2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology

A Framework for Developing Agent-Based Distributed Applications

Michel Oey Sander van Splunter Elth Ogston Martijn Warnier Frances M.T. Brazier
Faculty of Technology, Policy and Management
Delft University of Technology
Delft, The Netherlands
Email: {M.A.Oey,S.vanSplunter,E.Ogston,M.E.Warnier,F.M.Brazier}@tudelft.nl

***Abstract*—The development of large-scale distributed multi-agent systems in open dynamic environments is a challenge. System behavior is often not predictable and can only be evaluated by execution. This paper proposes a framework to support design and development of such systems: a framework in which both simulation and emulation play an important role. A distributed agent platform (AgentScape) is used to illustrate the potential of the framework.**

***Keywords*-multi-agent systems, agent-based simulation, emulation, development, distributed systems**

I. Introduction

Multi-agent systems are currently deployed in a large number of large-scale, distributed, open environments, such as the energy domain, crisis management, transportation, and sensor-networks [1], [2], [3]. These environments are typically characterized by the autonomy of the entities involved. Supporting the design and development of such distributed, large-scale agent applications is the topic this paper addresses.

Theoretical description and analysis (*design*), simulation (*simulation*), and deployment of a distributed application (*deployment*) are often the three phases involved. This paper adds another phase: *emulation*. Emulation provides a means to analyze system behavior in a distributed environment, introducing non-determinism, distribution, characteristics of real-life systems. In particular, during emulation only partial local views of system behavior are possible.

Incremental development is discussed in more depth in Section II. Section III presents the framework to support the design of distributed multi-agent applications. Section IV describes a prototype of the framework, illustrated with AgentScape as the runtime environment. Section V, in turn, illustrates the potential of the framework for agent applications. Section VI places the framework in the context of related work, and Section VII discusses the results.

II. Incremental Development

This paper proposes an incremental approach to multi-agent applications development, distinguishing four phases:

1) **design** - design of an application.
2) **simulation** - implementation and testing of a design by simulation.
3) **emulation** - refinement and testing of an implementation by emulation.
4) **deployment** - refinement, testing and actual deployment of an implementation in the real-world.

The main motivation behind this incremental approach is to separate concerns during design, focusing initially on the correctness of a design, for example, of a distributed coordination algorithm, abstracting from the practicalities of large-scale, distributed open environments. Truly distributed, open agent applications need to deal with practicalities such as network failures, network latencies, concurrency, asynchronous communication, dead locks, live locks, non-determinism, distributed nature, security, replication, fault tolerance, etc.

Simulations enable designers to focus on the functionality for which a system is to be designed, e.g., algorithms, and to simulate a distributed environment in different configurations often running on a single machine. Each configuration can test one or more specific versions of an application, each potentially focusing on a different aspect, all within a controlled environment. If tests fail, log-files/traces can be used to identify and fix faults. Tests can relatively easily be repeated with the same results.

In contrast, *emulation* provides a controlled, real distributed environment during design, including real network characteristics, such as network latencies and concurrency. Furthermore, emulation provides debugging support, such as logging and snapshots that enable analysis and testing of system behavior before deployment. The four development phases are briefly discussed below.

A. Design Phase

In the first phase, a design or model of a distributed multi-agent application is made, including models of each of the individual agents and their interaction. For convenience, this phase is assumed to include all the steps needed to design an application, such as requirements analysis, functional design, or even a formal verification. The outcome of this phase is a design describing the architecture of an application, the agents behaviors and their interaction.

978-0-7695-4191-4/10 $26.00 © 2010 IEEE
DOI 10.1109/WI-IAT.2010.134

B. Simulation Phase

The main goal of the second phase is to test the *functionality* of a design within a controlled environment to analyze system behavior without having to address (practical) issues such as network failures that complicate development. Typically, the behavior of a distributed agent application is analyzed on a stand-alone computer system, without actually running an actual distributed agent platform.

Different aspects of an application can be studied in isolation. For example, to test an application's communication protocol, first a faultless network can be simulated. In a later stadium, network failures can be simulated to test the application's ability to withstand lost connections and timeouts. Simulations can also be used to simulate conditions that are not easily tested in the real environment.

In addition, simulations provide debug/tracking facilities, such as logging and snapshots, to keep track of the progress of the application and to analyze errors as they occur. The ability to suspend a running application temporarily to allow inspection of the state of an application is also an option.

Unfortunately, system simulations also have their limitations. By definition, a simulation can only test aspects of an application that are supported by that simulation environment. Moreover, the models used in a simulation to simulate real distributed, open environments may not be sufficiently accurate. Models of non-determinism in distributed, open environments, for example, are not often realistic. To include realistic characteristics of distributed open, environments into system development, the next phase is needed.

C. Emulation Phase

In the emulation phase, an agent application is tested on a distributed agent platform that provides a runtime environment in which agents can communicate and possibly migrate between hosts. Emulation comes closer to the environment in which a system is to be deployed, than simulation. An emulation can run an agent application on multiple physical networked machines, testing the application in a real distributed setting with network latencies and race-conditions. Often the computer systems used for the (distributed) emulation (e.g., a small cluster on a LAN) are controlled to support debugging.

Debugging facilities in a distributed environment become more important, but are often more difficult to implement. In emulation, the underlying agent platform can provide support for logging, distributed measurement, and snapshots to inspect the state of the individual agents they host. Suspending a distributed application for the purpose of analysis is, however, not always an option.

D. Deployment Phase

The final phase in system development is deployment in the intended open environment. Whereas during the emulation phase machines were under control of the developers, in the deployment phase, the machines typically are not under their control. Debugging applications under these circumstances is typically limited to log messages, which may be inspected offline if the owners of the systems on which they are hosted are supportive.

III. Simulation/Emulation Framework

This paper proposes a *Simulation/Emulation Framework* to streamline the development of large-scale, distributed multi-agent applications, but which also benefits distributed applications in general.

A. Minimal Requirements

The design of the framework is based on the following minimum requirements:

- provide the ability to focus on the logic of a distributed agent application, i.e., to isolate its internal models and algorithms.
- provide support for debugging an application in different environments (such as logging, making snapshots, temporarily suspending an application).
- support the simulation and emulation phases

The framework this paper proposes provides a 'fixed' interface for multi-agent application design, supporting analysis of the effects of, for example, different (versions of) algorithms and communication patterns, during system design. The interface hides the details of the underlying runtime environment, supporting the transition between simulation and emulation as often as needed during design.

B. Architecture of the Framework

The proposed framework consists of three layers (see Figure 1). The top layer is the (distributed) multi-agent application itself - the system to be developed. The middle-layer is the fixed interface defined by the framework, which provides an application methods to create and debug a (distributed) simulation/emulation. The bottom layer, the backends, implements the underlying runtime environment. The framework supports multiple backends, each with its own characteristics: backends for simulation and emulation.

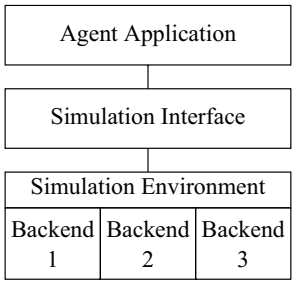

Figure 1. Overview of the Simulation Framework

C. Interface of the Framework

The interface layer defines a minimum set of functionality for multi-agent application design and development. The interface includes the following functionality:

- *agents* - the basic units in a multi-agent application.
- *communication* - agents send and receive messages.
- *protocols* - the logic/algorithms of each agent is defined as one or more protocols.
- *network* - different (distributed) network characteristics can be configured
- *logging* - each agent can log data for analysis purposes.
- *backend* - backends implement runtime environments.

D. Simulation/Emulation Backends

The backends of the framework are the runtime environments in which applications can run. Backends differ in their characteristics. One backend, for example, provides a simulation environment running on a single machine, single threaded in which communication between agents does not rely on actual network activity. Another backend provides a (distributed) emulation environment, running on multiple machines. Agents run on different machines and run in parallel, communicating via asynchronous message passing

Backends can be categorized as follows:

1) single machine, single thread
2) single machine, multiple threads
3) multiple machines, multiple threads, lock-step
4) multiple machines, multiple threads, asynchronous

These backend types progressively provide environments that come closer to a real distributed environment, providing more concurrency and network characteristics, but in turn make debugging more difficult.

The first type of backend is a very strict simulation environment on one machine, with a single thread of execution, running each agent sequentially. Consequently, this simulation can provide deterministic test runs, supporting evaluation and debugging of system behavior. This backend is typically used only to test basic correctness of the application in a simple scenario.

The second type of backend also runs on a single machine, but multiple threads of execution allow for (pseudo)concurrency. Agents run in parallel, each executing their own program. This simulation environment provides insight on how well a system is capable of handling concurrency. The application's behavior can be analyzed by suspending the simulation during execution and by analysis of logs.

The third type of backend runs on multiple (independent) machines. This environment is closer to emulation as the application actually runs on multiple machines. Agents run in parallel, but some (artificial) synchronicity can be used to provide some control over how the application progresses. For example, with lock-step, agents proceed to the next step in an algorithm only after all agents have finished the current step, but agents execute steps in parallel. Often the transition of going from a single-machine, synchronous, agent application to a multi-machine, asynchronous, application is a challenge. This lock-step approach provides a way to take the transition more gradually (see Section V). Unfortunately, as the application is now distributed over multiple machines, debugging, analyzing unexpected behavior and errors, becomes more difficult.

The final type of backend is closest to a fully distributed agent application. All agents run independently of each other. There is very little control over the application. Analysis of behavior and debugging, is mostly done by gathering and analyzing the logs of the agents.

The main advantage of the framework is that the interface is well-defined and that transitioning between the different phases relies solely on availability of an appropriate backend. Ideally an application is designed to transition without change. Unfortunately, this is not always possible. For example, a multi-agent application implicitly relying on agents being run in some sequential order, will run as expected in the first backend type, but not in the others unless explicit synchronization is built in. In general, the algorithms must change to cope with parallelism.

IV. Prototype Implementation and AgentScape

A prototype of the framework has been implemented in Java, which is supported on many different computer-architectures and operating systems. Currently, two backends have been implemented, a single-machine, multi-threaded simulation environment (type 2) and a multi-machine, multi-threaded emulation environment (type 4). The latter backend is built on the AgentScape* agent platform, which has been designed to support the design and deployment of large-scale, heterogeneous, secure, distributed agent systems [4]. In addition, the framework provides support for applications to define *steps* in order to run in lock-step (type 3 in Section III-D). Within AgentScape, *agents* are active entities that reside within *locations*, communicate with each other and access *services*. Agents may migrate between locations.

The simulation/emulation framework currently supports logging. Each agent has access to its private log where any type of log message can be recorded. The logs of all distributed agents can be sorted and filtered and is automatically gathered into a central point for further analysis. Creation of snapshots of the state of agents is currently being developed.

V. Usage Scenario: Distributed Auctions

This section discusses a typical usage scenario of the framework in the domain of distributed auctions. It illustrates how the framework can be used in the design, development, simulation, and emulation of distributed applications.

*http://www.agentscape.org

Combinatorial auctions (CA) are capable of delivering (semi-)optimal solutions to resource allocation problems. Gradwell et al. [5] compare the economic characteristics and the runtime performance of a market-based approach and a combinatorial auction, when both are applied to common data sets. They have designed and implemented a Multiple Distributed Auction (MDA) that consists of multiple Continuous Double Auctions (CDA), each trading one type of good (resource). The MDA implements a uni-processor agent-based simulation using the Repast framework [5].

In a follow-up study, the single-machine MDA simulation was reengineered into a distributed emulation [6]. The AgentScape multi-agent middleware is the underlying platform used to distribute the entities in the MDA over multiple hosts and to provide the necessary communication between them. The centralized (Repast-based) simulation uses a single thread of execution and runs bidding rounds in each auction sequentially. The AgentScape-based emulation, however, runs on multiple computers and runs auctions in parallel. Emulation also makes it possible to verify that the reengineering is (functionally) correct: By constraining the asynchronous nature of the emulation and synchronizing on the notion of rounds (lock-step) the results of the emulation and (Repast) simulation are compared.

The reengineering effort shows that going from a centralized simulation to a distributed emulation is not always straightforward. The results confirm that algorithms in the centralized simulation may need to change in a distributed environment to gain scalability. Furthermore, the study shows that incremental software development of MAS software is worthwhile. The combination of simulation and emulation allows the development efforts to focus on the concept of multiple auctions (simulation) while at the same time identifying the difficulties when actually deploying the algorithm in a distributed environment (emulation).

VI. Related Work

Agent-based simulations are an integral part of development. The use of standard simulation and development frameworks is, however, surprisingly rare. In 2007, Davidsson et al. [7] surveyed agent-based simulations reported in the literature. They noted that nearly half of the reviewed papers did not state how the simulations were implemented. Of the ones that did, many programs were written from scratch. The reasons why existing platforms are not used more extensively are not entirely clear.

A good selection of simulators and development frameworks does exist. They can be roughly divided into single-machine and multi-machine/distributed simulators. Repast [8], its successor Repast-S, and MASON [9] are examples of the first type. They are well-known agent-based modeling and simulation toolkits for single-machine simulations. However, for emulations, frameworks running on multiple machines are better suited. MACE3J [10] is a Java-based MAS simulation, integration and development testbed that runs on single and multi-processor environments. Farm [11] is a distributed environment for simulating large-scale multi-agent systems. ProtoPeer [12] is a prototyping toolkit for peer-to-peer systems development, supporting both event-driven simulation and live network deployment. Netbed [13] is a platform that integrates network simulation, emulation and the live deployment of wide-area distributed systems. However, both ProtoPeer and Netbed focus only on lower-level concepts, such as peers and networks, and therefore, are less suited for multi-agent applications. Some simulators/frameworks provide a hybrid environment to bridge the gap between software and hardware. In MASH [14], intended for embedded multi-agent systems (eMAS), real physical and virtual simulated agents cohabit. In the Java Agent Framework (JAF) [15] and its associated simulator, Multi Agent System Simulator MASS, agents can perform a mixture of both real and simulated activities.

The Simulation/Emulation framework presented in this paper explicitly defines an additional phase in system design and development: an emulation phase, specifically targeted to evaluation of system behavior in a distributed setting. The strength of this framework is that it targets multi-agent application development and uses an underlying agent platform to support emulation. Hence, the framework offers feedback at the desired level of multi-agent application development and not just on "lower-level" network issues. In addition, the framework offers a backend to AgentScape, enabling simple deployment due to extensive support for running agent systems in distributed environments.

VII. Discussion and Conclusions

The development of large-scale, distributed agent applications for open environments requires a careful balance of a wide range of concerns: a detailed understanding of the behavior of the algorithms being employed, a knowledge of effects and costs of operating in a distributed environment, and expertise in the performance requirements of the application itself. Without a good development methodology the complexity of this task quickly becomes unmanageable.

Studies rarely consider this development cycle as a whole. Scientific projects are often confined to the design of abstract algorithms and simulations to test their operation. Further development is left to application designers. This narrow focus can lead to researchers developing protocols that work well in theory, but not practice.

The Simulation/Emulation framework presented in this paper enables a more integrated approach to agent application development. It facilitates incremental development, analysis, and testing using simulation and emulation to close the gap between design and real world deployment. The behavior of agent applications is specified in a runtime

environment independent way. Backends enable development and debugging of the application in different runtime environments: simulation or emulation. Even though algorithms may have to be adapted when changing runtime environments to take advantage of multi-threading, add deadlock detection/prevention, etc, efforts will not have to be on implementing the supporting runtime environment itself. Furthermore, a common framework provides for a common code base, so that algorithms under development can easily be shared between teams. It makes explicit the differences between development phases, improving the focus of each. Experts from different areas can better identify and communicate issues arising from differences in assumptions or priorities.

A prototype implementation of the framework uses AgentScape to implement the emulation backend. AgentScape is a fully fledged agent operating system on which distributed agent-based applications are deployed. Changing the runtime environment from a single machine simulation to a distributed multi-machine emulation was a matter of changing one line of code. Currently, work is in progress to use the prototype framework in distributed energy resource management scenarios.

ACKNOWLEDGMENT

The authors thank Reinier Timmer and Evangelos Pournaras for their input. This research is partly supported by the NLnet Foundation http://www.nlnet.nl.

REFERENCES

[1] N. R. Jennings, "Agent-based computing: Promise and perils," in *16th Int. Joint Conf. on Artificial Intelligence (IJCAI-99)*, 1999, pp. 1429–1436.

[2] M. Wooldridge, "Agent-based software engineering," in *IEE Proceedings on Software Engineering*, 1997, pp. 26–37.

[3] E. Pournaras, M. Warnier, and F. M. T. Brazier, "A distributed agent-based approach to stabilization of global resource utilization," in *the International Conference on Complex, Intelligent and Software Intensive Systems (CISIS'09)*. IEEE, March 2009.

[4] N. J. E. Wijngaards, B. J. Overeinder, M. van Steen, and F. M. T. Brazier, "Supporting internet-scale multi-agent systems," *Data and Knowledge Engineering*, vol. 41, no. 2-3, pp. 229–245, June 2002.

[5] P. Gradwell and J. Padget, "A comparison of distributed and centralised agent based bundling systems," in *ICEC '07: Proc. of the 9th int. conference on Electronic commerce*. New York, NY, USA: ACM Press, 2007, pp. 25–34.

[6] P. Gradwell, M. A. Oey, R. J. Timmer, F. M. T. Brazier, and J. Padget, "Engineering large-scale distributed auctions," in *Proc. of the Seventh Int. Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, May 2008.

[7] P. Davidsson, J. Holmgren, H. Kyhlbäk, D. Mengistu, and M. Persson, "Applications of Agent Based Simulation," in *Multi-Agent-Based Simulation VII*, ser. LNCS, vol. 4442. Springer, 2007, pp. 15–27.

[8] M. J. North, N. T. Collier, and J. R. Vos, "Experiences creating three implementations of the repast agent modeling toolkit," *ACM Trans. Model. Comput. Simul.*, vol. 16, no. 1, pp. 1–25, 2006.

[9] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "MASON: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517–527, 2005.

[10] L. Gasser and K. Kakugawa, "MACE3J: fast flexible distributed simulation of large, large-grain multi-agent systems," in *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2002, pp. 745–752.

[11] B. Horling, R. Mailler, and V. Lesser, "Farm: A Scalable Environment for Multi-Agent Development and Evaluation," in *Advances in Software Engineering for Multi-Agent Systems*, A. G. C. Lucena, J. C. A. Romanovsky, and P. Alencar, Eds. Springer-Verlag, Berlin, February 2004, pp. 220–237.

[12] W. Galuba, K. Aberer, Z. Despotovic, and W. Kellerer, "ProtoPeer: a P2P toolkit bridging the gap between simulation and live deployement," in *Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. Brussels, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 1–9.

[13] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," Boston, MA, Dec. 2002, pp. 255–270.

[14] J.-P. Jamont and M. Occello, "A multiagent tool to simulate hybrid real/virtual embedded agent societies," in *WI-IAT '09: Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 501–504.

[15] R. Vincent, B. Horling, and V. R. Lesser, "An agent infrastructure to build and evaluate multi-agent systems: The java agent framework and multi-abent system simulator," in *Revised Papers from the International Workshop on Infrastructure for Multi-Agent Systems*. London, UK: Springer-Verlag, 2001, pp. 102–127.