agent. This differs from the queuing system utilised in [65].

- Agents do not refuse to execute CTs that have been submitted either locally by the user or remotely by another agent.

- All communication between agents, including the submission of CTs for execution and the transfer of execution results, is handled by FIPA ACL. In a

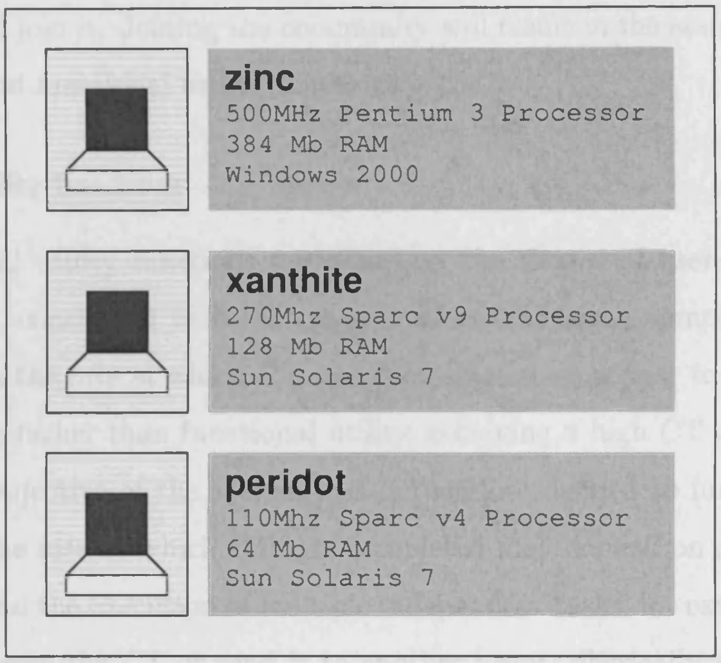


Figure 6.2: Host platforms for agents in the computational community.

There are three different hosts used in the experiments, zinc, xanthite, and peridot. The system configurations of these hosts are displayed. In all experiments, the local host's resources are utilised by resource agents only, and are not shared with other applications. The FIPA platform agents reside on a different host. Each host has a 10Mbs connection to the same Ethernet network.

realistic Grid environment, tasks would submitted using established protocols such as the GridFTP protocol provided by the Globus project [26].

- Users submit CTs of each type at a steady rate. The users of each agent are simulated using automated processes which submit new CTs to their agents with a specified rate.

6.3.2 Computational communities and utility functions

The resource agents in the system form a LEAF *computational community*, the objective of which is to satisfy user requirements by completing submitted tasks. Following deployment, all agents will attempt to discover a LEAF computational community and join it. Joining the community will result in the assignment of local performance and functional utility functions.

Functional utility functions

Local functional utility functions are based on the degree of user satisfaction an agent provides, considered to be the rate at which the agent completes submitted CTs. Although the rate at which CTs are completed may appear to be more suited to performance rather than functional utility, achieving a high CT completion rate is the overall objective of the system and is therefore defined as functional utility. Additionally, the rate at which CTs are completed may depend on cooperation between agents and the execution of multiple independent tasks, for example, deciding whether to process the CT or send it to another agent, physically sending the CT and executing the CT. It is these individual tasks that should be considered as parameters for performance utility functions. In further developments of the MAS, it would be possible to add learning behaviours to agents, where agents could learn to schedule the execution of CTs or choose to cooperate with specific agents based on the effects of these actions upon local functional utility.

Specifically, the local functional utility of agent μ, L_μ^F, is given by:

$$L_\mu^F = \frac{j_\mu}{k_\mu} \tag{6.1}$$

where j_μ is the number of CTs processed by agent μ (when a CT is submitted to another resource agent for execution due to the fact that the CT is not executable locally, the resource agent which submits the CT is considered to have processed the CT). k_μ is the number of CTs submitted by the users associated with agent μ. L_μ^F is computed following the completion of a CT by an agent. The global functional utility, G^F, is the average local functional utilities of agents belonging to the community.

Performance utility functions

Local performance utility functions are based on the time taken by agents to execute CTs. When an agent executes a CT, a *CT execution task* is created to handle the execution of the CT, and due to the fact that each CT follows an identical algorithm, the CT execution task provides an indication of the speed of performance of a given agent in this application domain. The LEAF toolkit automatically compiles statistics concerning average task execution times, and this data is used to compute performance utility functions. The local performance utility of agent μ, L_μ^P, is given by:

$$L_\mu^P = -\left[(\sum_{i=1}^{n} r^i)/n) \right] \tag{6.2}$$

where n is the number of CT execution tasks performed by agent μ, and r^i is the time taken (in seconds) to complete CT execution task i. Global performance utility, G^P, is the sum of local performance utilities.

6.4 Experimentation

Experiments were performed using three platforms, *zinc*, *xanthite*, and *peridot*. The hardware/software properties of these platforms are illustrated in figure 6.2. The computational community contains one agent of each service type, therefore enabling the community to satisfy any submitted CT. The significant events that take place in the system's lifetime are illustrated in figure 6.3. Once the resource agents begin to process CTs on behalf of the users, the system's temporal evolution is parameterised by a set of discrete time steps, $t \in \{0, 1, ...\}$, where one minute elapses between each time step. Experiments are divided into six categories, characterised by the platforms used to host the agents. These categories are as follows:

1. All agents reside on zinc. *(all-Z)*

2. All agents reside on xanthite. *(all-X)*

3. All agents reside on peridot. *(all-P)*

4. 2 agents reside on zinc, and 2 agents reside on peridot. *(mixed-ZP)*

5. 2 agents reside on zinc, and 2 agents reside on xanthite. *(mixed-ZX)*

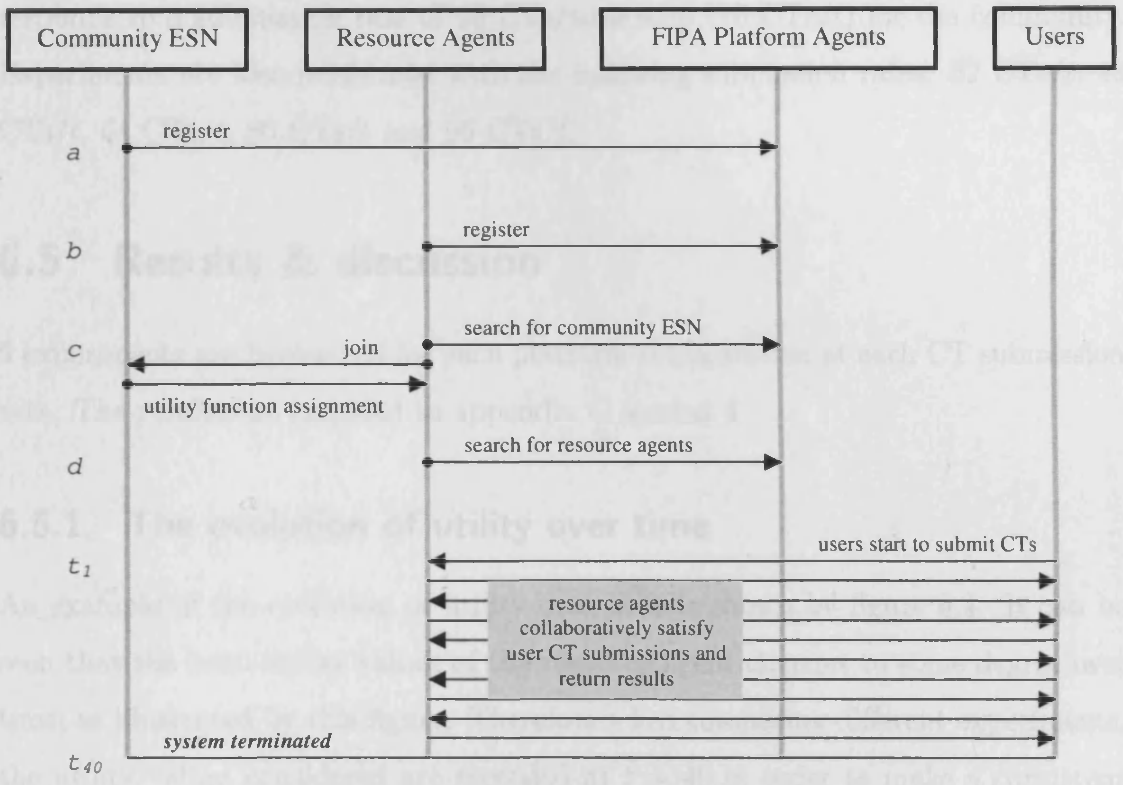

Community ESN
Resource Agents
FIPA Platform Agents
Users
a
register
b
register
c
join
search for community ESN
utility function assignment
d
search for resource agents
t_1
users start to submit CTs
resource agents collaboratively satisfy user CT submissions and return results
system terminated
t_{40}

Figure 6.3: Computational Grid based application behaviour.

The figure illustrates the behaviours of the components involved in the system. a - d correspond to the significant events that take place during the initialisation of the system so that agent's have joined the computational community, have been assigned utility functions, and are aware of the existence of the other resource agents in the MAS. The main behaviour of the system takes place in between t_1 and t_{40}, when resource agents process CTs submitted by the users, which involves the reactive behaviour described section 6.3.1.

6. 2 agents reside on peridot, and 2 agents reside on xanthite. *(mixed-PX)*

In each of these categories, six experiments are performed with different CT submission rates. An equal number of CTs of each type (C,V,S and I) are submitted in each case, starting at one CT of each type per agent per time step, which corresponds to a submission rate of 16 CTs/time step (16 CTs/t) for the community. Experiments are also performed with the following submission rates: 32 CTs/t, 48 CTs/t, 64 CTs/t, 80 CTs/t and 96 CTs/t.

6.5 Results & discussion

5 experiments are performed for each platform configuration at each CT submission rate. The results are included in appendix C section 4.

6.5.1 The evolution of utility over time

An example of the evolution of utility over time is shown by figure 6.4. It can be seen that the local utility values of the resource agent changes to some degree over time, as illustrated by this figure. Therefore when comparing different experiments, the utility values considered are recorded at $t = 40$ in order to make a consistent comparison of utilities by sampling values at the same time relative to the start of the system in each experiment. Global utility values, as the summation of local utilities, followed similar trends to the local utilities illustrated in figure 6.4.

6.6 The effects of platform configurations and submission rates on global utilities

6.6.1 Performance utility

It can be seen from figure 6.5 that the performance utility of certain platform configurations was clearly higher than others. The platform configuration *all-Z* obtained the highest performance utility by some distance, followed by the configurations

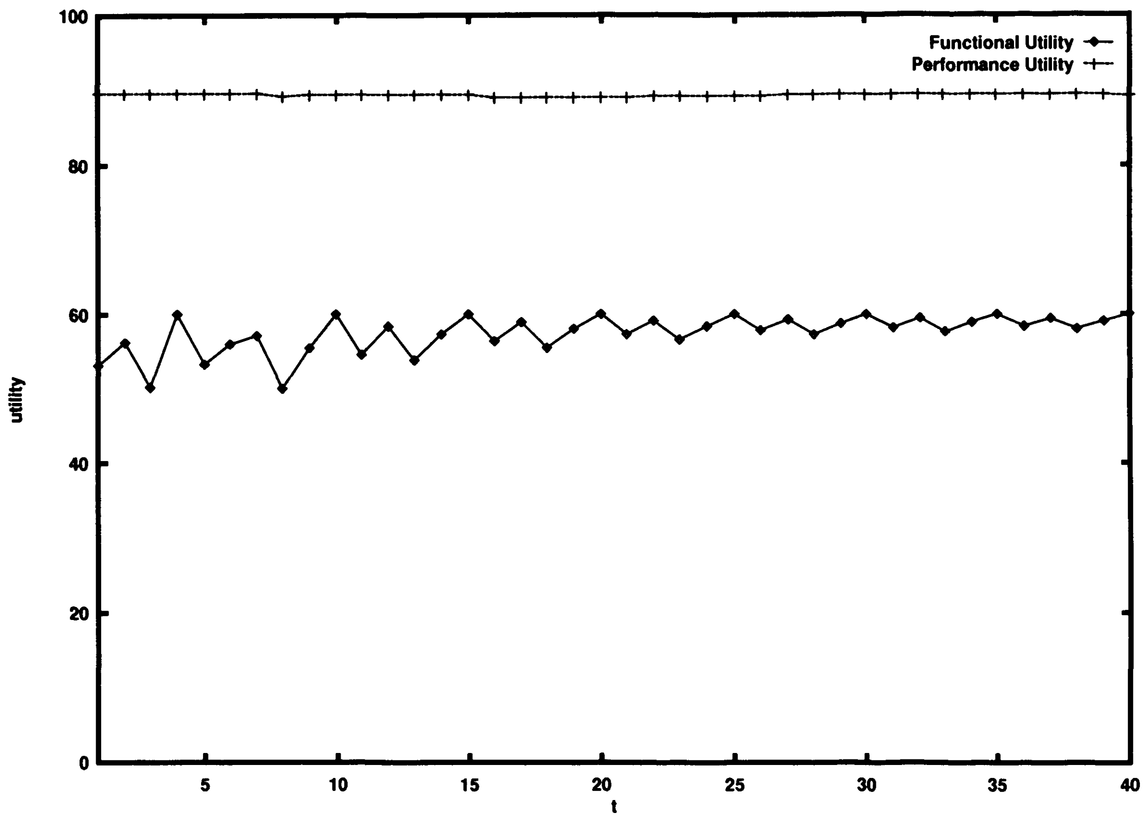


Figure 6.4: Local utility evolution over time for a resource agent belonging to the platform configuration *all-Z*.

CT submission rate equals 24 CTs/t. $t = 0$ corresponds to the time step at which users begin to submit CTs. Functional utility oscillates due to the fact that it is based on the CT execution rate of the agent, and this can be calculated more accurately as the number of CTs executed by the agent increases.

mixed-ZX and *mixed-ZP*. All of these platform configurations contained the host *zinc*, which had the highest CPU speed and available RAM of the four hosts. The local performance utility of agents residing on *zinc* was higher than that of all other hosts. This characteristic was expected, as was the low performance utility of the host with the lowest RAM/CPU speed, *peridot*, which participates in the platform configurations *all-P*, *mixed-ZP* and *mixed-PX*

It can be seen that increasing the CT submission rate did not have a significant effect on the global performance utility, with the exception of the low performing *all-P* and *mixed-PX* configurations, which experienced a decrease in performance utility as the CT submission rate was increased. This data has uncovered an important characteristic in the community's behaviour under these conditions, which would be of use in decisions concerning the deployment of agents. Specifically, performance utility functions have discovered CT submission rates at which the ability of the community to execute CTs degrades, which subsequently has an effect on the functional utility of the community. Significantly, the impact on functional utility may be observed across the community, rather than being limited to agents with low performance utilities. This is due to the fact that the functionality of each agent depends, to varying degrees, on the performance utilities of other agents.

6.6.2 Functional utility

The global functional utility of the community decreases in all platform configurations as CT submission rates are increased. This is due to the fact that functional utility is based on the ability of the community to satisfy user CT submissions, and this ability decreases as CT submission rates are increased.

It can be seen that the ordering of the platform configurations with respect to utilities are maintained, that is, high performance utility values are associated with high functional utility values. This shows that performance utility, which is based on the agent's ability to execute CTs quickly, has an effect on functional utility, which is based on the community's ability to process the CTs submitted by the users.

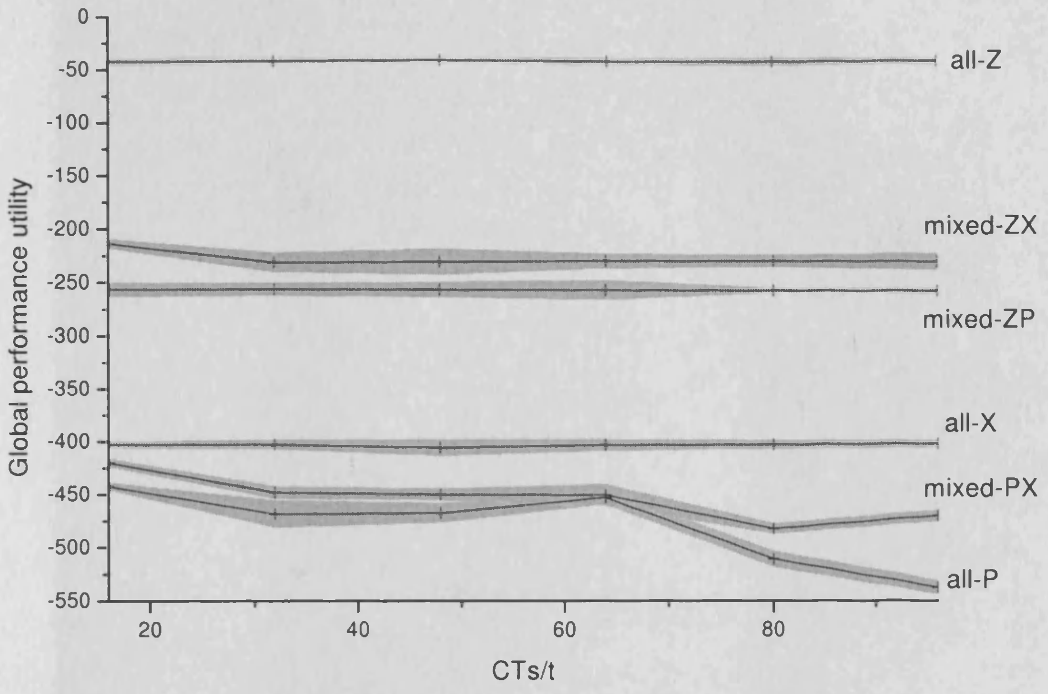


Figure 6.5: Global performance utility of the computational community. *The mean performance utility of each experiment category is plotted against the CT submission rate. The shaded areas represent the 95% confidence intervals around each mean.*

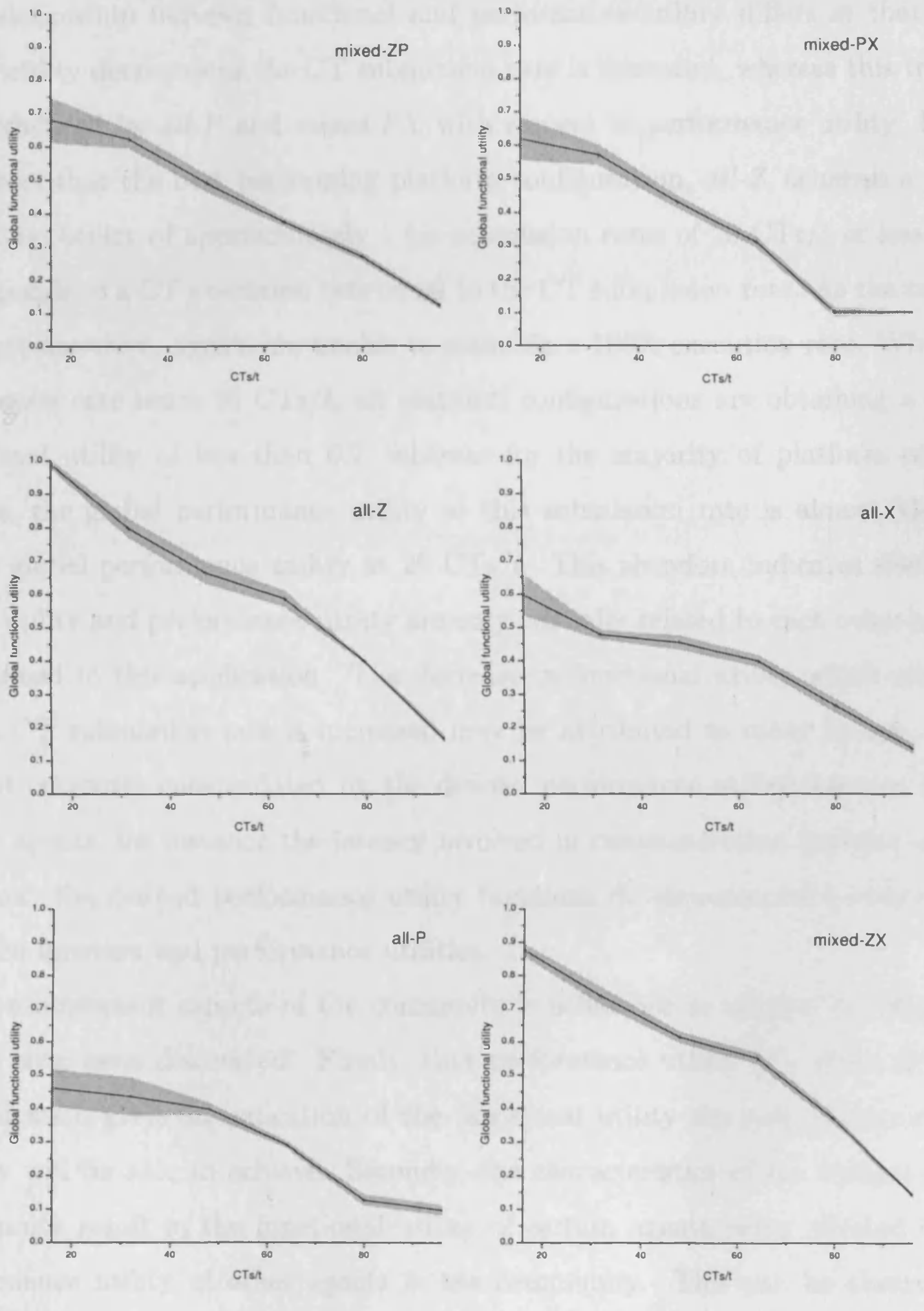


Figure 6.6: Global functional utility of the computational community.

The mean global functional utility of each experiment category is plotted against the CT submission rate. The shaded areas represent the 95% confidence intervals around each mean.

6.6.3 Relationship between functional and performance utility

The relationship between functional and performance utility differs in that functional utility decreases as the CT submission rate is increased, whereas this trend is only exhibited by *all-P* and *mixed-PX* with respect to performance utility. Figure 6.6 shows that the best performing platform configuration, *all-Z*, achieves a global functional utility of approximately 1 for submission rates of 20 CTs/t or less. This corresponds to a CT execution rate equal to the CT submission rate. As the submission rate increases, agents are unable to maintain a 100% execution rate. When the submission rate nears 96 CTs/t, all platform configurations are obtaining a global functional utility of less than 0.2, whereas for the majority of platform configurations, the global performance utility at this submission rate is almost identical to the global performance utility at 20 CTs/t. This therefore indicates that functional utility and performance utility are only partially related to each other as they are defined in this application. The decrease in functional utility which occurred as the CT submission rate is increased may be attributed to other factors, which are not currently encapsulated by the defined performance utility function for resource agents, for instance the latency involved in communication between agents. However, the defined performance utility functions do demonstrate a relationship between function and performance utilities.

Two important aspects of the community's behaviour in relation to this application have been discovered. Firstly, that performance utility of a given platform configuration gives an indication of the functional utility the computational community will be able to achieve. Secondly, the characteristics of the computational community result in the functional utility of certain agents being affected by the performance utility of other agents in the community. This can be observed by comparing the average performance utility of agents residing on *zinc*, plotted for the three platform configurations involving agents deployed on *zinc* (*all-Z*, *mixed-ZP* and *mixed-ZX*), illustrated in figure 6.7, to their average functional utility, illustrated in figure 6.8. Whereas average functional utility for *zinc* agents is higher in the platform configuration *all-Z* than the two others, performance utility show no

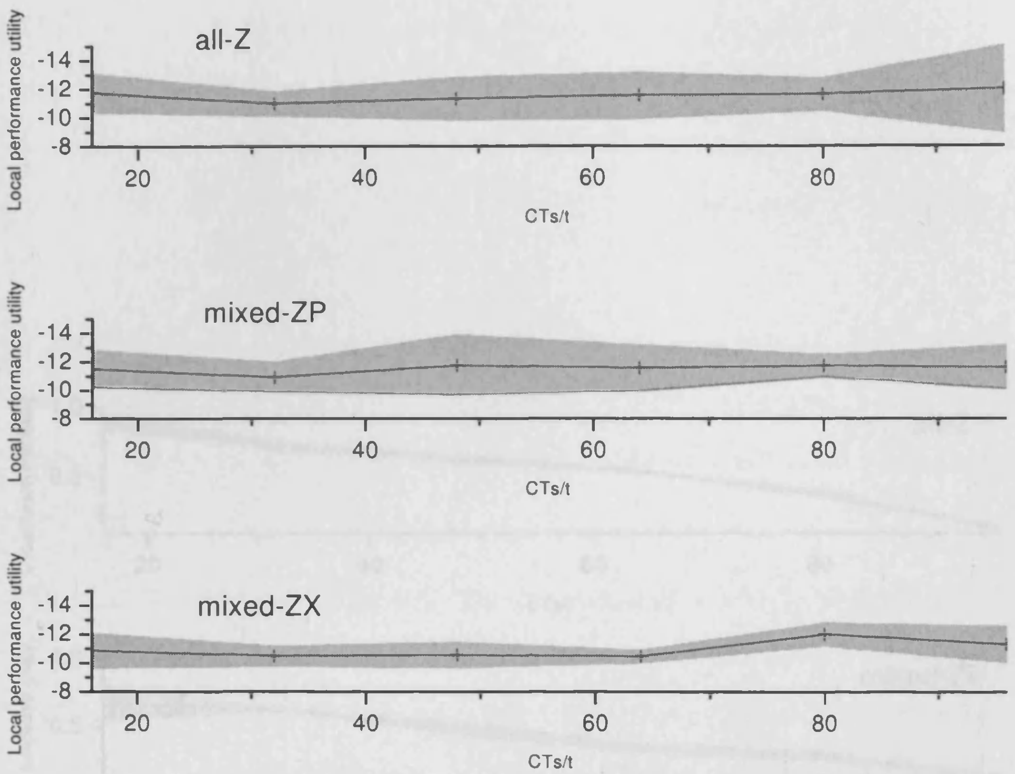


Figure 6.7: Local performance utility of *zinc* agents in the computational community. *The mean local performance utility of agents residing on* zinc *is plotted against the CT submission rate in three categories: all-Z, mixed-ZP, and mixed-ZX. The shaded areas represent the 95% confidence intervals around each mean.*

significant difference. The functional utility of agents hosted on *zinc* is affected due to the fact this utility is dependent on the ability of other agents in the system to execute CTs.

6.7 Conclusion

This application demonstrates how performance and functional utility functions can be defined for an application based on a computational Grid. In the experiments performed, properties have been discovered that are of use in the analysis of MAS behaviour with respect to the application, and this data is of use in making both online and offline decisions about the development and deployment of agents. The MAS designer may consider the following points:

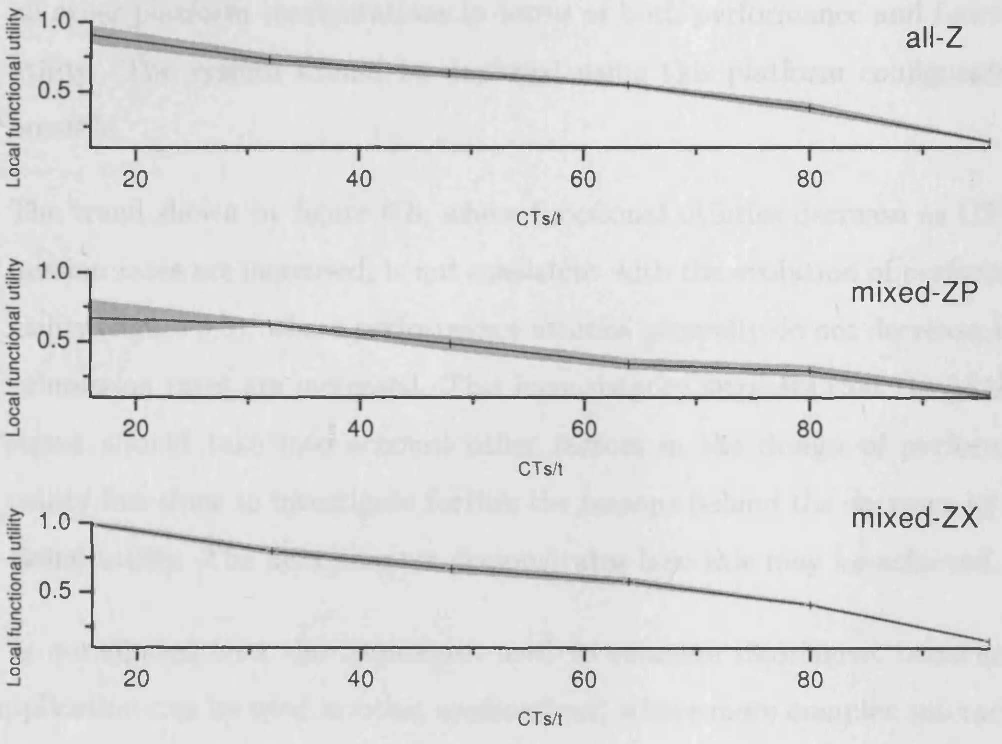


Figure 6.8: Local functional utility of *zinc* agents in the computational community. *The mean local functional utility of agents residing on* zinc *is plotted against the CT submission rate in three categories: all-Z, mixed-ZP, and mixed-ZX. The shaded areas represent the 95% confidence intervals around each mean.*

- From figure 6.5, it can be seen that the performance utility of the host configuration *all-P* starts to decline at CT submission rates of 32 CTs/t or greater. If the system should be deployed with the ability to cope with submission rates of 32 CTs/t or greater, the use of the platform configuration *all-P* should be avoided if at all possible.

- Figures 6.6 and 6.5 show that the platform configuration *all-Z* outperforms all other platform configurations in terms of both performance and functional utility. The system should be deployed using this platform configuration if possible.

- The trend shown by figure 6.6, where functional utilities decrease as CT submission rates are increased, is not consistent with the evolution of performance utility (figure 6.5), where performance utilities generally do not decrease as CT submission rates are increased. This inconsistency suggests that the MAS designer should take into account other factors in the design of performance utility functions to investigate further the reasons behind the decrease in functional utility. The next chapter demonstrates how this may be achieved.

It is anticipated that the techniques used to examine multiagent behaviour in this application can be used in other applications, where more complex interactions and behaviours exist.

CHAPTER 7

Scalability

7.1 Introduction

In this chapter, the scalability of the computational Grid based application presented in the previous chapter is analysed. This investigation focuses on observing the behaviours of agents as the number of agents in the system is increased. The results of this investigation are significant due to the fact that they indicate whether the LEAF approach is beneficial when developing large-scale MAS, and in particular, our aim is to discover the trends exhibited by performance and functional utilities as the number of agents are increased. Additionally, the aim is to gain a perspective on how many agents can function together under the following conditions:

- Multiple communities exist, therefore demonstrating a de-compositional approach to the utility function assignment process.
- Agents share system resources on various hosts.
- Agents belong to a single FIPA-OS agent platform.

The ability to define multiple communities is a key approach supported by LEAF to facilitate scalability, and involves the definition of a separate global utility function for each community, which in this case consists of four global utility functions, each of which is based around the execution of a specific CT type (C,V,S, or I).

7.2 LEAF scalability

With respect to Internet-scale applications of agent based technology, a large system is "deemed to consist of hundreds of agents, maybe a thousand, but not millions" [69]. LEAF is intended to be able to function on this scale, however, an individual LEAF community (centred around a single ESN) is not. It is envisaged that if the LEAF approach is to be implemented on a large scale, individual communities will involve not more than hundreds of agents, where each community is based around common objectives, or possibly a geographical location/organisation.

The investigation described in this chapter focuses on the support provided by LEAF when agent systems are scaled upwards in terms of the number of agents involved in a MAS. The scalability of agent coordination techniques, which in LEAF are based on COIN, is not the focus here due to the fact that scalability in this context has already been explored [72, 67, 74, 73, 71]. The aspects of scalability of interest in this investigation are the effects on performance and functional utilities as the MAS is deployed on larger scales. Of particular interest are answering the following questions:

- How do performance and functional utilities change as agents compete for the same system resources?
- Are performance and functional utilities useful in understanding the behaviors of agents/MAS with respect to scalability?

The areas identified above are impacted by the efficiency of FIPA-OS, however, we do not undertake an evaluation of the performance of FIPA-OS in general, and the results obtained and conclusions presented here apply to LEAF only, which is implemented using FIPA-OS. Appendix B illustrates some scalability issues with respect to performance utility, while [19] discusses scalability issues within the context of FIPA compliant systems and FIPA-OS in particular.

The computational Grid based application was chosen to investigate scalability as this application involves multiple interactive relationships between agents, resulting in the regular exchange of FIPA ACL messages. Additionally, the application involves the assignment of performance and functional utility functions, and the transfer of observable properties (the local utilities of agents are transferred in order to compute global utilities), therefore encapsulating the key features of the LEAF framework.

7.3 Decomposing system objectives into multiple communities

A key approach to scalability in LEAF is that of enabling multiple communities to exist, where each community is centered around a particular set of objectives. The application presented in the previous chapter is altered as follows:

1. Four CT type specific communities replace the computational community. There are therefore computational (C), visualisation (V), storage (S), and instrumentation (I) communities. Each community is independent in the sense that is not affected by the maximisation of the global utilities of other communities. More precisely, there are no conflicts of interest between communities, for example, the execution of a CT of type C does not inhibit the execution of type V CTs, due to the fact that resources of type C are able to execute type C CTs only (resource agents are not capable of executing more than one CT type). The implications of scenarios in which conflicts between communities may exist are discussed in chapters 8 and 9.

2. Previously, in the event of a resource agent being unable to execute a computational task (CT), resulting in the delegation of this task to another agent capable of executing it, the agent sending the CT is considered to have processed the CT, and it's local utility is updated accordingly. This is now altered so that the resource agent to which the CT is delegated is considered to have processed the CT.

3. A new global performance utility function, described in section 7.4 is introduced.

The first alteration is made in order to demonstrate the use of multiple communities within LEAF. The second alteration is necessary in order to compute accurate global utility functions for these communities, so that, for instance, a type C community is not associated with the execution of CTs of types V, S, and I.

7.4 Global performance utility

In the previous chapter, it was observed that functional utility was related to more than just the speed of performance of resource agents. It was hypothesised that the functional utility of the system was being affected by the latency involved in sending CTs over FIPA ACL. In order to investigate this hypothesis, a new global performance utility function is introduced, which is based on the time taken between the sending and receiving of CTs by agents. Global performance utility, G^P is given by:

$$G^P = -\left[(\sum_{i=1}^{n} t^i)/n \right] \tag{7.1}$$

where n is the number of requests for remote CT executions made by the agents in the system, and t^i is the time taken (in milliseconds) for the ith CT execution request to reach it's receiver once sent. The performance utility therefore reflects the average time taken for an agent in the system to send a CT over FIPA ACL, and the higher the value of G^P, the faster agents are able to transfer CTs.

7.5 Experimentation

Experiments were performed on 8 different hosts, each of which has a 10Mbs connection to the same Ethernet network and the following properties:

- *Operating system:* Sun Solaris 7
- *Processor:* 360 MHz Sparc v9 Processor
- *Memory:* 128 MB RAM

Number of agents residing on each host	Number of agents in each community	Total number of agents in the system
2	4	16
3	6	24
4	8	32
5	10	40
6	12	48
7	14	56
8	16	64
9	18	72
10	20	80

Table 7.1: Scalability experiments.

Experiments performed in order to investigate scalability using the computational Grid based application.

- *Java Version:* Version 1.3.0

In all experiments, each agent was assigned a single simulated user providing a CT submission rate of 12 CTs/t. Initially, two resource agents of each type (C,V,S,I) were deployed, with one agent deployed on each host. Subsequent experiments were then performed, increasing the number of agents residing on each host by one in each experiment (see table 7.1) from 2 to 10 agents. It was decided to perform experiments which ranged from deploying 2 to 10 agent on each host due to the fact that each agent is deployed using a separate Java Virtual Machine (JVM). Each JVM uses a considerable amount of memory, and more agents could have been deployed if multiple agents had been executed on the same JVM. Multiple agents were not deployed on the same JVM so that agents were executed using completely separate independent processes, which is how we generally anticipate agent systems to be deployed. FIPA platform agents and LEAF ESNs were deployed on a separate host from the LEAF agents (they did not compete for system resources with the resource agents).

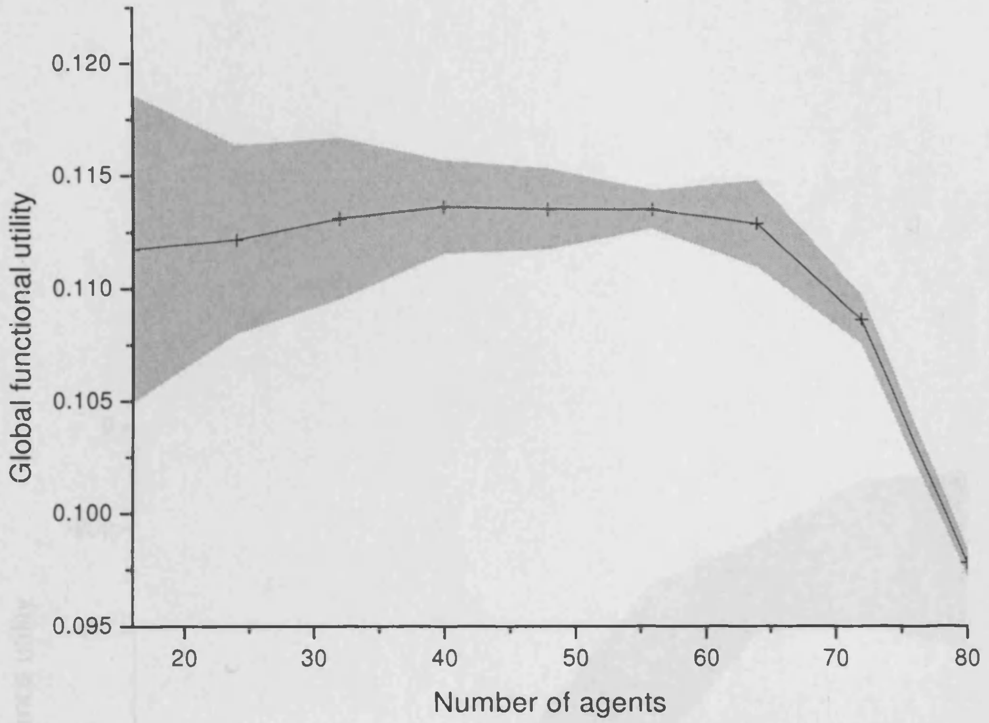


Figure 7.1: Global functional utility of resource agents.

The mean global functional utility at t=40 is plotted against the number of agents in the system. The shaded area represents the 95% confidence interval around the mean.

7.6 Results and observations

10 experiment categories were performed as illustrated in table 7.1. For each of the 9 experiment categories, the system was executed 5 times, and the results averaged. Results are included in appendix C section 5. The results obtained are presented as follows:

- Figure 7.1 plots the average local functional utility of agents at $t = 40$ against the total number of agents deployed.
- Figure 7.2 plots the average local performance utility of agents at $t = 40$ against the total number of agents deployed.
- Figure 7.3 plots the average global performance utility of communities at $t = 40$ against the total number of agents deployed.

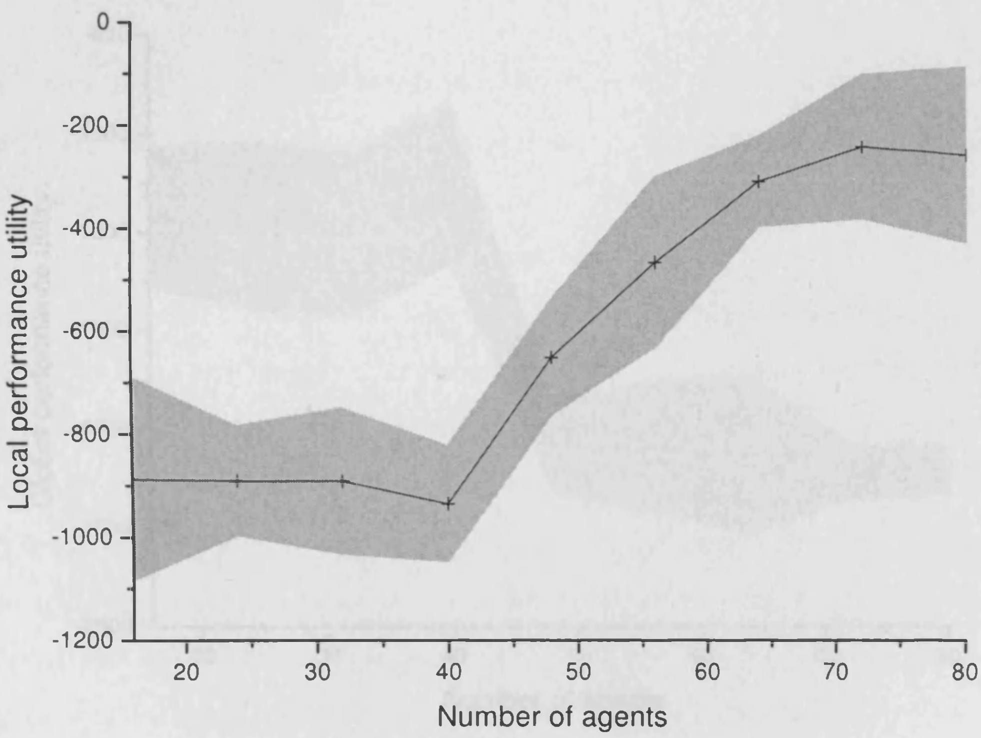


Figure 7.2: Average local performance utility of resource agents.

The mean local performance utility at t=40 is plotted against the number of agents in the system. The shaded area represents the 95% confidence interval around the mean.

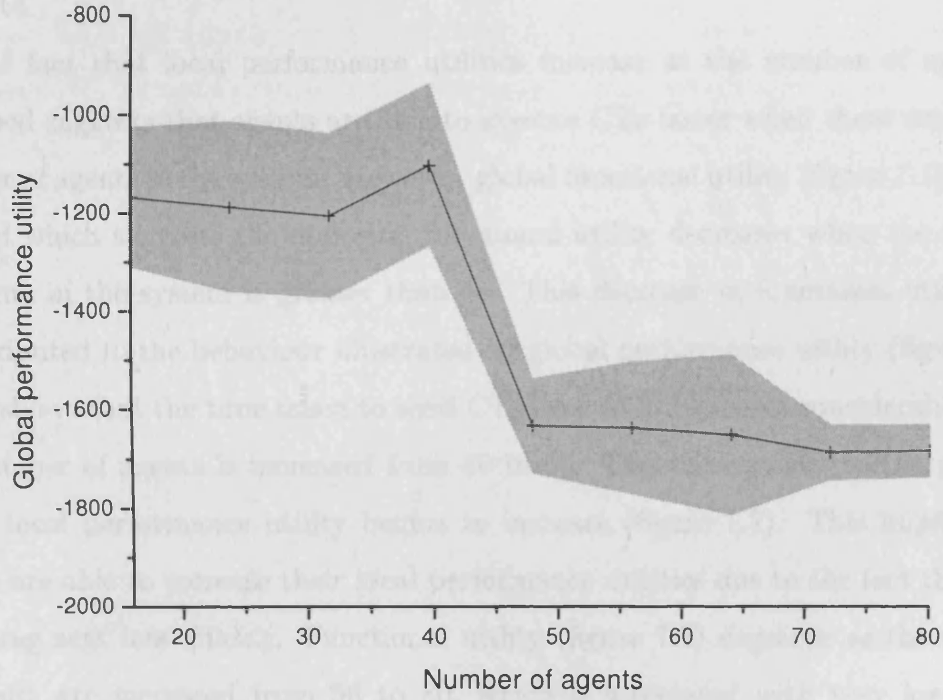


Figure 7.3: Average global performance utility of the computational community. *The mean global performance utility at t=40 is plotted against the number of agents in the system. The shaded area represents the 95% confidence interval around the mean.*

Firstly, it can be seen that the global performance utility (figure 7.2) of the system follows a trend that was not anticipated: local performance utility is higher for agents deployed in experiments with 50-80 agents that it is with lower numbers of agents. The local performance utilities of agents were expected to decline as the number of agents residing on hosts increased, as an increasing number of agents compete for the same resources (processor time and memory), which should increase the time taken to execute CTs; however the results illustrated in figure 7.2 show the opposite.

The fact that local performance utilities increase as the number of agents is increased suggests that agents are able to execute CTs faster when there are a large number of agents in the system. However, global functional utility (figure 7.1) follows a trend which suggests the opposite; functional utility decreases when the number of agents in the system is greater than 60. This decrease in functional utility can be attributed to the behaviour illustrated by global performance utility (figure 7.3), which shows that the time taken to send CTs over ACL increases considerably when the number of agents is increased from 40 to 56. This corresponds to the point at which local performance utility begins to increase (figure 7.2). This implies that agents are able to increase their local performance utilities due to the fact that CTs are being sent less quickly. Functional utility (figure 7.1) degrades as the number of agents are increased from 56 to 80, which is associated with very low global performance utility (figure 7.3).

To summarise, the investigation has shown that as the scale of the MAS is increased, the following occurs:

1. Functional utility, which encapsulates the primary objectives of the system, is initially unaffected by an increase in the number of agents deployed on hosts, until the number of agents is greater than 56, when functional utility decreases. This shows that at this point, the system's ability to achieve it's design goals is compromised.

2. Local performance utility, which encapsulates each agent's CT execution speed, increases as the number of agents is increased from 40 upwards.

3. Global performance utility, which is based on the efficiency of sending CTs over ACL, decreases when the system contains more than 40 agents.

This therefore shows that the initial emergence of a scalability issue with respect to this application occurs with sending CTs over FIPA ACL when the system contains more than 40 agents, where more than 5 agents are deployed on each of the hosts. Although additional factors [1], other than those currently encapsulated by G^P and L^P, may affect the performance/behaviour of the community, performance utility as it has been defined here is of use in detecting an important aspect of MAS behaviour. If this system were deployed on a different network infrastructure, or different hosts, or using different versions of Java, the effects on performance utilities may be very different. Additionally, the analysis of this factor in implementation decisions, for instance the consideration of an alternative method of sending CTs, may also have an effect. The engineering of the application in this way, where performance utility is optimised, provides a different perspective when compared to the optimisation of any high-level decision making or learning processes that may be implemented by agents optimising functional utility.

7.7 Conclusions

It has been demonstrated that performance and functional utilities can be of use in assessing the scalability of this application. In particular, it has been shown how the defined utility functions exhibit different trends that are of use in analysing MAS behaviour. Local performance utilities, which are based on the speed of performance of an agent, and global performance utilities, which are based on the performance of interacting agents, were shown to have contrasting properties. It was shown that the global performance utility degraded considerably at a certain point when the number of agents in the MAS was increased. The discovery of this property suggests that the MAS designer may carry out the following in order to improve the system's

[1]Other factors may include the underlying routines performed by FIPA-OS, the efficiency of which are not currently measured by performance utility functions, but do affect the speed with which agents are able to send, receive and process CTs

performance:

- If the number of CTs that are submitted to the system fluctuates over time, one approach would be to develop agents with the ability to reason about whether to send CTs (the CTs that are not executable locally, and must be sent to another agent) immediately or to hold them in a queue. Agents with this ability could queue CTs when global performance utility is low, meaning that CTs are being delivered slowly, in order avoid further reducing CT delivery times by sending more messages. CTs placed in the queue could be held in the queue until global performance utility increases, which would occur during time periods in which the number of CTs that are being submitted to the system is lower.

- Equipping agents with a memory of previous experiences could be effective in improving performance. It has been shown in chapter 6 that agents residing on certain platforms perform better than others, and one approach would be to develop agents that are capable of learning to choose where to send CTs. Agents would rank other agents with which they have interacted before, based on the time taken by an agent to execute a CT once it has been sent. When sending a CT, an agent would then choose to interact with an agent that has responded quickly in previous interactions.

- Due to the fact that the time taken to send CTs over ACL is a major issue with respect to the scalability of the system, the MAS designer could choose to deliver CTs using a different method. Utilising socket communications, which are faster than the Java RMI based FIPA ACL communication protocol, should improve system performance.

An important aspect of the application described here is that the use of multiple community ESNs has been demonstrated. A single ESN was deployed to represent each of the four (C,V,S and I) communities, where each ESN manages utility function assignment in its corresponding community. The LEAF approach allows multiple communities to be defined around decomposed system objectives, and this provides a scalable approach to engineering LEAF MAS by avoiding the single point-of-failure disadvantage of using a single community ESN.

CHAPTER 8

Summary, lessons learnt and conclusion

8.1 Summary

The thesis is now summarised by examining the work presented in each chapter.

Chapter 1 The aim of chapter 1 was to introduce the concept of a MAS, introduce the work presented in this thesis, and to state the hypothesis on which this thesis is based. The hypothesis consisted of two parts, firstly, that utility function assignment can be implemented as means of multiagent coordination for FIPA compliant agents, and secondly, that a novel approach to defining utility as two separate types, performance and functional utilities, can enable properties of a system's behaviour to be discovered that a MAS designer can utilise to improve the system.

Chapter 2 Chapter 2 surveyed the areas of agent research related to this thesis. Firstly, techniques for coordination in multiagent systems were discussed in order to put into context the method of coordinating agents employed by LEAF, which is based on COIN. Following this brief introduction of coordination techniques in general, the advantages of learning agents and the COIN framework are discussed. The advantages of learning agents with respect to coordination are that learning agents can automatically optimise their behaviours and reduce the need for manually configuring coordination strategies using other techniques. It is concluded that the COIN framework provides a domain independent, scalable coordination technique that incorporates learning agents, and is subsequently adopted as the coordination model upon which LEAF is based. Agent interoperability techniques are surveyed, with the emphasis on FIPA and KQML. Finally, the chapter discusses toolkits which provide support for FIPA or learning technologies.

Chapter 3 The LEAF framework is presented in chapter 3. The development of this framework is the first step towards demonstrating that the hypothesis presented in chapter 1 holds true. The LEAF framework implements the COIN approach to coordination, which is based on assigning utility functions to agents. Assigned utility functions must be supported by various parameters which may not be available to an agent locally, and the LEAF framework introduces the ESN entity which distributes these parameters to LEAF agents. In order to provide a framework capable of testing the hypothesis of this thesis, the LEAF framework defines the following classes of utility:

1. *Local performance utility:* quantifies the performance engineering related aspects of an individual agent's behaviour.

2. *Local functional utility:* quantifies the agent's success in achieving abstract individual objectives.

3. *Global performance utility:* quantifies the performance engineering related aspects of an interacting community of agents.

4. *Global functional utility:* quantifies the success of a community of agent's with respect to their achievement of globally defined, application specific objectives.

Chapter 3 shows how a mechanism may be implemented to support the COIN framework when implemented by distributed FIPA compliant agents. Specifically, it is concluded that a MAS coordinated using COIN must be supported by an entity such as the ESN, which assigns utility functions and manages the supply of parameters to these functions. It is concluded that utility function assignment should take place via FIPA ACL messages, and that utility function parameters, which can change relatively quickly, should be communicated using the faster communication mechanism provided by sockets.

Chapter 4 In order to apply the LEAF framework, a software toolkit building on the FIPA-OS platform is presented. The LEAF toolkit supports the development of agent systems which conform to the LEAF framework. Support is provided by a Java API which extends the API provided by FIPA-OS. FIPA-OS is a widely used agent construction toolkit which claims to simplify and speed up the development of agent based systems [52]. The FIPA-OS model has been examined in detail in chapter 4, and the LEAF API which extends FIPA-OS is designed in the same style to add components to this model which support learning and coordination based on the COIN framework.

Chapter 5 Chapter 5 demonstrates an application of LEAF by approaching a problem involving the coordination of buyer agents in different markets. This problem was chosen due to the difficulty in finding a solution without centralised control, where agents execute instructions originating from a centralised entity. Additionally, the problem was chosen in order to demonstrate a potential practical application of LEAF. Although a number of simplifying assumptions are made in developing the market model utilised, the application is grounded in a practical domain. An investigation of multiagent learning supported by LEAF is undertaken, and the results presented show that the buyer agents are able to out-perform self-interested agents and that the application is able to scale adequately as the number of buyer agents is increased. Additionally, an investigation is made into the effects of agent learning when remote parameters are updated with varying frequencies, and results are presented which show that the performance of agents degrades gracefully as remote parameters are updated less frequently.

Chapter 6 In chapter 6, the focus is on demonstrating that the second part of the hypothesis stated in chapter 1 holds true. Specifically, the use of performance utility is investigated within the context of an application based on a computational

Grid. It is shown that performance utility and functional utility are partially related within this application, and that performance and functional utilities can be used to analyse different aspects of MAS behaviour. It is demonstrated that the performance utility of an agent is highly dependent on the host on which the agent is deployed, and that performance utility subsequently affects the ability of a MAS to obtain high values of functional utility.

Chapter 7 Chapter 7 investigated the scalability of the computational Grid based application by increasing the number of agents in the system. A key approach supported by LEAF is demonstrated, where multiple communities may be defined around which agents can optimise global utility functions. By introducing a new performance utility function, it was shown that message delivery times were a significant factor in the scalability of the computational Grid based application, and various possible improvements to the system were highlighted as a result of the use of functional and performance utilities to analyse the behaviour of the MAS.

8.2 Conclusion

The contributions of this thesis are as follows:

- The development of a FIPA compliant toolkit for developing learning based agents which can be coordinated using COIN.
- An extension to the definition of utility which is shown to be of use in highlighting properties of a MAS which may be utilised to improve system performance.
- Electronic market and computational Grid based applications which highlight potential applications of the developed toolkit.

The development of LEAF and its subsequent application to the market based application presented in chapter 5 supports the first part of our hypothesis. It has been shown that LEAF provides a useful implementation of ideas from COIN, and can be used to coordinate the behaviours of learning agents. The approach enables the coordination of independent learners by parameterising each agent's payoff function with non-local information. More complex applications may utilise more advanced learning techniques than those used in chapter 5, including Q-learning [61] agents. Q-learning agents are utilised in an electronic market based scenario by Tesauro and Kephart [64, 63] to make decisions in competitive market places. One of the advantages of using LEAF in such scenarios is that LEAF agents are able to interact with FIPA compliant agents and agent platforms, therefore allowing the agents to be deployed in practical settings. Although much theoretical research has taken place in the areas of coordination and multiagent learning, the development of LEAF has been motivated by a desire to implement ideas within fully interoperable FIPA compliant agent systems. FIPA compliance was chosen as it currently provides the most promising agent standardisation approach for promoting Internet-scale agent interoperability. In particular, this approach aims to minimise the number of simplifying assumptions and abstractions made when developing coordination frameworks, because it is our belief that there are important considerations to be made when applying ideas in practical scenarios. This has been demonstrated by the implementation of LEAF using FIPA-OS, where various implementation issues such as problems with large-scale agent interaction have inhibited MAS performance. Appendix B describes these issues in more detail.

As discussed in chapter 2, FIPA publishes specifications for developing agents which standardise agent communication protocols, message transport mechanisms, agent management mechanisms, service discovery mechanisms and other aspects of software agent implementations. LEAF has been implemented as a FIPA-compliant platform, which therefore allows LEAF agents to interoperate with other non-LEAF, FIPA-compliant agents developed by different organisations. Due to efforts such as Agentcities, which promote the adoption of FIPA, and the large number of organisations which participate in the development of the FIPA standards, we believe that FIPA-compliance will be crucial in allowing LEAF agents to interact with many currently existing agent systems and also systems developed in the near future. We believe that FIPA agent systems will become increasingly widespread and that LEAF agents will therefore be able to take advantage of a large-scale environment in which

LEAF communities can be deployed to achieve application specific objectives.

The LEAF toolkit is a useful piece of software to emerge from the research conducted, and plans exist to make the software available publicly as an open-source implementation. FIPA-OS was chosen as an existing toolkit with which LEAF could be integrated. FIPA-OS provides an implementation of the FIPA specifications which reduced the amount of work required to implement LEAF. It would be possible to integrate LEAF with other FIPA compliant agent toolkits such as JADE and Zeus as the LEAF framework is compatible with any FIPA compliant implementation. It was highlighted in chapter 2 that there are very few agent construction toolkits that provide support for learning technologies. The DirectIA toolkit focuses on developing AI for videogames, and provides no support for any agent interoperability standard such as FIPA or KQML. The ABLE toolkit is the most closely related toolkit to LEAF that provides learning technologies, however in contrast to FIPA-OS and JADE it does not provide full support for the FIPA specifications. ABLE provides support for coordination via the Autotune agent which is able to directly adjust, using learning techniques, the parameters belonging to the agents it controls. This differs from the COIN coordination model adopted by LEAF where only the utility functions of agents are manipulated by the controller (in LEAF, the ESN is the controller), therefore allowing for a greater degree of agent autonomy.

The work presented in chapters 6 and 7, which focuses on an agent based approach to managing resource in computational Grids is related to the framework presented in [54] and the utilisation of COIN in [65] to coordinate agents in a similar scenario. In retrospect, the demonstration of performance utility in chapters 6 and 7 perhaps did not illustrate the advantages of performance utility as clearly as was hoped, however trends in performance utilities were shown to be of use in analysing a system's behaviour (therefore supporting the second part of the stated hypothesis) and revealing potential improvements that could be made to the system in order to enhance performance. We have therefore learnt that the approach taken in defining performance and functional utilities can be of benefit in developing MAS. If the host/platform on which a software agent is deployed is considered to be a separate issue to the algorithms used by the agent to achieve its objectives, it makes sense to define a utility with respect to the progress made by the algorithm (functional utility), and a utility of the agent dependent on implementation (performance utility). Specifically, the notion of functional/performance utilities may be more beneficial in more complex domains, where:

- Agents are able to migrate to different platforms in order to maximise their performance utilities, possibly approaching the maximisation of functional and performance utilities as two separate learning processes.
- The maximisation of functional utility by individual agents conflicts with the maximisation of performance utility. This may necessitate a trade-off between working to maximise functional or performance utility, depending upon the platform on which agents are deployed.

In order to decentralise the utility function assignment process, the notion of separate communities was introduced. This approach is introduced to promote scalability, as the assignment of utility functions around a single ESN, by which all remote parameters are distributed to agents, clearly becomes difficult when a community is large. Communities may undertake complex activities, and individual agents are assumed only to provide a subset of these activities. For a community to operate effectively, it must utilise the capabilities of a number of agents, where each agent provides only a small part of the aggregate behaviour that a community is expected to exhibit.

The existence of multiple communities are found to be an emergent property of many existing large scale networks [30], and this would suggest that this approach is potentially beneficial when engineering scalable MAS. In chapter 7, the use of multiple communities was demonstrated with a system consisting of four communities, where the objectives (utility functions) of these communities were independent. The application presented in chapter 5 demonstrated how a community can consist of agents residing in different environments (markets), cooperating indirectly as a result of the optimisation of their assigned utility functions. Within these applications, the notion of a community represents a coherent group of agents committed to a common set of objectives. Forming such communities may be beneficial in forming trust and altruistic relationships between agents, therefore encouraging cooperation and coordination. An agent trusts another agent when it can rely on the quality of the services and information provided by the agent. Altruistic behaviour occurs when an agent behaves in a way that is of benefit to another agent, or set of agents, but is of no benefit to itself. As discussed in chapter 3, the COIN coordination framework adopted by LEAF supports utility functions with social context, which results in an agent being rewarded for the effects of its actions upon the community of which it is a member. An agent which learns to optimise a local utility function with social context may therefore learn to perform actions that would appear to be altruistic towards other agents in the community.

The utility assignment approach adopted in this thesis is a useful framework for forming agent communities, where the key features of the community formation process are:

- Agents are free to join and leave communities with the objective of maximising local utility.
- When an agent joins a community, the agent's utility reflects its effect on that community. This is achieved though utility function assignment.

These two properties encourage community formation, as agents will tend to remain as members of the communities which they positively effect, therefore allowing communities to be formed in a decentralised, self-organising process. The utilisation of this process provides a means of community formation without the need for centralised computation and with no requirement for agents to communicate as communities are formed. However, the use of utility assignment to form multiagent communities provides no guarantees of the optimality of the groups formed, and the time taken to form stable communities depends on the complexity of the learning problem faced by agents and the performance of the learning techniques they employ. Furthermore, the objectives of communities must be independent in order to avoid competition between communities; LEAF coordinates agents within specific communities, and provides no mechanism for coordination between agents belonging to different communities. This technique must be further developed, where scenarios are examined in which global objectives are not decomposed into independent community utility functions. A detailed investigation into community formation using utility function assignment is beyond the scope of this thesis, and is an exciting avenue for further research. Various other ideas concerning aspects of further work in relation to LEAF are explored in chapter 9.

CHAPTER 9

Further work

9.1 Introduction

In this chapter, potential areas of further research and possible development activities, arising from the work presented in this thesis, are explored.

The LEAF toolkit as described in this thesis is intended primarily as a tool enabling our hypothesis to be tested. Although the toolkit provides a useful API that can utilised by developers in order to create MAS conforming to the LEAF framework, a wide range of improvements and further developments are possible.

9.2 Graphical user interfaces and visual development tools

As described in this thesis, LEAF provides no visual tools/graphical user interfaces (GUIs) to assist the MAS developer. Potentially, this discourages users who do not happen to be experienced Java programmers. Integrating GUIs within LEAF may provide a more accessible development process and provide a less time consuming development process. In particular, the following visual tools would benefit the MAS developer:

- *Code generation tools:* Visual tools used to specify software components and generate corresponding code segments are common features of integrated development environments (IDEs) and various other software packages. Similar approaches within the context of LEAF would allow developers to construct utility functions and agent behaviours visually by manipulating components in a visual builder tool.

- *Visualisation consoles:* The ability of the developer to debug MAS is dependent on the often difficult task of observing the behaviour of agents as a system evolves. This could be supported by providing consoles displaying the evolution of local/global utilities, remote parameters and observable properties of agents/communities, and additionally, GUIs may be used to deploy agents/communities.

9.3 Interaction between communities

As discussed in chapter 8, the LEAF framework must be extended to allow multiple communities to be deployed where interdependence between communities may exist. Three potential solutions to this problem should be investigated:

- *Communication between ESNs:* It may be advantageous for ESNs to share information and assign utility functions parameterised by remote parameters which include information from other communities. In this way it may be possible to engineer utility functions with social context provided by other communities.

- *Hierarchical community structures:* In the same way that WLU is assigned to agents based on their contribution to a community, the possibility exists for communities to be assigned similar utilities based on a hierarchical structure, as illustrated in figure 9.1.

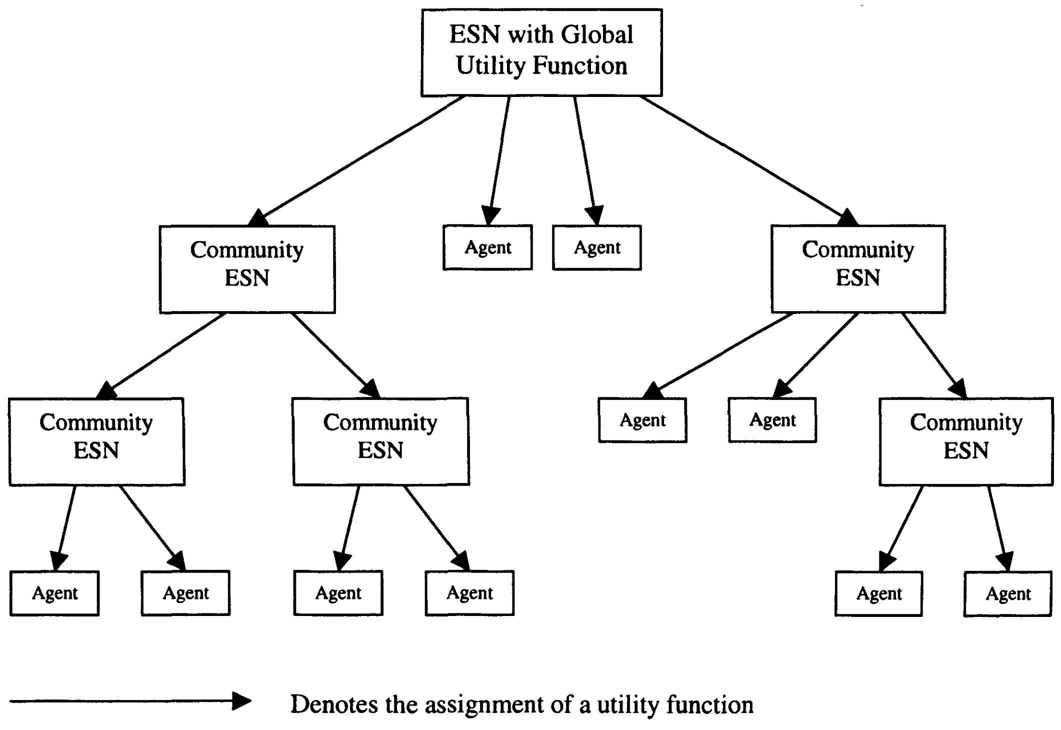


Figure 9.1: Hierarchical structuring of communities

The figure illustrated how community ESNs may be structured in a hierarchy where utility functions are assigned to ESNs in addition to agents.

9.4 Security

The security considerations involved with LEAF are generally the same issues applicable to MAS in general [16], with the addition of security implications relating to:

1. *Utility function assignment:* In addition to the regular communication over FIPA ACL, LEAF agents utilise utility function assignment, which in turn influence agent behaviour. The possibility of "Trojan Horse" utility functions exists, where the computation of utility functions may trigger malicious code, or simply inhibit agent behaviour.

2. *Community membership:* There exists the potential for malicious agents to join LEAF communities in order to lower global utility or generally wreak havoc.

3. *Observable properties and remote parameters:* The transfer of observable properties and remote parameters via socket connections provides additional communication layers at which security needs to be considered.

In order to deploy LEAF beyond its current status as a research tool, a comprehensive assessment of these security problems and their possible solutions is required. A discussion of trust and security issues with respect to FIPA in [51] concludes that the current FIPA specifications are "easy to exploit in order to disrupt access to service providers, to deny services to other users and to masquerade as other users breaking their privacy". Although security is often not considered by agent researchers, FIPA has recognised this issue, and a working group is currently active in this area. The fact remains, however, that the FIPA specifications currently do not define a specific mechanism or policy for secure communication, and therefore a requirement of future work is to monitor the progress of the FIPA working group active in this area, and to consider the implementation of security features in LEAF.

9.5 Fault-tolerance and persistence

A potential drawback of the LEAF approach to supporting the assignment of utility functions is that the community ESN involved in this process exists as a single point of failure for a community. Within a specific LEAF community, an ESN maintains important dynamic information concerning:

- Observable properties obtained from agents in the community.
- Remote parameters, which are dynamically transferred to agents in the community.
- Information concerning the current membership of the community.
- The global utility of the community.

In chapter 5, the ability of a community to continue to function and continue to learn was investigated under conditions which simulated the temporary failure in communications with the community ESN. Although this investigation demonstrated that a community is able to continue to function while the ESN is unavailable, it was assumed that all of the aforementioned information maintained by the ESN was recoverable by the ESN after any temporary faults. Generally, faults related to the ESN may potentially occur due to hardware, software or network problems, some of which may enable the ESN to retain its state prior to the fault, and some of which may require the ESN to be re-initialised, possibly on a different host. A possible area of further work is to examine the faults that can occur with agents and ESNs, and ways in which they can be most efficiently resolved. Possible solutions may involve making certain parts of agent/ESN states persistent, and provide mechanisms which enable ESNs/agents to be effectively recovered.

9.6 Further applications

A key area of further work will be to explore the application of LEAF within large-scale applications. The applications presented in this thesis were developed primarily to investigate the concepts underpinning LEAF and demonstrate key ideas.

In order to fully explore the benefits of the approach defined by LEAF, and also further develop the approach, research may involve applications with multiple organisations/institutions, where issues arise due to the fact that agents belonging to multiple owners interact. These issues may include the prevention of agents with certain owners belonging to certain communities, and various other restrictions placed upon the interactions made by agents. Additionally, the applications described in chapters 5 and 6 may be further developed and applied on a larger scale. The possibility of deploying LEAF agents as part of the Agentcities network is also to be investigated.

The agent behaviours used within the applications presented in this thesis have been fairly simple, and further work should investigate the use of more complex behaviours, and possibly agents which employ plans. It may be possible to employ more sophisticated learning techniques such as hierarchical reinforcement learning [43] to aid the learning processes of agents in such scenarios.

9.7 Extending communities to form Web services

The notion of a community in LEAF currently refers to a set of objectives, where these objectives are performed by the agents belonging to the community. Although the membership of the community may change as agents join and leave the community, the ESN represents the same community. A possible extension of the LEAF community is to enable communities to provide information regarding the community they represent, and also to provide application specific agent based services via the agents belonging to a community.

Web services [31] currently provide a technology which allows organisations to offer functionality via the Internet – "Web services are the next step in the evolution of the World Wide Web and allow programmable elements to be placed on Web sites where others can access distributed behaviours" [8]. Support for discovering Web services is provided by the Universal Description, Discovery and Integration (UDDI) specifications [8], which enable information regarding Web services to be published and discovered. The LEAF community ESN is an entity which could be extended to provide an interface between LEAF and Web services. The LEAF community ESN is able to provide information regarding the agents belonging to the community, the utilities of both the community and agents within the community, and the observable properties of agents within the community. Work is currently being carried out into the presentation of LEAF communities in the form of Web services, and potentially, more complex services could also be provided by community ESN Web services, including the definition of utility functions and the invocation of agent based services via an ESN's Web service interface.

APPENDIX A

Acronyms

ABLE	Agent Building and Learning Environment
ACC	Agent Communication Channel
ACL	Agent Communication Language
AI	Artificial Intelligence
AID	Agent Identifier
AMS	Agent Management System
API	Application Programming Interface
CAP	Credit Assignment Problem
COIN	Collective Intelligence
CT	Computational Task
DAI	Distributed Artificial Intelligence
DF	Directory Facilitator
ESN	Environment Service Node

FIPA Foundation for Intelligent Physical Agents

FIPA-OS FIPA-Open Source

FTP File Transfer Protocol

GridFTP Grid File Transfer Protocol

GUI Graphical User Interface

IBM International Business Machines Corporation

IDE Integrated Development Environment

IIOP Internet Inter-Orb Protocol

IPMT Internal Platform Message Transport

JADE Java Agent Development Environment

JESS Java Expert System Shell

JRE Java Runtime Environment

JVM Java Virtual Machine

KQML Knowledge Querying and Manipulation Language

LEAF Learning Agent based FIPA-compliant Community Toolkit

MAS Multiagent System

MASIF Mobile Agent Systems Interoperability Facility

MDP Markov Decision Process

MOP Market Oriented Programming

MTS Message Transport System

NASA National Aeronautics and Space Administration

OMG Object Management Group

PD Prisoner's Dilemma

QoS Quality of Service

RL Reinforcement Learning

RMI Remote Method Invocation

TCP Transmission Control Protocol

TOC Tragedy of the Commons

UDDI Universal Description, Discovery and Integration

WLU Wonderful Life Utility

APPENDIX B

FIPA-OS performance and scalability issues

The approach taken in this thesis is motivated by the belief that there are important considerations to be made when applying ideas in practical scenarios such as FIPA compliant agent systems. The LEAF toolkit, described in chapter 4, utilises FIPA-OS for an implementation of the FIPA specifications. Some implementation issues relating to the use of FIPA-OS are now demonstrated through an empirical investigation into *(a)* the difference in message delivery times between agents residing on the same host and agents residing on different hosts, and *(b)* the efficiency of communication between FIPA-OS agents as the number of agents interacting with each other is increased. The aim of this investigation is to demonstrate how implementation issues become important when dealing with the practical implementation of MAS, and in particular, that the scalability of an implementation is an important factor. This study is of relevance due to the fact that while many approaches concentrate on learning/coordination models and their scalability (as is the case in COIN), leaving implementation issues to be optimised separately, LEAF aims to incorporate both of these issues through performance and functional utilities. The data collected here shows some of the issues that can be encapsulated by performance utility. All experiments use FIPA-OS version 2.1.0 and Java version 1.3.1.

B.1 Comparison of message delivery times on single and multiple hosts

The time taken to send a FIPA ACL message, containing a content object of varying sizes, is made between two agents residing on the same host, and two agents residing on different hosts. Delivery time is considered to be the time between (a) the invocation of the `forward` method by the sender agent, and the invocation of the `handleX` method of the receiver agent. [1] Experiments are performed using message sizes starting at 125KB, incremented by 125KB until a maximum of 1000KB is reached. For each message size, 20 messages were sent between the agents and the average delivery times are displayed in figure B.1. Results are included in full in appendix C section 6.

It can be seen that message delivery times are affected by the size of the message, and whether or not the agents reside on the same host.

B.2 Message delivery time and memory usage with respect to multiagent scalability

The focus here is to observe the effects on message delivery times and agent memory usage as greater number of agents interact with each other. A modified version of the "ping agent" [3] provided as part of the tutorials packaged with FIPA-OS is used in this investigation.

1. The agent registers with the AMS agent of the FIPA agent platform.
2. The agent registers with the DF agent of the FIPA agent platform.
3. The agent searches the DF agent in order to find all other test agents.
4. The agent sends a message to all discovered agents (agents may be discovered via the DF search performed in (3), or when a message is received from a

[1]The `forward` and `handleX` methods are used to send and receive messages using FIPA-OS, where X is the performative received.

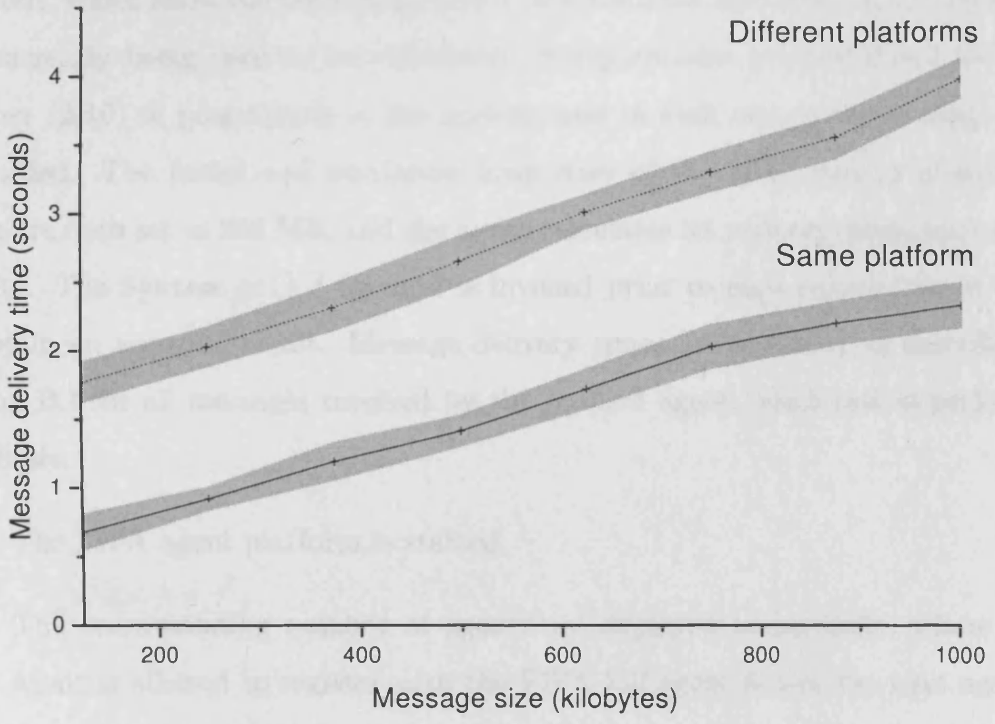


Figure B.1: Comparison of message delivery times.

The figure shows the results of the comparison between single and multiple hosts with different message content sizes. Mean message delivery times are plotted for each platform and message size. The shaded areas represent the 95% confidence intervals around each mean.

previously unknown agent).

5. The agent waits for 5 minutes and then goes back to step (4).

The FIPA-OS ping agent is modified so that messages sent are 1000 KB in size, and so that memory usage and message delivery times may be recorded. Memory usage is obtained via Java's `Runtime.freeMemory()` and `Runtime.totalMemory()` methods, which allow the percentage of the Java Runtime Environment (JRE) memory currently being used to be calculated. 9 experiments are performed for each number (2-10) of ping agents in the system, and in each test, a single ping agent is profiled. The initial and maximum heap sizes of the JRE used to invoke this agent are both set to 256 MB, and the agent calculates its memory usage once every minute. The `System.gc()` [2] method is invoked prior to each calculation in order to obtain an accurate result. Message delivery times are obtained as described in section B.1 for all messages received by the profiled agent. Each test is performed as follows:

1. The FIPA agent platform is started.

2. The corresponding number of agents are deployed sequentially, where each agent is allowed to register with the FIPA DF agent before the next agent is deployed. The profiled agent is always the last agent to be deployed, meaning that when it performs a DF search, it discovers all the agents that will participate in the test.

3. When 15 minutes has elapsed since all agents were deployed, the MAS is terminated and the average memory usage and message delivery times are recorded.

The FIPA agent platform and profiled agent are both deployed on a host with a 500MHz Pentium 3 Processor, running Windows 2000 with 384 MB RAM. Other

[2] According to the Java Core API documentation [6], "Calling the `System.gc()` method suggests that the Java Virtual Machine expend effort toward recycling unused objects in order to make the memory they currently occupy available for quick reuse. When control returns from the method call, the Java Virtual Machine has made a best effort to reclaim space from all discarded objects".

agents are deployed on separate hosts, each of which has a 360 MHz Sparc v9 processor, running Sun Solaris 7 with 128MB RAM. Each host has a 10Mbs connection to the same Ethernet network.

The results of each batch of 9 experiments are averaged and are included in appendix C section 7. Results are displayed in figures B.2 and B.3, which visualise the scalability of message delivery times and memory usage respectively. Figure B.2 shows that in particular, message delivery times do not scale well as the number of agents is increased. This illustrates the potential for a "tragedy of the commons" [33] type scenario with respect to the ping agents, where if we were to assume that ping agents increase their utility by pinging other agents, the rational decision to continue to increase local utility by each ping agent may work towards the detriment of the collective by drastically reducing the ability of all agent to send messages efficiently. This result shows a similar trend to the investigation described in chapter 7, where the time taken to deliver computational tasks increases rapidly as the number of resource agents is increased.

B.3 Summary

The difficulties associated with deploying large scale MAS under the conditions described above illustrate how unexpected factors may inhibit coordination models that may function perfectly in theory. The introduction of performance utility is an approach towards integrating factors based on the issues described above into the LEAF coordination framework. Performance utility allows implementation and performance engineering related issues to be categorised as being conceptually different from functional utility, which relates to the notion of utility most often applied to theoretical multiagent research.

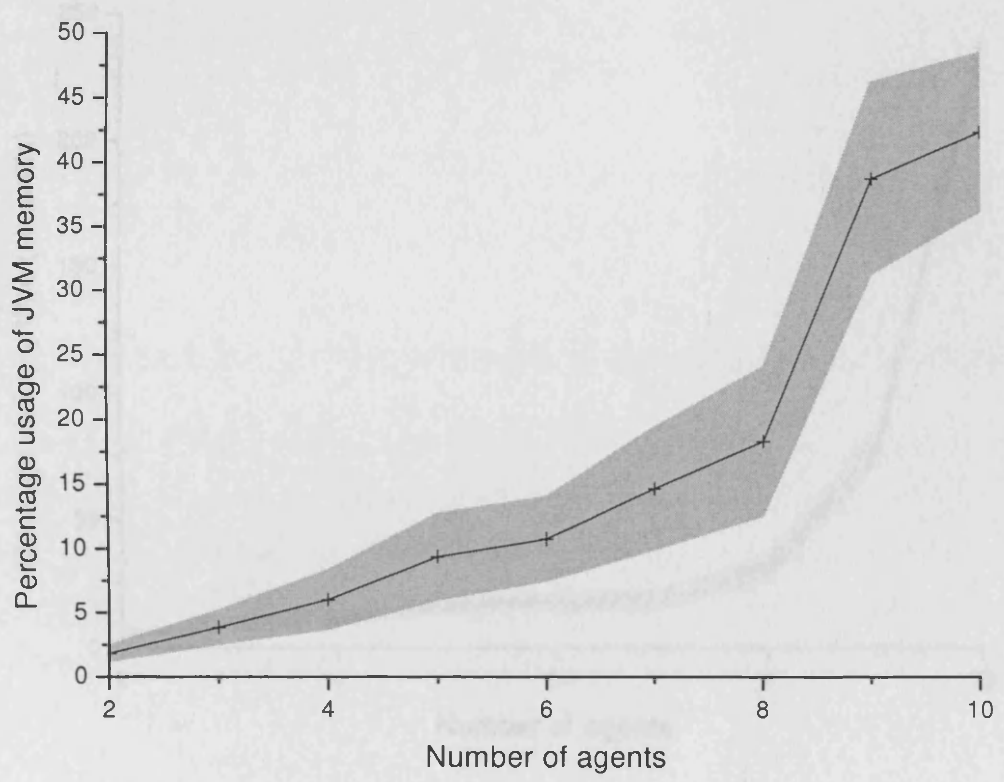


Figure B.2: Memory usage of the profiled ping agent.
The mean memory usage of the agent is plotted against the number of agents in the system. The shaded area represents the 95% confidence interval around the mean.

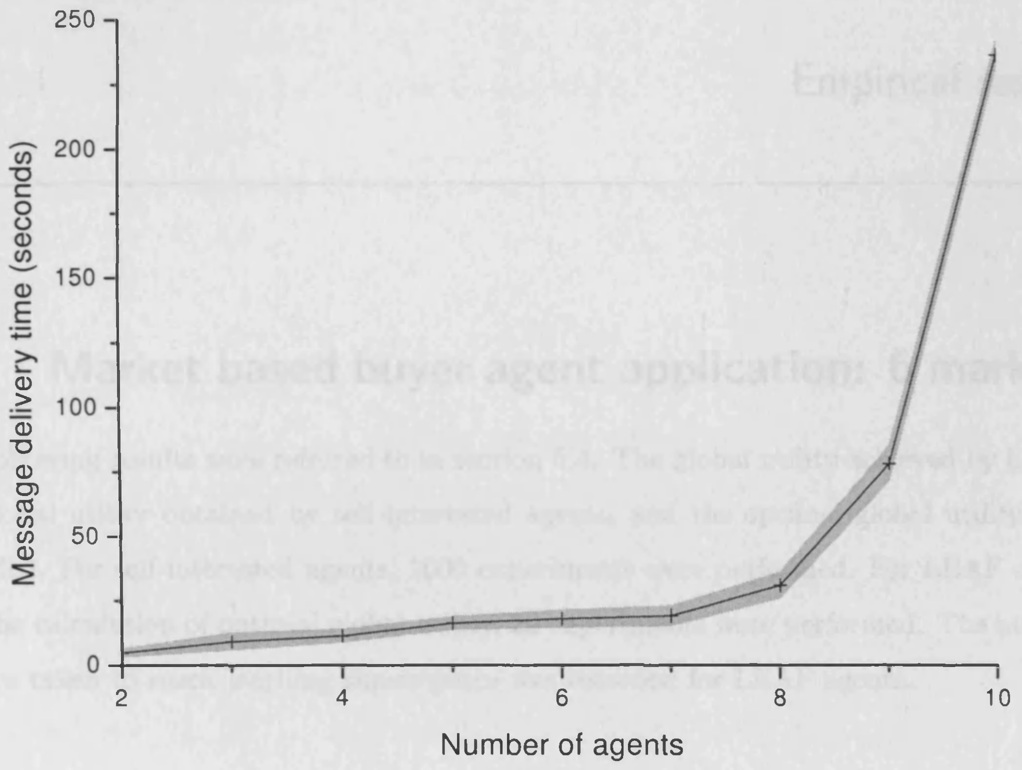


Figure B.3: Message delivery time.

The mean message delivery time for all messages received by the profiled agent is plotted against the number of agents in the system. The shaded area represents the 95% confidence interval around the mean.

APPENDIX C

Empirical results

C.1 Market based buyer agent application: 6 markets

The following results were referred to in section 5.4. The global utility achieved by LEAF, the global utility obtained by self-interested agents, and the optimal global utility were recorded. For self-interested agents, 1000 experiments were performed. For LEAF agents and the calculation of optimal global utility, 20 experiments were performed. The number of days taken to reach learning convergence was recorded for LEAF agents.

LEAF global utility

Global utility values: 374 545 514 482 509 527 543 470 472 534 475 497 484 472 532 454 454 526 492 474

Mean:491.5 Standard deviation:40.276

95% confidence lower bound:472.16 upper bound:510.839

LEAF learning convergence

Number of days taken to converge: 107 118 124 138 103 107 106 104 125 106 103 105 99 108 105 125 104 106 106 116

Mean:100.75 Standard deviation:60.829

95% confidence lower bound:348.592 upper bound:407.008

Optimal global utility

Global utility values recorded: 524 545 536 494 519 527 553 524 539 534 555 497 553 536 537 539 523 526 584 497

Mean:532.1 Standard deviation:21.491

95% confidence lower bound:521.78 upper bound:542.419

Self-interested global utility

Global utility values recorded: 284 475 472 290 383 273 390 378 393 382 393 478 372 364 397 379 470 348 477 388 361 289 297 434 560 287 490 485 371 375 385 388 477 384 385 300 476 293 375 473 382 384 379 468 374 291 389 454 467 290 350 378 467 395 198 362 381 342 473 457 291 382 292 392 286 462 395 478 293 364 378 375 384 393 293 380 298 295 476 397 364 390 387 483 396 375 569 489 376 476 285 387 292 331 276 375 382 195 362 395 386 474 481 361 366 278 473 454 383 369 275 532 294 465 388 482 287 360 281 386 276 386 383 387 296 380 477 392 371 382 454 379 363 373 192 353 379 386 185 293 293 476 366 388 369 284 296 476 384 395 455 293 486 356 295 472 460 362 377 295 374 453 375 383 358 398 269 375 370 476 370 476 395 474 378 280 292 290 289 376 455 394 295 470 479 462 379 290 295 462 480 480 439 461 296 378 273 192 397 468 390 385 290 375 292 377 477 458 387 461 443 382 384 439 387 378 388 387 293 395 392 278 368 360 381 382 295 476 381 352 378 377 370 298 384 391 374 464 374 391 363 460 385 343 388 386 479 379 373 293 386 470 472 288 379 368 369 372 278 387 351 289 449 465 385 385 482 465 472 380 295 371 472 369 264 383 285 390 471 481 289 379 388 395 566 433 385 382 384 386 298 347 279 386 372 385 382 546 377 489 464 379 378 294 457 393 194 391 284 379 290 473 297 398 376 282 474 378 292 387 376 436 386 379 540 379 467 390 356 288 367 376 282 460 393 195 378 463 379 377 367 388 394 275 378 290 382 192 291 378 379 296 471 294 378 273 291 287 484 474 357 463 394 390 442 379 197 377 494 386 290 282 479 469 396 375 458 294 480 366 452 479 351 377 575 428 295 286 291 374 355 276 382 465 393 373 572 191 376 277 390 282 484 383 284 295 458 379 481 290 463 285 482 462

```
376 480 360 299 392 383 286 383 369 482 353 388 487 367 380 481 282 477 385 388
387 469 430 387 299 371 286 467 481 475 383 363 335 475 385 375 389 462 285 385
479 382 392 473 475 382 388 545 449 385 366 388 392 370 360 457 466 390 295 372
380 287 378 283 392 297 482 369 363 476 479 365 477 385 361 465 384 397 353 377
370 453 373 472 378 388 464 455 390 379 367 378 294 295 288 351 370 273 285 459
388 291 373 459 368 298 391 373 292 385 374 345 299 398 254 466 279 362 386 493
368 444 293 363 378 382 546 471 385 380 490 392 361 389 375 336 358 378 385 384
474 396 293 286 283 472 440 388 268 288 489 432 398 388 372 482 388 368 386 471
296 285 466 488 378 378 295 392 297 381 495 348 386 471 381 291 339 544 384 291
480 439 371 373 292 484 393 292 396 281 381 393 469 385 293 381 464 377 367 358
464 370 485 386 471 285 371 295 297 371 436 384 295 386 335 357 394 390 483 468
395 350 285 471 382 381 354 379 279 386 452 367 368 387 471 385 365 295 483 368
295 385 290 385 369 288 482 391 454 369 293 451 384 391 469 282 459 276 474 482
345 394 289 466 364 373 382 282 373 377 193 463 382 290 380 368 295 550 343 380
482 381 388 295 388 377 398 385 289 382 284 478 378 198 343 462 391 291 366 396
371 374 287 367 372 479 369 384 392 466 291 271 472 379 380 295 460 385 386 278
470 398 466 374 373 481 363 374 482 378 193 274 453 465 292 394 378 398 355 360
392 382 343 433 387 388 349 483 372 444 260 288 291 392 384 393 381 386 381 570
376 281 295 379 385 388 375 382 358 382 375 458 379 392 385 387 491 289 358 472
300 378 350 480 475 377 382 385 466 460 376 299 394 383 286 375 370 296 393 387
284 276 383 389 386 489 198 368 474 291 467 466 378 378 379 376 288 486 387 471
483 497 372 471 456 373 469 294 484 382 480 473 389 286 370 367 197 482 414 400
381 367 454 377 454 368 296 192 360 296 296 447 473 294 441 367 374 284 444 380
293 385 462 379 374 394 475 376 378 378 360 380 292 474 365 375 374 476 382 369
467 389 476 483 488 376 457 481 391 450 455 296 387 480 384 281 457 351 379 282
286 393 390 288 383 386 280 389 370 458 297 383 380 384 376 278 386 294 455 466
491 278 370 493 373 288 379 298 288 291 448 382 290 491 383 455 355 466 471 484
345 395 468 386 373 378 381 352 383 557 395 497 369 569 472 390 381 477 286 385
471 288 473 391 373 366 297 390 374 392 552 376 376 294 361 477 473 372 292 355
286 445 272 445 457 282
```

Mean:382.525 Standard deviation:69.604

95% confidence lower bound:378.211 upper bound:386.839

C.2 Market based buyer application: 7-20 markets

The following results were referred to in section 5.5.3. For self-interested agents, 1000 experiments were performed for each number of markets. For LEAF agents 20 experiments were performed for each number of markets.

7 Markets

LEAF global utility

Global utility values recorded: 601 565 609 624 632 445 457 639 649 564 659 657 589 666 598 633 599 579 582 667

Mean:600.7 Standard deviation:60.76

95% confidence lower bound:571.523 upper bound:629.875

LEAF learning convergence

Number of days taken to converge: 106 123 112 138 105 103 106 107 106 128 101 104 107 105 111 106 105 106 114 103

Mean:109.8 Standard deviation:9.418

95% confidence lower bound:571.525 upper bound:629.875

Self-interested global utility

Global utility values recorded: 650 480 390 381 384 387 457 460 385 374 480 367 298 295 388 466 383 290 490 487 285 378 380 399 378 388 380 384 675 466 476 379 300 373 394 381 479 295 467 447 486 464 389 489 396 373 384 388 383 291 442 383 485 379 466 476 284 396 378 385 474 368 486 370 375 394 388 476 462 489 475 488 396 369 474 297 289 384 491 280 286 289 358 490 476 382 451 394 461 365 475 486 273 298 667 439 481 478 298 367 440 358 363 290 391 395 470 456 476 464 461 383 284 454 485 482 383 485 355 615 491 487 440 477 384 386 480 375 457 452 484 451 389 389 472 385 362 670 460 285 355 273 291 480 388 466 197 630 381 384 391 369 644 660 470 376 481 472 391 475 390 486 477 385 488 458 473 374 385 395 459 467 486 460 644 339 290 385 482 486 451 483 394 466 472 468 475 386 292 470 478 385 638 458 379 477 369 381 464 471 379 470 388 276 480 456 366 484 476 467 468 472 661 281 368 372 470 447 381 391 379 377 440 458 475 464 354 488 387 476 383 357 468 377 365 633 639 293 457 356 457 397 459 385 446 661 470 373 655 394 380 385 386 472 395 477 451 482 291 372 388 286 377 292 380 385 383 474 668 396 383 450 475 478 495 439 478 393 475 369 295 478 390 267 281 388 390 487 485 466 470 383 373 384 360 462 477 470 281 462 375 469 376 388 289 368 469 448 388 650 389 467 367 392 493 464 379 387 647 389 291 477 291 396 388 644 379 296 371 381 460 490 285 288 372 469 389 290 381 487 390 369 347 479 474 391 386 288 385 490 297 473 662 396

```
641 375 389 473 294 668 453 399 384 264 384 373 389 658 484 644 288 366 355 388
473 479 383 358 358 479 384 487 482 386 383 473 397 473 486 371 492 646 389 377
381 380 389 478 467 388 380 470 655 394 377 282 296 680 359 642 488 290 271 381
365 389 388 383 381 381 649 357 447 389 383 642 387 383 390 397 382 470 462 454
290 285 485 585 466 479 378 477 388 386 388 398 374 379 386 469 636 446 475 465
466 474 469 492 446 392 458 488 397 362 382 643 387 476 390 389 291 479 395 390
392 397 670 483 387 383 473 473 461 387 474 485 662 291 291 491 479 377 387 475
465 389 390 465 482 294 292 469 381 293 296 382 643 450 476 479 670 391 479 382
671 479 419 382 692 477 357 476 482 484 469 476 482 475 393 291 285 300 474 478
395 378 199 380 475 481 380 298 296 645 492 648 379 468 459 454 384 469 469 435
461 379 289 458 483 462 377 379 465 487 288 386 394 641 483 298 488 284 271 475
456 476 637 465 474 479 375 377 394 386 478 373 471 456 660 378 379 291 198 389
486 466 382 386 384 451 391 388 442 474 379 490 383 467 290 292 375 379 486 482
382 479 390 376 386 318 484 692 374 673 453 296 392 466 447 380 372 277 483 396
367 461 296 392 296 394 293 481 480 386 294 291 386 293 294 371 487 376 480 292
397 434 384 381 270 656 659 393 483 459 293 382 457 479 478 393 657 382 615 366
355 374 294 291 480 687 649 390 377 454 458 457 361 457 390 442 477 332 482 383
479 457 461 370 464 441 576 389 494 283 380 493 359 477 374 481 285 443 474 293
469 397 480 462 478 282 389 389 389 482 387 290 432 486 379 451 279 468 476 293
359 481 467 388 464 489 617 482 291 475 356 470 437 486 383 447 471 381 384 474
269 379 454 376 376 464 494 390 458 299 392 367 291 273 463 391 377 286 358 388
376 372 474 291 468 357 476 291 390 387 468 488 464 382 298 289 374 477 461 465
389 360 466 375 358 291 396 292 456 384 606 392 377 469 374 292 381 465 377 388
378 380 383 280 368 371 378 625 389 462 386 473 396 458 462 373 479 370 680 387
386 381 352 380 645 374 657 664 483 392 454 450 385 381 372 459 363 464 477 637
472 656 453 297 436 469 378 380 195 355 286 383 467 373 280 294 362 465 647 465
675 470 299 459 469 484 384 354 384 375 361 375 475 458 292 392 379 475 481 392
379 256 370 473 444 392 478 474 481 464 395 483 368 371 441 281 620 400 288 287
383 475 390 481 468 377 487 383 369 388 295 285 454 376 365 386 368 377 362 455
638 467 484 295 395 488 466 489 480 376 481 379 459 477 464 382 469 445 449 480
293 376 381 359 377 494 294 469 297 398 457 294 394 362 486 466 381 378 371 462
385 642 292 480 472 295 391 466 444 380 453 485 381 482 645 286 367 387 288 477
665 443 468 372 382 364
```

Mean:421.803 Standard deviation:88.473

95% confidence lower bound:416.319 upper bound:427.287

8 Markets

LEAF global utility

Global utility values recorded: 737 720 685 683 462 734 679 695 703 704 766 703 662 681
719 643 736 732 748 674

Mean:693.300 Standard deviation:62.726

95% confidence lower bound:663.181 upper bound:723.419

LEAF learning convergence

Number of days taken to converge: 105 107 105 134 107 104 118 112 104 108 105 107 142
106 106 106 106 105 106 115

Mean:110.400 Standard deviation:10.190

95% confidence lower bound:105.507 upper bound:115.293

Self-interested global utility

Global utility values recorded: 483 384 480 631 458 676 385 489 372 387 776 386 390 454
466 444 393 294 353 478 461 454 370 372 448 389 384 482 367 384 385 466 384 477
368 469 386 491 292 386 479 392 299 470 373 387 489 388 380 384 393 293 456 476
354 481 371 486 469 746 396 397 288 392 486 768 393 488 290 388 474 390 370 351
486 372 462 374 386 472 393 471 466 389 650 383 485 389 381 388 387 391 747 381
774 395 299 394 632 277 458 464 292 289 476 379 279 735 485 461 724 381 756 475
471 446 452 381 481 479 461 378 293 667 466 380 752 373 477 390 377 371 487 483
466 385 459 487 467 482 485 394 456 389 391 756 497 486 478 476 473 481 469 386
480 386 476 388 384 485 388 481 464 379 382 359 373 381 395 387 468 466 391 478
366 468 375 468 393 380 489 484 379 669 751 470 292 373 707 376 660 705 472 472
289 471 474 280 376 390 478 479 483 648 469 378 296 470 479 369 381 492 466 485
349 377 370 706 480 487 443 383 482 451 482 453 486 478 743 287 277 470 479 486
282 467 281 470 478 377 391 485 281 496 381 483 761 381 480 362 470 481 371 392
474 380 383 384 376 462 765 727 280 472 397 379 484 666 479 392 444 364 484 298
469 377 455 465 478 410 468 480 392 456 272 384 730 384 474 491 387 492 484 472
392 378 740 469 382 483 475 484 481 479 466 449 736 490 707 746 480 468 374 283
480 480 376 479 485 289 776 378 478 480 451 389 462 494 384 377 365 478 369 378
463 383 396 384 360 486 479 435 470 466 480 476 279 394 485 751 603 475 475 369
468 466 482 376 454 374 292 720 493 383 390 464 754 374 483 438 479 489 487 395
388 453 381 383 392 492 377 194 767 397 392 346 477 732 373 487 481 391 383 471
381 472 728 476 475 281 395 293 474 291 389 446 381 699 473 391 482 442 392 449
384 483 380 469 361 399 475 352 289 464 382 482 385 465 390 476 379 469 493 468
674 468 463 392 482 457 483 479 294 479 486 490 481 486 455 445 479 467 393 387
699 388 392 473 375 384 392 482 699 468 391 485 475 381 473 392 479 368 483 736

```
479 386 394 475 486 395 658 483 281 353 283 378 384 377 485 384 485 726 368 381
446 482 438 370 468 472 369 284 390 395 294 483 722 391 374 486 491 472 293 726
468 297 474 450 292 746 477 380 386 767 471 462 386 486 487 373 466 375 761 385
491 378 465 382 491 441 357 666 491 375 727 473 388 661 391 386 475 378 396 483
379 484 472 482 387 387 458 486 394 452 379 383 469 390 378 488 487 293 385 497
472 386 486 284 486 387 483 466 296 731 464 497 481 481 390 719 385 363 367 481
394 480 377 485 299 375 370 349 467 460 489 471 751 478 364 385 762 370 292 478
463 479 478 735 283 383 394 775 752 381 468 472 469 479 458 487 762 711 384 283
279 492 389 457 469 383 458 300 714 488 653 483 290 466 455 467 481 476 758 750
369 386 474 465 489 453 695 466 382 375 382 377 747 364 470 392 476 374 394 391
393 475 388 745 476 391 383 464 369 393 375 486 296 386 477 476 394 375 295 753
477 458 760 383 359 492 388 378 452 487 292 712 363 483 380 479 392 369 480 461
492 375 379 381 486 744 474 377 483 387 756 383 472 275 371 297 378 381 476 490
468 486 472 361 487 740 373 281 388 388 739 397 757 391 479 485 384 375 390 463
465 371 386 375 397 388 728 468 743 363 389 369 282 482 454 386 470 667 444 462
472 361 720 474 477 482 482 381 480 467 474 394 420 443 369 378 774 486 750 390
395 287 389 466 464 392 487 197 483 763 495 392 396 371 705 380 464 386 481 389
345 398 474 490 273 467 470 379 390 474 758 469 429 738 498 453 295 373 479 465
384 385 379 395 456 483 352 291 375 488 385 385 392 488 475 387 375 475 381 458
373 479 291 741 383 395 464 472 383 767 454 757 766 388 722 460 384 480 381 657
375 464 741 721 459 465 445 375 471 385 475 353 474 470 480 499 473 473 447 376
485 387 481 707 458 394 485 377 447 367 475 379 370 483 457 386 494 376 486 469
388 772 372 391 479 470 753 383 382 381 389 477 357 484 650 487 483 457 385 384
294 381 379 488 387 292 478 473 494 480 478 380 353 486 759 471 385 649 478 471
485 747 446 463 761 382 742 383 489 465 397 486 381 482 488 387 380 747 472 491
394 363 474 390 474 389 464 460 377 470 383 472 385 486 477 474 457 384 395 470
483 478 641 393 470 752
```

Mean:454.190 Standard deviation:111.205

95% confidence lower bound:447.298 upper bound:461.082

9 Markets

LEAF global utility

Global utility values recorded: 785 819 802 775 784 799 774 779 831 787 767 749 824 748 810 828 807 832 847 802

Mean:797.450 Standard deviation:27.843

95% confidence lower bound:784.081 upper bound:810.819

LEAF learning convergence

Number of days taken to converge: 108 108 106 107 107 112 105 113 110 106 105 125 106 106 105 107 107 106 106 105

Mean:108.000 Standard deviation:4.577

95% confidence lower bound:105.802 upper bound:110.198

Self-interested global utility

Global utility values recorded: 476 294 745 378 474 293 727 477 390 380 380 748 482 754 468 480 482 475 485 391 818 477 354 485 373 397 494 298 469 466 391 477 387 751 389 496 460 457 481 392 383 483 390 298 468 386 477 378 374 830 384 495 390 478 469 453 483 475 765 489 470 475 390 455 487 482 737 460 390 378 477 377 382 813 368 383 741 490 455 471 383 481 388 468 493 490 370 392 748 719 477 300 473 485 386 389 485 393 369 490 467 756 384 395 465 475 491 458 743 391 294 478 469 381 473 761 390 845 492 489 367 381 460 395 739 732 473 682 369 379 471 479 820 882 863 476 480 862 764 852 489 468 737 388 484 383 494 453 393 445 372 760 731 370 388 397 391 495 375 382 474 480 834 387 396 478 386 839 391 813 776 385 393 393 388 838 488 831 482 756 758 840 745 288 489 483 459 376 845 376 389 481 463 481 484 842 825 395 377 742 370 385 488 827 399 442 834 478 482 395 388 390 489 286 473 471 380 479 478 395 859 822 484 487 476 395 288 388 744 825 474 475 359 489 860 471 288 801 872 467 477 387 819 673 487 387 799 480 387 858 484 384 841 400 384 485 492 397 374 492 452 473 392 475 370 483 376 469 386 483 453 481 389 467 434 495 481 459 291 480 488 485 475 487 761 828 466 480 392 458 476 379 383 474 444 761 485 845 495 393 481 487 465 389 492 489 491 480 855 453 487 474 485 391 486 390 295 490 759 846 802 479 478 387 481 389 389 392 392 385 479 482 470 476 479 394 489 475 390 474 295 385 494 834 488 477 776 488 393 772 458 389 728 394 477 457 388 378 824 465 480 748 393 468 389 385 846 778 443 759 851 805 492 469 494 458 484 377 391 472 482 476 466 739 491 374 737 390 387 873 483 475 387 390 870 486 380 837 762 827 459 483 489 471 395 395 494 483 457 713 483 832 456 478 282 487 393 387 478 387 852 478 472 393 736 389 478 838 388 395 387 489 490 397 473 459 292 479 400 821 491 454 441 391 481 834 393 385 296 469 460 467 486 388 747 391 771 492 473 487 830 840 481 489 485 733 484 383 489 370 394 454 477 733

```
493 362 482 488 463 843 468 300 480 823 489 495 489 378 380 485 493 385 395 467
394 392 390 389 460 474 477 820 394 364 465 479 465 466 483 382 656 373 779 477
292 459 857 492 383 490 483 477 476 388 480 665 487 485 481 494 362 494 493 491
840 388 468 740 471 372 268 354 844 467 394 485 863 480 481 467 474 468 479 771
824 290 386 692 392 488 386 391 473 475 475 487 385 484 395 483 468 767 394 283
389 868 740 387 476 384 373 705 479 476 487 466 387 443 475 299 389 378 367 482
386 387 485 485 481 293 489 745 291 492 396 473 841 479 483 485 385 388 386 480
489 484 483 485 864 480 389 762 468 478 474 386 397 484 760 482 747 465 856 392
485 389 488 482 361 280 470 481 489 387 481 462 389 813 730 493 295 765 769 472
464 477 290 477 479 495 474 844 839 490 476 847 481 719 385 755 472 474 365 746
734 471 836 366 733 372 490 467 388 781 363 739 478 482 482 476 490 845 472 370
454 750 379 490 481 816 465 481 489 383 397 475 741 487 476 479 490 461 488 472
470 489 382 476 854 476 495 455 737 298 366 485 379 855 475 481 471 482 756 381
477 372 490 369 351 475 293 380 385 486 376 479 372 770 390 747 379 467 748 486
490 383 397 487 385 477 471 832 373 468 474 474 481 480 478 385 487 489 384 479
474 741 375 393 379 676 473 391 382 461 841 397 485 484 447 490 372 482 460 374
839 490 471 381 485 382 368 473 477 477 479 371 480 293 472 848 383 483 449 807
397 289 475 389 381 838 485 469 485 485 828 471 479 388 749 394 285 736 481 484
824 361 383 856 380 389 455 380 486 365 381 706 484 385 459 444 374 388 492 298
473 481 374 844 395 385 471 843 461 477 386 743 496 847 835 374 378 395 395 473
480 479 477 825 472 477 468 475 837 469 491 483 388 386 462 751 457 380 444 465
451 394 841 477 467 371 489 460 385 393 826 757 495 482 287 842 475 484 492 298
380 470 385 491 865 395 490 463 455 687 449 391 387 486 799 392 838 473 471 492
368 393 381 475 477 474 462 477 395 292 487 382 483 484 387 393 471 495 437 853
482 483 477 480 490 466 840 437 740 289 490 471 389 396 476 487 483 483 386 383
763 487 389 481 756 472 454 834 298 485 470 473 459 780 365 855 716 752 389 485
496 469 380 383 394 372
```

Mean:502.257 Standard deviation:149.571

95% confidence lower bound:492.987 upper bound:511.527

10 Markets

LEAF global utility

Global utility values recorded: 823 864 922 892 862 861 906 898 898 910 928 945 931 885 865 880 899 922 933 904

Mean:896.400 Standard deviation:30.636

95% confidence lower bound:881.690 upper bound:911.110

LEAF learning convergence

Number of days taken to converge: 107 106 115 124 107 106 105 109 105 123 106 105 104 110 110 105 109 107 104 106

Mean:108.650 Standard deviation:5.706

95% confidence lower bound:105.910 upper bound:111.390

Self-interested global utility

Global utility values recorded: 474 735 476 485 388 480 473 398 479 482 850 486 840 949 755 382 822 480 497 860 461 481 905 368 495 297 484 381 484 849 390 834 480 484 294 860 456 947 297 742 841 493 383 483 459 395 456 387 391 858 463 393 483 486 774 489 875 396 470 380 770 390 462 830 473 392 494 480 486 840 465 961 487 398 394 483 475 867 472 394 396 755 390 451 858 481 481 831 482 808 833 762 364 465 869 477 476 381 485 484 481 472 387 488 470 471 851 297 479 392 374 482 473 495 490 394 377 761 480 842 442 477 395 853 744 489 829 481 386 866 789 854 379 489 386 392 386 823 475 485 467 377 372 392 487 466 895 849 476 845 468 467 455 378 489 488 844 938 469 860 477 491 458 389 385 384 496 468 779 478 479 478 464 485 837 745 482 863 491 900 491 896 837 372 939 376 922 486 343 767 867 490 482 792 463 471 484 385 476 849 384 952 846 487 680 484 385 488 293 746 381 455 463 298 478 388 388 855 479 815 468 797 482 756 460 745 816 475 396 482 475 826 480 377 446 830 395 392 491 777 869 479 471 472 375 795 396 396 366 368 394 462 856 840 858 492 480 481 488 488 478 384 470 482 376 800 946 484 479 386 457 469 456 481 484 485 480 388 927 353 488 393 494 484 462 728 388 388 381 847 392 372 481 752 495 434 290 478 492 472 387 488 833 844 468 377 477 375 462 479 388 844 389 481 487 744 483 471 363 475 472 286 383 467 755 393 389 464 392 483 482 814 823 392 490 834 747 390 846 829 653 479 452 478 828 471 465 851 372 944 774 953 472 370 853 492 930 382 491 475 473 482 291 488 364 470 490 787 829 854 385 467 465 866 393 872 481 485 479 476 390 489 836 776 475 804 294 755 484 489 377 491 881 483 477 294 399 388 488 485 749 477 491 745 449 493 829 485 921 288 398 485 494 770 471 915 380 457 482 481 356 392 382 473 851 937 373 823 953 465 823 835 476 491 787 939 847 770 852 847 954 845 490 919 473 471 492 839 392 749 492 396 482 495 375 491 907 454 484 478 475 454 489 491 398 381 382 398 465 396 392 488 393 475

```
473 850 487 469 488 363 931 489 395 484 489 490 939 394 457 468 486 385 482 467
833 490 488 380 838 388 480 478 394 470 480 496 490 485 751 483 471 487 763 485
382 494 490 389 430 385 394 466 483 392 945 290 486 493 389 863 469 471 456 482
444 777 487 454 935 394 850 731 477 474 382 482 477 485 397 936 475 862 475 480
487 824 493 479 476 466 744 394 496 479 842 474 485 805 480 766 754 376 391 393
476 461 470 370 472 473 383 846 479 380 924 385 476 480 382 475 845 874 485 478
862 488 808 812 471 462 469 493 808 481 390 385 845 930 465 855 457 464 488 492
813 473 470 470 486 786 477 761 466 866 483 745 770 707 463 463 383 460 488 474
922 729 954 864 389 840 744 462 924 389 860 482 284 392 465 478 386 389 376 485
476 469 485 803 391 487 469 386 816 382 384 372 486 465 455 460 871 941 479 495
384 485 784 485 484 467 483 483 471 391 492 463 480 450 492 477 465 489 858 472
854 946 475 466 643 491 494 474 394 466 881 289 457 490 390 486 486 851 391 485
472 777 385 810 851 471 394 821 488 836 463 494 919 474 793 462 480 474 377 388
470 848 738 376 814 473 862 846 924 472 481 386 485 299 486 388 811 467 927 497
393 832 394 482 844 484 484 477 486 378 487 480 490 819 483 491 493 491 459 931
490 483 478 886 476 831 423 459 459 872 493 377 395 366 464 858 468 494 484 481
852 392 387 481 965 484 481 950 489 766 474 455 361 457 869 840 459 852 483 487
478 849 873 943 837 470 494 476 386 390 481 760 489 483 298 392 482 480 480 394
750 415 390 844 449 471 375 481 489 838 838 478 750 802 491 490 477 377 383 489
483 447 840 475 473 457 384 487 867 488 486 392 773 941 471 486 395 485 470 734
472 820 459 483 477 485 461 485 463 857 471 827 392 490 841 487 490 390 470 489
908 448 489 493 485 853 458 496 485 815 839 808 765 469 473 767 464 481 475 483
489 388 482 386 483 486 391 742 482 475 470 762 852 743 467 838 390 463 376 462
904 771 380 477 856 755 882 756 835 836 299 470 394 878 765 493 379 394 293 828
478 739 484 834 485 852 468 477 488 485 473 461 839 384 453 481 468 483 466 493
479 436 881 477 473 949 878 483 385 481 388 458 382 863 468 929 472 863 943 381
481 375 473 385 375 818
```

Mean:555.297 Standard deviation:182.3836961

95% confidence lower bound:543.993 upper bound:566.601

11 Markets

LEAF global utility

Global utility values recorded: 965 921 999 987 976 982 1033 910 1004 956 917 977 943 895 878 900 944 972 964 996

Mean:955.950 Standard deviation:41.471

95% confidence lower bound:936.037 upper bound:975.863

LEAF learning convergence

Number of days taken to converge: 112 106 106 123 105 141 133 124 104 105 111 105 108 105 116 132 121 113 117 106

Mean:114.650 Standard deviation:11.037

95% confidence lower bound:109.350 upper bound:119.950

Self-interested global utility

Global utility values recorded: 730 484 478 946 488 491 953 912 847 394 472 396 846 481 478 487 473 907 908 394 461 492 943 742 481 757 927 394 484 470 390 859 346 385 476 483 380 397 497 451 761 481 379 387 488 763 861 921 814 486 484 485 474 488 393 858 772 482 480 858 380 477 475 877 392 483 835 949 953 493 460 471 389 398 485 463 471 476 849 466 381 476 386 393 955 841 1059 485 824 794 949 392 491 763 944 825 485 948 485 932 477 832 469 381 746 466 483 855 492 376 1010 468 489 879 967 735 854 466 948 781 957 474 395 942 482 479 465 483 493 433 492 472 850 837 490 832 381 481 377 491 371 845 491 365 475 481 483 396 383 464 386 484 469 1017 488 473 377 473 489 390 488 959 473 389 387 861 863 486 394 460 928 744 357 398 930 942 846 837 859 921 762 485 475 467 395 952 860 487 886 475 482 498 392 848 854 479 488 861 390 468 747 475 480 954 472 489 481 493 488 389 844 474 491 479 732 474 488 486 942 466 488 463 921 376 776 481 467 472 392 472 483 485 481 493 937 490 391 827 391 488 492 833 857 388 932 853 759 372 482 922 472 770 741 481 483 483 482 468 372 849 299 940 473 844 476 476 480 479 474 945 482 463 947 470 384 772 385 935 485 367 483 373 398 496 398 838 391 488 475 869 487 379 468 959 818 380 487 478 1047 489 471 487 475 493 387 372 763 478 468 705 366 488 913 470 955 485 488 484 489 915 907 471 804 960 484 479 476 468 483 493 937 778 489 478 934 840 395 482 481 385 735 395 389 398 831 385 855 484 861 394 491 924 488 481 932 478 481 391 467 487 487 487 915 394 852 493 493 944 936 487 377 846 738 936 480 831 859 478 493 490 1059 478 856 857 800 487 480 463 480 875 820 739 394 298 466 374 481 476 480 484 487 471 391 828 480 471 494 388 463 485 490 462 385 851 757 371 487 862 485 914 496 931 391 484 488 753 907 825 948 474 396 475 495 472 741 482 389 467 396 466 833 866 483 455 298 971 390 480 949 928 478 811 817 769 476 763 389 490 479 834 385 474 390 896 857 467 480 824 887 898 754 366 813 478

```
842 479 480 942 490 493 398 467 495 840 380 491 759 852 384 384 483 482 467 952
863 833 845 879 483 864 957 841 479 492 396 941 393 944 385 476 825 939 927 385
471 477 909 490 982 864 838 770 489 465 392 497 394 1025 491 471 393 475 933 475
843 773 489 477 394 297 484 398 491 469 466 861 387 462 397 943 471 840 866 389
488 851 385 391 836 486 463 961 837 388 495 964 485 376 853 939 756 815 387 491
379 488 466 487 839 938 832 855 485 473 963 380 483 861 491 446 853 817 957 840
933 490 384 480 479 815 484 480 493 485 491 457 865 487 480 387 484 838 846 908
914 487 867 392 480 927 482 473 828 921 934 473 394 742 490 484 451 491 421 296
490 293 475 873 462 479 843 477 857 959 856 935 847 488 847 480 457 490 871 492
898 487 852 446 472 837 477 492 479 948 473 378 498 483 390 825 479 951 873 943
857 490 937 844 459 479 913 490 387 464 286 479 470 486 481 856 495 483 843 759
491 964 972 866 743 868 378 296 389 925 834 488 468 480 498 857 838 951 469 481
857 743 480 496 456 465 461 853 391 469 477 482 474 294 481 854 289 392 921 867
490 494 469 488 958 467 491 918 829 473 493 946 398 466 927 482 738 944 483 480
468 473 485 471 487 928 491 869 731 843 482 756 475 379 876 959 929 482 375 380
474 490 1032 929 496 918 839 478 481 865 382 801 845 910 939 485 939 389 489 396
745 481 936 365 392 484 490 474 487 382 435 394 493 468 485 914 1029 490 1012 473
487 960 1042 381 489 493 487 492 948 483 492 480 490 496 499 475 486 473 451 387
908 487 459 473 874 475 917 462 489 489 493 397 490 948 471 475 480 900 397 485
377 455 469 833 486 472 386 488 869 471 484 479 487 476 489 392 943 1055 389 931
387 476 380 860 869 368 482 476 394 388 489 477 886 386 783 945 466 474 490 485
480 476 884 916 484 850 833 399 851 394 398 907 486 1036 478 474 919 397 385 858
387 483 855 492 867 397 471 393 383 832 395 480 852 783 949 457 488 782 495 387
955 955 473 463 931 830 823 919 747 398 484 945 814 460 363 491 952 939 489 464
389 488 490 390 393 957 493 399 484 479 489 859 389 936 833 483 482 936 492 476
901 490 471 391 478 863 724 484 394 379 905 469 374 933 488 485 479 478 487 489
492 807 833 467 482 488
```

Mean:601.942 Standard deviation:209.244

95% confidence lower bound:588.973 upper bound:614.911

12 Markets

LEAF global utility

Global utility values recorded: 999 1094 990 991 1058 1053 1085 1040 1053 1042 1036 1009 1092 918 1016 1081 1041 1072 1012 1019

Mean:1035.050 Standard deviation:42.779

95% confidence lower bound:1014.509 upper bound:1055.591

LEAF learning convergence

Number of days taken to converge: 113 111 107 106 134 128 106 105 126 127 109 115 142 106 112 136 141 104 108 125

Mean:118.050 Standard deviation:12.972

95% confidence lower bound:111.821 upper bound:124.279

Self-interested global utility

Global utility values recorded: 767 855 491 481 394 467 1037 865 479 476 384 952 490 1028 389 862 473 477 826 375 399 473 1114 486 479 393 932 381 933 947 384 486 844 489 878 477 492 968 490 855 466 488 483 483 976 495 381 935 487 479 933 932 965 958 485 471 480 473 491 914 481 937 386 489 967 480 483 489 884 872 996 934 394 390 393 839 478 1030 919 491 876 477 489 495 493 924 942 937 856 848 488 398 457 449 464 941 495 947 479 758 841 1029 907 486 497 467 858 479 478 490 871 480 872 487 1049 932 459 1115 475 485 839 489 941 466 1030 465 488 486 473 926 394 375 393 955 491 481 470 879 480 454 835 484 841 486 392 756 482 926 393 483 394 486 1054 923 918 476 875 974 472 488 475 392 360 399 485 492 481 390 873 940 935 836 483 495 898 471 472 494 906 955 488 935 928 467 395 363 946 958 941 988 868 481 495 736 935 1023 488 482 483 933 488 482 928 955 852 847 492 468 478 856 956 759 489 488 1151 479 392 480 893 457 395 941 482 460 950 923 844 483 955 914 866 494 388 478 384 935 962 391 957 492 935 964 481 481 845 477 849 484 849 954 1028 474 490 391 955 388 384 476 484 972 953 389 494 487 481 483 394 485 929 1004 487 493 1052 957 868 490 864 395 964 867 834 967 481 387 861 943 1012 810 359 493 737 478 657 1032 480 392 1044 1024 774 1010 470 493 952 488 382 832 490 480 949 1031 481 929 482 484 492 962 946 957 495 497 918 491 490 448 1002 868 934 836 480 475 392 1034 1018 489 1112 486 846 747 825 868 762 845 385 919 1110 484 478 491 488 391 388 952 478 959 953 492 483 486 830 482 481 478 452 477 957 851 851 862 465 869 865 926 479 498 479 495 477 483 1045 398 853 455 465 944 488 386 373 488 863 495 753 475 847 843 488 356 296 378 486 764 485 826 926 908 1013 868 480 865 488 951 479 496 475 493 389 392 492 385 917 485 932 471 932 943 486 490 479 949 940 985 868 859 387 974 452 480 937 943 971 485 464 478 484 925 1049 387 450 913 496 492 826 1050 496 492 492 479 394 937 488 906 856 854 463 478 873 1022 867 489 480 955

```
861 958 940 475 831 1010 384 397 771 495 956 955 397 388 853 484 872 852 994 929
481 807 494 470 487 496 469 470 385 489 759 1041 472 832 470 832 944 1025 468 951
483 484 382 945 859 950 908 495 850 847 1016 873 476 454 493 841 467 934 396 755
974 1022 1047 919 945 843 960 377 494 491 397 935 952 879 945 747 945 384 486 859
477 928 959 944 842 869 485 434 473 480 840 486 1049 835 492 389 491 482 292 376
937 489 961 1013 481 464 498 1008 967 936 491 485 498 469 859 844 858 868 487 862
1004 488 489 490 862 472 391 381 928 465 915 875 867 894 395 488 494 913 456 475
487 935 838 847 388 469 859 482 398 482 851 454 975 932 483 490 451 383 938 765
461 849 493 1042 943 457 1040 465 476 944 854 942 487 828 486 755 934 492 485 467
1047 367 493 488 975 481 481 943 490 841 484 1032 1004 949 848 466 483 756 396
808 932 390 977 482 482 969 486 489 484 1049 772 486 482 471 390 846 971 470 392
494 477 470 481 482 479 835 398 456 954 469 491 390 959 936 384 482 481 393 474
484 464 836 295 479 473 474 482 489 492 389 952 860 497 392 754 920 948 492 487
470 491 1021 494 384 926 852 483 484 490 1026 909 478 388 395 394 283 1046 1038
1072 381 492 956 465 869 823 952 918 488 460 472 487 943 491 979 478 954 767 395
386 934 395 393 462 483 977 482 489 1056 470 862 492 1149 479 473 390 1041 489
944 473 388 483 491 484 865 822 487 979 466 829 950 866 400 484 400 493 844 1041
474 958 494 916 494 775 958 394 480 390 932 1025 460 492 865 931 432 455 489 468
941 941 832 857 468 974 480 478 968 383 878 470 921 490 494 929 384 918 945 482
479 941 495 962 465 480 1000 454 385 488 358 491 973 461 952 390 923 489 479 484
967 391 488 491 398 484 875 918 475 461 892 1023 1043 907 465 871 950 486 467 485
494 855 488 491 481 932 960 874 484 923 486 478 378 466 475 487 491 488 397 484
1008 850 822 895 905 1023 477 480 484 921 490 489 487 1064 392 484 864 463 482
477 398 486 494 483 935 959 488 464 929 390 744 488 490 483 938 472 1024 463 924
490 910 381 468 955 486 479 479 486 963 482 477 957 462 489 491 951 846 489 845
485 879 955 390 486 969 294 494 385 380 943 867 481 477 469 1025 375 474 900 490
866 478 900 393 863 939 395 930 857 923 1029 938 489 819
```

Mean:661.007 Standard deviation:235.47

95% confidence lower bound:646.413 upper bound:675.601

13 Markets

LEAF global utility

Global utility values recorded: 1157 1160 1044 1184 1239 1171 1203 1229 1142 1116 1049
1190 977 1161 1170 1177 1207 1196 1095 1153

Mean:1151.000 Standard deviation:65.907

95% confidence lower bound:1119.353 upper bound:1182.647

LEAF learning convergence

Number of days taken to converge: 143 120 105 109 115 106 108 105 107 112 131 129 111
130 111 106 107 104 104 108

Mean:113.550 Standard deviation:11.124

95% confidence lower bound:108.209 upper bound:118.891

Self-interested global utility

Global utility values recorded: 478 961 490 1032 465 1024 842 946 763 969 1018 943 495
1031 902 473 991 1012 473 953 490 492 485 491 843 469 841 957 1013 858 925 963
491 935 486 493 483 479 972 474 496 490 1067 494 1036 445 473 869 487 1058 489
483 1022 950 490 922 922 841 867 493 962 829 479 490 856 939 982 477 494 963 954
484 490 493 484 1171 493 948 395 494 465 461 497 466 857 1026 381 970 484 480 947
484 974 478 949 1021 480 1050 929 833 482 386 480 919 494 488 489 955 965 893 938
493 1066 488 819 868 494 967 1230 387 479 871 940 930 486 894 486 731 491 936 930
963 463 1000 1035 853 390 942 1006 948 960 973 970 944 478 476 1013 943 931 488
397 484 491 1016 905 483 1018 483 475 958 858 944 915 491 479 816 488 937 387 495
954 1018 493 399 486 472 484 851 491 948 854 475 484 1049 1040 1205 935 917 1027
955 454 865 840 486 924 1009 485 385 957 859 1048 384 490 940 491 928 858 1027
954 480 934 921 859 485 487 927 484 1004 386 389 1021 1054 488 1018 388 936 1056
390 391 954 398 956 494 952 921 1002 492 1034 488 901 924 820 931 481 488 935 847
478 494 964 841 487 493 941 385 745 490 831 474 981 962 949 489 867 923 924 491
490 865 1028 871 953 485 920 912 477 471 397 961 964 470 860 494 962 483 1034 491
864 473 488 942 1041 487 487 484 839 852 485 948 375 485 944 490 1044 900 1226
1023 493 473 490 479 765 482 940 482 1028 934 1031 483 1054 957 962 945 929 942
398 393 489 484 483 1042 481 472 483 947 1006 480 956 471 484 495 1021 922 862
458 1177 484 491 858 950 1046 483 974 458 450 1041 940 494 920 816 930 484 488
467 846 835 493 386 295 982 1048 1214 864 491 954 852 1038 470 495 1028 466 495
483 484 1045 946 480 876 374 474 1051 490 493 840 857 490 477 1053 472 950 1037
1032 850 955 386 486 485 485 856 944 391 363 490 484 396 947 481 489 493 870 869
482 959 770 487 849 1046 486 489 737 492 488 852 1029 488 495 830 936 946 491 495
926 494 490 857 491 932 391 488 494 1045 940 481 390 487 926 492 937 493 1037 488

```
945 393 1048 490 942 850 851 1026 488 479 480 858 1034 471 488 816 469 478 1023
484 470 1014 1025 487 947 487 494 922 954 488 490 909 851 396 484 482 873 392 486
1039 1042 1039 962 1069 483 1020 982 488 941 482 853 384 386 959 731 862 480 942
460 866 476 468 948 497 388 496 482 485 934 846 476 1018 498 946 1060 490 931 930
929 955 490 1063 874 485 480 479 477 859 1049 833 484 390 1058 941 393 1021 1052
867 926 1026 906 839 488 843 1070 966 491 1028 867 393 763 479 951 848 964 937
1022 473 920 486 483 870 1015 495 397 1038 492 481 951 1062 489 939 498 486 780
470 1010 1019 1036 975 958 1023 396 474 486 491 486 850 1015 865 1027 970 891 926
945 1030 861 387 951 437 492 487 467 944 869 460 846 940 396 487 1006 1047 388
963 926 479 854 482 489 1057 930 479 960 496 395 1030 1035 484 487 820 480 955
1046 874 1033 953 954 495 1003 850 936 1060 955 475 380 874 959 394 902 474 486
461 473 1025 480 1076 491 479 396 395 949 459 485 482 856 827 392 871 481 484 461
931 495 491 939 395 478 957 1031 951 398 946 1014 479 497 479 944 479 480 968 936
489 956 1014 390 486 486 479 937 484 912 394 474 1038 1031 930 755 491 1007 965
937 392 947 492 1029 1055 963 458 477 925 931 491 482 1238 448 496 1038 944 447
497 394 478 466 474 477 481 376 375 875 486 950 1041 963 956 390 841 483 931 965
379 483 480 490 392 491 850 481 1032 497 964 481 927 971 962 941 488 475 918 966
464 955 992 480 937 1044 466 863 933 834 489 846 487 484 485 487 1022 467 756 482
494 1036 484 930 485 390 1003 948 953 1022 953 1052 949 943 455 489 482 842 924
497 453 947 489 1038 473 484 979 442 945 963 976 874 486 957 482 843 467 842 944
490 496 967 491 1158 925 480 469 487 1059 968 868 462 396 470 859 947 942 844 475
956 481 491 844 869 937 1029 922 861 840 393 922 476 492 952 485 1205 1080 466
485 482 843 475 490 942 938 490 920 1046 447 481 972 480 485 492 940 952 965 1034
475 297 484 957 934 472 960 483 961 493 953 483 865 482 397 467 473 1037 942 834
1036 873 1032 484 956 1045 1040 466 967 473 1056 384 1045 378 964 493 500 968 876
927 1240 392 396 906 477 485 480 842 476 1050 470 1016 1023 855 482 866 482 947
894 937 461 953 487 490 963 1061 748 967 476 1015 490 473 492 1062 386 478 485
493 904 866 477 1003 819 456 1010 934 486 478 486 745 850 477
```

Mean:723.277 Standard deviation:248.283

95% confidence lower bound:707.889 upper bound:738.665

14 Markets

LEAF global utility

Global utility values recorded: 1247 1249 1250 1267 1240 1237 1253 1219 1241 1251 1275 1021 1219 1277 1274 1190 1174 1297 1291 1206

Mean:1233.900 Standard deviation:59.369

95% confidence lower bound:1205.393 upper bound:1262.407

LEAF learning convergence

Number of days taken to converge: 106 106 105 107 111 110 107 105 109 107 107 106 115 107 105 124 108 105 105 107

Mean:108.100 Standard deviation:4.483

95% confidence lower bound:105.948 upper bound:110.252

Self-interested global utility

Global utility values recorded: 1013 478 953 476 920 1053 484 482 1018 882 491 1034 493 482 473 952 1014 1349 489 948 485 391 395 1007 963 1058 493 1039 1056 494 1036 1036 485 1069 962 950 869 947 984 484 489 487 1233 477 491 988 487 483 1010 956 1048 941 483 968 1044 490 938 1030 1039 490 874 497 957 942 938 971 1035 951 972 395 939 947 946 947 964 491 1054 393 1012 936 481 931 1038 478 492 368 859 465 494 484 490 463 950 951 481 942 1055 1041 1026 479 970 937 488 954 382 1034 488 470 1044 934 393 1050 964 396 1067 371 1286 487 954 462 492 494 1030 975 946 838 1339 1042 1264 972 939 1041 1313 493 928 471 1008 482 819 1238 925 392 943 482 488 938 480 482 497 492 1032 491 469 942 485 931 941 387 496 487 1013 846 472 474 493 470 481 474 942 1052 956 940 398 397 379 931 1025 483 912 492 486 1348 485 948 1020 498 486 393 952 495 944 863 844 297 959 1066 1045 943 931 487 939 959 855 1262 938 1061 1041 1047 1050 857 464 963 999 944 489 921 841 951 400 488 493 473 397 823 957 978 396 476 1054 851 1058 1002 1026 490 943 861 494 497 394 1043 1047 828 1023 487 489 954 487 921 960 863 960 474 960 470 960 1256 945 934 1004 479 1290 1045 940 957 959 484 1027 1057 484 920 491 1045 472 1028 965 500 992 491 491 1039 481 994 853 495 1033 961 950 493 1050 491 478 496 864 469 489 395 496 1066 1224 932 1062 481 967 390 977 865 1331 927 494 449 486 1001 917 398 948 966 481 929 948 492 495 842 976 489 396 491 481 956 937 464 968 1285 1323 971 484 958 969 958 948 473 454 495 1016 941 1043 395 1056 494 836 487 826 476 394 1023 487 953 484 1047 462 853 485 478 489 874 393 1039 1022 946 1035 394 1049 892 492 933 490 493 1033 474 982 393 470 397 482 499 486 471 1037 489 481 877 455 848 493 963 497 858 828 971 962 955 957 1064 496 483 941 1043 479 482 488 1015 964 491 1029 390 1017 498 485 485 1077 944 488 1037 952 970 978 920 864 973 852 475 995 960 904 1044 975 864 1225 947 374 494 487 1023 1007 950 482 881 935 498 919 386 818

```
1047 862 871 918 490 923 1031 956 955 930 1035 857 476 485 470 392 962 966 488
497 1034 476 473 864 1056 493 839 1046 1020 487 477 933 934 938 767 940 1030 864
496 852 471 483 490 1019 481 487 849 494 845 1042 939 477 495 1037 1043 968 950
955 490 784 394 950 494 472 1049 498 481 847 475 720 380 963 908 481 944 479 492
952 928 1057 472 1034 834 480 1018 483 1043 495 915 391 958 491 496 390 832 458
380 484 493 478 936 1057 932 492 390 1027 868 877 1067 984 493 949 396 1068 1313
973 953 976 849 970 931 485 480 484 1029 387 931 868 1070 485 495 487 482 1031
952 1227 1057 480 954 824 866 1032 1022 964 495 963 489 1275 482 1037 479 497 470
836 490 957 843 1013 944 476 1045 1026 447 840 1004 483 1043 1022 486 1029 385
970 980 378 950 865 985 862 1021 1047 490 467 876 479 469 1047 966 931 390 494
873 955 1034 475 875 961 970 394 461 481 486 949 947 929 483 1012 1040 1028 474
481 946 485 481 479 902 764 950 919 956 473 838 486 945 460 490 456 475 1024 488
918 955 964 1009 854 482 1055 918 490 1016 1046 1030 936 858 476 950 997 846 489
480 1039 1028 1036 469 849 487 967 476 1028 486 479 864 869 958 488 484 487 971
474 478 486 397 1223 486 493 869 854 1041 875 949 486 875 971 1035 1033 1275 394
866 389 495 484 1019 953 484 483 955 1037 481 936 1025 957 1017 1028 481 968 485
958 873 467 1054 1226 933 481 863 1025 495 480 397 953 484 966 488 964 1039 1049
477 475 496 962 459 1035 991 497 492 480 489 948 484 479 925 386 947 390 477 494
871 855 1021 1347 972 463 487 1151 1034 1035 479 1218 1264 1045 845 488 479 970
1044 1014 485 867 1050 389 1216 856 395 474 1323 489 945 939 472 866 482 486 492
479 1052 930 1039 849 478 933 484 1321 485 1037 484 925 951 1034 870 1053 485 478
1034 959 481 478 488 949 1020 471 466 1050 979 484 484 458 943 481 926 479 479
484 894 861 485 937 1023 928 480 471 488 486 896 1037 923 495 853 1034 1043 1060
1033 863 1063 483 493 490 975 958 952 919 478 1077 466 476 490 1217 481 492 477
1293 944 480 985 488 1038 947 496 1046 1075 1058 483 492 478 743 493 914 491 937
498 767 905 486 913 962 1075 387 949 473 489 933 488 1054 489 926 1046 843 1071
996 492 935 486 942 486 945 497 947 1190 487 457 951 925 494 488 906 937 955 475
479 992 468 878 490 940 396 969 1055 474 933 390 1309 914 481 493 491 1038 1268
487 467 497 479 872 1019 1020 1217 483 946 474 1036 480 962 492
```

Mean:765.379 Standard deviation:264.008

95% confidence lower bound:749.016 upper bound:781.742

15 Markets

LEAF global utility

Global utility values recorded: 1341 1347 1387 1407 1380 1288 1397 1000 1339 1376 1383 1238 1363 1329 1269 1380 1348 1388 1287 1393

Mean:1332.000 Standard deviation:91.032

95% confidence lower bound:1288.290 upper bound:1375.710

LEAF learning convergence

Number of days taken to converge: 121 118 106 108 127 121 106 106 107 104 117 109 129 111 122 107 104 110 105 105

Mean:112.150 Standard deviation:8.113

95% confidence lower bound:108.254 upper bound:116.046

Self-interested global utility

Global utility values recorded: 974 489 1037 488 497 488 470 929 945 871 887 484 1035 959 1052 966 940 1047 977 477 467 954 1039 473 499 493 481 1076 1312 1022 492 980 930 875 1056 963 493 1058 395 483 495 496 490 484 482 487 480 1365 953 484 395 947 968 1379 925 1032 1074 1012 941 494 480 397 486 964 498 1056 493 1401 490 871 1066 1316 1048 1049 852 948 1082 467 495 482 472 1045 468 1331 469 486 1034 959 479 1021 1045 849 492 486 1026 1049 484 1210 476 1043 1036 1055 1088 494 1029 474 859 936 492 484 861 940 1081 497 1059 1033 852 1035 491 1037 480 465 945 495 947 955 959 950 1405 890 494 962 948 1041 1023 1044 1045 1047 485 952 1037 1301 1013 917 1031 497 1048 962 488 933 1331 1320 494 947 1038 476 1059 495 918 958 483 1036 836 956 1045 972 921 1063 1036 474 1377 465 496 487 1285 486 487 989 397 951 474 490 963 382 936 493 956 904 958 1036 1051 1417 392 946 1045 941 1039 1045 491 950 1038 1020 958 963 388 1403 1021 1042 481 488 1047 1045 869 481 1039 864 919 475 1030 480 1027 840 488 472 486 951 377 1407 923 1042 495 1006 495 970 1335 482 1415 478 950 1040 1074 1032 471 490 1047 480 946 1053 467 1053 958 495 1267 489 1026 936 494 950 477 493 487 942 478 488 1012 951 382 1044 490 484 1406 490 1064 959 877 966 1076 1013 1452 488 1199 1054 486 477 490 1040 942 1035 1067 940 878 960 473 1012 1057 933 480 381 491 1058 1028 1054 1074 1055 487 932 959 493 388 953 863 490 1018 939 1034 921 857 479 983 1317 1230 1061 490 1050 496 488 1045 1054 973 941 483 460 1053 1055 1381 1047 963 1049 1031 1038 951 480 954 1046 1404 1312 939 1044 939 488 959 940 1039 391 936 485 489 1425 484 879 480 1034 1036 483 838 1042 490 957 1060 472 488 489 388 477 484 965 488 940 1041 487 861 942 1025 1036 484 967 494 949 486 469 1062 967 483 1050 495 940 475 484 1027 1058 1061 480 495 1310 491 1028 392 1299 924 1005 482 957 478 1018 486 1265 1045 965 493 1056 955 944 944 1024 477 934 1010 1041 488 955 927 488 1061 1038 1411 928 1023 1048

475 478 396 391 1009 480 977 500 499 1371 947 999 1296 485 941 942 1057 950 967
980 487 930 494 488 464 491 959 491 1010 1060 952 932 459 493 486 493 1291 1053
495 1388 871 1061 495 489 488 966 1014 1035 954 479 394 1282 948 381 1288 974
812 1035 967 482 959 1024 916 1045 1303 489 1324 968 985 469 1052 479 967 490 963
483 976 475 931 1057 496 482 491 1301 1051 994 388 938 383 486 951 489 1050 1032
1040 389 1045 1045 1047 1033 486 981 498 936 970 482 491 390 1045 875 960 493 488
944 943 1023 1066 490 955 1048 1040 953 983 1039 1026 473 946 479 1385 992 1045
477 930 965 1375 496 869 391 962 1054 1054 944 1370 1029 1334 939 948 384 1046
1008 933 829 1046 1014 1050 999 489 934 1054 1041 939 491 487 472 997 911 1067
470 1418 1386 955 479 850 943 474 500 489 1034 492 1059 394 484 485 496 939 480
999 490 958 482 492 1019 478 944 480 980 952 931 1051 884 1063 1062 1239 486 1071
966 486 480 949 1058 472 1051 942 995 492 1052 468 942 399 869 496 1046 488 1049
490 1041 1069 469 944 480 863 1360 1328 1039 480 1028 1366 962 1313 486 1066 495
983 993 480 486 384 495 493 998 1056 967 490 1042 1021 1417 488 1041 1019 1044
1071 395 467 488 940 491 1389 1394 949 471 458 954 975 478 955 1054 964 955 1317
1066 1047 950 1053 928 1019 1039 492 1028 910 865 487 490 485 493 398 853 1255
1424 983 1039 1035 966 953 1026 1061 1078 1062 950 484 927 979 946 1328 857 972
1022 497 479 1034 944 946 1034 1007 395 1023 490 955 489 1046 939 495 930 487
952 489 950 856 948 488 962 1050 1307 1071 1048 460 482 1050 486 959 484 486 1061
944 1062 945 484 479 967 1019 482 978 398 958 876 1073 486 972 1053 480 487 1050
489 1331 952 963 1455 939 496 1052 1010 488 1032 932 491 1387 477 1048 1404 489
951 490 483 906 491 929 1071 490 393 988 486 926 947 496 1053 1308 937 962 486
488 385 1071 1068 1072 850 864 1020 968 1363 487 1313 958 1040 1057 1044 926 1056
1197 457 960 941 484 957 489 939 1241 956 476 485 1407 1055 1401 940 1063 481 965
476 478 494 950 848 959 855 941 1053 482 956 1219 474 965 868 1271 926 1048 842
482 1039 492 485 993 492 469 974 976 490 896 1036 496 482 1045 487 493 1027 1319
1051 1034 484 470 483 1063 1055 480 492 1035 1034 1067 948 494 484 942 976 933
1035 475 480 953 483 1029 947 984 1222 480 484 395 1019 869 484 960 491 492 382
1042 495 1053 1031 495 481 469 1017 493 959 487 476 1009 1058 468 446 1060 478
1072 1341 386 956 486 966 472 958 1071 955 1045 388 951 476 487 936 1027 1030 477
486 476 1056 499 484

Mean:827.803 Standard deviation:287.089

95% confidence lower bound:810.009 upper bound:845.597

16 Markets

LEAF global utility

Global utility values recorded: 1397 1443 1454 1429 1383 1434 1459 1392 1478 1489 1472 1392 1413 1464 1433 1428 1455 1384 1032 1454

Mean:1414.250 Standard deviation:95.575

95% confidence lower bound:1368.358 upper bound:1460.142

LEAF learning convergence

Number of days taken to converge: 108 111 111 112 118 108 119 106 120 104 111 106 123 111 116 107 142 108 107 115

Mean:113.150 Standard deviation:8.604

95% confidence lower bound:109.019 upper bound:117.281

Self-interested global utility

Global utility values recorded: 490 871 490 1422 496 941 948 1054 1002 1069 481 971 485 492 493 1062 478 1024 480 1016 1028 1030 1035 945 955 947 1386 486 488 1032 1008 949 951 1042 1044 965 1053 1029 1037 859 486 1264 1027 972 1034 482 1067 496 927 1041 490 478 480 972 850 848 1047 1035 1420 875 484 968 1045 953 1432 1039 1521 922 493 1050 1340 1477 497 1027 960 954 493 1021 1050 486 962 980 474 480 490 1022 1396 1015 1032 948 945 957 1333 935 472 396 940 1048 952 481 488 968 393 1064 1057 1034 485 490 1391 1080 1052 1036 1040 974 1041 1043 948 1028 963 490 471 497 465 395 951 1030 1045 494 953 492 946 1046 497 489 495 480 907 1055 1055 1477 1043 473 486 469 1071 485 1035 486 1058 983 397 489 1028 1304 496 1030 1070 920 949 1056 868 1353 900 481 1369 1027 972 485 491 1400 482 479 975 494 1047 1034 485 1401 928 937 1026 946 1406 1038 393 1086 1398 1021 1061 1055 480 1018 491 965 954 1030 490 1041 1038 953 875 1006 1479 946 1043 498 974 1054 1412 1023 945 951 1060 1068 974 949 489 941 379 483 1402 1397 1048 486 482 489 488 1393 997 978 849 1060 956 1073 1047 486 496 1062 1048 946 1052 1027 479 493 1003 1045 1493 485 977 1513 1080 1034 492 1390 1026 940 1420 931 492 1034 1030 1062 492 1394 488 484 1060 851 1035 1514 497 1005 1034 1046 1062 1381 1030 951 971 959 1024 489 496 1037 1004 962 1047 482 492 494 1036 491 959 956 966 968 491 972 396 975 1009 972 957 475 492 492 475 942 943 1378 1046 1454 1054 1315 494 845 488 1074 1395 482 1026 951 934 1041 1061 491 907 985 487 1036 1039 493 1064 481 1056 494 1422 947 1410 944 495 1338 399 946 1064 473 858 1022 1060 966 1021 487 476 941 878 492 965 495 1041 491 947 1044 486 1058 1004 968 1409 1057 950 941 1036 1438 982 491 489 498 1027 1275 1018 974 1037 1069 1034 483 1383 1037 492 1504 837 494 1342 958 884 876 1044 1432 1029 1052 396 937 1380 960 488 1062 1040 1057 1489 1047 1014 1088 486 480 960 1055 870 957 1024 947 1042 1061 867 493 1250 489 1053 949 472 1059

```
1035 1045 490 1431 955 494 488 1376 1341 1063 398 1052 493 1062 1035 1054 1273
939 482 498 883 1027 962 998 1034 399 486 1061 491 1080 473 471 1412 489 485 498
490 1391 984 907 1056 495 973 1074 494 490 937 1044 1343 855 1427 1063 1028 486
1028 395 479 1075 1048 1043 482 381 1039 476 492 491 957 1048 925 473 1006 1039
493 1434 941 945 398 487 482 481 1302 1032 494 1028 474 1041 947 1299 484 955 493
1051 1043 483 495 1061 1418 482 482 939 976 1432 1085 1014 495 1051 968 863 1054
1038 1471 963 1041 875 957 1031 1547 942 971 1271 1294 1045 489 1057 1321 1331
1275 484 1039 1066 1036 493 494 1055 943 493 1413 492 1055 1032 489 1033 472 1419
1478 1070 1048 490 970 493 498 1037 945 491 1438 969 958 397 877 1417 485 1403
934 486 492 948 1462 938 935 485 1408 1049 1311 487 1354 497 948 483 1040 931 487
474 1024 1059 1035 938 1058 1380 1057 1043 1037 486 1408 1442 963 944 1452 1064
490 1052 956 1036 484 970 1440 1392 494 1529 1399 1019 974 1439 392 978 488 852
1056 1393 485 1054 861 391 954 1040 1034 964 1065 491 1040 1488 874 1062 962 942
871 1070 489 946 478 486 953 992 1081 1048 940 1300 1035 952 1029 937 1444 922
491 491 1032 1066 869 1063 1053 1043 1338 867 1311 494 492 491 1037 1060 1025 490
488 1047 392 1043 1485 1020 978 876 1036 933 942 1052 1049 1377 1038 945 918 395
394 1064 1511 873 487 491 495 945 492 1044 475 938 1422 1426 974 1026 970 1377
479 908 860 486 400 482 865 1029 487 373 391 492 489 1044 482 489 490 1442 477
849 475 481 970 1054 1046 487 1061 491 1036 468 1398 482 492 1048 999 1511 1042
969 1066 953 979 947 1055 1030 1061 1065 952 834 955 1544 1065 1050 962 971 1059
942 392 485 397 1014 962 1050 874 924 1374 934 955 1032 1046 493 967 490 1056
1019 1005 483 494 1031 1429 1400 1056 1045 1386 483 1431 1037 1432 1077 1048 1065
1057 1387 487 498 477 1373 1020 493 955 1082 1055 1041 1052 1078 480 949 952 1022
1017 944 1046 1457 1043 1044 1001 477 1044 487 913 496 490 491 1074 489 960 485
493 1418 499 1006 925 1366 490 932 494 1389 966 1046 855 1039 483 1068 1060 490
490 483 1045 978 954 478 485 954 485 1043 945 492 924 930 492 1022 964 908 488
486 483 483 1401 399 483 1067 1292 947 942 1021 1043 1065 931 964 862 494 491
1052 853 965 1003 484 1051 1324 1066 1057 1070 1066 818 1022 943 1043 400 1038
962 1072 1018 1058 1404 495 481 950 991 487 487 494 1060 986 960 482 1064 496
1044 974 1031 1045 1055 953 1024 1412 873 1047 1064 480 1420 1046 1090 1069 1027
943 848 1031 1067 959 1015 460 938 1046 930 1039 963 1026 1065 1053 491 1037 965
1025 1078 474 1064 1407 468 487 389 1486 946 491 380 491 1011 1051
```

Mean:895.209 Standard deviation:303.267

95% confidence lower bound:876.413 upper bound:914.005

17 Markets

LEAF global utility

Global utility values recorded: 1403 1492 1522 1468 1607 1582 1528 1573 1544 1485 1545 1502 1492 1545 1572 1555 1393 1499 1496 1456

Mean:1512.950 Standard deviation:56.103

95% confidence lower bound:1486.011 upper bound:1539.889

LEAF learning convergence

Number of days taken to converge: 112 109 105 106 126 111 133 106 118 143 134 112 145 108 107 119 114 106 108 127

Mean:117.450 Standard deviation:12.804

95% confidence lower bound:111.302 upper bound:123.598

Self-interested global utility

Global utility values recorded: 486 843 1069 470 965 493 1046 1075 498 944 464 488 494 1033 495 498 1383 1054 956 977 495 962 479 477 1061 962 489 912 1321 1055 1053 495 492 1018 1032 881 1375 1049 490 962 1055 960 490 1047 907 1057 474 477 488 385 959 1068 944 1077 1046 487 1048 1478 490 948 1410 1066 963 963 490 1053 1038 1062 493 1258 494 1404 1438 1365 913 967 951 487 1446 491 1449 962 1313 1008 972 1004 1070 1402 1475 491 487 395 939 974 1047 483 1397 939 1321 1032 1294 1038 395 494 1072 1064 397 1016 1511 968 1401 491 480 489 948 958 956 1039 1021 480 867 1389 489 1048 972 497 394 1373 498 491 1308 976 965 944 1059 490 463 957 1054 941 1402 1033 1498 1045 1041 980 995 496 390 965 498 489 940 467 959 483 958 1413 952 970 482 481 486 1510 1059 1403 957 965 1051 493 966 982 943 1079 388 497 1376 974 1057 494 1065 960 1066 494 1283 1054 1016 1338 1524 1062 486 489 1391 1505 491 973 1023 1470 1054 482 461 479 1063 1423 1068 1055 964 1008 975 967 1064 935 1026 1404 878 476 493 965 1044 1040 1047 966 488 864 1488 1399 1386 968 1308 488 482 1340 1451 483 1068 996 965 1059 944 495 1052 1074 1035 1044 955 492 1013 494 1037 1069 940 492 488 1384 482 968 1546 1040 959 946 1038 1050 1432 1507 487 958 484 397 1390 1026 386 1027 483 1481 999 1032 1061 1046 1056 961 1463 1055 974 1056 964 1045 474 1338 492 1044 489 1041 1008 487 1060 944 960 489 1036 395 1031 1070 1035 1050 971 490 1077 1032 1029 1058 1034 1078 1521 874 1512 948 968 1532 1053 1024 1047 967 1412 1045 1032 1047 1073 493 1017 1072 1045 1025 495 963 1061 979 942 960 1306 968 1413 1012 1060 1489 481 486 1027 1066 946 1406 1065 1051 490 1374 1513 1059 490 987 482 483 1540 481 973 488 1027 487 1035 1540 958 492 955 1054 974 1537 481 482 966 1351 945 877 1451 1412 495 490 1045 868 1036 482 1041 1015 486 1067 480 1071 882 1407 959 1533 1443 1483 492 1048 1360 941 969 488 1066 493 936 942 1324 1328 1195 454 490 1037 482 1060 397 1013 497 1050 1432 962 1070

1073 969 494 483 1055 1409 1028 1418 488 957 958 1040 1034 1495 394 485 1057 1059
493 1073 1038 483 965 1066 1052 489 1412 487 1024 951 488 1400 1049 481 1020 1030
1055 1064 1075 954 1062 1015 1068 1063 493 1292 953 1065 1409 865 1072 479 1489
1482 1059 484 485 940 495 487 874 1040 1059 1024 940 1552 1415 1337 1458 1407
946 1038 492 1413 491 479 395 884 1398 1020 1053 1062 1585 1060 861 956 917 483
930 493 1324 1042 489 482 917 1053 481 492 486 1329 1031 1505 1520 1297 493 485
484 1056 1036 1478 1030 488 1044 483 1079 482 480 1012 1044 908 488 1039 1059 484
998 952 1018 1033 1038 1339 1344 1045 1018 1510 958 949 1045 1043 1027 1059 485
955 1055 487 1057 493 964 1055 933 1401 472 917 1368 500 980 1023 1045 1012 1060
1046 1012 949 1554 1038 1071 939 1050 1012 490 1412 1013 947 1050 485 1073 488
1050 485 490 973 1065 1350 490 1437 1060 1069 987 487 970 468 945 1473 942 915
1532 945 1503 493 1067 494 480 946 485 1061 1048 968 974 1044 990 1024 945 498
946 1053 471 959 1416 478 977 493 1022 1070 471 489 1429 962 946 483 497 1517 970
1030 1376 936 1313 1052 1056 1405 484 1044 487 490 1018 1040 1464 1063 1069 876
481 1026 1061 489 489 1061 478 1042 959 1019 963 1220 931 493 1067 1458 1034 961
483 493 976 1041 858 496 934 1032 490 1393 1035 490 936 492 1019 1313 497 1050
1032 492 478 940 484 493 490 1502 1458 1437 1027 486 498 1045 480 1533 495 1034
976 1066 1073 477 1375 1060 1057 487 1039 1049 1436 492 1384 1456 485 1031 1038
1032 1050 1378 479 1482 484 1071 1034 1036 928 1052 1067 1437 1024 969 973 483
1459 979 494 1061 1393 948 1061 1340 973 1077 488 1505 1040 1052 951 492 960 490
1055 1006 872 1046 1513 1016 1048 498 972 494 1383 445 1398 1072 474 485 964 1044
1497 1399 1044 489 1044 1034 1527 1317 948 976 393 1037 1069 974 1038 1347 1039
1082 497 487 1065 970 1039 934 1514 1376 961 947 1017 485 1430 962 957 1015 491
489 488 944 1323 1056 1054 1054 479 1063 940 1417 1433 477 1047 1078 1022 1057
480 396 955 966 958 1032 492 480 1061 943 1073 1009 1044 496 979 1037 1053 493
1040 485 869 1047 1039 1054 495 1507 1500 1042 483 1032 485 963 1062 1050 1321
1081 392 1063 946 972 1038 857 1061 1055 1036 1480 963 1342 1054 486 1052 1063
1036 974 1068 1503 943 1024 1047 961 884 1435 1511 489 1061 967 1491 944 958 1051
478 487 1043 978 1037 959 489 1038 1045 1476 1024 1407 1514 964 1521 1051 1045
1412 1379 485 956 1024 951 1510 938 1059 1053 1046 1066 1049 1063 951 1057 489
949 1046 1057 489 1071 493 1057 1320 1062 924 483 1484 494 1024 960 1493 398 1025
961 493 1072 1444 497 1050 1031 479 1063 397 1078 975 1035 1051 1044 490 1499
1043 490 964 966 490 478 1403 1515 852 486 1063 1044

Mean:987.049 Standard deviation:318.209

95% confidence lower bound:966.408 upper bound:1007.690

18 Markets

LEAF global utility

Global utility values recorded: 1665 1576 1512 1565 1504 1462 1534 1606 1552 1566 1595 1557 1562 1588 1540 1543 1565 1463 1509 1557

Mean:1551.050 Standard deviation:47.237

95% confidence lower bound:1528.368 upper bound:1573.732

LEAF learning convergence

Number of days taken to converge: 126 106 136 124 105 109 109 110 106 166 112 107 105 138 118 106 148 106 147 106

Mean:119.500 Standard deviation:18.159

95% confidence lower bound:110.781 upper bound:128.219

Self-interested global utility

Global utility values recorded: 1054 1516 968 1053 1046 1074 942 479 878 1050 1030 1022 1510 1492 468 495 1472 1075 1544 1428 1507 496 490 1408 1055 1068 1438 1070 489 1595 1050 1397 497 1073 1038 1422 1533 1043 1055 1497 1496 983 495 1079 1058 492 1066 1066 1052 917 1057 1026 1040 1051 495 497 1014 966 490 1066 1045 490 1060 956 1041 1056 490 943 949 487 498 914 1058 969 1055 965 1030 1515 1044 1070 499 1014 1057 1500 487 486 1065 1409 1081 1068 484 490 1069 1055 478 1006 1051 1059 1515 495 1052 1057 1038 970 957 489 1521 494 968 1077 1025 1043 1056 487 1422 487 492 1036 1034 1055 978 1452 496 478 483 1040 950 1512 1072 1049 1512 493 1060 486 1032 947 490 1064 1026 1058 1054 982 495 1069 937 1342 1040 942 492 1042 1071 1050 1413 1039 1050 493 1043 485 493 1049 1399 1430 980 497 1054 1059 1052 477 1026 494 1063 1067 487 1044 494 964 1416 950 1435 1517 1061 493 966 469 1062 479 1002 944 952 975 495 1050 487 491 973 491 1043 1067 1260 463 1045 961 1496 1046 1032 983 496 1053 1032 1010 1070 1382 1475 1027 954 969 1040 991 491 1060 1592 960 1032 1464 1082 1446 1061 492 1597 1035 495 1046 498 1526 1056 1043 1648 496 1064 489 1034 496 497 979 1029 1060 1040 1597 1391 1047 948 1074 476 1391 1452 1493 1046 1427 923 1059 1401 1054 495 1062 490 1319 1053 1047 995 1500 1058 1059 935 1483 1453 1387 1063 830 1048 1031 943 396 1534 1422 492 495 1040 1087 1059 1049 1499 959 949 1064 1409 1020 969 476 1440 486 1610 1061 1043 954 480 1431 979 981 1397 484 490 1037 1068 1078 1375 1060 1643 1072 1069 1514 495 1074 1046 965 961 1060 1432 1069 1507 1037 1055 1054 846 1069 1063 1002 1262 1037 491 1538 970 1054 1055 1055 479 490 492 1507 1538 485 495 957 493 1061 1042 966 1035 1051 1509 492 976 1048 963 1065 463 1050 490 1468 840 1035 1053 482 1049 495 1047 1436 978 1038 1004 1472 493 1468 979 1060 496 1057 1506 475 492 484 492 1570 1486 1451 1054 1033 1028 1054 948 495 1053 1041 1503 495 1533 979 1052 477 482 1023 1068

```
496 1341 1405 944 1041 1063 1059 1049 386 1042 488 479 1074 1033 484 1409 913
483 1575 1036 1430 393 949 1043 1540 483 1048 494 1052 495 984 1493 1057 1416 964
1487 1395 1040 487 491 483 950 1050 1064 1056 1029 953 485 1435 1328 1537 1069
1438 1008 486 972 391 927 1480 1044 1045 942 1322 493 957 1026 1042 490 494 1023
972 481 1050 942 949 1017 497 490 1043 1500 943 954 1057 1485 1591 492 487 495
1506 962 960 1313 1060 1437 1045 1072 1041 1389 1054 1315 497 1067 1050 1023 1632
1032 496 1034 1049 1065 494 1046 482 1044 1482 490 493 946 1064 1521 1039 991 491
489 974 974 976 938 1378 1042 1053 960 1495 1493 880 1060 1424 1062 882 494 1581
970 1021 490 1027 954 1070 479 1067 1066 1037 1063 1391 1041 1039 1067 951 1527
480 1064 1300 492 1036 1397 1037 957 962 1456 475 880 1035 1047 1536 1503 1023
1054 1054 1064 473 492 942 1064 965 1494 486 1549 1042 1503 483 1604 1056 1041
1060 496 1019 1054 961 394 486 960 485 1054 398 1049 1013 494 1047 1067 1045 1051
1065 479 1044 1057 955 1049 1047 1065 1434 483 943 872 1058 1469 1634 493 1059
947 1030 1044 1047 955 1537 499 1038 481 888 1061 962 1055 1058 489 492 492 1069
1572 1506 975 1535 1039 962 1054 967 1502 496 1081 1508 494 1614 473 1035 1052
1036 1069 1491 1063 1046 1051 967 1054 1081 484 1536 1415 1054 489 1056 981 493
486 1392 1491 965 928 1068 954 1061 499 486 863 1045 945 489 1060 1517 487 1369
1016 1503 1330 492 1030 958 1070 1071 976 1050 1066 1074 1605 1050 968 883 482
956 1356 1043 1362 487 1013 492 494 1263 489 974 488 460 946 1609 1061 956 1295
951 1036 485 487 494 1050 873 492 496 494 1390 1524 1062 931 1060 493 970 459
491 959 1049 1081 944 481 1446 1561 1025 1053 475 1047 1046 490 973 1409 984 1457
1429 1050 479 493 958 490 1357 498 1076 1413 951 1497 1051 485 1515 967 1070 1051
1485 1029 1460 487 1074 1406 1018 1532 955 495 1062 1045 958 1042 1066 489 1325
494 494 1041 1007 950 1409 1402 1064 488 1067 853 979 957 923 899 964 1433 966
491 497 1055 1045 1029 486 1411 970 976 963 965 1502 955 1053 1426 1327 967 1050
1497 1568 1067 488 1055 497 957 483 1517 953 1056 1428 1040 1067 957 963 929 1063
1341 1419 1047 1075 1050 1065 495 1492 1484 495 498 923 395 488 493 1589 1034 974
930 1065 494 1451 498 1417 485 980 1058 1400 486 917 959 1542 1032 1499 1404 1077
996 952 1063 488 1060 399 1636 494 1065 873 1338 851 488 492 1072 492 1487 975
1032 1053 950 1530 1061 490 1036 971 1540 980 488 485 493 981 1023 1419 951 967
1015 1516 1508 963 486 1535 1028 865 1445 1042 1461 400 477 1071 486 1055 1457
1039 1077 1057 493 962 1057 480 1073 877 1419 1073 962 957 486 483 1015 973 466
1069 1587 961 482 1410 1523 489 1436 1385 1076 488 1059 466 1436 1413 1060 1624
947 962 998 1022 1455 1425 472 857
```

Mean:987.049 Standard deviation:333.028

95% confidence lower bound:966.408 upper bound:1007.690

19 Markets

LEAF global utility

Global utility values recorded: 1638 1580 1578 1762 1534 1569 1568 1801 1712 1738 1675 1597 1738 1567 1695 1709 1695 1719 1674 1577

Mean:1656.300 Standard deviation:79.264

95% confidence lower bound:1618.240 upper bound:1694.360

LEAF learning convergence

Number of days taken to converge: 135 108 113 109 106 107 112 105 107 105 130 107 116 119 154 148 127 135 131 137

Mean:120.550 Standard deviation:15.374

95% confidence lower bound:113.168 upper bound:127.932

Self-interested global utility

Global utility values recorded: 1514 1501 494 1585 1050 1052 1499 952 1550 1054 1514 1020 1524 1406 958 961 1370 1579 1036 1053 1525 1070 1078 1515 492 1443 478 1038 960 1429 1056 1568 1406 959 1079 1463 1484 1044 1043 864 1034 1069 492 1066 1415 845 1033 967 491 1071 1372 1043 1047 474 972 1580 1341 1499 1482 1548 497 972 1487 1066 491 490 1466 1076 1627 1487 490 1490 967 1398 1529 1384 1029 1606 480 1069 1465 1042 1077 1568 1049 1557 1036 1486 1519 484 1055 1499 1067 1015 1410 489 958 1061 1052 1075 1629 1032 1059 1041 1058 1514 1629 1066 1061 1023 1043 492 1051 1524 1751 1060 1056 493 1454 1037 1060 1508 1072 486 1079 1516 1039 1040 1745 1489 1061 1054 1068 1520 483 1510 1555 488 492 1495 1612 1047 1065 960 1077 1522 1482 1067 1071 1506 495 1535 948 1050 1063 1047 1442 1772 953 1571 1053 958 387 1496 1629 1069 1047 482 1067 1381 1406 1436 1065 980 1580 1474 1457 973 1411 1066 1504 1402 1609 1531 1063 1089 1063 1064 467 1075 976 1045 955 1062 1071 1031 1514 1041 1039 483 1055 1579 1482 489 961 487 1062 1611 1326 495 473 1034 1051 1045 494 1451 1049 945 393 1493 1489 968 1602 493 875 1013 493 974 1067 468 1061 1051 497 1332 1046 488 1518 1487 1621 1496 1061 1416 498 478 1082 1051 483 497 1609 482 1072 488 1563 1043 1426 1004 1492 940 1058 488 1059 965 399 1534 1787 498 1419 1024 1567 1626 1069 1068 1516 1061 1057 1048 1054 493 1051 1023 488 961 482 1432 1418 1010 489 1571 1599 1424 1071 1558 1495 1065 972 1430 1589 979 1772 475 492 1081 1018 1518 1568 488 1622 500 1043 1515 967 494 966 493 1528 497 1574 1063 949 1424 494 1031 1517 1022 1518 1395 1405 1597 475 1438 1473 484 1073 1035 1354 1350 484 1465 961 1051 1048 935 988 1533 1059 966 1048 1056 1057 1458 1515 1478 499 1023 1494 1647 1042 1064 1420 1051 1399 1419 495 1032 492 1608 392 1381 1640 498 1071 1064 1030 495 1611 1054 1053 1438 481 1426 1419 1059 1052 1061 960 494 1036 396 1015 957 1414 1568 1506 1791 1504 967 490 488 1065 1056 1070 1604

492 1085 1012 1501 1064 1422 1416 476 1412 1429 1445 1083 977 1031 1043 967 1032
1576 1057 1466 1483 1599 1540 1051 1066 487 1082 1048 1055 1065 1059 956 1057
1061 496 1066 938 1028 1472 1585 1471 1060 1044 978 944 1047 496 1526 980 1055
1025 1070 486 1508 1518 966 1046 964 958 1052 1605 1041 1289 487 1446 1543 1068
921 1725 1057 489 489 483 973 1604 490 1522 979 966 1594 1371 1064 479 492 950
1651 496 1037 1474 1010 1051 1409 1047 1569 1035 1426 496 1061 1445 1623 1062
1047 1024 934 484 963 1039 1054 1054 1052 1514 1062 1039 484 496 1448 976 963
1022 1055 1448 1466 1069 837 1044 1547 483 1047 1515 1557 1054 490 479 496 983
1049 1615 1533 1035 1533 1400 486 1068 492 1035 1062 494 487 1032 1062 959 970
1050 491 1462 987 1012 493 496 1416 489 1038 1314 495 1062 1574 495 495 483 1048
491 1398 1057 1045 1068 488 1035 487 1064 1050 1516 1068 1017 1063 907 1500 1498
1075 1020 1381 1054 492 1501 1011 1515 969 1065 1002 1052 1075 1495 1048 1596
1059 1055 955 1058 1053 916 1056 500 1517 1065 973 479 920 1055 488 969 1055 979
1501 469 493 1522 1068 486 1448 1612 1029 1506 956 1058 948 1575 959 1388 492
483 917 486 1063 470 1433 1491 1056 1500 1023 1066 496 1061 1505 482 493 495 1068
1062 499 490 1047 1024 1533 1058 1023 496 1053 1050 475 1501 962 971 1077 986 956
1428 1474 1577 1062 1530 1023 1616 1058 477 1051 490 1062 1471 488 1043 495 1569
490 1632 1060 1043 1471 1032 488 965 478 941 1359 1610 954 487 1507 479 1047 1032
1066 977 1056 1533 1520 485 1093 1543 1048 1058 456 1038 489 948 498 982 1059
1041 985 1065 1052 1506 486 1045 496 962 490 395 476 1499 1060 489 1030 964 490
990 498 1049 497 1487 1053 1048 1071 1059 1049 1443 1439 983 1540 1465 1011 955
1044 1493 1511 491 1048 1041 1048 1405 976 1586 1049 1039 1050 1053 494 1522 1036
1522 1511 1055 488 1656 1083 1453 1063 478 1058 1621 1073 490 961 1065 1464 492
1083 1085 1419 388 1073 1598 491 493 1604 950 1052 1076 1452 1600 1542 1080 1062
1535 961 1026 973 967 487 946 1005 1044 1046 1052 1046 1068 1763 955 483 1078
1053 488 494 393 498 496 1600 489 1512 1038 961 1037 472 492 487 1500 1484 1508
1455 1059 1594 852 1058 1068 1070 1055 845 1589 1064 971 496 1499 1525 964 1060
1015 1037 481 988 1050 1600 1040 492 1481 1049 1544 1437 976 966 487 495 1029 964
1508 494 1463 1391 943 1418 1067 1051 1587 937 493 1329 1080 1444 1538 1056 485
495 496 1056 490 1314 980 478 1063 955 1061 1412 485 497 495 1024 1504 1010 493
1071 1042 498 1045 972 1057 489 487 1023 1063 1054 1063 1053 1031 1425 391 484
1011 1503 970 497 1048 490 954 1052 1075 1067 1038 497 1607 956 1791 953 1049 490
1519 972 1498 488 954 972 1505 1064 1074 1370 1066 1025 1052 493 1070 957 1531
1080 489 1424 1788 1027 489 496 1469 1074 960 493 1069 976 1489 486 1501 489 950
485 979 1625 1446 1520 1023 1067 1514 1474 494 908

Mean:1064.381 Standard deviation:364.52

95% confidence lower bound:1041.788 upper bound:1086.974

20 Markets

LEAF global utility

Global utility values recorded: 1787 1710 1766 1789 1767 1775 1810 1756 1837 1789 1852 1666 1797 1850 1574 1810 1675 1808 1701 1732

Mean:1762.550 Standard deviation:69.280

95% confidence lower bound:1729.284 upper bound:1795.816

LEAF learning convergence

Number of days taken to converge: 122 108 116 110 106 125 105 138 112 111 107 139 107 107 122 147 147 106 131 111

Mean:118.850 Standard deviation:14.291

95% confidence lower bound:111.988 upper bound:125.712

Self-interested global utility

Global utility values recorded: 1590 1034 962 1057 1404 1619 1524 489 935 490 1035 1064 1069 1593 1505 952 1518 1473 499 1508 1046 1041 1037 1058 1051 1052 1051 1553 489 1500 1031 1049 1603 1610 1046 1513 487 1645 1408 491 1520 1046 1431 485 490 1032 1058 1487 1532 490 1550 1060 1077 498 1066 1586 476 922 1052 1568 1061 1062 496 1046 1044 1428 494 1062 1083 484 1624 1079 488 1466 1039 1053 1060 1066 1635 1057 1428 1050 1466 1070 1063 1484 1576 1412 1429 490 491 1072 480 1408 1513 1500 1627 1533 1535 1069 1623 1432 1066 1563 1076 1061 1568 1058 494 1062 1073 492 1498 1050 1050 1072 1385 490 1501 969 946 480 1425 1571 1049 1887 1519 1547 1057 1062 1078 979 1026 1077 398 1055 1519 909 1041 1055 1521 1068 1514 1515 1083 1052 1017 1058 1077 1059 870 1537 1819 1070 1636 1063 1513 1468 1059 494 1468 1494 496 495 485 1063 1388 975 1480 490 1047 1062 1029 1499 1061 1073 1037 1897 1052 497 944 494 1064 1599 1523 1612 1599 1437 1060 495 1072 1877 1069 1613 1057 1567 1056 1067 492 1594 1509 488 1066 1516 1015 954 492 961 1044 1455 1407 1056 1443 1485 1912 1084 1074 1484 1032 1040 1616 1610 1494 1529 1596 1067 1030 1540 496 965 1071 495 456 1059 1034 1046 1595 1537 393 1391 1073 1058 1022 477 1542 1030 1048 484 1040 1068 493 1056 1060 1812 1065 1485 1063 1006 1061 1078 957 1442 1053 1636 1045 1561 956 1070 1039 1063 963 1528 1598 1044 1052 1477 956 1063 1579 393 1041 1081 494 1068 1532 1075 1874 1481 1048 956 1477 1515 1526 494 1505 1784 489 972 1012 1046 1068 1411 1327 943 493 1541 970 1623 981 938 1055 1050 1044 963 482 1040 943 1059 1043 1608 1052 966 1071 1503 1060 1511 1045 1034 1066 489 1041 480 488 491 964 1062 483 1070 1535 1493 1502 961 490 1593 1029 1514 1548 1055 1526 495 1578 981 1507 1072 1611 1479 1614 493 1528 1068 1058 944 968 974 1533 1070 1064 492 484 491 487 494 1599 491 1081 975 488 1046 1037 1426 1487 1047 1063 1073 1512 1020 1070 1460 1456 958 1055 963 1052 1538 1510 1058 1032 1079 1529 1582

```
1075 491 1506 1071 1047 477 966 1061 1604 1051 955 1398 495 1624 1506 1541 1605
942 497 1058 487 1614 484 1433 1489 1049 1615 1072 962 1442 1778 1539 1494 1058
970 1332 1044 1072 1024 1048 1047 484 1075 1039 1046 1517 1064 1832 1420 494 1016
1053 973 1072 994 1521 1062 1539 1536 1066 865 1582 1033 493 1037 1507 1058 1058
1073 1012 495 1067 1851 1064 493 1057 1444 1078 1533 492 1054 1428 1609 962 1073
1041 1056 1504 1056 1476 973 1020 1523 1409 1065 1077 1053 963 1496 486 1556 1602
1045 496 982 493 1509 1070 1448 1083 945 488 489 1445 1059 1521 1056 1534 1057
1579 1518 1457 456 1051 496 1592 488 483 1535 489 970 1043 495 493 1061 1058 1533
1513 1485 980 1065 1059 1497 1073 1022 1525 1057 496 1570 1059 1069 491 495 1066
1075 1608 951 1053 1077 1526 1074 1069 1517 1047 1040 1051 1799 493 949 953 1078
1608 1493 1081 1076 494 1510 470 477 1051 1063 867 1528 932 1505 1021 1055 1603
491 493 496 1524 1646 1548 1580 1393 1032 1417 1589 1034 495 1504 1494 1042 1523
1437 1066 1024 489 1041 498 950 876 1481 490 956 960 1060 1043 1082 1080 1620
1051 956 1497 1443 1064 973 1440 1039 1059 891 1604 1487 1492 1601 1030 1567 956
1027 495 1061 1059 493 1051 1017 493 1603 1060 1054 1508 482 1063 1041 1054 1052
1064 1625 1062 489 1489 1045 1053 1058 1532 1426 478 1499 1040 1032 486 1612 497
485 498 1036 1520 1509 966 1522 1026 490 1020 1505 1079 1052 1503 1619 1523 1493
495 1526 975 1465 1479 1492 1524 1071 1529 1488 1879 932 490 1057 1063 1064 944
489 1470 1072 485 1044 474 1060 1025 1075 1062 1017 1042 1562 930 1062 956 1472
1409 1768 1481 1844 1442 1038 983 1048 1494 1501 1491 1054 1027 940 486 1055 494
1058 962 1548 1508 939 493 1526 494 491 977 1533 1062 1051 1511 493 976 1068 1042
1068 1522 1572 1451 1608 1063 1548 1600 1466 939 1053 1061 1410 1489 1046 1498
1065 1048 1571 1431 1081 493 1628 1080 1045 1054 1037 1517 486 961 956 1413 455
1071 489 1062 1066 487 867 1048 1037 1062 1369 494 1042 1054 494 495 1060 1526
1637 1068 1072 1051 1386 1049 1424 1513 1643 1420 966 1622 979 1048 1528 495 1610
1035 1445 1540 1568 498 1532 493 1624 1472 1486 1080 966 1071 1642 1076 1045 1600
495 489 1063 490 1058 1048 492 1059 1615 1591 1433 394 1495 499 1040 1616 1048
1840 1440 1057 1060 1545 1041 1505 1015 491 493 1064 1422 491 1555 1371 1482 1514
1026 1059 1408 1421 475 1528 1487 1479 1608 1067 1603 1017 1063 1042 497 397 488
968 964 1435 1041 498 493 476 1044 1617 494 1056 1061 1056 1612 1457 492 1068
1632 487 1598 1039 1583 1079 1064 939 971 1601 481 1048 1050 1054 1022 1492 1524
1050 1543 477 951 1572 1075 482 1494 988 1050 1538 1050 492 490 960 1544 1900
1033 1079 1043 1581 1534 1017 1533 497 1063 1546 479 483 488 1614 1531 1063 1060
1040 1056 492 1024 492 1632 1436 1539 981 1030 493 1053 1408 1066 490 485 1042
1050 1045 480 497 1493 1645 1064 487 1055 485 1049 494 495 1004 494 487 1059 493
1576 1880 1030
```

Mean:1114.999 Standard deviation:372.895

```
95% confidence lower bound:1091.887 upper bound:1138.111
```

C.3 Market based buyer application: varying remote parameter update probabilities

These results are referred to in section 5.6.2. 20 experiments were performed for each different number of markets for each different value of U.

C.3.1 U=0.1

6 Markets

LEAF global utility

Global utility values recorded: 474 423 456 512 546 494 482 472 451 509 383 454 508 472 480 446 421 468 508 559

Mean:475.900 Standard deviation:42.046

95% confidence lower bound:455.711 upper bound:496.089

LEAF learning convergence

Number of days taken to converge: 106 109 105 105 105 107 140 105 105 104 106 149 104 104 127 145 107 105 105 101

Mean:112.200 Standard deviation:14.962

95% confidence lower bound:105.016 upper bound:119.384

7 Markets

LEAF global utility

Global utility values recorded: 468 458 365 588 625 415 635 616 448 623 387 421 474 650 290 648 645 469 639 621

Mean:524.250 Standard deviation:115.508

95% confidence lower bound:468.787 upper bound:579.713

LEAF learning convergence

Number of days taken to converge: 105 120 107 107 131 108 103 105 105 105 109 139 125 107 118 105 105 107 145 113

Mean:113.450 Standard deviation:12.369

95% confidence lower bound:107.511 upper bound:119.389

8 Markets

LEAF global utility

Global utility values recorded: 745 464 722 379 626 690 702 480 644 686 391 696 642 671 669 703 452 370 740 711

Mean:609.150 Standard deviation:131.044

95% confidence lower bound:546.227 upper bound:672.073

LEAF learning convergence

Number of days taken to converge: 107 106 109 126 116 121 109 106 109 107 103 113 106 105 107 107 115 107 105 105

Mean:109.450 Standard deviation:5.898

95% confidence lower bound:106.618 upper bound:112.282

9 Markets

LEAF global utility

Global utility values recorded: 835 801 825 807 835 799 817 841 777 778 707 730 771 800 440 815 821 791 695 767

Mean:772.600 Standard deviation:88.242

95% confidence lower bound:730.229 upper bound:814.971

LEAF learning convergence

Number of days taken to converge: 106 107 111 139 104 103 112 106 106 106 104 105 130 106 107 114 125 103 103 113

Mean:110.500 Standard deviation:9.822

95% confidence lower bound:105.784 upper bound:115.216

10 Markets

LEAF global utility

Global utility values recorded: 907 854 932 867 773 798 885 701 465 447 845 918 839 898 463 906 884 871 888 843

Mean:799.200 Standard deviation:156.344

95% confidence lower bound:724.129 upper bound:874.271

LEAF learning convergence

Number of days taken to converge: 107 106 106 106 107 105 109 122 103 104 107 112 111 120 116 106 105 105 107 109

Mean:108.650 Standard deviation:5.184

95% confidence lower bound:106.161 upper bound:111.139

11 Markets

LEAF global utility

Global utility values recorded: 927 977 808 949 958 981 865 880 998 476 912 940 936 946 871 929 474 932 948 927

Mean:881.700 Standard deviation:145.842

95% confidence lower bound:811.672 upper bound:951.728

LEAF learning convergence

Number of days taken to converge: 106 113 125 106 106 105 132 118 114 106 106 108 106 108 112 119 118 115 109 110

Mean:112.100 Standard deviation:7.305

95% confidence lower bound:108.593 upper bound:115.607

12 Markets

LEAF global utility

Global utility values recorded: 983 899 1012 825 966 816 476 994 1066 999 900 987 486 1132 949 1077 1026 1082 499 889

Mean:903.150 Standard deviation:197.175

95% confidence lower bound:808.473 upper bound:997.827

LEAF learning convergence

Number of days taken to converge: 135 121 105 122 141 106 104 118 123 117 121 142 153 106 112 104 111 142 108 103

Mean:119.700 Standard deviation:15.315

95% confidence lower bound:112.346 upper bound:127.054

13 Markets

LEAF global utility

Global utility values recorded: 477 929 1166 1132 1025 1078 980 1186 1018 1171 1090 480 1022 984 934 1005 1131 921 1001 915

Mean:982.250 Standard deviation:192.322

95% confidence lower bound:889.903 upper bound:1074.597

LEAF learning convergence

Number of days taken to converge: 110 151 130 122 107 105 107 128 124 151 106 106 118 129 107 122 114 112 104 112

Mean:118.250 Standard deviation:14.101

95% confidence lower bound:111.479 upper bound:125.021

14 Markets

LEAF global utility

Global utility values recorded: 1278 1197 992 966 1037 1002 1004 1229 879 484 1266 1002 846 385 1248 993 1023 1313 1131 1222

Mean:1024.850 Standard deviation:244.829

95% confidence lower bound:907.291 upper bound:1142.409

LEAF learning convergence

Number of days taken to converge: 113 144 107 106 105 116 112 118 114 112 104 106 105 113 108 106 119 105 104 111

Mean:111.400 Standard deviation:9.029

95% confidence lower bound:107.065 upper bound:115.735

15 Markets

LEAF global utility

Global utility values recorded: 1364 1307 1293 1340 1371 1312 1355 1346 997 906 1293 1052 1294 1290 1073 378 1249 1006 467 1016

Mean:1135.450 Standard deviation:285.483

95% confidence lower bound:998.370 upper bound:1272.530

LEAF learning convergence

Number of days taken to converge: 110 106 105 106 134 107 106 104 112 145 110 107 109 116 108 106 106 106 117 142

Mean:113.1 Standard deviation:12.358

95% confidence lower bound:107.166 upper bound:119.034

16 Markets

LEAF global utility

Global utility values recorded: 1419 1375 1053 974 1032 1375 1046 1448 1387 1363 1069 1463 1483 1393 1333 1398 1279 1420 1046 1284

Mean:1282 Standard deviation:172.958

95% confidence lower bound:1198.951 upper bound:1365.049

LEAF learning convergence

Number of days taken to converge: 106 114 107 143 143 109 106 115 124 116 110 106 107 124 125 113 106 106 130 107

Mean:115.85 Standard deviation:11.931

95% confidence lower bound:110.121 upper bound:121.579

17 Markets

LEAF global utility

Global utility values recorded: 1325 1556 1028 1550 1041 485 1401 1521 1547 1514 1515 1447 1019 459 1406 945 1325 1411 1055 1043

Mean:1229.65 Standard deviation:334.913

95% confidence lower bound:1068.836 upper bound:1390.464

LEAF learning convergence

Number of days taken to converge: 120 140 141 110 106 109 110 107 115 103 105 158 138 105 109 142 114 107 107 116

Mean:118.1 Standard deviation:16.193

95% confidence lower bound:110.325 upper bound:125.875

18 Markets

LEAF global utility

Global utility values recorded: 1424 1530 1554 1522 1520 1434 1601 1511 1585 1506 1518 1485 925 1530 1379 1559 1493 1452 1539 1028

Mean:1454.75 Standard deviation:172.949

95% confidence lower bound:1371.705 upper bound:1537.795

LEAF learning convergence

Number of days taken to converge: 118 105 114 109 115 112 112 103 133 108 125 115 110 114 105 152 118 149 106 108

Mean:116.55 Standard deviation:13.617

95% confidence lower bound:110.012 upper bound:123.088

19 Markets

LEAF global utility

Global utility values recorded: 1513 1776 1521 477 853 1396 1530 1024 1525 1657 1673 1511 1504 1572 1461 1424 1713 1630 1546 1050

Mean:1417.8 Standard deviation:322.799

95% confidence lower bound:1262.803 upper bound:1572.797

LEAF learning convergence

Number of days taken to converge: 113 122 135 138 108 106 123 107 144 125 130 115 144 106 144 135 104 108 111 122

Mean:122 Standard deviation:14.187

95% confidence lower bound:115.188 upper bound:128.812

20 Markets

LEAF global utility

Global utility values recorded: 1593 1629 1559 1772 1810 1587 1704 1820 1841 1548 1611 1821 1767 1542 1790 1528 1509 1768 1477 1590

Mean:1663.3 Standard deviation:123.442

95% confidence lower bound:1604.027 upper bound:1722.573

LEAF learning convergence

Number of days taken to converge: 148 106 107 123 107 113 108 113 138 108 135 152 106 109 106 106 144 116 119 182

Mean:122.3 Standard deviation:20.934

95% confidence lower bound:112.248 upper bound:132.352

C.3.2 U=0.05

6 Markets

LEAF global utility

Global utility values recorded: 345 458 366 359 552 417 327 257 368 403 443 290 344 479 283 457 362 383 491 416

Mean:390 Standard deviation:75.381

95% confidence lower bound:353.804 upper bound:426.196

LEAF learning convergence

Number of days taken to converge: 105 112 107 106 134 136 123 140 104 107 130 120 103 110 129 104 105 106 109 103

Mean:114.65 Standard deviation:12.617

95% confidence lower bound:108.592 upper bound:120.708

7 Markets

LEAF global utility

Global utility values recorded: 523 384 577 386 408 506 371 454 417 612 559 439 389 523 353 431 532 601 481 394

Mean:467 Standard deviation:81.778

95% confidence lower bound:427.733 upper bound:506.267

LEAF learning convergence

Number of days taken to converge: 144 114 107 104 111 106 128 107 127 105 101 104 116 114 105 106 105 108 108 112

Mean:111.6 Standard deviation:10.455

95% confidence lower bound:106.58 upper bound:116.62

8 Markets

LEAF global utility

Global utility values recorded: 479 374 518 376 391 375 580 337 386 607 651 509 454 490 378 445 384 656 443 701

Mean:476.7 Standard deviation:110.15

95% confidence lower bound:423.81 upper bound:529.59

LEAF learning convergence

Number of days taken to converge: 106 106 104 105 106 114 142 108 113 105 105 105 142 105 104 106 107 157 105 102

Mean:112.35 Standard deviation:15.449

95% confidence lower bound:104.932 upper bound:119.768

9 Markets

LEAF global utility

Global utility values recorded: 294 603 852 465 719 466 541 754 564 331 632 298 383 448 791 388 378 578 455 697

Mean:531.85 Standard deviation:168.508

95% confidence lower bound:450.938 upper bound:612.762

LEAF learning convergence

Number of days taken to converge: 108 102 107 104 154 111 109 107 113 104 107 108 114 106 105 105 116 103 119 122

Mean:111.2 Standard deviation:11.428

95% confidence lower bound:105.713 upper bound:116.687

10 Markets

LEAF global utility

Global utility values recorded: `471 876 451 748 641 706 482 489 911 378 683 457 464 848 677 512 821 743 515 694`

Mean:628.35 Standard deviation:164.217

95% confidence lower bound:549.498 upper bound:707.202

LEAF learning convergence

Number of days taken to converge: `111 103 105 116 129 109 127 104 107 105 106 104 117 107 106 109 107 134 105 106`

Mean:110.85 Standard deviation:9.092

95% confidence lower bound:106.484 upper bound:115.216

11 Markets

LEAF global utility

Global utility values recorded: `963 435 848 474 455 461 678 480 759 722 823 476 460 871 876 470 739 752 722 774`

Mean:661.9 Standard deviation:177.637

95% confidence lower bound:576.605 upper bound:747.195

LEAF learning convergence

Number of days taken to converge: `109 106 125 116 111 107 105 121 155 107 113 133 109 111 106 117 141 106 118 108`

Mean:116.2 Standard deviation:13.277

95% confidence lower bound:109.825 upper bound:122.575

12 Markets

LEAF global utility

Global utility values recorded: `920 637 705 771 908 832 769 743 928 774 364 725 489 905 372 477 752 890 817 847`

Mean:731.25 Standard deviation:176.835

95% confidence lower bound:646.34 upper bound:816.16

LEAF learning convergence

Number of days taken to converge: `154 160 123 105 105 115 127 106 118 162 106 113 132 106 105 107 116 106 106 115`

Mean:119.35 Standard deviation:18.717

95% confidence lower bound:110.362 upper bound:128.338

13 Markets

LEAF global utility

Global utility values recorded: 910 863 856 900 961 817 908 889 716 477 879 476 656 713 719 719 802 927 830 860

Mean:793.9 Standard deviation:137.442

95% confidence lower bound:727.905 upper bound:859.895

LEAF learning convergence

Number of days taken to converge: 120 121 112 129 110 110 105 126 133 106 105 132 105 106 135 105 113 109 110 110

Mean:115.1 Standard deviation:10.518

95% confidence lower bound:110.05 upper bound:120.15

14 Markets

LEAF global utility

Global utility values recorded: 833 742 751 795 420 893 900 946 805 779 470 1030 451 1249 841 1242 1000 819 866 776

Mean:830.4 Standard deviation:217.736

95% confidence lower bound:725.851 upper bound:934.949

LEAF learning convergence

Number of days taken to converge: 109 129 115 108 121 136 127 110 112 106 119 115 122 150 106 157 110 106 157 111

Mean:121.3 Standard deviation:16.614

95% confidence lower bound:113.323 upper bound:129.277

15 Markets

LEAF global utility

Global utility values recorded: 1161 1010 830 901 1033 1030 974 906 482 727 868 842 704 1140 918 1355 1184 1029 376 485

Mean:897.75 Standard deviation:249.381

95% confidence lower bound:778.005 upper bound:1017.495

LEAF learning convergence

Number of days taken to converge: 104 123 132 110 109 107 108 105 116 133 106 105 107 114 107 108 124 107 107 111

Mean:112.15 Standard deviation:8.869

95% confidence lower bound:107.891 upper bound:116.409

16 Markets

LEAF global utility

Global utility values recorded: 814 921 866 489 1304 1206 804 935 1362 473 918 1248 971 964 1049 783 1387 802 1042 1117

Mean:972.75 Standard deviation:252.942

95% confidence lower bound:851.296 upper bound:1094.204

LEAF learning convergence

Number of days taken to converge: 122 107 107 106 107 115 106 107 125 104 138 105 109 105 105 116 119 138 132 116

Mean:114.45 Standard deviation:11.199

95% confidence lower bound:109.073 upper bound:119.827

17 Markets

LEAF global utility

Global utility values recorded: 1337 1285 854 1387 1095 1249 459 1141 1257 1209 1245 811 1329 1229 1046 1219 473 904 1263 925

Mean:1085.85 Standard deviation:269.506

95% confidence lower bound:956.442 upper bound:1215.258

LEAF learning convergence

Number of days taken to converge: 122 107 107 106 107 115 106 107 125 104 138 105 109 105 105 116 119 138 132 116

Mean:114.45 Standard deviation:11.199

95% confidence lower bound:109.073 upper bound:119.827

18 Markets

LEAF global utility

Global utility values recorded: 1261 475 1017 1207 1231 1391 955 492 1459 1536 1029 1012 489 931 488 1366 1180 997 1302 909

Mean:1036.35 Standard deviation:334.005

95% confidence lower bound:875.972 upper bound:1196.728

LEAF learning convergence

Number of days taken to converge: 122 107 107 106 107 115 106 107 125 104 138 105 109 105 105 116 119 138 132 116

Mean:114.45 Standard deviation:11.199

95% confidence lower bound:109.073 upper bound:119.827

19 Markets

LEAF global utility

Global utility values recorded: 1148 1265 1516 1254 1287 932 1005 1448 1711 918 1209 1034 490 829 1382 1418 1391 1276 1112 984

Mean:1180.45 Standard deviation:277.992

95% confidence lower bound:1046.968 upper bound:1313.932

LEAF learning convergence

Number of days taken to converge: 110 117 119 108 133 114 122 120 115 118 135 121 116 121 118 118 112 103 113 153

Mean:119.3 Standard deviation:10.82

95% confidence lower bound:114.105 upper bound:124.495

20 Markets

LEAF global utility

Global utility values recorded: 1315 1266 1309 1247 962 859 831 1378 1063 1407 1484 1434 946 1556 992 1373 1441 1041 1222 1019

Mean:1207.25 Standard deviation:223.545

95% confidence lower bound:1099.911 upper bound:1314.589

LEAF learning convergence

Number of days taken to converge: 136 122 108 161 141 108 116 119 107 122 106 106 107 110 120 105 115 106 106 110

Mean:116.55 Standard deviation:14.555

95% confidence lower bound:109.561 upper bound:123.539

C.3.3 U=0.01

6 Markets

LEAF global utility

Global utility values recorded: 296 254 371 378 320 422 457 465 440 427 379 361 520 371 347 355 327 311 479 382

Mean:383.1 Standard deviation:67.389

95% confidence lower bound:350.742 upper bound:415.458

LEAF learning convergence

Number of days taken to converge: 127 237 285 103 102 160 162 104 111 106 103 104 192 133 105 118 106 195 104 106

Mean:138.15 Standard deviation:51.954

95% confidence lower bound:113.204 upper bound:163.096

7 Markets

LEAF global utility

Global utility values recorded: 435 350 354 380 335 440 344 548 645 377 364 290 355 442 472 479 542 530 648 396

Mean:436.3 Standard deviation:102.212

95% confidence lower bound:387.221 upper bound:485.379

LEAF learning convergence

Number of days taken to converge: 301 332 268 361 104 210 103 107 147 106 105 104 103 133 143 107 205 122 115 176

Mean:167.6 Standard deviation:84.024

95% confidence lower bound:127.254 upper bound:207.946

8 Markets

LEAF global utility

Global utility values recorded: 477 441 643 456 461 594 533 614 431 640 438 427 428 451 474 603 566 481 651 562

Mean:518.55 Standard deviation:81.989

95% confidence lower bound:479.182 upper bound:557.918

LEAF learning convergence

Number of days taken to converge: 103 107 106 342 146 113 103 238 107 106 103 104 114 367 127 189 243 123 331 106

Mean:163.9 Standard deviation:89.722

95% confidence lower bound:120.818 upper bound:206.982

9 Markets

LEAF global utility

Global utility values recorded: 567 584 688 456 630 720 567 710 551 684 345 667 725 462 635 621 702 690 686 652

Mean:617.1 Standard deviation:101.664

95% confidence lower bound:568.284 upper bound:665.916

LEAF learning convergence

Number of days taken to converge: 153 103 310 119 140 250 133 105 344 105 105 124 106 143 355 104 107 208 116 222

Mean:167.6 Standard deviation:84.305

95% confidence lower bound:127.12 upper bound:208.08

10 Markets

LEAF global utility

Global utility values recorded: 405 758 624 714 727 466 700 722 786 473 639 716 866 645 670 337 745 743 455 606

Mean:639.85 Standard deviation:141.208

95% confidence lower bound:572.046 upper bound:707.654

LEAF learning convergence

Number of days taken to converge: 320 106 106 104 118 163 346 106 146 102 574 103 299 107 289 104 103 153 105 457

Mean:195.55 Standard deviation:138.264

95% confidence lower bound:129.16 upper bound:261.94

11 Markets

LEAF global utility

Global utility values recorded: 702 453 839 626 813 749 388 775 460 676 702 657 473 638 733 733 632 774 673 675

Mean:658.55 Standard deviation:125.367

95% confidence lower bound:598.353 upper bound:718.747

LEAF learning convergence

Number of days taken to converge: 104 142 221 151 104 118 150 106 252 156 177 103 262 141 118 325 105 104 236 159

Mean:161.7 Standard deviation:64.412

95% confidence lower bound:130.771 upper bound:192.629

12 Markets

LEAF global utility

Global utility values recorded: 797 788 831 813 830 673 936 623 971 721 471 598 865 816 808 746 928 857 455 770

Mean:764.85 Standard deviation:140.483

95% confidence lower bound:697.395 upper bound:832.305

LEAF learning convergence

Number of days taken to converge: 105 159 143 105 105 157 139 125 454 103 104 219 161 143 102 344 193 102 328 106

Mean:169.85 Standard deviation:96.928

95% confidence lower bound:123.309 upper bound:216.391

13 Markets

LEAF global utility

Global utility values recorded: 978 771 788 364 800 693 771 840 360 895 871 490 883 657 974 919 924 971 722 843

Mean:775.7 Standard deviation:185.378

95% confidence lower bound:686.687 upper bound:864.713

LEAF learning convergence

Number of days taken to converge: 121 315 104 417 164 105 114 106 157 145 129 175 106 118 107 108 105 239 103 208

Mean:157.3 Standard deviation:82.427

95% confidence lower bound:117.721 upper bound:196.879

14 Markets

LEAF global utility

Global utility values recorded: 1274 746 473 959 726 454 1095 782 1006 1008 388 901 736 813 1071 897 997 931 798 862

Mean:845.85 Standard deviation:222.767

95% confidence lower bound:738.885 upper bound:952.815

LEAF learning convergence

Number of days taken to converge: 118 102 102 106 139 463 176 205 180 165 107 141 102 298 106 174 106 108 106 166

Mean:158.5 Standard deviation:86.93

95% confidence lower bound:116.759 upper bound:200.241

15 Markets

LEAF global utility

Global utility values recorded: 874 1134 1140 1099 1064 883 795 952 922 797 835 1134 1160 959 1193 1094 886 963 1105 1119

Mean:1005.4 Standard deviation:131.82

95% confidence lower bound:942.104 upper bound:1068.696

LEAF learning convergence

Number of days taken to converge: 358 204 136 105 212 103 133 112 235 144 212 115 346 106 176 647 104 105 120 106

Mean:188.95 Standard deviation:132.425

95% confidence lower bound:125.364 upper bound:252.536

16 Markets

LEAF global utility

Global utility values recorded: 898 466 1323 1212 1188 675 1143 917 1210 447 390 849 823 1367 1004 1036 861 1173 483 827

Mean:914.6 Standard deviation:301.39

95% confidence lower bound:769.882 upper bound:1059.318

LEAF learning convergence

Number of days taken to converge: 310 209 128 163 105 154 110 448 366 106 400 107 104 153 121 246 193 172 105 217

Mean:195.85 Standard deviation:106.154

95% confidence lower bound:144.878 upper bound:246.822

17 Markets

LEAF global utility

Global utility values recorded: 1006 1056 941 1256 1223 487 932 1290 1300 1157 1012 973 1018 848 1080 1136 1192 813 1185 485

Mean:1019.5 Standard deviation:229.569

95% confidence lower bound:909.269 upper bound:1129.731

LEAF learning convergence

Number of days taken to converge: 369 110 262 103 216 105 122 105 105 142 272 167 216 310 330 104 293 289 302 105

Mean:201.35 Standard deviation:93.865

95% confidence lower bound:156.279 upper bound:246.421

18 Markets

LEAF global utility

Global utility values recorded: 829 961 923 974 1188 846 965 483 1329 923 965 745 953 1235 832 785 1192 1330 486 1233

Mean:958.85 Standard deviation:241.446

95% confidence lower bound:842.916 upper bound:1074.784

LEAF learning convergence

Number of days taken to converge: 120 104 272 199 117 104 116 105 127 107 261 106 181 384 147 183 131 330 125 142

Mean:168.05 Standard deviation:81.780

95% confidence lower bound:128.782 upper bound:207.318

19 Markets

LEAF global utility

Global utility values recorded: `1333 949 1306 1008 1038 942 1291 998 1293 1028 990 946 1268 469 938 1257 1010 1336 1308 1187`

Mean:1094.75 Standard deviation:214.389

95% confidence lower bound:991.808 upper bound:1197.692

LEAF learning convergence

Number of days taken to converge: `102 106 148 400 210 105 340 105 335 124 113 120 107 249 202 104 319 218 120 104`

Mean:181.55 Standard deviation:97.448

95% confidence lower bound:134.759 upper bound:228.341

20 Markets

LEAF global utility

Global utility values recorded: `1258 1476 1355 1177 1231 945 1402 828 998 835 1047 916 1021 477 1429 1289 1021 1218 1287 1270`

Mean:1124 Standard deviation:248.343

95% confidence lower bound:1004.754 upper bound:1243.246

LEAF learning convergence

Number of days taken to converge: `102 236 154 305 106 140 105 526 153 105 125 102 294 169 337 205 120 104 111 131`

Mean:181.5 Standard deviation:109.613

95% confidence lower bound:128.867 upper bound:234.133

C.4 Computational task processing application

These results are referred to in section 6.5. For each different CT submission rate, 5 experiments were performed for each of the different platform configuration categories. In all cases the utility values were recorded at the $t = 40$ time step, as explained in section 6.5.1.

Global functional utility

		global functional utility at t=40 for the 5 experiments performed							95% confidence interval	
CTs/t	platform configuration	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	mean	standard deviation	lower bound	upper bound
16	all-Z	0.99063	0.99688	0.97031	1.00000	0.99219	0.99000	0.01162	0.97387	1.00613
32	all-Z	0.79844	0.81875	0.77500	0.80156	0.79297	0.79734	0.01578	0.77545	0.81924
48	all-Z	0.69583	0.65521	0.64219	0.68229	0.67708	0.67052	0.02156	0.64059	0.70045
64	all-Z	0.55820	0.56289	0.59219	0.59375	0.59063	0.57953	0.01744	0.55532	0.60374
80	all-Z	0.41563	0.41813	0.41406	0.41219	0.40031	0.41206	0.00692	0.40246	0.42167
96	all-Z	0.15156	0.14818	0.15182	0.15651	0.15208	0.15203	0.00297	0.14791	0.15615
16	all-X	0.65000	0.56094	0.55000	0.60156	0.63281	0.59906	0.04359	0.53856	0.65957
32	all-X	0.48359	0.47188	0.49219	0.46797	0.47422	0.47797	0.00981	0.46435	0.49159
48	all-X	0.44844	0.44271	0.44479	0.47760	0.45885	0.45448	0.01434	0.43457	0.47439
64	all-X	0.40000	0.39570	0.40938	0.38086	0.40977	0.39914	0.01188	0.38265	0.41563
80	all-X	0.28156	0.26781	0.26875	0.24281	0.26906	0.26600	0.01414	0.24637	0.28563
96	all-X	0.14583	0.13724	0.13854	0.12057	0.13151	0.13474	0.00942	0.12167	0.14781
16	all-P	0.46406	0.47344	0.48125	0.39375	0.49375	0.46125	0.03927	0.40674	0.51576
32	all-P	0.50078	0.40625	0.42500	0.40156	0.44219	0.43516	0.04007	0.37954	0.49078
48	all-P	0.41250	0.41042	0.38438	0.38906	0.38906	0.39708	0.01328	0.37865	0.41552
64	all-P	0.29453	0.28320	0.29219	0.28984	0.29961	0.29188	0.00605	0.28348	0.30027
80	all-P	0.13500	0.14000	0.12875	0.11344	0.11969	0.12738	0.01087	0.11229	0.14246
96	all-P	0.10286	0.09141	0.10833	0.08229	0.10000	0.09698	0.01024	0.08277	0.11119
16	mixed-PX	0.58906	0.57188	0.66563	0.60156	0.67031	0.61969	0.04535	0.55675	0.68263
32	mixed-PX	0.55000	0.58906	0.55703	0.56641	0.60000	0.57250	0.02129	0.54295	0.60205
48	mixed-PX	0.43281	0.41875	0.42083	0.44115	0.44844	0.43240	0.01279	0.41465	0.45014
64	mixed-PX	0.31367	0.29609	0.31563	0.32578	0.30625	0.31148	0.01107	0.29611	0.32686
80	mixed-PX	0.08938	0.09938	0.09875	0.10469	0.11250	0.10094	0.00850	0.08915	0.11273
96	mixed-PX	0.09896	0.09505	0.10078	0.09844	0.10104	0.09885	0.00241	0.09552	0.10219
16	mixed-ZP	0.75000	0.63594	0.69688	0.68281	0.63281	0.67969	0.04837	0.61254	0.74683
32	mixed-ZP	0.64063	0.61406	0.63594	0.59297	0.63203	0.62313	0.01963	0.59588	0.65037
48	mixed-ZP	0.48750	0.50833	0.48646	0.50625	0.46667	0.49104	0.01701	0.46743	0.51466
64	mixed-ZP	0.36680	0.36875	0.37695	0.37031	0.38711	0.37398	0.00827	0.36250	0.38546
80	mixed-ZP	0.27531	0.27000	0.27594	0.26844	0.25656	0.26925	0.00780	0.25842	0.28008
96	mixed-ZP	0.12943	0.12109	0.12734	0.11510	0.11927	0.12245	0.00589	0.11428	0.13062
16	mixed-ZX	0.88906	0.87656	0.86094	0.89063	0.87188	0.87781	0.01237	0.86064	0.89499
32	mixed-ZX	0.74375	0.72734	0.73438	0.76797	0.72188	0.73906	0.01811	0.71392	0.76420
48	mixed-ZX	0.59583	0.62604	0.61563	0.61198	0.62135	0.61417	0.01157	0.59810	0.63023
64	mixed-ZX	0.53945	0.55898	0.54336	0.56719	0.55078	0.55195	0.01132	0.53623	0.56767
80	mixed-ZX	0.38250	0.37469	0.36250	0.37688	0.36375	0.37206	0.00865	0.36005	0.38407
96	mixed-ZX	0.14557	0.14453	0.14297	0.13646	0.14453	0.14281	0.00367	0.13772	0.14791

Global performance utility

		global performance utility at t=40 for the 5 experiments performed							95% confidence interval	
CTs/t	platform configuration	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	mean	standard deviation	lower bound	upper bound
16	all-Z	-40.814	-43.457	-42.9766	-49.3314	-40.8979	-43.495	3.474	-48.317	-38.674
32	all-Z	-41.1587	-40.5082	-40.4179	-45.1458	-42.8445	-42.015	2.003	-44.795	-39.235
48	all-Z	-41.2422	-40.5325	-46.5756	-40.6974	-42.665	-42.343	2.511	-45.828	-38.858
64	all-Z	-41.3175	-44.2468	-41.8051	-41.063	-41.3768	-41.962	1.305	-43.773	-40.151
80	all-Z	-46.4916	-51.2448	-46.1272	-49.7912	-46.6151	-48.054	2.314	-51.266	-44.842
96	all-Z	-43.266	-40.5485	-45.8045	-47.0424	-50.025	-45.337	3.617	-50.357	-40.317
16	all-X	-403.103	-400.371	-405.686	-402.597	-403.509	-403.053	1.907	-405.700	-400.407
32	all-X	-404.888	-400.947	-406.737	-402.149	-399.614	-402.867	2.909	-406.904	-398.830
48	all-X	-402.305	-402.023	-405.052	-405.876	-414.896	-406.030	5.232	-413.293	-398.768
64	all-X	-401.205	-406.387	-400.451	-408.964	-402.445	-403.890	3.644	-408.948	-398.833
80	all-X	-401.427	-403.028	-407.338	-404.577	-401.324	-403.539	2.507	-407.018	-400.059
96	all-X	-405.272	-401.827	-402.601	-399.759	-403.958	-402.683	2.099	-405.596	-399.771
16	all-P	-441.405	-440.415	-441.576	-442.174	-446.009	-442.316	2.159	-445.313	-439.319
32	all-P	-458.479	-460.863	-474.986	-467.938	-479.624	-468.378	9.011	-480.885	-455.870
48	all-P	-473.998	-470.534	-459.919	-471.617	-461.839	-467.581	6.282	-476.301	-458.862
64	all-P	-454.488	-455.615	-447.398	-450.324	-458.239	-453.213	4.325	-459.215	-447.210
80	all-P	-504.135	-510.951	-518.277	-509.744	-512.527	-511.127	5.097	-518.201	-504.052
96	all-P	-542.207	-541.182	-533.106	-536.824	-533.067	-537.277	4.327	-543.283	-531.272
16	mixed-pX	-420.064	-419.214	-417.411	-423.717	-419.263	-419.934	2.327	-423.164	-416.704
32	mixed-pX	-448.001	-450.618	-449.574	-441.681	-450.927	-448.160	3.798	-453.432	-442.888
48	mixed-pX	-451.962	-450.914	-446.25	-446.862	-454.491	-450.096	3.490	-454.940	-445.252
64	mixed-pX	-451.581	-460.548	-441.977	-453.51	-447.407	-451.005	6.930	-460.624	-441.385
80	mixed-pX	-481.834	-487.449	-478.697	-482.532	-483.097	-482.722	3.143	-487.084	-478.359
96	mixed-pX	-467.255	-472.735	-468.757	-467.337	-474.923	-470.201	3.453	-474.993	-465.409
16	mixed-ZP	-254.594	-251.867	-261.603	-260.716	-254.026	-256.561	4.331	-262.572	-250.550
32	mixed-ZP	-252.799	-259.857	-258.948	-258.65	-251.189	-256.289	3.986	-261.822	-250.756
48	mixed-ZP	-259.75	-253.218	-255.263	-264.212	-252.839	-257.056	4.853	-263.792	-250.321
64	mixed-ZP	-254.131	-267.63	-261.47	-254.381	-252.585	-258.039	6.367	-266.877	-249.202
80	mixed-ZP	-257.164	-257.634	-258.529	-260.105	-259.641	-258.615	1.260	-260.363	-256.866
96	mixed-ZP	-260.89	-257.735	-258.487	-257.828	-259.986	-258.985	1.395	-260.921	-257.049
16	mixed-ZX	-216.78	-212.211	-216.623	-211.192	-211.836	-213.728	2.739	-217.530	-209.927
32	mixed-ZX	-233.593	-239.874	-224.001	-231.838	-227.049	-231.271	6.132	-239.782	-222.760
48	mixed-ZX	-235.559	-223.652	-226.421	-243.659	-226.203	-231.099	8.351	-242.690	-219.508
64	mixed-ZX	-226.832	-228.625	-233.581	-227.453	-237.617	-230.821	4.634	-237.253	-224.390
80	mixed-ZX	-230.659	-234.898	-234.723	-226.666	-228.882	-231.166	3.616	-236.185	-226.147
96	mixed-ZX	-236.703	-226.611	-230.395	-238.236	-226.985	-231.786	5.421	-239.310	-224.262

C.4.1 Profiled zinc agent

The following results were referred to in section 6.5.1. The local functional and performance utility of an agent residing on *zinc* is recorded over time.

time step	local functional utility	local performance utility
1	0.5	-10.452
2	0.541667	-10.452
3	0.5	-10.36
4	0.604167	-10.36
5	0.533333	-10.36
6	0.555556	-10.404
7	0.571429	-10.404
8	0.5	-10.712
9	0.555556	-10.67
10	0.6	-10.67
11	0.545455	-10.643
12	0.583333	-10.628
13	0.538462	-10.628
14	0.511905	-10.609
15	0.6	-10.609
16	0.5625	-10.953
17	0.588235	-10.953
18	0.555556	-10.953
19	0.578947	-10.893
20	0.6	-10.893
21	0.571429	-10.893
22	0.590909	-10.842
23	0.565217	-10.842
24	0.583333	-10.842
25	0.6	-10.842
26	0.576923	-10.842
27	0.592593	-10.637
28	0.571429	-10.637
29	0.586207	-10.637
30	0.6	-10.637
31	0.580645	-10.637
32	0.59375	-10.637
33	0.575758	-10.637
34	0.588235	-10.637
35	0.6	-10.637
36	0.583333	-10.637
37	0.594595	-10.648
38	0.578947	-10.637
39	0.589744	-10.637
40	0.6	-10.787

C.4.2 Average local utility of zinc agents

The following results were used to generate figures 6.8 and 6.7. The average local functional and local performance utility of agents residing on the host *zinc* were recorded in each different platform configuration involving *zinc* at all CT submission rates.

Functional utility

		average functional utility at t=40							95% confidence interval	
CTs/t	platform configuration	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	mean	standard deviation	lower bound	upper bound
16	all-Z	0.9906	0.9969	0.9703	1.0000	0.9922	0.9900	0.0116	0.9739	1.0061
32	all-Z	0.7984	0.8188	0.7750	0.8016	0.7930	0.7973	0.0158	0.7754	0.8192
48	all-Z	0.6958	0.6552	0.6422	0.6823	0.6771	0.6705	0.0216	0.6406	0.7004
64	all-Z	0.5582	0.5629	0.5922	0.5938	0.5906	0.5795	0.0174	0.5553	0.6037
80	all-Z	0.4156	0.4181	0.4141	0.4122	0.4003	0.4121	0.0069	0.4025	0.4217
96	all-Z	0.1516	0.1482	0.1518	0.1565	0.1521	0.1520	0.0030	0.1479	0.1561
16	mixed-ZP	0.7219	0.7094	0.5156	0.6750	0.7375	0.6719	0.0903	0.5465	0.7973
32	mixed-ZP	0.6313	0.6359	0.6406	0.6109	0.6188	0.6275	0.0123	0.6104	0.6446
48	mixed-ZP	0.5042	0.5354	0.4833	0.4594	0.4458	0.4856	0.0357	0.4361	0.5352
64	mixed-ZP	0.3211	0.3461	0.3492	0.3781	0.3102	0.3409	0.0265	0.3041	0.3778
80	mixed-ZP	0.3344	0.2644	0.2831	0.2969	0.2725	0.2903	0.0275	0.2521	0.3284
96	mixed-ZP	0.1125	0.1344	0.1161	0.1063	0.1411	0.1221	0.0149	0.1014	0.1428
16	mixed-ZX	0.9344	0.9219	0.8250	0.9344	0.9156	0.9063	0.0461	0.8422	0.9703
32	mixed-ZX	0.7156	0.7609	0.7109	0.7078	0.7547	0.7300	0.0256	0.6944	0.7656
48	mixed-ZX	0.6219	0.6042	0.6333	0.6375	0.6615	0.6317	0.0211	0.6024	0.6609
64	mixed-ZX	0.5445	0.5375	0.5320	0.5547	0.5500	0.5438	0.0092	0.5310	0.5565
80	mixed-ZX	0.3650	0.3619	0.3988	0.4156	0.3744	0.3831	0.0232	0.3509	0.4154
96	mixed-ZX	0.1359	0.1432	0.1354	0.1464	0.1458	0.1414	0.0053	0.1340	0.1487

Performance utility

		average performance utility at t=40							95% confidence interval	
CTs/t	platform configuration	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	mean	standard deviation	lower bound	upper bound
16	all-Z	-10.2035	-10.8642	-10.7442	-12.3328	-10.2245	-10.8738	0.8684	-12.0792	-9.6685
32	all-Z	-10.2897	-10.1271	-10.1045	-11.2865	-10.7111	-10.5038	0.5007	-11.1987	-9.8088
48	all-Z	-10.3106	-10.1331	-11.6439	-10.1743	-10.6662	-10.5856	0.6277	-11.4569	-9.7144
64	all-Z	-10.3294	-11.0617	-10.4513	-10.2658	-10.3442	-10.4905	0.3262	-10.9433	-10.0377
80	all-Z	-11.6229	-12.8112	-11.5318	-12.4478	-11.6538	-12.0135	0.5786	-12.8165	-11.2104
96	all-Z	-10.8165	-10.1371	-11.4511	-11.7606	-12.5063	-11.3343	0.9042	-12.5893	-10.0793
16	mixed-ZP	-12.2895	-10.6246	-12.0950	-12.1498	-10.3665	-11.5051	0.9288	-12.7942	-10.2159
32	mixed-ZP	-10.2679	-11.9908	-10.5233	-11.4525	-10.4765	-10.9422	0.7428	-11.9732	-9.9112
48	mixed-ZP	-10.8063	-13.9598	-12.5342	-11.3776	-10.1173	-11.7590	1.5160	-13.8632	-9.6549
64	mixed-ZP	-12.4599	-12.9215	-10.8044	-10.1469	-11.5217	-11.5709	1.1437	-13.1583	-9.9834
80	mixed-ZP	-12.3999	-11.7690	-11.8602	-11.7302	-10.8436	-11.7206	0.5596	-12.4973	-10.9439
96	mixed-ZP	-11.5287	-13.3070	-11.7034	-10.2919	-11.3188	-11.6300	1.0857	-13.1370	-10.1229
16	mixed-ZX	-12.3800	-12.7001	-12.2540	-10.5387	-10.8793	-11.7504	0.9720	-13.0995	-10.4013
32	mixed-ZX	-10.7979	-10.6928	-10.5107	-12.0539	-10.9159	-10.9942	0.6108	-11.8420	-10.1465
48	mixed-ZX	-12.9723	-11.8317	-10.3632	-10.6956	-10.7948	-11.3315	1.0692	-12.8155	-9.8475
64	mixed-ZX	-12.4457	-10.4858	-10.3173	-12.9335	-11.5442	-11.5453	1.1585	-13.1533	-9.9373
80	mixed-ZX	-11.8162	-10.4318	-11.4400	-11.6932	-12.6889	-11.6140	0.8109	-12.7396	-10.4884
96	mixed-ZX	-10.5654	-12.1063	-15.7042	-10.7181	-10.7814	-11.9751	2.1746	-14.9934	-8.9568

C.5 Computational task processing application: scalability

These results are referred to in section 7.6. The global performance utility, average local performance utility and average local functional utility values were recorded for each platform configuration at the $t = 40$ time step. The number of agents deployed on each host ranged from 2 to 10 agents. For each number of agents, 5 experiments were performed.

Global functional utility

	global functional utility at t=40							95% confidence interval	
number of agents	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	mean	standard deviation	lower bound	upper bound
16	0.11159	0.11224	0.11549	0.10365	0.11576	0.11174	0.00490	0.10494	0.11854
24	0.11285	0.10911	0.11311	0.11632	0.10938	0.11215	0.00299	0.10801	0.11630
32	0.11400	0.11628	0.11348	0.10931	0.11230	0.11307	0.00255	0.10953	0.11661
40	0.11500	0.11141	0.11286	0.11391	0.11469	0.11357	0.00147	0.11154	0.11561
48	0.11168	0.11363	0.11398	0.11510	0.11293	0.11346	0.00127	0.11170	0.11523
56	0.11261	0.11362	0.11414	0.11373	0.11310	0.11344	0.00059	0.11262	0.11426
64	0.11211	0.11357	0.11090	0.11439	0.11309	0.11281	0.00135	0.11094	0.11468
72	0.10929	0.10906	0.10868	0.10735	0.10851	0.10858	0.00075	0.10753	0.10962
80	0.09773	0.09833	0.09727	0.09789	0.09776	0.09780	0.00038	0.09727	0.09833

Global performance utility

	global performance utility at t=40							95% confidence interval	
number of agents	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	mean	standard deviation	lower bound	upper bound
16	-1171.00	-1288.36	-1058.00	-1247.67	-1074.56	-1167.92	102.07	-1309.60	-1026.24
24	-1226.00	-1230.88	-1135.71	-1339.08	-1006.86	-1187.71	124.15	-1360.03	-1015.39
32	-1381.00	-1151.30	-1054.35	-1226.79	-1217.02	-1206.09	119.55	-1372.03	-1040.16
40	-984.00	-977.47	-1202.62	-1134.58	-1223.27	-1104.39	117.57	-1267.58	-941.20
48	-1645.00	-1520.44	-1631.06	-1687.89	-1684.15	-1633.71	67.91	-1727.96	-1539.46
56	-1521.00	-1672.18	-1634.81	-1776.51	-1588.72	-1638.64	95.50	-1771.19	-1506.10
64	-1578.00	-1747.89	-1488.59	-1693.85	-1754.72	-1652.61	115.84	-1813.39	-1491.83
72	-1652.00	-1724.80	-1657.81	-1736.00	-1669.22	-1687.97	39.43	-1742.69	-1633.24
80	-1656.00	-1739.50	-1700.17	-1646.08	-1687.28	-1685.81	37.27	-1737.54	-1634.07

Local performance utility

number of agents	average local performance utility at t=40 exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	mean	standard deviation	95% confidence interval lower bound	upper bound
16	-936.240	-1012.620	-663.793	-837.131	-987.809	-887.518	142.016	-1084.637	-690.400
24	-919.868	-806.619	-810.089	-956.768	-959.433	-890.555	76.661	-996.961	-784.150
32	-719.641	-900.555	-972.846	-963.558	-901.462	-891.612	101.890	-1033.036	-750.189
40	-929.663	-857.657	-913.236	-1071.613	-906.039	-935.642	80.598	-1047.511	-823.772
48	-596.743	-736.281	-563.687	-628.319	-737.755	-652.557	80.419	-764.179	-540.935
56	-590.073	-271.531	-513.861	-471.752	-493.812	-468.206	118.623	-632.854	-303.558
64	-331.252	-331.955	-386.515	-216.387	-294.526	-312.127	62.786	-399.275	-224.979
72	-337.973	-237.012	-184.485	-352.321	-117.319	-245.822	100.233	-384.946	-106.699
80	-163.428	-263.497	-249.238	-168.483	-465.123	-261.954	122.358	-431.786	-92.121

C.6 FIPA-OS message delivery platform comparison

These results are referred to in section B.1. For each different message size, 20 messages were sent between 2 agents and the average message delivery time was recorded. The average time taken to deliver messages between agents on the same, and different hosts were recorded for each message size.

Agents residing on the same host

message	message size							
number	125KB	250KB	375KB	500KB	625KB	750KB	875KB	1000KB
1	0.411	0.813	1.069	1.042	1.682	2.369	2.216	2.502
2	0.413	1.357	1.386	1.282	1.58	2.174	1.888	2.73
3	0.839	0.874	1.194	1.543	1.804	1.226	2.51	2.655
4	0.583	1.1	0.841	1.205	2.108	1.704	2.817	2.373
5	0.472	0.738	1.816	1.161	1.906	1.532	2.32	2.529
6	1.209	0.881	1.257	1.531	1.493	2.318	2.238	2.294
7	0.401	0.725	1.16	1.294	1.405	1.715	1.704	3.101
8	0.493	0.719	1.446	1.429	1.699	2.024	1.861	2.162
9	1.3	0.745	1.273	1.452	2.077	2.161	2.067	2.364
10	0.588	0.918	1.171	1.345	1.79	2.296	2.388	2.088
11	1.092	1.042	1.125	2.059	2.245	2.331	2.26	2.612
12	0.458	0.769	1.868	1.271	1.511	2.287	2.857	2.439
13	0.975	0.76	0.962	1.282	1.969	2.161	2.253	1.996
14	0.569	1.043	0.932	1.622	2.129	1.691	2.105	1.98
15	0.471	0.994	0.981	1.64	1.504	1.554	1.913	1.995
16	0.591	1.111	1.01	1.33	1.289	1.977	1.894	1.918
17	0.639	0.901	1.309	1.403	1.911	2.264	2.007	1.72
18	0.638	1.184	1.254	1.729	1.092	2.327	2.166	2.724
19	0.737	0.854	0.772	1.5	1.702	2.152	1.779	2.336
20	0.671	0.868	0.988	1.153	1.472	2.298	2.706	2.046
mean	0.6775	0.9198	1.1907	1.41365	1.7184	2.02805	2.19745	2.3282
standard deviation	0.269868	0.174435	0.284442	0.2345	0.305255	0.336717	0.332363	0.343031
95% confidence interval								
upper bound	0.547918	0.836042	1.05412	1.301051	1.571827	1.86637	2.03786	2.163488
lower bound	0.807082	1.003558	1.32728	1.526249	1.864973	2.18973	2.35704	2.492912

Agents residing on different hosts

message number	message size							
	125KB	250KB	375KB	500KB	625KB	750KB	875KB	1000KB
1	2.151	1.807	1.793	2.436	3.182	3.533	3.575	3.762
2	2.027	1.904	3.13	3.028	3.312	2.82	3.371	3.836
3	1.927	2.177	2.551	2.5	2.73	3.062	3.722	3.293
4	1.287	1.99	2.482	2.55	3.464	3.36	3.769	3.574
5	1.352	2.129	2.467	2.611	3.359	3.337	2.886	4.143
6	1.519	2.132	2.395	3.298	2.75	3.15	3.467	4.18
7	1.81	1.779	2.927	2.848	2.908	3.251	3.912	4.019
8	2.189	1.636	2.115	2.78	3.159	3.262	3.078	4.366
9	1.38	1.943	2.486	2.569	2.957	3.071	3.392	4
10	1.393	2.099	2.383	1.729	2.779	3.209	3.954	4.204
11	1.803	2.635	2.555	2.96	3.083	3.666	3.337	3.867
12	1.885	2.539	2.049	3.033	3.331	3.049	3.196	3.985
13	1.531	1.822	2.099	2.326	2.9	3.236	3.934	3.858
14	1.552	2.329	2.194	2.304	2.586	3.795	4.078	3.954
15	1.61	1.907	1.925	2.54	2.739	3.845	3.69	4.307
16	1.996	2.169	2.513	2.928	3.124	3.53	3.455	4.151
17	2.116	2.398	1.83	2.642	2.687	3.343	3.254	4.376
18	2.311	2.035	1.773	2.061	2.908	3.218	3.362	3.5
19	1.879	1.442	2.199	3.106	2.913	3.526	3.54	3.719
20	1.688	1.7	2.393	2.789	3.277	2.66	3.98	4.325
mean	1.7703	2.0286	2.31295	2.6519	3.0074	3.29615	3.5476	3.97095
standard deviation	0.308212	0.301039	0.357448	0.375888	0.258076	0.298974	0.327141	0.299998
95% confidence interval								
upper bound	1.622307	1.884051	2.141315	2.471411	2.883481	3.152592	3.390518	3.826901
lower bound	1.918293	2.173149	2.484585	2.832389	3.131319	3.439708	3.704682	4.114999

C.7 FIPA-OS scalability investigation

These results are referred to in section B.2. 9 experiments were performed for each different number of agents in the system. The average time taken to deliver messages between agents, and the average percentage usage of JVM memory were recorded.

number of agents	percentage JVM usage in different experiments											95% confidence interval	
	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	exp. 6	exp. 7	exp. 8	exp. 9	mean	standard deviation	lower	upper
2	1.796	2.298	2.512	0.577	1.855	1.310	1.776	1.025	3.220	1.819	0.799	1.167	2.470
3	4.372	5.571	5.868	3.795	2.643	1.943	5.864	1.975	2.613	3.849	1.640	2.512	5.186
4	4.292	8.341	9.384	2.659	2.354	3.951	6.636	6.550	9.911	6.009	2.839	3.694	8.324
5	9.953	15.072	5.756	5.689	3.663	11.394	6.464	11.466	14.625	9.343	4.132	5.974	12.711
6	5.919	10.986	7.040	15.026	6.448	15.209	15.582	7.783	12.535	10.725	4.018	7.449	14.001
7	10.561	12.857	11.799	12.357	10.800	21.726	14.655	26.908	10.046	14.634	5.806	9.901	19.368
8	26.022	8.528	17.417	22.950	14.399	23.352	9.535	27.786	14.529	18.280	7.065	12.520	24.040
9	37.001	39.835	22.352	31.597	44.115	40.044	56.353	38.691	38.010	38.667	9.118	31.233	46.100
10	39.053	41.885	38.268	32.444	49.698	48.934	38.755	55.375	35.454	42.207	7.526	36.071	48.344

number of agents	average message delivery time in different experiments											95% confidence interval	
	exp. 1	exp. 2	exp. 3	exp. 4	exp. 5	exp. 6	exp. 7	exp. 8	exp. 9	mean	standard deviation	lower	upper
2	2.635	5.507	6.500	5.484	3.138	6.397	2.537	3.374	7.335	4.767	1.852	3.258	6.277
3	8.905	6.628	10.975	15.854	8.249	7.571	2.628	7.158	13.708	9.075	3.957	5.849	12.301
4	12.547	7.482	11.320	6.536	10.047	13.945	16.091	12.211	12.093	11.363	2.995	8.921	13.805
5	18.445	12.767	12.425	14.671	19.437	19.059	19.235	13.180	17.001	16.247	2.979	13.818	18.676
6	20.813	21.452	16.865	24.216	16.079	10.732	14.784	17.815	18.096	17.872	3.977	14.630	21.115
7	14.839	19.460	21.156	14.051	19.371	12.834	25.645	25.656	18.102	19.013	4.665	15.210	22.816
8	39.889	35.795	29.188	32.505	38.524	25.002	24.059	27.870	25.526	30.929	6.000	26.037	35.821
9	74.929	77.950	70.535	69.698	72.888	91.906	73.351	82.259	89.561	78.120	8.115	71.503	84.736
10	240.380	228.039	238.231	236.005	241.685	230.182	232.648	230.922	245.879	235.997	6.003	231.102	240.891

APPENDIX D

The LEAF Ping Agent tutorial

D.1 Introduction

This tutorial gives an overview of how to utilise the LEAF toolkit to implement utility function assignment. This tutorial builds on the corresponding FIPA-OS Ping Agent tutorial [3], which describes the construction of a FIPA-OS agent which does the following when deployed:

1. Registers with the local FIPA agent platform
2. Searches for other Ping Agents, and adds any discovered agents to the "ping-list".
3. Responds to incoming ping messages and adds the sending agent to the ping-list if not already listed.
4. Periodically pings all agent in the ping list. Pings are FIPA ACL messages following the FIPA-request conversation protocol.

This tutorial aims to demonstrate to the reader the key differences in using the LEAF toolkit as opposed to using FIPA-OS directly, in addition to the implementation of the following:

1. A ping community, which Ping Agents join when they are deployed.
2. Local utility function assignment and the management of observable properties/remote parameters.

The full source code associate with this tutorial may be obtained from the LEAF website at:
`http://www.cs.cardiff.ac.uk/User/S.J.Lynden/leaf`

D.2 The Ping Agent

Initialising the agent

The Ping Agent is based around the `PingAgent` class, which extends the `LeafNode` class. When invoked, the `PingAgent` class constructor invokes the `LeafNode` constructor as follows:

```
super( platform, name, ''cs.cf.ac.uk'', ''ping-agent'');
```

Where `platform` specifies the location of the FIPA-OS platform profile, `name` specifies the name of the agent, `cs.cf.ac.uk` is the owner of the agent, and `ping-agent` defines the Ping Agent's LEAF agent type. Following the invocation of the constructor, the Ping Agent is registered with the local agent platform AMS and DF agents. The Ping Agent specifies two observable properties, which are visible to community ESNs and utility functions for the purposes of computing local and global utility. These observable properties are declared as follows:

```
addObservableProperty(''number-of-pings'',''0'');
```
Declares an observable property that is used to record the number of ping messages that the agent has received.

```
addObservableProperty(DefaultOntology.PERFORMANCE_UTILITY,''0'');
```
Declares an observable property that is used to record the agents current performance utility value.

Searching for other agents

Whereas the FIPA-OS Ping Agent uses a Java `List` data structure to store references to other agents, references are automatically stored by the local agent database when using LEAF. To add agents to the local agent database, the `discoverAgents` method is used as follows:

```
discoverAgents( ''ping-agent'', false );
```

The first parameter of this method is used to specify that the search should identify agents with the Ping Agent LEAF agent type. The second parameter is used to specify that a federated search should not be performed (only the local platform DF agent will be queried).

Joining a Ping Community

The Ping Agent joins the first Ping Community it is able to find by searching the local platform DF agent. In order to perform a non-federated search of the DF, the following code is used:

```
discoverCommunities( ''ping-community'', false );
```

Which adds to the local community database community ESNs of the type given by the first parameter. Once this search has completed, LEAF will invoke the agent's `doneCommunitySearch` method, which then joins a community as follows:

```
joinCommunity( (AgentID) newESN.get(0) );
```

Where `newESN` is a Java `List` object (containing the FIPA agent IDs (AIDs) of discovered communities) passed as a parameter to the `doneCommunitySearch` method. The agent joins the first community ESN contained in this list, indexed by 0.

Agent behaviour

The main agent behaviour consists of responding to ping messages received from other agents, and sending ping agents to all known agents every 5 minutes. Whereas the FIPA-OS Ping Agent uses an IdleTask to wait for incoming messages, LEAF agents utilise an incoming message queue to store messages. The queue is examined for new messages using:

```
Conversation c = popIncomingConversation();
```

When incoming ping messages are received, or it is time to send ping messages to other agents, specific tasks are used to carry out these functions, which are now described.

D.2.1 Tasks

The Ping Agent uses three tasks, the PingAllTask, PingTask, and PingResponse-Task. Each of these tasks is defined by a single class, extending the `LeafTask` class. The functionality of these tasks, and the relationships between them, are illustrated in figure D.1.

The PingAllTask

The PingAllTask is used by an agent to ping all agents known to the agent. The only difference between the PingAllTask defined here and the PingAllTask as defined in the FIPA-OS is that LEAF is able to take advantage of the local agent database, whereas the FIPA-OS agent must use a separate data structure the maintain references to other agents.

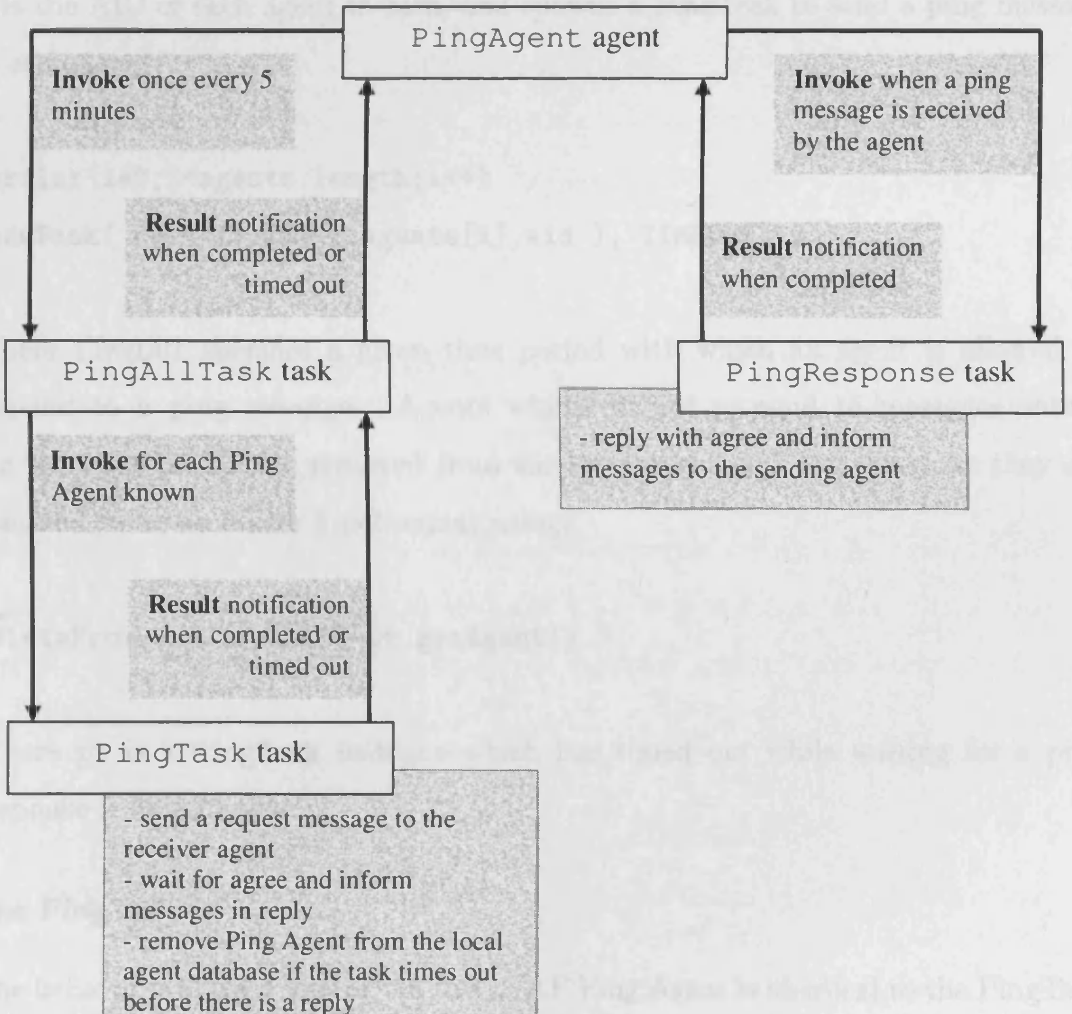

PingAgent agent
Invoke once every 5 minutes
Result notification when completed or timed out
Invoke when a ping message is received by the agent
Result notification when completed
PingAllTask task
PingResponse task
- reply with agree and inform messages to the sending agent
Invoke for each Ping Agent known
Result notification when completed or timed out
PingTask task
- send a request message to the receiver agent
- wait for agree and inform messages in reply
- remove Ping Agent from the local agent database if the task times out before there is a reply

Figure D.1: The Ping Agent task structure

The LEAF PingAgent is able to determine which agents to ping as follows:

```
LeafAgent[] agents = searchLocalAgentDB( ''ping-agent'');
```

Each agent is then sent a ping message using the PingTask. The following code gets the AID of each agent in turn, and spawns a PingTask to send a ping message to each agent:

```
for(int i=0;i<agents.length;i++)
 newTask( new PingTask( agents[i].aid ), TIMEOUT );
```

Where `TIMEOUT` specifies a given time period with which an agent is allowed to respond to a ping message. Agents which do not respond to messages within the time out period are removed from the local agent database (because they are assumed to be no longer functioning) using:

```
deleteFromLocalAgentDB( pt.getAgent() );
```

Where `pt` is a `PingTask` instance which has timed out while waiting for a ping response from an agent.

The PingTask

The behaviour of the PingTask in the LEAF Ping Agent is identical to the PingTask in the FIPA-OS agent. A ping message is sent using the FIPA request performative, and the PingTask waits for a reply (an agree performative followed by an inform performative) following the FIPA-request conversation protocol.

The PingResponseTask

A PingResponseTask is launched by the main agent behaviour when a ping message has been removed from the incoming message queue. In the same way as the FIPA-OS Ping Agent, the LEAF PingResponseTask sends agree and inform messages in reply to the sent ping request message. This task also attempts to add the sending agent to the local agent database:

```
LeafAgent a = new LeafAgent( msg.getSenderAID() , ''ping-agent'' );
addToLocalAgentDB(a);
```

This will construct a new `LeafAgent` instance, `a`, which encapsulates the reference to the sender of the ping message. The `addToLocalAgentDB` returns a `boolean` value, equal to true if the agent was added to the local agent database, or false if the agent was already present in the database.

D.3 Local Utility functions

D.3.1 Performance Utility

Performance utility is based on the memory usage of a Ping Agent, and is encapsulated by a class implementing the `LocalUtilityFunction` class. The `compute` method of this class, invoked by LEAF to compute utility, calculates utility using the following code:

```
double percentageMemoryUsage = Double.parseDouble(
   source.getPerformanceData(DataSourceConstants.MEMORY_USAGE,
null));
return 1-(percentageMemoryUsage/100);
```

This calculates the performance utility of the agent to which the utility function is assigned as $1 - (p/100)$, where p is the percentage available memory used by the agent. It is necessary to use the `Double.parseDouble` method in order to convert the `String` return type from the `source.getPerformanceData` method.

D.3.2 Functional Utility

Functional utility is based on the average number of pings received per minute by agents belonging to the same community as the Ping Agent. The following lines of code are used within the local utility function `compute` method:

```
float pings = Float.parseFloat(source.getRemoteParameter
   (esnSource,''global-pings-per-minute''));
```

Declares the `pings` variable, obtained by retrieving the "global-pings-per-minute" remote parameter, distributed by the ESN identified by the `esnSource` field, which holds a FIPA-OS `AgentID` object. The `esnSource` field is assigned this value by the ESN prior to the assignment of the utility function.

```
float minutes = (float) (System.currentTimeMillis()- t1)/60000;
```

Declares the `minutes` variable, which holds the number of minutes since the community was formed. The variable `t1`, which records the time in milliseconds when the community was formed, is assigned by the ESN prior to the assignment of the utility function.

```
return (double) pings / minutes;
```

Finally, the local utility value is returned as the average number of ping messages received per minute within the community.

D.4 The Ping Community ESN

The Ping Community ESN is implemented by a class extending the `ESN` class. When invoked, the constructor of this class invokes the `ESN` class constructor as follows:

```
super(platform,name,''cs.cf.ac.uk'',''ping-community'');
```

Where `platform` specifies the location of the FIPA-OS platform profile, `name` specifies the name of the ESN, `cs.cf.ac.uk` is the owner of the ESN, and `ping-community` is a constant defines the Ping Community's LEAF community type. Following the invocation of the superclass constructor, the Ping Community is registered with the local agent platform AMS and DF agents.

D.4.1 Utility function assignment

When a Ping Agent joins a community represented by a community ESN, the `joiningLeafAgent(LeafAgent)` method is invoked by LEAF. This method is an abstract class defined in the `ESN` class which must be implemented by application specific ESNs. When this method is invoked, a Ping Community assigns utility functions to the joining agent as follows:

```
assignUtilityFunctions(agent,new FunctionalUtilityFunction(getAID()),
new PerformanceUtilityFunction(),new String[] ''number-of-pings'',
DefaultOntology.PERFORMANCE_UTILITY,0 );
```

Which creates and assigns both performance and functional utility functions to the Ping Agent specified by `agent`. The third parameters specifies that the ESN requires two observable properties, "number-of-pings" and the performance utility of the agents. Finally, the fourth parameter specifies the time period with which observable properties should be updated by the Ping Agent, where the value 0 is used to indicate that observable properties should be sent each time their values change.

D.4.2 Global utility

The ESN calculates global performance utility as the sum of the local performance utilities of the agents in the community, and global functional utility is calculated as the average number of ping messages received by agents belonging to the community since the community ESN was deployed. In order to compute global utility and distribute the "global-pings-per-minute" remote parameter to ping agents, the Ping ESN must periodically retrieve the specified remote parameters. This is achieved using:

```
getRemoteProperty(agent,''number-of-pings'');
```

and

```
getRemoteProperty(agent,DefaultOntology.PERFORMANCE_UTILITY);
```

Where **agent** is the Ping Agent from which the observable properties should be retrieved. Once these observable properties are obtained from each agent in the community, the ESN is able to calculate functional utility by summing the total number of pings received by agent in the community and dividing this number by the number of minutes for which the community has existed. The global performance utility is obtained by summing the individual local performance utility values of the agents in the community.

D.4.3 Remote parameters

The ESN must distribute the "global-pings-per-minute" remote parameter to all Ping Agents, which is achieved by:

```
for(int i=0;i<agents.length;i++)
 pushParameter(agents[i],''global-pings-per-minute'',ppm);
```

Where `ppm` is the value of pings received per minute calculated by the ESN. The ESNs behaviour involves repeatedly re-calculating global utilities and remote parameters, and re-distributed these parameters to the agents.

D.5 Deployment

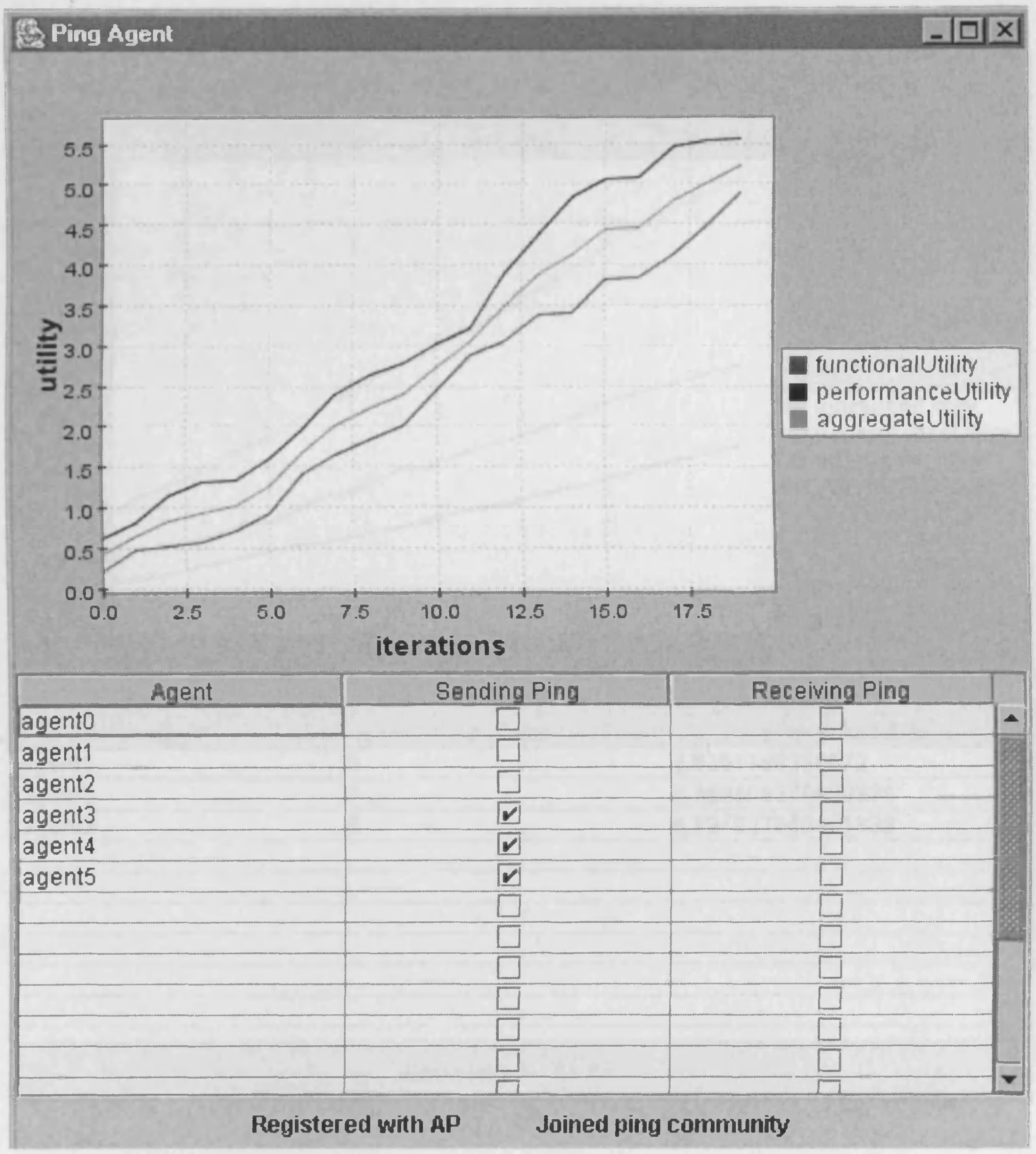
The Ping Community ESN must be deployed before the agents, which can be achieved from the command prompt as follows:

```
java leaf.agentlib.ping.PingCommunity <platform_profile> <name>
```

Assuming that the local FIPA agent platform specified by the profile located by `platform_profile` is available, this will create a LEAF community with a name specified by `<name>`. Ping Agents may then by deployed as follows:

```
java leaf.agentlib.ping.PingAgent <platform_profile> <name>
```

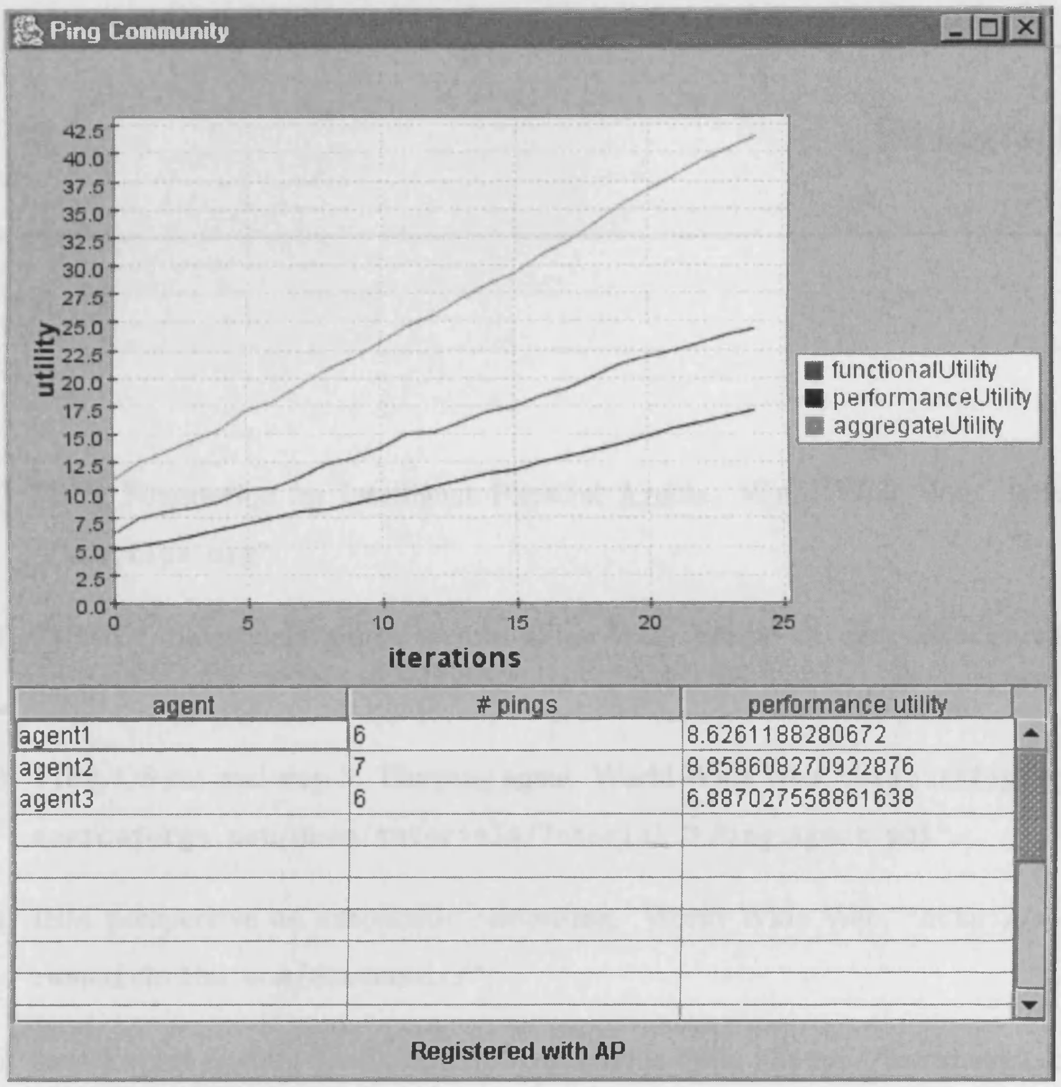
Which creates, initialises, and starts the behaviour of a Ping Agent with the specified name. Both the ESN (see figure D.3) and agents (see figure D.2) utilise graphical user interfaces (GUIs) to visualise their utility functions and behaviours.



Agent	Sending Ping	Receiving Ping
agent0	☐	☐
agent1	☐	☐
agent2	☐	☐
agent3	✓	☐
agent4	✓	☐
agent5	✓	☐
	☐	☐
	☐	☐
	☐	☐
	☐	☐
	☐	☐
	☐	☐

Figure D.2: The Ping Agent GUI

Utility function visualisation is aided by the use of JFreeChart [7]. The default aggregation function (sum) is used to define the aggregate of local performance and local functional utility. In this example, the agent is in the process of pinging agents 3,4 and 5.



agent	# pings	performance utility
agent1	6	8.6261188280672
agent2	7	8.858608270922876
agent3	6	6.887027558861638

Figure D.3: The Ping Community GUI

Utility function visualisation is aided by the use of JFreeChart [7]. The default aggregation function (sum) is used to define the aggregate of global performance and global functional utility.

Bibliography

[1] FIPA: Foundation for Intelligent Physical Agents. World Wide Web, "http://www.fipa.org".

[2] FIPA-OS developers guide. World Wide Web, "fipa-os.sourceforge.net/docs/".

[3] FIPA-OS tutorial step 3: The ping agent. World Wide Web, "http://fipa-os.sourceforge.net/docs/tutorials/Tutorial_3_Ping_Agent.pdf".

[4] IBM perspective on autonomic computing. World Wide Web, "http://www.research.ibm.com/autonomic/".

[5] Java Expert System Shell (JESS). World Wide Web, "http://herzberg1.ca.sandia.gov/jess/".

[6] Java version 1.3.0 documentation. World Wide Web, "http://java.sun.com/j2se/1.3/docs/api/".

[7] Jfreechart. World Wide Web, "http://www.jfree.org/jfreechart/index.html".

[8] UDDI technical white paper. World Wide Web, "http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf", 2000.

[9] E. Alonso, M. d'Inverno, D. Kudenko, M. Luck, and J.Noble. Learning in multi-agent systems. *Knowledge Engineering Review*, 16(3):277–284, 2001.

[10] W. Brian Arthur. Bounded rationality and inductive behavior (the El Farol problem). *American Economic Review*, 84:406–411, 1994.

[11] N.M. Avouris and L. Gasser. *Distributed Artificial Intelligence: Theory and Praxis.* Kluwer Academic Publishers, 1992.

[12] F. Bellifemine, A Poggi, and G. Rimassa. JADE A FIPA-compliant agent framework. In *Proceedings of the Fourth International Conference on the Practical Application of Intelligent Agent and Multi Agent Technology (PAAM99)*, pages 97–108, London, U.K., 1999. The Practical Application Company Ltd.

[13] S.H. Bierman and L. Fernandez, editors. *Game Theory with Economic Applications.* Addison-Wesley, 1993.

[14] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills, and Y. Diao. ABLE: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, 41(3):350–371, 2002.

[15] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence : from natural to artificial systems.* Oxford University Press, 1999.

[16] N. Borselius. Security in multi-agent systems. In *Proceedings of the 2002 International Conference on Security and Management (SAM'02)*, pages 31–36, Las Vegas, Nevada, USA, June 2002. Computer Science Research, Education and Applications Press.

[17] K. Bouzouba and B. Moulin. KQML+: An extension of KQML in order to deal with implicit information and social relationships. In *Proceedings of the Eleventh International Florida Artificial Intelligence Research Society Conference, Sanibel Island, Florida, USA*, pages 289–293. AAAI Press, 1998.

[18] M. Breugst, S. Choy, L. Hagen, M. Hoft, and T. Magedanz. Grasshopper - an agent platform for mobile agent-based services in fixed and mobile telecommunications environments. In A.L.G. Hayzelden and J. Bigham, editors, *Software*

Agents for Future Communication Systems, pages 326–357. Springer Verlag, 1999.

[19] P. Buckle, T. Moore, A. Treadway, S. Tarkoma, and S. Poslad. Scalability in multi-agent systems: the FIPA-OS perspective. In *LNAI Vol 2403, UKMAS proceedings*, pages 110–130. Springer Verlag, 2002.

[20] M. Calisti, E. Rollon, and S. Willmott. E-banking services for automated agent-based trading. In *Proceedings of the first International Workshop on Challenges in Open Agent Systems. (Held in conjunction with AAMAS 2002, Bologna, Italy)*. World Wide Web, "`http://www.agentcities.org/Challenge02/Proc/`", 2002.

[21] Philip R. Cohen and Hector J. Levesque. Communicative actions for artificial agents. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 65–72, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA.

[22] R.H. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. *Advances in Neural Information Processing Systems*, 8:1017–1023, 1996.

[23] M. desJardins, E. Durfee, C. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 4:13–22, 1999.

[24] T.W. Finin, Y. Labrou, and J. Mayfield. KQML as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM)*, pages 456–463, Gaithersburg, Maryland, USA, November 1994. ACM Press.

[25] D. Fitoussi and M. Tennenholtz. Choosing social laws for multi-agent systems: Minimality and simplicity. *Artificial Intelligence*, 119:61–102, 2000.

[26] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[27] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: enabling scalable virtual organisations. *Journal on High Performance Computing Applications*, 15:200–222, 2001.

[28] I.T. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, San Francisco, CA, USA, 1999.

[29] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Agent Theories, Architectures, and Languages (ATAL '96)*, pages 21–35, Budapest, Hungry, August 1996. Springer-Verlag.

[30] M. Girvan and M.E.J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99:8271–8276, 2002.

[31] K. Gottschalk, S. Graham, H. Kreger, and J. Snell. Introduction to Web Services architecture. *IBM Systems Journal*, 41(2):170–177, 2002.

[32] J.N. Hagstrom and R.A. Abrams. Characterizing Braess' paradox for traffic networks. In *Proceedings of the IEEE 2001 Conference on Intelligent Transpotation Systems*, pages 837–842. IEEE Computer Society Press, 2001.

[33] Garrett Hardin. The Tragedy of the Commons. *Science*, 162:1243–1248, 1968.

[34] E. R. Harold. *Java Network Programming.* O'Reilly and Associates, 1997.

[35] J. Holland. Genetic algorithms. *Scientific American*, pages 44–50, July 1992.

[36] J. Holland. *Emergence: From Chaos to Order.* Addison-Wesley, 1997.

[37] J. Hu and M.P. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm. In *Proceedings of the 15th International Conference on Machine Learning*, pages 242–250, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.

[38] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[39] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[40] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[41] R. MacEntire and D. McKay. KQML lite specification. Technical Report ALP-TR/03, Lockheed Martin Mission Systems Technical Report, 1998.

[42] P. Maes, R. Guttman, and A. Moukas. Agents that buy and sell: Transforming commerce as we know it. *Communications of the ACM (CACM)*, 42(3):81–91, 1999.

[43] R. Makar, S. Mahadevan, and M. Ghavamzadeh. Hierarchical multiagent reinforcement learning. In *Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS 2001)*, pages 67–68, Montreal, Canada, May 2001. ACM Press.

[44] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, B. Virdhagriswaran, and J. White. MASIF: The OMG Mobile Agent System Interoperability Facility. In Kurt Rothermel and Fritz Hohl, editors, *Proceedings of the International Workshop on Mobile Agents (MA'98)*, volume 1477 of *Lecture Notes in Computer Science*, pages 50–67, Stuttgart, September 1998. Springer-Verlag Inc.

[45] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, Hewlett Packard Laboratories, 2002.

[46] H.P. Nii. *Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures*, pages 447–474. Computation & Intelligence. MIT Press, 1995.

[47] P. O'Brien and R. Nicol. FIPA - Towards a standard for software agents. *BT Technology Journal*, 16:3:51–59, 1988.

[48] Andrea Omicini and Franco Zambonelli. Tuple centres for the coordination of internet agents. In *Proceedings of the 1999 ACM symposium on Applied Computing*, pages 183–190, San Antonio, Texas, United States, 1999. ACM Press.

[49] B.J. Overeinder, N.J.E. Wijngaards, M. van Steen, and F.M.T. Brazier. Multi-agent support for Internet-scale Grid management. In *Proceedings of the AISB 2002 Symposium on AI and Grid Computing*, pages 18–22, London, UK, 2002. ACM Press.

[50] D.T. Pham and D. Karaboga. *Intelligent Optimisation Techniques.* Springer, London, UK, 2000.

[51] S. Poslad and M. Calisti. Towards improved trust and security in FIPA agent platforms. In *Proceedings of the Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies*, pages 87–90, Barcelona, Spain, 2000. ACM Press.

[52] S.J. Poslad, P. Buckle, and R. Hadingham. The FIPA-OS agent platform: Open source for open standards. In *Proceedings of Fifth International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, pages 355–368, Manchester, UK, 2000. The Practical Application Company Ltd.

[53] O. F. Rana and Luc Moreau. Issues in building agent-based Computational Grids. In *Proceedings of the UK Multi-Agent Systems (UKMAS 2000) Workshop.* World Wide Web, "`http://citeseer.ist.psu.edu/523418.html`", December 2000.

[54] O.F. Rana, D. Bunford-Jones, D. W. Walker, M Addis, M. Surridge, and K. Hawick. Agent based resource discovery for dynamic clusters. In *Proceedings of the IEEE Conference on Cluster Computing*, pages 353–355, Chemnitz, Germany, November 2000. IEEE Computer Society Press.

[55] T. Sandholm and R. Crites. Multiagent reinforcement learning in the Iterated Prisoner's Dilemma. *Biosystems*, 37:147–166, 1995.

[56] J.R. Searle. *Speech acts: An essay in the philosophy of language.* Cambridge University Press, Cambridge, 1969.

[57] Sandip Sen and Mahendra Sekaran. Individual learning of coordination knowledge. *Journal of Experimental & Theoretical Artificial Intelligence*, 10(3):333–356, 1998.

[58] Y. Shoham and M. Tennenholtz. Co-learning and the evolution of social activity. Technical Report STAN-CS-TR-94-1511, Department. of Computer Science, Stanford University, 1994.

[59] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, 1980.

[60] Bjarne Stroustrup. What is Object-Oriented programming? *IEEE Software*, 5:10–20, 1988.

[61] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An introduction.* MIT Press, Cambridge, MA, USA, 1998.

[62] K. Sycara, K. Decker, , and D. Zeng. Intelligent agents in portfolio management. In N. Jennings and M. Woolridge, editors, *Agent Technology: Foundations, Applications, and Markets*, pages 267–283. Springer, 1998.

[63] Gerald Tesauro. Pricing in agent economies using neural networks and multi-agent Q-Learning. *Sequence Learning: Paradigms, Algorithms, and Applications (Lecture Notes in Computer Science)*, 1828:288–307, 2001.

[64] Gerald Tesauro and J. Kephart. Pricing in agent economies using multi-agent Q-learning. *Autonomous Agents and Multi-Agent Systems*, 5:289–304, 2002.

[65] K. Tumer and J. Lawson. Collectives for multiple resource job scheduling across heterogeneous servers. In *Proceedings of the Second International Joint Conference on Autonomous Agents Multiagent Systems, July 14-18, 2003, Melbourne, Australia*, pages 1142–1143. ACM Press, 2003.

[66] K. Tumer and D. Wolpert. Avoiding Braess' paradox through Collective Intelligence. Technical Report NASA-ARC-IC-99-124, NASA, 1999.

[67] Kagan Tumer, Adrian K. Agogino, and David H. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems, ACM Press*, pages 378–385, Bologna, Italy, 2002.

[68] Michael P. Wellman. Market-oriented programming: Some early lessons. In *S. Clearwater (editor), Market-Based Control: A Paradigm for Distributed Resource Allocation.* World Scientific, 1996.

[69] N. Wijngaards, M. van Steen, and F. Brazier. On MAS scalability. In Tom Wagner and Omer Rana, editors, *Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS and Scalable MAS.*, pages 121–126. AAAI, 2001.

[70] S.N. Willmott, J. Dale, B. Burg, C. Charlton, and P. O'Brien. Agentcities: A Worldwide Open Agent Network. *Agentlink News*, 8:13–15, November 2001.

[71] D. Wolpert, J. Sill, and K. Tumer. Reinforcement learning in distributed domains: Beyond team games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 819–824, Seattle, Washington, USA, 2001.

[72] D. Wolpert and K. Tumer. An introduction to Collective Intelligence. Technical Report NASA-ARC-IC-99-63, NASA, 1999.

[73] D. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.

[74] David H. Wolpert, Kagan Tumer, and Jeremy Frank. Using Collective Intelligence to route internet traffic. In *Advances in Neural Information Processing Systems-11 Proceedings (NIPS 1998)*, pages 952–958, Denver, 1998. The MIT Press.

[75] David H. Wolpert, Kevin R. Wheller, and Kagan Tumer. General principles of learning-based multi-agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 77–83, Seattle, WA, USA, 1999. ACM Press.

[76] M. Wooldridge. Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37, 1997.