MACSIM: A SIMULINK ENABLED ENVIRONMENT FOR MULTI-AGENT SYSTEM SIMULATION [1]

Peter Mendham, Tim Clarke

Intelligent Systems Group, University of York, Heslington, York, YO10 5DD, UK
http://www.elec.york.ac.uk/intsys/

Abstract: Agent-based approaches to software and algorithm development have received a great deal of research attention in recent years and are becoming widely utilised in the construction of complex systems. This paper presents MACSim a novel simulation environment which allows a multi-agent system to be embedded into the industry standard system, Simulink. The architecture of MACSim is discussed and a case study is presented where a number of MACSim agents are used to control the behaviour of a Boeing 747 in simulation. *Copyright ©2005 IFAC*

Keywords: Artificial intelligence; simulators; fault tolerance; complex systems.

1. INTRODUCTION

Agent-based approaches to software and algorithm development have received a great deal of research attention in recent years and are becoming widely utilised in the construction of complex systems (Aström *et al.*, 2001; Jennings, 2000; Parunak, 2000). Agents use their own localised knowledge for decision-making, supplementing this with information gained by communication with other agents. Remaining independent of any kind of centralised control whilst taking a local view of decisions gives rise to a tendency for robust behaviour (Parunak, 1997). The distributed nature of such an approach also provides a degree of tolerance to faults – both those originating in the software/hardware system itself and in the wider environment (Kaminka and Tambe, 2000; Kaminka and Tambe, 1998). It is for these reasons that a multi-agent system has been considered a suitable model on which to base an intelligent control system for complex systems requiring a large degree of autonomy (Voos, 1999*a*; Voos, 1999*b*; Voos, 2000; Mendham and Clarke, 2003*a*; Mendham and Clarke, 2003*b*).

A concise definition of a multi-agent systems, or indeed an agent, is much debated. However, it is generally agreed that an agent has a several defining characteristics (Wooldridge, 2002; Jennings, 2000):

- *Autonomy* Every agent has at least one independent thread of execution.
- *Situatedness* Agents are situated in an environment, and are capable of interacting with that environment.
- *Communication* Agents may communicate with each other through the passing of messages.
- *Intelligence* Where agents are defined as intelligent, this usually implies that each agent has its own knowledge, goals and the means to act towards those goals, which are all local to the agent.

[1] Corresponding author P. Mendham. Email pdm104@ohm.york.ac.uk, Tel. +44 (0) 1904 432823, Fax +44 (0) 1904 432335.

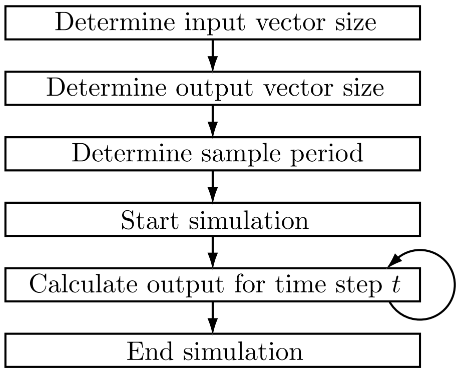
A multi-agent system (MAS) is therefore a multi-threaded environment with the facilities to pass information between threads in the form of messages.

There is a great deal of potential in using a MAS to interface to, or represent a dynamic system (for example, for modelling the dynamics of an emergent system). Inspired by this requirement, this paper details a novel method of simulating a MAS in a manner compatible with the industry standard environment Simulink (Mathworks Inc., 1999).

The paper begins by detailing the requirements for a MAS interfaced with Simulink. Attention is focused on discrete-time systems with fixed numbers of inputs/outputs. In response to these requirements a multi-agent environment, MACSim(for Multi-Agent Control Simulation), is detailed. This is accompanied by a discussion of the facilities MACSim has to offer designers and developers. To illustrate the potential of MACSim we present a case study, applying a MAS to the task of controlling a non-linear model of the Boeing 747 aircraft. The paper concludes with a discussion of the future potential of the MACSim environment.

2. MULTI-AGENT SYSTEMS IN SIMULATION

The Simulink S-Function Application Program Interface (API) provides a framework for creating custom Simulink blocks written in a number of high level languages including C/C++, Fortran and Matlab's native language. This paper is concerned with the specification of a framework which will support the development of simulations of a practical MAS. There should be the potential to interface the MAS simulation to any other Simulink block. As a practical MAS would inevitably be a sampled system, a fixed-rate, discrete time simulation cycle is chosen for simulation. The basic cycle is shown in Figure 1.



Determine input vector size
Determine output vector size
Determine sample period
Start simulation
Calculate output for time step t
End simulation

Fig. 1. Basic Simulink simulation cycle

Simulink has the capability to carry out integration on behalf of any custom S-function. This requires the S-function to posses an internal state, and to calculate state derivatives. However, such a structure may not be applicable to a MAS in all cases.

Importantly, a MAS in simulation must have the ability to continue executing *in parallel* with the simulation cycle shown in Figure 1. The MAS environment should present capabilities allowing agents to synchronise with the cycle where required.

Finally, the MAS environment must have the potential for 'mathematics heavy' operations, including matrix manipulation. The way in which these facilities are provided will be largely dependent on the choice of implementation language.

To summarise, the MAS simulation framework should have the following characteristics:

- A fixed-rate, discrete time simulation with no explicit internal state.
- True multi-threading in parallel with Simulink's simulation cycle.
- The capability to synchronise with Simulink operations.
- Mathematical facilities.

Simple experiments with multi-threading inside S-functions indicated that there was the potential for instability in Simulink during operations involving multiple threads. The causes of these issues were difficult to isolate. However, instabilities became more apparent if a thread of execution was allowed to continue in parallel with Simulink's own simulation cycle. If threading was removed the system became stable again. This advocates the requirement for an alternative approach to providing a multi-threaded environment to ensure stability.

3. REALISING MACSIM

The MACSim environment is written as a framework of C++ classes for use by MAS simulation developers. An object-oriented language is more suitable for the development of agents. Combining this with the need for a high execution speed, C++ is an obvious choice. To ensure stability, MACSim was given a client-server architecture, separating the multi-agent system from Simulink as shown in Figure 2.

A lightweight client S-function runs in the Simulink environment. This communicates with the server MACSim environment through Windows named pipes. Two pipes are used, one for passing configuration information and a second for passing simulation information. This allows these two pro-

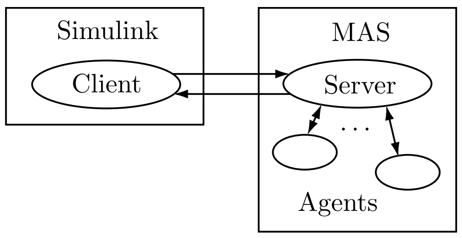


Fig. 2. MACSim client-server architecture

cesses to be run asynchronously. All communication across these pipes is conducted in a fixed-size message format. Queries are sent by the client and responded to by the server. There are three types of message used on the configuration pipe:

- number of inputs query;
- number of outputs query;
- sample period query.

Messages on the simulation pipe all have the same format. A simple packet is used, comprising:

- a header containing size and time step information;
- space for the current input vector, which is filled in by the client before the query message is sent;
- space for the resulting output vector, filled in by the server in response.

Once simulation packets arrive at the MACSim server, they are passed on to the agent environment.

MACSim provides a class encompassing the basic behaviour of an agent. This allows multiple agents to exist within the agent environment. The environment provides essential agent facilities such as coordination and messaging. Upon receipt of simulation information the agent environment posts messages to each of the running agents, requesting them to carry out any operations necessary to prepare outputs for the specified time step. If an agent has nothing to do, it immediately returns a message to the environment indicating that its work for the current time step is complete. Other agents may query the environment for the values of the current inputs, compute outputs and pass these back to the environment. Only then will these agents inform the environment that their work is complete. Once the environment has time step completion messages from all agents, the output values are passed to the pipe server for return to Simulink. The messaging therefore provides synchronisation with Simulink's simulation cycle. It is important to note that, despite the synchronisation, agents may continue to execute after sending the time step completion message.

The agent environment is therefore responsible for the following tasks:

- Acting as a central register of current agents, allowing the dynamic 'birth' and 'death' of agents.
- Coordinating synchronisation with the MACSim server and therefore with Simulink.
- Responding to queries for the current input and time step information.
- Storing the current set of outputs and providing a mechanism for agents to alter these.
- Providing a facility for broadcasting messages across the agent population.

In turn, there is a minimum set of actions that an agent must be capable of performing in order to be compatible with the MACSim environment:

- All agents must register their existence with the environment and inform the environment before they cease to exist, of this intention.
- All agents must be capable of sending the time step completion message in response to a time step update message from the environment. Without this response the environment will wait indefinitely.
- If an agent is presented with a query message that it does not understand, it must respond to indicate this. Again, without this response the environment may wait indefinitely when attempting a broadcast query.

MACSim provides a set of agent classes which implement this fundamental behaviour. This may be extended for further functionality.

Mathematical operations are catered for through the provision of a set of matrix and vector classes. These classes use C++ operator overloading to encourage the use of a coding style which looks as similar as possible to the mathematical operations it produces. Matrix classes also provide fundamental operations such as inversion and exponential.

The facilities offered by MACSim were originally designed with multi-agent control in mind. However, the system could easily be used for many other applications including studies oriented towards artificial life.

4. CASE STUDY: MULTI-AGENT FLIGHT CONTROL

The Intelligent Systems Group at the University of York is actively researching multi-agent control. As part of this research, a test simulation was required which provided control of a Boeing 747 aircraft. A basic non-linear model of the Boeing 747 was created as a Simulink block and MACSim was used as the execution environment for the multi-agent control (Mendham *et al.*, 2004). Figure 3 shows the model and multi-agent controller running in Simulink.

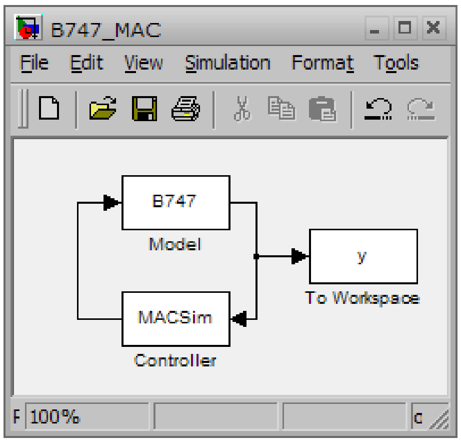


Fig. 3. Boeing 747 Model and MACSim Controller in Simulink

The granularity of the data available supported the creation of five agents: four agents controlling actuators (active agents), and one passive agent providing information about lift and drag of the major lifting surfaces, plus a gravity model. The actuators under control are:

- inboard ailerons;
- a simple engine model;
- inboard and outboard elevators, locked together;
- upper and lower rudders, locked together.

Each active agent has a partial non-linear model describing the effects of the actuator it represents on the vehicle as a whole. The passive agent describes the effect of states such as attitude and forward velocity on the vehicle. Periodically, agents share locally linearised versions of this information to reach consensus on a set of decoupled vehicle models. The number of models produced, and the agents involved in each of these models, varies dynamically depending on flight conditions. Using these models, agents control the vehicle optimally according to a predefined flight path. The system response to a simple unit step in altitude is shown in Figure 4.

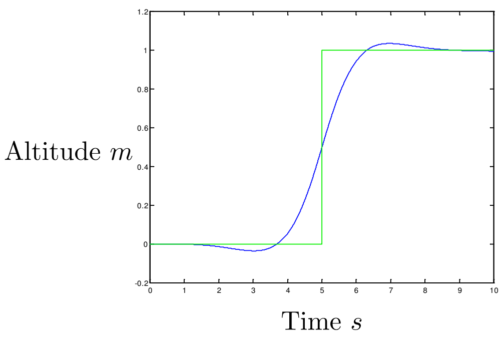


Fig. 4. Response to a demanded step change in altitude

Running under the MACSim framework, the multi-agent controller uses two threads for each agent plus another two for the environment. It is possible to allow a linearisation process to occur concurrently with Simulink, synchronising with the simulation cycle where necessary. Broadcast messaging is used to identify groups of agents involved in decoupled models known as working parties (WPs). Messaging across ‘mailing lists’ is then used inside WPs to carry out control tasks. The mathematical facilities offered by MACSim are crucial in control computation.

5. RELATIONSHIP WITH RELATED ENVIRONMENTS

There is a large number of existing agent development environments, a few of which are suitable for studying the dynamics of agent interaction. We will now focus on three well established environments largely used for artificial life research: Swarm (Minar *et al.*, 1996), Breve (Klein, 2002) and StarLogo (Resnick, 1994). Although very different, each of these environments is concerned with visualising the dynamics of a MAS. In Swarm, the environment is treated by the system as another agent and any data must be collected by purpose-built observer agents. Breve provides an environment that includes a physics engine allowing physical modelling and collision detection. It also supports 3D visualisation. Whereas Swarm is programmed in Objective-C or Java and Breve in an interpreted language known as ‘steve’, StarLogo is based on the simple Logo language with extensions to support a very large number of ‘turtles’. Originally intended as a tool to aid the understanding of complex and massively parallel systems, StarLogo has proved useful as a simple but powerful tool to visualise large systems of fairly simple agents. StarLogo does not provide facilities for the extraction of dynamic data.

MACSim does not provide facilities specifically aimed at massive parallelism, though there is no reason that it could not be used in this manner. The key feature, when compared with the three environments mentioned above, is the ability to extract data from the MAS in ‘simulation time’, process it through some other dynamic system, and return it back to the MAS. This is not possible in any of the three environments mentioned. With careful programming it might be possible in Swarm, but agent synchronisation would be an issue.

MACSim provides no visualisation, relying on the tools available in Simulink and Matlab. This places the emphasis on the developer to ensure that suitable data is passed out of the MACSim block to create meaningful visualisations.

6. SUMMARY AND CONCLUSIONS

Multi-agent systems have shown great potential in a number of fields. Investigating the dynamics of agent interaction is an important field of study. The multi-agent control simulation environment, MACSim, allows a MAS to be integrated into the industry standard Simulink environment either to study the dynamics of the MAS or to use the MAS dynamics to control other elements of the Simulink model. MACSim provides a framework for the development of agents in C++ and acts as an enabling platform for applying MAS in practical simulations. Difficulties with Simulink have been overcome using a client-server architecture and providing synchronisation mechanisms to ensure that MACSim agents take their proper place in Simulink's simulation cycle.

Recent work on MACSim has been driven by the requirements of a flight control demonstration, presented briefly in this paper as a case study. Future work will increase the scope of MACSim by developing the framework to provide tools for the investigation of reactive agent system dynamics and emergence. A formalised framework for control agent development will also be developed once a clear picture of the fundamental principles of agents for control emerges.

MACSim will be made freely available from the Intelligent Systems Group at the University of York's web site from March 2005. Refer to `http://www.elec.york.ac.uk/intsys/`.

REFERENCES

Aström, Karl, Pedro Albertos, Morgens Blankes, Alberto Isidori, Walter Schaufelberger and Ricardo Sanz (editors) (2001). *Control of Complex Systems.* Springer-Verlag.

Jennings, Nicholas R. (2000). On agent-based software engineering. *Artificial Intelligence* **117**(2), 227–926.

Kaminka, Gal A. and Milind Tambe (1998). What is wrong with us? improving robustness through social diagnosis. In: *AAAI'98 Proceedings, Madison, WI, USA.*

Kaminka, Gal A. and Milind Tambe (2000). Robust agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research (JAIR)* **12**, 105–147.

Klein, Jon (2002). Breve: a 3d simulation environment for the simulation of decentralized systems and artificial life. In: *Proceedings of Artificial Life VIII: the 8th International Conference on the Simulation and Synthesis of Living Systems, Sydney, NSW, Australia.* MIT Press.

Mathworks Inc. (1999). *Simulink Version 3.* Mathworks Inc.. Natick, MA.

Mendham, Peter and Tim Clarke (2003*a*). Dependable intelligent control through the use of multiple intelligent agents. In: *Proceedings of the 16th International Conference on Systems Engineering, ICSE2003, Coventry, UK* (Keith J. Burnham and Olivier C. L. Haas, Eds.). Vol. 1. pp. 478–483.

Mendham, Peter and Tim Clarke (2003*b*). Growing dependablility using a multi-agent approach to fault tolerance. In: *Proceedings of the 54th Congress of the International Astronautical Federation, IAF-03-U.2.a.04, Bremen, Germany.*

Mendham, Peter, Andrew Pomfret and Tim Clarke (2004). Dependable dynamic control using distributed intelligent agents. In: *Proceedings of the 55th Congress of the International Astronautical Federation, IAF-04-A.4.04, Vancouver, Canada.*

Minar, Nelson, Roger Burkhart, Chris Langton and Manor Askenazi (1996). The Swarm simulation system: A toolkit for building multiagent simulations. Technical Report 96-06-042 (Working Paper). Santa Fe Institute. Santa Fe, NM, USA.

Parunak, H. Van Dyke (1997). 'Go to the ant': Engineering principles from natural multiagent systems. *Annals of Operations Research* **75**, 69–101.

Parunak, H. Van Dyke (2000). A practitioners' review of industrial agent applications. *Autonomous Agents and Multi-Agent Systems* **3**(4), 389–407.

Resnick, Mitchel (1994). *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds.* MIT Press.

Voos, Holger (1999*a*). Market-based algorithms for optimal decentralized control of complex dynamic systems. In: *Proceedings of the 38th Conference on Decision and Control, Phoenix, AZ, USA.* Vol. 4. IEEE. pp. 3295–3296.

Voos, Holger (1999*b*). Market-based control of complex dynamic systems. In: *Proceedings of the IEEE Internation Symposium on Intelligent Control, Intelligent Systems and Semiotics, Cambridge, MA, USA.* pp. 284–289.

Voos, Holger (2000). Intelligent agents for supervision and control: A perspective. In: *Proceedings of the 15th IEEE International Symposium on Intelligent Control (ISIC 2000), Rio, Patras, Greece.* pp. 339–344.

Wooldridge, Michael (2002). *An Introduction to MultiAgent Systems.* John Wiley and Sons, Ltd.