ABSTRACT
Multiagent systems (MAS) research is well established yet there is little development of agent technology in industry. It has been suggested that this is due, in part, to the unavailability of support tools for Agent Oriented Software Engineering (AOSE). This paper suggests requirements for Integrated Development Environments (IDEs) to support MAS construction. We suggest that an IDE can be built as its own MAS which allows it to be decoupled from any particular agent framework thereby allowing it to be platform independent.

Categories and Subject Descriptors
I.2.11 [Distributed Artificial Intelligence]: Multiagent systems.

General Terms
Experimentation, Standardization.

Keywords
Agent oriented software engineering, debugging, maintenance, development environments.

1. INTRODUCTION
MAS research is reaching a significant level of maturity but its industrial take-up is still low [7, 14]. This may be because there is a widely held view that agent-oriented engineering is radically different to other software engineering approaches or it may be because MAS based development imposes a greater level of complexity then other approaches to software construction [3]. The necessity for tools and techniques to ease this complexity has been recognised by various authors [6, 7, 11] but the results are not effective enough to provide a complete methodology [1] and implementing a multiagent system is “still a daunting task” [2].

In this paper we identify two different levels of abstraction: the agent-level (tools which aid the development of individual agents) and the MAS-level. The MAS-level focuses on properties exhibited by the MAS as a whole, it is concerned with the semantic content of agent-agent messages but represents the agents themselves as black box entities.

Although the choice of MAS platforms is currently rich and most have some level of tool support, some tools are typically missing or have received too little attention [4, 8, 13]. The first of these can be found in debugging MAS. Due to the reactive nature of agent based code, implementation and debugging individual agents can be more complex than with some traditional systems. However an even greater challenge [13] lies in debugging systems containing multiple agents where there is tendency for software tasks to be distributed across agents and the macro-level behaviour of the system may be hard to predict from the apparent behaviour of its agents. This makes good debugging tools essential [13] yet they have not received sufficient attention [4, 8, 13]. Where debugging has been addressed, it has been at the agent-level, rather than debugging the MAS as a whole [8].

Secondly the necessity for tools extends beyond development to deployment and post-implementation monitoring. Unfortunately these tasks are also inadequately supported [1, 3].

One reason why MAS-level activities (debugging, deployment, post implementation administration and maintenance) have lacked development may be that it has been easier to focus efforts towards the agent-level. The most often used strategy is to extend existing object-based methods to deal with MAS [10]. However, MAS have fundamental differences to object oriented systems and there is a lack of tools that focus on the differences.

As a result, current tools for agent based development are focused on the internals of agents and tend to be limited to the development phase. A recent survey [2] indicated that current IDEs for MAS tend only to provide the kinds of support associated with object-oriented IDEs, focusing on programming issues rather than the MAS-level and serving primarily to automate tedious coding tasks.

This paper examines the provision of software tools for MAS deployment, monitoring and debugging and considers minimal common requirements for a MAS-level IDE. We recognise the need (discussed by Bordini et al. [2]) to decouple agent tools from any specific agent framework. We consider the experience of deploying this MAS IDE as its own MAS, an approach which
achieves a greater level of decoupling between the IDE and any agent framework and also allows the IDE to be extended.

2. REQUIREMENTS OF A MAS IDE
The sections below outline various categories of MAS-level tools and their requirements. These have been derived while developing MAS applications and from evaluation of an experimental IDE constructed as part of this investigation. These tools are presented (as relevant) in the following sections.

2.1 Representing MAS Structure
A simple MAS is a collection of homogeneous agents which can be presented as black box components with messaging interfaces. Further inspection of any element would reveal the internals of an agent. With larger MAS, there is sometimes a need to place entire sections of MAS in a black box. These groups of agents (termed organisations [11] or holons) can often be treated as single entities and presented as single black-box components. However, for developers examining agent-agent activity occurring within a holon it is necessary to view intra-holon agents and messages so multiple views must be provided.

The experimental IDE presents a MAS as a tree structure that can be expanded and collapsed as necessary, this allows users to view MAS information at different level of structural abstraction. Views are dynamically updated as agents join/leave a MAS and as messages are routed between agents.

2.2 Capturing Agent Messages
MAS activity is primarily driven by agent interactions and MAS behaviour is most clearly exhibited by the messages transmitted between agents. Interaction monitoring has been identified as important by the MAS community and is used as criteria for comparing toolkits [5]. With most systems there will be a large number of messages, some which may cause the macro-level behaviour of the system to change. Tools must capture these messages and present them in a coherent manner. During any given phase of debugging, developers will only be interested in interactions involving certain subgroups of agents – messages relating to specific scenarios of activity which can be verified against design scenarios [13].

To remove the burden of message filtering from the developer, a message capture tool must make it possible to identify subsets of messages by providing some filtering mechanisms to allow users to specify constraints on which messages are displayed. Currently some tools provide facilities like conversation tracking [5] which allow messages to be viewed based on time frames and filter messages according to the agents which send/receive them.

During debugging there may also be a need to manually drive agent interactions so tools should allow users to hand-code messages and view the effects. Some platforms provide a messaging agent for this purpose [5].

2.3 Logging and Playback
Visualisation of MAS behaviour has been recognised as being a "notoriously difficult task" [12] without tool support. Tools should allow developers to view and monitor the whole or selected sections of a running MAS. This involves viewing the structure, composition, messaging activity, errors and any other behaviour in conjunction.

Observing MAS in realtime is difficult because the activity occurs too fast for human analysis. It is important that this behaviour can be captured for offline analysis and "played back", allowing users to step through execution. The experimental IDE allows activity to be played back at slow speeds or message-by-message.

2.4 Managing Deployment
MAS consist of independently executing entities and so do not have a single starting point like other types of applications [3]. This makes launching a MAS more complex yet is only addressed by few platforms [3]. A MAS toolkit should allow the specification of agent instances and their dependencies (constraints on the order in which they are started up) and be able to start up an entire MAS or holon from a specification.

Once launched, administrators of MAS will need to monitor the MAS for conditions that need correction [3]. Messages from underlying agent frameworks or operating systems may indicate some level of failure or information about changes to the MAS environment. They may be caused by specific MAS behaviour or indicate changes which will have knock-on effects, these should be captured and reported as part of the monitoring process.

Some conditions may indicate failures that require agent repair or redundancy. Conditions like network loads may need correction by reconfiguring MAS structure. This means that a MAS IDE should allow administrators to relocate agents and change the MAS structure.

3. DESIGN PHILOSOPHY
Bordini et al. identify a number of priorities to enable wider development of agent based software [2]. They highlight the need to integrate MAS development environments with existing IDEs, implying that a key challenge is to develop a MAS IDE that can integrate across different MAS frameworks and different existing (object-based) IDEs but acknowledge that this is difficult due to "unavoidably tight coupling of agent IDEs and agent platforms" [ibid, p.40].

We have attempted to reduce the coupling of the agent IDE from the agent framework while still allowing it to integrate with existing (object based) IDEs.

3.1 Decoupling from Framework
Reduced coupling between the IDE and the agent framework has been achieved by constructing the IDE as a MAS in its own right, we use the term organisation to describe this MAS. Given the ideology that agents written in different languages, using different paradigms should all be able to communicate as long as they use some agreed protocol, an IDE deployed in the form of a MAS is inherently less tightly coupled than one relying on method calls. Any framework which has the capability to keep the organisation informed about its structure, messaging, etc. can use the organisation and its tools. The requirement is only that the organisation must be informed about agents joining / leaving the system and be copied into agent-agent messages.

A further benefit in deploying the IDE in the form of its own MAS is that it allows new tools to be added as additional agents.

To date we have experimented with this IDE in two ways (i) with a virtual MAS (a shell which allows a developer to message the IDE directly) and (ii) using the Boris framework (www.agent-domain.org) with agents written in Java and Lisp. Boris is an open
3.2 Decoupling of Agent-level Tools

Different application areas require different aspects of agency, some may have specific needs for brokering, others may require agents to be mobile. The agent-level needs to recognise different approaches to agent implementation provided by the various frameworks. Broadly we consider two different categories (i) frameworks where agents are based on extensions of existing languages (Java for example) and (ii) those based on specialised agent definition languages.

In the first case neither MAS-level concepts nor many aspects of agency are explicitly visible in the program code. Agent-level activities like debugging and editing the code that defines an agent are similar to those carried out with code not involving agents and existing IDEs, editors, etc. provided for the base language are suitable for agent-level tasks.

In the second case, where agents are defined in a specialised language, the framework has an obligation to its developer community to provide appropriate agent-level tools if it intends widespread use. Currently the tendency is either to provide a plug-in to an existing IDE (eg: The Living Systems Developer, a commercial IDE deployed as an Eclipse (www.eclipse.org plug-in) or to provide a specialised IDE. These specialised agent-level "MAS IDEs" typically follow the model set out by the object-oriented IDEs [2].

Consequently, whether framework specific or not, agent-level tools can be provided in the form of a conventional IDE. The only requirement for a framework-independent MAS-toolkit is an ability to link to these object IDEs. This integration has been achieved by specifying links to those IDEs so individual agents can be opened for debugging, editing, recompilation using an existing IDE.

4. SUMMARY

Evaluation confirmed the importance of tools and supported the view that the two levels of abstraction (the agent layer and the MAS layer) involve different types of activity. We are satisfied that the MAS IDE can integrate agent-level tools like object IDEs. This is simply achieved by allowing agents to be inspected by different IDEs, according to their type (base language, etc).

The MAS IDE is decoupled from any framework used to specify/deploy agents. This is achieved by implementing the IDE as a MAS in its own right. All information provided to it is in the form of messages to its agents and, these messages are transmitted under an agreed protocol. As a result, the IDE is not dependent on any framework-specific facilities. To demonstrate this empirically we have linked the IDE to a user controllable shell that masquerades as a running MAS sending messages to the IDE about the structure and message passing in this virtual MAS. Since this controllable shell/virtual MAS is running independent of any framework the IDE cannot use framework-specific features to report on it.

Adding new MAS-level tools is also readily achieved. Since the IDE is deployed as a MAS new tools can be provided in the form of additional agents. For example, a framework which supports mobility may provide the IDE with agents which report on and/or manipulate the mobile characteristics of its agents.

5. REFERENCES