LEAF: A FIPA Compliant Software Toolkit for Learning based MAS

Steven Lynden and Omer F. Rana
Department of Computer Science, University of Wales, Cardiff, UK.
{s.j.lynden,o.f.rana}@cs.cf.ac.uk

ABSTRACT
This paper introduces LEAF, a FIPA compliant software toolkit for developing learning based multiagent systems. The FIPA-OS agent development toolkit is extended to include support for learning agents using techniques such as reinforcement learning, Q-learning and neural networks. A coordination mechanism is also provided that facilitates self-organising/emergent behaviour, using an approach based on the utility assignment techniques of Collective Intelligence (COIN). A novel contribution of our work is that the notion of agent utility is extended to form two separate definitions of utility: the traditional functional utility of agents (defined in terms of how well the system is achieving its objectives), and a performance utility, which is based on performance engineering related issues.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Coherence and coordination Intelligent agents Multiagent systems

General Terms
Design, Experimentation

1. INTRODUCTION
The study of machine learning based multiagent systems (MAS) has revealed a promising technology for developing adaptable and self-organising systems with emergent behaviours. There is now a need to apply systems with these properties to practical, real-world problems, and bridge the gap between theoretical research and practical problem solving. The application of agent based methods to real-world problems brings with it the need for standards that can facilitate large scale interoperability. This issue is addressed by the Foundation for Intelligent Physical Agents (FIPA) [3] specifications, one implementation of which is FIPA-OS [5]. The aim of the work described in this paper is to provide an extension of FIPA-OS that customises it to enable the development of coordinated, machine learning based MAS.

Although the investigation of a wide range of applications is intended, a number of properties that should be part of the infrastructure of all these applications have been identified. In order to determine its behaviour, a learning agent requires some form of payoff function, which gives an indication of the benefit to the agent of the actions it has performed; in this work, the agent’s local utility function provides this, and the goal of each agent’s internal learning process is to maximise its local utility function. In the same way that a local utility function rates the behaviour of a single agent, a global utility function rates the performance of MAS. If a MAS is designed to be fully cooperative, with globally defined objectives, a coordination problem exists due to the fact that although agents strive to maximise their local utility, they may indeed be working at cross purposes with other agents, and/or be working to lower world utility. The Collective Intelligence (COIN) framework introduces a coordination mechanism where agents are assigned local utility functions with the essential property that an increase in local utility implies an increase in the global utility. Various schemes for assigning utility functions are described in [1][2].

We introduce LEAF (the Learning Agent based FIPA compliant agent toolkit), which provides support for the development of MAS utilising COIN based local utility function assignment techniques. We believe the COIN approach is widely applicable for building agent systems employing various mixtures of machine learning and rule based paradigms. The aim of our work is to investigate practical implementations of the use of utility function assignment in a variety of application domains. The importance of interoperability is recognised, and for this reason, LEAF utilises FIPA-OS, allowing LEAF agents to interact with other FIPA compliant agents. LEAF provides an infrastructure for the development of learning agents, and assigning utility functions to agents. The notion of utility is extended to include performance utility, which gives an indication of the performance of agents in terms of performance engineering related aspects, such as speed and CPU/memory usage etc.

2. UTILITY ASSIGNMENT
Researchers into MAS related fields have long realised the difficulties involved with methods that define agent behaviour using approaches based on centralised control, where all agents maintain communications with a single entity that controls agent behaviour based on the MAS state. Centralised methods can involve tremendous costs in communication and computation, and also provide the weakness of a single point of failure for the entire MAS.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAAMI'02, July 15-19, 2002, Bologna, Italy.
Copyright 2002 ACM 1-58113-480-0/02/0007 ...$5.00.
Research has tended towards completely decentralised methods and autonomous agents, where the coordination of agents is the challenging issue. The framework on which LEAF is based lies between decentralised and centralised control; a centralised entity is involved in utility function assignment, but no instructions are given to agents, agents are autonomous in the sense that their behaviour is governed by an internal learning process, but agents are required to learn to maximise the assigned functions. The centralised entity responsible for local utility assignment in LEAF is the environment service node (ESN).

Among the various utility functions assigned to agents in COIN, some require centralised communication, whereas others can be computed by information available locally to an agent [2]. It is therefore clear that any practical implementation of COIN techniques requires some means of dynamically providing information (that is not available locally) to agents concerning MAS state. Therefore, a key feature of the local utility functions that can be assigned to agents in LEAF is that the functions can be parameterised by data that is not normally available locally to the agent. In MAS, an agent’s behaviour can affect various aspects of the system that are not observable locally by the agent, but must be considered in the computation of local utility in order to engineer coordinated agents. In the LEAF system, the ESN entity is responsible for dynamically providing the parameters needed to compute local utility functions where parameters are not available locally to the agent. The ESN also maintains the global utility function. The information needed to compute local and global utility functions is obtained periodically by the ESN from the agents in the system. The functionality of the ESN is illustrated in figure 1. The parameters provided by the ESN enable utility functions to be parameterised by various group dynamics, and can allow an agent’s learning process to take into account the effect of its actions on factors that are not observable locally.

2.1 Performance Utility

In the LEAF system, the traditional notion of utility, relating to the agents high-level behaviour, is referred to as functional utility. LEAF supports an additional definition of utility, performance utility, which is a function of the agents performance in relation to speed, CPU/memory usage, and other performance engineering related aspects. Performance utility is intended to be of use to the designer of the MAS, in order to assess the effects of the decisions made during design/implementation. These decisions could include the platforms on which agents are hosted, whether behaviour is defined by Java algorithms or implemented in Prolog, the structure of behaviour algorithms, the complexity of rules, the data structures used in implementation, and various initial parameters passed to the agents at creation time.

Performance utility can be utilised online in adaptive management processes [3], to make decisions about MAS at runtime, and group agents with similar performance utilities. The relationship between functional and performance utility may also be investigated using the LEAF environment.

3. DEVELOPING MAS USING LEAF

The development of LEAF agents is tailored towards Java programmers with some knowledge of FIPA-OS. It is our intention that the features of the toolkit outlined here can be utilised in a wide range of applications. The software toolkit provides a convenient API for constructing learning based MAS, with access to various learning algorithms. Agents, and ESNs, are constructed by extending Java objects provided by LEAF, which provide the agent developer with a set of methods governing the utility functions assignment process.

LEAF provides a library of tasks objects, which can be utilised to implement various agent behaviours. Tasks can be used to build agent behaviour in a modular fashion out of reusable modules. As an illustrative example, a task could be used to define the behaviour required to engage in a negotiation process with other agents according to certain constraints, for example, the maximum price to pay for an item, or by what amount to increment offers when engaging in an auction. Another example of the use of tasks is to encapsulate learning processes, such as RL algorithms, or decision tree based learning algorithms. The task library will consist of tasks that are developed applying LEAF to various application domains, which is the aim of future research. Where possible tasks will be developed to be as domain independent as possible, resulting in reusable tasks that can be utilised as part of the LEAF toolkit.

4. CONCLUSION

The LEAF agent development toolkit has been described, which aims to provide support for the development of learning based MAS. We have outlined the extension of the notion of agent utility to form two separate utility functions, performance utility and functional utility. In future work, the relationship between performance and functional utility will be investigated, and MAS developed using LEAF will be applied to various domains.

5. REFERENCES