USING META-AGENTS TO BUILD MAS MIDDLEWARE

S. C. Lynch
School of Computing, University of Teesside, Tees Valley, TS1 3BA, Middlesbrough, U.K.

Keywords: Meta-agents, Platforms, Middleware, Distributed systems.

Abstract: Various multiagent platforms exist, each providing a range of individual capabilities but typically their implementations lack the flexibility to allow developers to adapt them to the differing needs of individual applications. This paper investigates the design of a kernel for MAS middleware based on primitive meta-agents. We specify these meta-agents and examine how they can be used to realise the capabilities required by multiagent platforms. We examine how changes in the organisation of meta-agents produce MAS platforms with differing behaviours. We evaluate the meta-agent approach by experimentation, demonstrating how modifications in meta-agent behaviour can provide different strategies for agent communication, scoping rules and connectivity with other tools.

1 INTRODUCTION

Many MAS platforms are currently available to developers, these platforms vary in their range of capabilities, the facilities they offer and their notions of agency. Some platforms concentrate on support for BDI agents (Mascardi et al, 2005), others on mobility (Cabri et al, 2006; Suna & Fallah-Seghrouchni, 2005), others intend to be more general purpose (Bellifemine et al, 2008). Some platforms provide programmer support in the form of debuggers and developer tools, others do not (Bordini et al, 2006). With the exception of MadKit (Gutknecht & Ferber, 2000), none of the platforms we surveyed allowed system developers to modify the underlying behaviour of the platform or adapt the functions of its middleware. This inflexibility has been noted by other authors (eg: Fonseca, 2006). For MAS developers this means that they can build systems on top of platforms but in doing so must work within the constraints imposed by the platform they are using.

Some authors have suggested building MAS platforms using modifiable components. Kind and Padget suggested meta-agents (Kind & Padget, 1999) and others have discussed the use of meta-actors for middleware (Sen & Agha, 2002). This work may have influenced the design of some MAS platforms (Mulet et al, 2006) but meta-agent design patterns are not an area for significant investigation in the agent research community; there has been little discussion of the way meta-agents are constructed and the ways they may be reconfigured to provide multiagent platforms and middleware with differing capabilities. Not only that but MAS platforms do not present their users with a meta-agent layer but instead provide a rigid structure for their middleware which imposes various limitations and may predetermine the system architecture, the nature of communication channels and/or the scale of platform's middleware.

MadKit, built from a kernel of low-level agents, was presented as an exception to this norm, providing an extensible platform which could be customised by its users (Gutknecht & Ferber, 2000). Although it was not widely adopted as a base-layer or middleware solution the wide-ranging potential of MAS suggests that the use of a meta-agent micro-kernel is worth further study.

Our main aim is to use configurations of meta-agents to support distributed systems of application-level agents and to consider how changing the organisation of these meta-agents produces MAS platforms with differing behaviours. To this end we investigate using meta-agents to provide a generic but extensible middleware which may be reconfigured to suit the requirements of individual applications or their deployment needs.

In this paper we build a small set of primitive meta-agents and show how they can be organised into different configurations. These configurations, which are themselves distributed systems of meta-agents, form platforms to support application-level

Lynch S..
USING META-AGENTS TO BUILD MAS MIDDLEWARE.
DOI: 10.5220/0003149603850388
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 385-388
ISBN: 978-989-8425-41-6

agents. The differences in configurations produce adaptations in the characteristics of the underlying systems which change the behaviour of applications built on them.

We show how meta-agents can be constructed from a small set of language-level primitives and investigate how they can be used to support strategies for communication between application agents in a distributed environment as well as the behaviour of those agents. The resulting meta-agent configurations provide adaptable middleware solutions for application MAS. The design lends itself for implementation in various languages and permits easy connectivity with other systems, we have successfully used the design to build platforms in Java, Lisp and C# and have developed links to Galaxy, .NET and JADE.

2 A META-AGENT SUBSYSTEM

Conceptually each application agent rests on a small set of interconnected and interacting meta-agents which provide the application agent with its abilities to encapsulate behaviour and to communicate. In a distributed environment agents may reside on different physical or virtual machines. Our aim is for meta-agents to be light-weight and offer maximum opportunity for reuse and reconfiguration so we separate the functions of behaviour and communication into different meta-agent classes. In the following discussion a *Portal* is defined as a specialisation of meta-agent which provides communication between agents and a *socket-agent* is another specialisation of meta-agent which routes messages between machines through a socket.

For the purposes of our discussion here we consider meta-agents to be small autonomous software entities which (i) operate in their own process thread and can thereby be concurrently active with other meta-agents (ii) have an inward communication stream capable of queuing incoming messages (iii) can encapsulate behaviour including that which allows them to send massages to other meta-agents and manage deliberation cycles.

The design of meta-agents is such that incoming messages are queued until their thread is idle then the least recent message is passed to the meta-agent message-handler and its thread is rescheduled. The message handler is used to encapsulate behaviour which occurs in response to messages. Meta-agents are specified as objects so the nature of this behaviour is not restricted and may involve modification of instance or environmental data.

Other details concerning the structure of agents, the functions they may perform and the nature of their inter-agent communication is left unrestricted.

A high-level (abstract) meta-agent can be specified by extending the concept of a *blocking queue*. A blocking is a queue which puts the current thread into a waiting state if it is asked for data when it is empty. Assuming the class *blocking-queue* exists with the capability to *enqueue* and *dequeue* data, meta-agents can be specified as shown below. Instances of meta-agent have a name and a link to a *portal*.

```
class meta-agent
  extension-of: blocking-queue
  variables: name, portal

  constructor method( args... )
  | super.constructor( args... )
  | thread{ loop-forever
  |    msg-handler( dequeue() )}

  method msg-handler( msg )
  |  ;; reactive behaviour to
messages
```

The construction/instantiation of a meta-agent creates a new process thread for the agent which continually extracts messages from the agent's queue and passes them to its message handler, the reactive behaviour for responding to messages is specified in the message handler. The use of a single thread for an agent ensures that messages it receives are processed sequentially. A minor modification to meta-agents allows them to process their messages concurrently. To achieve this each *msg-handler* runs its own thread.

Portals are meta-agents which manage communication for other meta-agents so when one agent sends a message to another it is sent via the sending agent's portal. The message contains the name of the agent which is to be its final recipient. The agent's send-message method handles this as follows:

```
meta-agent.send-message(recipient, msg)
| portal.enqueue(
|    wrap(name, recipient, msg)
```

For the sake of simplicity we assume here that all meta-agents have locally unique names and portals map these names onto agent instances in order to forward messages onto the appropriate queue for any agent (alternative arrangements of meta-agents can be used to produce different white/yellow pages systems but these are not discussed here). Note also that the function wrap is used to pack the recipient name and the message into a single, faceted form.

Using portals allows meta-agents to communicate with each other by names but removes the need for them to locate each other since this becomes the portal's responsibility. The simplest specification for portal is as follows:

```
class portal extension-of: meta-agent
 variables: routing-table

 method msg-handler(msg)
 | routing-table.get(
 |     recipient-of(msg)).enqueue(msg)

method add-agent( name, agent )
 | routing-table.set( name, agent )
```

A portal's routing table is a hashing structure which maps agent names onto their queues (or their agent instances).

The specification of portal allows multiple agents to share the same portal but since it also implicitly allows portals to connect to other portals (by virtue of them being defined as agents themselves) it also allows each agent to have its own dedicated portal This provides the flexibility to develop both peer-to-peer architectures for agent communication or hub-based architectures (or hybrid approaches) – see Figure 1 where A1 and A2 are standard meta-agents and P1 and P2 are portals depicted with their routing tables, arrows indicate movement of message data between components.

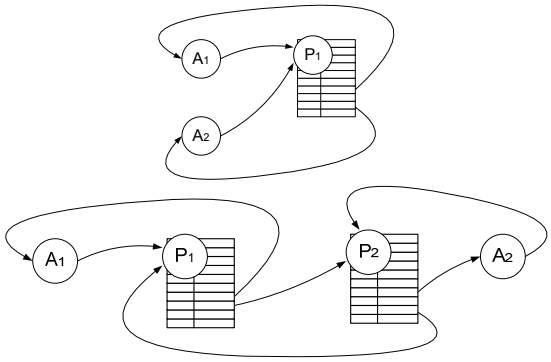

Figure 1: Agents sharing a single portal (top) and with independent portals (below).

A third genre of meta-agent is responsible for handling communication between execution spaces (physical or virtual machines). In this discussion we use sockets for this and name the new meta-agent *socket-agent*. The socket agent has input and output streams, any data sent to it via its input stream will be forwarded to its portal and its message handler writes all messages to its output stream.

Since socket agents may be accessed by a portal's routing table (like any other kind of meta-agent) their specification now allows portals and the agents attached to them to communicate between execution spaces and between machines on a network (see Figure 2 where S1 and S2 are socket agents and the heavy line shows socket based communication).

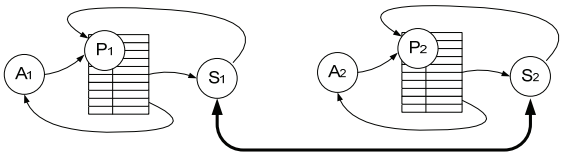

Figure 2: Use of socket agents.

3 EVALUATION

We have used the meta-agents specified in this paper to rebuild pre-existing middleware and MAS platforms used in our institution, successfully developing meta-agent subsystems in Java, Lisp and C# which allow distributed agents specified in different languages to freely interact. These subsystems have been further developed to link to the Galaxy Communicator architecture .NET services and JADE and have been successfully used as stable platforms for larger scale MAS projects involving multimodal dialog and other areas of research.

Here we present various ways that meta-agent subsystems can be extended and/or adjusted to change their characteristics.

First: extending the meta-agent subsystem to make it easier for users to specify application level agents primarily requires additional classes for application-agents and for portals to include some utility methods to aid usability.

Second: modification to meta-agents allows us to change the underlying virtual network architecture for agent messaging. This is of interest since the requirements for MAS deployed over long distance networks or on mobile devices are different to those deployed on a small local cluster of machines. This approach allows us to construct (i) peer-to-peer architectures, this is achieved by enforcing the condition that each agent has its own portal which is not shared directly with other agents (ii) a hub architecture where behaviour and communication are placed in different software components, with agents managing behaviour and (as above) portals managing communication (iii) tree and network topologies which can be formed by connecting portals in different configurations.

Third: with tree topologies it is possible to introduce scoping rules for agents. Agents declare

their scope at registration, this scope can be described as "global" or can name a portal. Global agents are visible to all other agents (their details are included in the routing-tables of all portals). This behaviour can be achieved with minor modifications to the agent registration system.

When the scope of an agent is declared with the name of a portal, the agent is only visible to other agents in the same tree/sub-MAS, the details of the non-global agent are included only in the routing-tables of portals in the same sub-MAS so only other agents in this sub-MAS may send messages to the non-global agent.

Fourth: despite the need for run-time analysis of MAS, problems exist with trying to collect the necessary run-time information. A possible solution is to encourage/insist that programmers add code to their agents to capture their run-time status at specified points in execution and relay it to some central monitoring system. However developers are unlikely to comply and would need to operate according to a set of standards which would impose additional burdens on development.

A better solution may be obtained by using meta-agents. Information about MAS structure can be obtained by examining the details of agent registration and messages exchanged between application agents provide details/traces of system activity. Since portals route various meta-data, including that describing registration and user-agent messaging, portals can be readily modified to forward that meta-data to a monitoring system without disrupting any other system activity.

In practice we implement the monitoring system as its own MAS and modify the message receiver of portals so the monitor is copied in to relevant information.

4 CONCLUSIONS

This paper has highlighted a limitation with the agent platforms and middleware which are currently available – they are not designed to allow system developers to modify their behaviour so cannot be tailored to suit the needs of developers.

Influenced by related work on meta-agents and actors we have specified a small set of meta-agents, light-weight components which lend themselves to modification and may readily be configured into different patterns.

We have used patterns of interacting meta-agents to form distributed subsystems which function as MAS platforms and middleware. Different configurations of these meta-agent patterns can be made to exhibit different properties and influence the characteristics of the resulting platforms they produce thereby providing adaptable frameworks for a variety of MAS applications.

REFERENCES

Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G. 2008. JADE: A software framework for developing multi-agent applications. Lessons learned. Inf. Softw. Technol. 50, 1-2 (Jan. 2008), 10-21.

Bordini, R., Braubach, L., Dastani, M., Seghrouchni, A.E.F., Gomez-Sanz, J.J., Leite, J., O'Hare, G., Pokahr, A., and Ricci, A. “A Survey of Programming Languages and Platforms for Multi-Agent Systems”. Informatica, 2006, 30(1), 33-44.

Cabri, G., Ferrari, L., Leonardi, L. and Quitadamo, R. 2006. Strong Agent Mobility for Aglets Based on the IBM JikesRVM. ACM symposium on Applied computing (Dijon, France) ACM.

Fonseca, S. P. 2006. Engineering degrees of agency. Proceedings of Software engineering for large-scale multi-agent systems, SELMAS (Shanghai, China, 2006) ACM Press, New York, NY.

Giret, A. and Botti, V., “Holons and Agents”, Journal of Intelligent Manufacturing 2004, Vol. 15 No.5 pp. 645-659. Springer Netherlands.

Gutknecht, O. and Ferber, J. The Mad Kit Agent Platform Architecture. In Infrastructure for Agents, Multi-agent Systems, and Scalable Multi-agent Systems, 3-7, (2000).

Kind, A. and Padget, J. Towards Meta-Agent Protocols, LNCS 1624, 30-42, 1999.

Mascardi, V., Demergasso, D. and Ancona, D. 2005. Languages for Programming BDI-style Agents: an Overview. In Proceedings of WOA (Camerino, Italy, 2005). Pitagora Editrice Bologna.

Massonet, P., Deville, Y., and Neve, C. 2002. From AOSE Methodology to Agent Implementation. In Proceedings of AAMAS (Bologna, Italy, 2002) ACM Press, New York, NY.

Mulet, L, Such, J M. and Alberola, J M., Performance evaluation of open-source multiagent platforms. AAMAS (Japan) ACM Press, New York, NY, 2006.

Sen, K., Agha, G., "Thin Middleware for Ubiquitous Computing," Process Coordination and Ubiquitous Computing, CRC Press, 2002.

Suna, A. and Fallah-Seghrouchni, A., E. 2005. A Mobile Agents Platform: Architecture, Mobility and Security Elements. LNCS, 3346 / 2005, 126-146.