Evaluating Agent Architectures Using Simulation

B. S. Logan
School of Computer Science
University of Nottingham
Nottingham NG8 1BB, UK
{bsl}@cs.nott.ac.uk

Abstract

In this position paper, we present an overview of recent work on using simulation to evaluate agent architectures. We characterise the kinds of evaluation that can be performed using simulation, and highlight some of the key issues in using simulation for evaluation. Two main trends in recent work are then identified: an increasing adoption of ideas from the 'mainstream' simulation community (particularly parallel discrete event simulation), and an increasing focus on the interoperability of simulators. We characterise the features that distinguish MAS simulation classical parallel discrete event simulation problems and highlight recent work which attempts to address these problems. We argue that, together, these developments point to the emergence of new software engineering methodologies for MAS, in which simulation (and hence evaluation) plays a central role throughout the development process.

Introduction

Multi-agent systems are often extremely complex and it can be difficult to formally verify their properties. As a result, design and implementation remains largely experimental, and experimental approaches are likely to remain important for the foreseeable future. In this context, simulation has a key role to play in the design and analysis of agent architectures and systems. Simulation allows a degree of control over experimental conditions and facilitates the replication of results in a way that is difficult or impossible with a prototype or fielded system, allowing the agent designer or researcher to focus on key aspects of the system. In addition, in many cases, the simulation runs faster than real time, allowing the investigation of a large number of alternative scenarios. It is therefore not surprising that over the last two decades, a wide range of MAS simulators and testbeds have been developed (Durfee & Montgomery 1989; Pollack & Ringuette 1990; Atkin *et al.* 1998; Sloman & Poli 1996; Anderson 2000; Schattenberg & Uhrmacher 2001; Gasser & Kakugawa 2002; Riley & Riley 2003), and simulation has been applied to a wide range of MAS research and design problems, from models of complex individual agents employing sophisticated internal mechanisms to models of large scale societies of relatively simple agents which focus more on the interactions between agents.

In this position paper, we present an overview of recent work on using simulation to evaluate agent architectures. We characterise the kinds of evaluation that can be performed using simulation, and highlight some of the key issues in using simulation for evaluation. We then go on to identify two important recent trends in MAS simulation: an increasing adoption of ideas from the 'mainstream' simulation community (particularly parallel discrete event simulation), and an increasing focus on the interoperability of simulators. We argue that, taken together, these developments point to the emergence of new software engineering methodologies for MAS, in which simulation (and hence evaluation) plays a central role throughout the development process.

MAS Simulation

In what follows, it will be useful to distinguish between benchmarks, testbeds and 'simulation proper'. As used here, a *benchmark* is a standard agent task or problem (or set of problems) which is taken to be representative of the problems is a given domain (or characterising certain features of problems in that domain). The evaluation criteria for a benchmark are usually quantitative, and may be explicitly stated (e.g., problem size, or performance score on a task), or implicit, such as CPU time. Like benchmarks, *testbeds* embody a standard problem; however a testbed differs from a benchmark in providing some means of evaluating one or more aspects of an architecture, e.g., by providing code which implements part of the problem. For example, Tileworld (Pollack & Ringuette 1990), Phoenix (Cohen *et al.* 1989), Gamebots (Kaminka *et al.* 2002), AgentCities (`www.agentcities.org`) and RoboCup (`www.robocup.org`) can all be seen as testbeds in this sense. Evaluating an agent architecture using a testbed usually involves either interfacing an implementation of the agent with the testbed code using a well defined API or communication protocol, as in, e.g., Gamebots, or redefining part of a 'standard' agent supplied with the testbed. In some cases, testbeds limit architectural design choices (if only in terms of the implementation language used). However they can reduce the implementation effort required to perform an evaluation, and may offer additional benefits in terms of standardising the calculation of a performance metric. The main limitation of benchmarks and testbeds is that, although they are representative of some common types of MAS problems, they do not cover the full spectrum of agent design problems.

Simulators generalise testbeds in two ways: they allow user-defined problems, and they allow parts of the agent architecture itself to be simulated rather than implemented (for example, systems such as JAMES (Schattenberg & Uhrmacher 2001) and SIM_AGENT (Sloman & Poli 1996) allow different parts of the agent to be modelled at different levels of detail). In implementation terms, simulators can be viewed as being intermediate between benchmarks and testbeds, in providing tools for developing an evaluation rather than an implementation of a predefined problem or environment as with a testbed. However in an important sense, simulators can be seen as subsuming both benchmarks and testbeds. With careful design of interfaces (see below), simulators can be used to implement standard benchmarks and testbeds as *simulation components* which can be freely reused by agent researchers and developers.

Simulation-based evaluation usually targets quantitative measures of performance, e.g., the ability to perform some (real or synthetic) task, or non-functional requirements such as CPU or elapsed time. However simulation can also be used to investigate qualitative properties, such as possible system failure modes. In some cases, exhaustive testing is possible, allowing verification of correctness properties. The ability to specify the problem allows simulation-based evaluation to be tailored to a particular research question or agent development task. The disadvantage is that the more problem specific the evaluation, the less informative the results are likely to be to the agent research and development community as a whole. Similarly, the more abstract the model of the agent, the less reliable the performance measures, such as run time etc., will be, in that they depend critically on the assumptions made in developing the simulation of (parts of) the agent. The ability to transition from simulated agent components to real agent implementations and from abstract benchmark problems to domain specific problems and evaluations are therefore key goals in simulator development.

Recent Trends in MAS Simulation

Early agent testbeds and simulators were often fairly ad-hoc, and typically adopted a centralised, asynchronous or time-driven approach to simulation. In a centralised simulator, all the agents are either simulated by a single process or connect to a single process which simulates the environment and manages interactions between the agents. Such simulators are (relatively) simple to build and, in the case of asynchronous simulation where CPU or wall clock time is taken as a measure of simulation time, easy to integrate with existing agent code. Time-driven simulation, in which simulation time advances in fixed timesteps, usually requires more control over agent execution, but has the advantage that results are independent of CPU load.

Centralised simulators have been used to study a wide range of agent phenomena. For example, Tileworld has been used to study commitment strategies (i.e., when an agent should abandon its current goal and replan) (Kinny & Georgeff 1991; Pollack *et al.* 1994) and in comparisons of reactive and deliberative agent architectures (Pollack & Ringuette 1990), and SIM_AGENT has been used to investigate the effectiveness of affective and deliberative control in simple agent systems (Scheutz & Logan 2001). However for simulations of MAS with thousands or tens of thousands of agents, the scalability of centralised simulators is an issue.

More recently there has been an increasing adoption of ideas from the "mainstream" simulation community, and in particular work in discrete event and distributed simulation.

Discrete Event Simulation

In discrete event simulation, the evolution of a system is represented as a sequence of events. Each event occurs at a fixed point in time and marks a change of state in the system. Discrete event techniques have been used in a number of recent MAS simulators. For example, the SPADES system (which forms the basis of the new RoboCup 3D testbed (Riley 2003)) builds on ideas from discrete event simulation and the JAMES system (Schattenberg & Uhrmacher 2001) uses the DEVS Discrete Event System Specification modelling formalism widely used within the simulation community. These systems utilise a hybrid discrete event simulation approach (sometimes called "Software in the loop"), in which the 'agents' to be simulated may be models of agents (e.g., situated automata), or they may be implementations of agents or agent components in a simulated environment, or a mixture of the two. Agent components with difficult to predict latencies, such as deliberation, are not modelled but called directly from the simulation, with the actual computational time (possibly scaled) being used to determine the time of the next event. Alternatively, the nominal time required to execute each primitive action performed by a (simulated or implemented) agent can be mapped into logical times for events in the simulation.

Distributed Simulation

At the same time, there has been an increasing focus on distributed simulation. Distributed simulation addresses two key problems of existing MAS simulations and simulators: scalability and simulation re-use.

The computational requirements of simulations of many large multi-agent systems far exceeds the capabilities of a single computer. Each agent may be a complex system in its own right (e.g., with sensing, planning, inference, etc. capabilities), requiring considerable computational resources, and many agents may be required to investigate the behaviour of the system as a whole or even the behaviour of a single agent. As researchers have attempted to simulate larger and more complex MAS, they have increasingly turned to distributed approaches to simulation. Distributed simulation exploits the natural parallelism of MAS. Simulation components can be distributed so as to make the best use of available computational resources, allowing agent researchers and developers to run agent simulations in less time and/or investigate multi-agent systems which are simply to large to be effectively simulated on a single computer.

Distribution also promotes *inter-operability* of simulators and simulation components. No one simulator or testbed is, or can be, appropriate to all agents and environments. Investigating a particular problem therefore frequently entails

the development of a new simulation. The effort required to develop a new simulation from scratch is considerable. There is therefore a strong incentive to reuse existing simulation components, toolkits and testbeds for a new problem. For example, in demonstrating that a particular result holds across a range of agent architectures or environments, it would often be conventient to be able to re-use (parts of) existing simulators and testbeds. However, at present, simulations developed for different simulators typically don't inter-operate, making it more difficult to re-use simulation components. Combining a simulation of an agent architecture developed for one simulator with a simulation of an environment developed for another typically involves reimplementation of one or both components. If many architectures must be simulated in the same environment or the same architecture simulated in several different environments, the problem is compounded.

The last decade has witnessed an explosion of interest in distributed simulation as a strategic technology for linking simulation components of various types at multiple locations to create a common virtual environment. The culmination of this activity was the development of the High Level Architecture (HLA), a framework for simulation reuse and interoperability developed by the US Defence Modelling and Simulation Office (Kuhl, Weatherly, & Dahmann 1999). Using HLA, a large-scale distributed simulation can be constructed by linking together a number of geographically distributed simulation components (or federates) into a single, larger simulation (or federation). The federates may be written in different languages and run on different machines. HLA (with minor revisions) has been adopted as an IEEE standard (IEEE 1516) (IEEE 2000) and is likely to be increasingly widely adopted within the simulation community. As such, HLA-compliance will be an increasingly important feature of agent simulators, allowing inter-operation with other simulations, re-use of agent simulation components and the distribution of agent and other simulation components across multiple computers to increase the overall performance of a global simulation. HLA-compliant agent simulators are now starting to appear, e.g., the HLA_AGENT system developed at the University of Nottingham (Lees *et al.* 2004) (`www.agents.cs.nott.ac.uk/simulation/hla_agent`) and HLA-RePast developed at the University of Birmingham (Minson & Theodoropoulos 2004).

The Problem of Shared State

However, while conventional distributed simulation can bring real benefits from an inter-operability point of view, the speedups that can be attained in practice (particularly for situated MAS) are often more limited. The simulation of situated agents (e.g., robots situated in a physical environment, or characters in a computer game or interactive entertainment situated in a virtual environment) presents particular challenges which are not addressed by standard parallel discrete event simulation (PDES) models and techniques. While the modelling and simulation of agents, at least at a coarse grain, is relatively straightforward, it is harder to apply conventional PDES approaches to the simulation of the agent's environment.

In a conventional decentralised event-driven distributed simulation the simulation model is divided into a network of Logical Process (LPs). Each LP maintains its own portion of the simulation state and LPs interact with each other in a small number of well defined ways. The topology of the simulation is determined by the topology of the simulated system and its decomposition into LPs, and is largely static.

In contrast, the interaction of agents in a situated MAS is often hard to predict in advance. Different kinds of agent have differing degrees of access to different parts of the environment at different times. The degree of access is dependent on the range of the agent's sensors (read access) and the actions it can perform (write access). For example, what a mobile agent can sense is a function of the actions it performed in the past which is in turn a function of what it sensed in the past. As a result, it is difficult to predict which parts of the simulation state an agent can or will access without running the simulation. This makes it hard to determine an appropriate topology for a MAS simulation a priori, and simulations of MAS typically have a large shared state, the agents' environment, which is only loosely associated with any particular process. This shared state can form a bottleneck, limiting the speedups that can be attained.

The efficient simulation of systems with large shared state is therefore a key problem in the distributed simulation of MAS. In our own work on the PDES-MAS project (Logan & Theodoropoulos 2001), we have investigated a new approach to parallel discrete event simulation of MAS in which both the agents and the environment are distributed. In PDES-MAS, the shared state is loosely associated with a group of special Communication Logical Processes (CLPs), and the distribution of state (i.e., its allocation to CLPs) changes at run time in response to the events generated by the agents during the simulation. This approach facilitates load balancing and data distribution, by moving state computationally closer to the agents which access it (see `www.cs.bham.ac.uk/research/pdesmas`).

Discussion

We believe that, together, these developments point to the emergence of new software engineering models for agent based systems in which simulation (and hence evaluation), rather than being seen as a tool for a particular stage or phase in development (e.g., proof of concept or pre-deployment), plays a central role throughout the development process. In the future we can envisage a development process which starts with abstract models of a small number of agents which are progressively refined and extended to give more complex models of larger numbers of agents in which progressively more of the agent's functionality is implemented by 'real code' rather than being simulated. Successive stages of refinement and elaboration ultimately result in the deployed system. Such approaches require the ability to model at multiple scales, the integration of different models of time, and the ability to seamlessly transition models from simulated implementations to real ones. This is an extremely challenging and ambitious goal, but some initial progress has been made. For example, the MACE3J

system (Gasser & Kakugawa 2002) simulates MAS models seamlessly across a variety of scales and architecture types, from single PCs to heterogeneous distributed GRID environments, allowing the developer to progressively relax control over uncertainty, bringing the simulation closer to reality. Such simulation-based approaches complement more formal approaches to agent development, such as proof based and model based verification, as they can be applied to larger and more complex systems whose behaviour is hard to formalise.

As deployed agent systems become larger and more complex and increasingly interact with other systems and user communities, the need for agent simulation methodologies and tools will only increase. However we can look forward to the emergence of flexible, efficient and generic platforms for simulation of multi-agent systems.

References

Anderson, J. 2000. A generic distributed simulation system for intelligent agent design and evaluation. In Sarjoughian, H. S.; Cellier, F. E.; Marefat, M. M.; and Rozenblit, J. W., eds., *Proceedings of the Tenth Conference on AI, Simulation and Planning, AIS-2000*, 36–44. Society for Computer Simulation International.

Atkin, S. M.; Westbrook, D. L.; Cohen, P. R.; and Jorstad., G. D. 1998. AFS and HAC: Domain general agent simulation and control. In Baxter, J., and Logan, B., eds., *Software Tools for Developing Agents: Papers from the 1998 Workshop*, 89–96. AAAI Press. Technical Report WS–98–10.

Cohen, P. R.; Greenberg, M. L.; Hart, D. M.; and Howe, A. E. 1989. Trial by fire: Understanding the design requirements for agents in complex environments. *AI Magazine* 10(3):32–48.

Durfee, E. H., and Montgomery, T. A. 1989. MICE: A flexible testbed for intelligent coordination experiements. In *Proceedings of the Ninth Distributed Artificial Intelligence Workshop*, 25–40.

Gasser, L., and Kakugawa, K. 2002. MACE3J: Fast flexible distributed simulation of large, large-grain multi-agent systems. In Castelfranchi, C., and Johnson, W. L., eds., *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, 745–752. Bologna: ACM Press.

2000. IEEE Standard for modeling and simulation (M&S) High Level Architecture (HLA) — Framework and rules. IEEE. (IEEE Standard No.: 1516-2000).

Kaminka, G. A.; Veloso, M. M.; Schaffer, S.; Sollitto, C.; Adobbati, R.; Marshall, A. N.; Scholer, A.; and Tejada, S. 2002. GameBots: A flexible test bed for multiagent team research. *Communications of the ACM* 45(1):43–45.

Kinny, D., and Georgeff, M. P. 1991. Commitment and effectiveness of situated agents. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91)*, 82–88. Sydney, Australia: Morgan Kaufmann.

Kuhl, F.; Weatherly, R.; and Dahmann, J. 1999. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*. Prentice Hall.

Lees, M.; Logan, B.; Oguara, T.; and Theodoropoulos, G. 2004. HLA_AGENT: Distributed simulation of agent-based systems with HLA. In *Proceedings of the International Conference on Computational Science (ICCS'04)*, LNCS, 907–915. Krakow, Poland: Springer.

Logan, B., and Theodoropoulos, G. 2001. The distributed simulation of multi-agent systems. *Proceedings of the IEEE* 89(2):174–186.

Minson, R., and Theodoropoulos, G. 2004. Distributing RePast agent-based simulations with HLA. In *Proceedings of the 2004 European Simulation Interoperability Workshop*, number 04E-SIW-046. Edinburgh: Simulation Interoperability Standards Organisation and Society for Computer Simulation International. Paper No. 04E-SIW-046.

Pollack, M. E., and Ringuette, M. 1990. Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 183–189. Boston, MA: AAAI.

Pollack, M. E.; Joslin, D.; Nunes, A.; Ur, S.; and Ephrati, E. 1994. Experimental investigation of an agent commitment strategy. Technical Report TR 94–31, University of Pittsburgh, Pittsburgh, PA 15260.

Riley, P., and Riley, G. 2003. SPADES — a distributed agent simulation environment with software-in-the-loop execution. In Chick, S.; Sánchez, P. J.; Ferrin, D.; and Morrice, D. J., eds., *Winter Simulation Conference Proceedings*.

Riley, P. 2003. SPADES: System for parallel agent discrete event simulation. *AI Magazine* 24(2):41–42.

Schattenberg, B., and Uhrmacher, A. M. 2001. Planning agents in JAMES. *Proceedings of the IEEE* 89(2):158–173.

Scheutz, M., and Logan, B. 2001. Affective vs. deliberative agent control. In *Proceedings of the AISB'01 Symposium on Emotion, Cognition and Affective Computing*, 1–10. AISB.

Sloman, A., and Poli, R. 1996. SIM_AGENT: A toolkit for exploring agent designs. In Wooldridge, M.; Mueller, J.; and Tambe, M., eds., *Intelligent Agents II: Agent Theories Architectures and Languages (ATAL-95)*. Springer–Verlag. 392–407.