An Agent-Based Framework for Developing 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 agent applications in large-scale, distributed, open environments is a challenge. This paper proposes a framework to support development and deployment of distributed agent applications. The trajectory from design to real-world deployment starts with simulation and ends with emulation. The framework is illustrated with AgentScape, a distributed agent platform.**

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

I. Introduction

The multi-agent systems paradigm is particularly applicable to applications in 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. Distributed identity and integrity management are crucial, mandating a framework to support heterogeneity, fault-tolerance and security.

Development of distributed systems and algorithms entails: theoretical description and analysis (*design*), simulation (*simulation*), and deployment of a distributed application (*deployment*). This paper focuses on the second phase.

A structured development approach is proposed in which the second phase is extended to include not only single system simulation but also *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.

The framework this paper proposes provides a 'fixed' interface for multi-agent system 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.

Incremental development is discussed in more depth in Section II. Section III presents the framework designed to support the design of distributed multi-agent systems. Section IV details the design of the framework, illustrated for AgentScape as the runtime environment. Section V, in turn, illustrates the potential of the framework for agent applications in two different domains. 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 system development, distinguishing the following 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. True 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. 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.

Unfortunately, even though simulation provides a powerful tool to develop distributed agent applications, it has its limitations. By definition, a simulation can only test aspects of an application that are supported by a specific simulator. Moreover, the models a specific simulator uses 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 gap between simulation and real-life deployment of applications, needs to be addressed.

The incremental approach to multi-agent system design proposed above explicitly includes *emulation*. Emulation in a controlled distributed environment during design, with debugging support, such as logging and snapshots, enables 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 system, of each of the individual agents and any interaction between agents is made. For convenience, this phase is assumed to include all of the steps necessary to design an application, such as requirements analysis, functional design, or even a formal verificatio. The outcome of this phase is a design that describes the architecture of an application, the agents behavior and their interaction.

B. Simulation Phase

The main goal of the second phase, simulation, 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, during simulation, the behavior of a distributed agent application is analyzed on a stand-alone computer system, without actually running an actual distributed agent platform.

To study different aspects of an application in isolation, the configuration of the simulation environment can be changed. For example, to first focus on the correctness of an application's communication protocol, the simulation environment can first provide (i.e., simulate) a faultless network. In a later stadium, network failures can be simulated to test an 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 often provide useful debug/tracking facilities. For example, most simulators provide logging facilities (e.g., the ability to make snapshots) to keep track of the progress of the application and to analyze errors as they occur. Some simulators provide the ability to suspend a running application temporarily to allow inspection of the state of an application.

C. Emulation Phase

In the emulation phase, an agent application is tested on an actual agent platform that provides a run-time 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 are controlled to support debugging. For example, first a small cluster of machines within a LAN may be used, after which a larger cluster of physically distributed machines may be deployed.

Debugging facilities in a distributed environment become more important, but are often more difficult to implement. However, for multi-agent systems the agent platform on which they run 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 section proposes a framework to support transitioning between these phases, focussing on simulation and emulation. The purpose of the proposed *Simulation/Emulation Framework* is to streamline the development of large-scale, distributed multi-agent system applications, but can also benefit less ambitious agent applications or 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, and, if possible, deployment of an application.

By isolating the logic of an agent application, this logic can be designed and evaluated, without having to worry about details of the environment. Most of the code developed to evaluate the logic of an agent application can remain intact when going from one phase to the next. As mentioned above, debugging is an important part of development. The framework must provide facilities to help debug a distributed application, including support for distributed measurements and/or sampling. Lastly, the framework must provide different environments for simulation and emulation. If possible, the framework should support deployment in the application's intended environment.

B. Architecture of the Framework

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

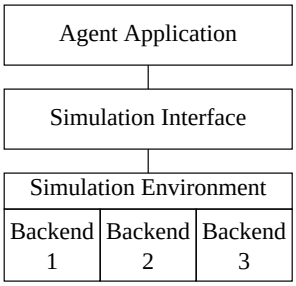

Figure 1. Overview of the Simulation Framework

C. Interface of the Framework

The interface layer defines a minimum set of functionality needed for multi-agent system design and development. The current interface contains the following minimum set of functionality:

- *agents* - the basic units in a multi-agent application
- *communication* - agents communicate by sending and receiving messages.
- *protocols* - the logic/algorithms of each agent is defined as one or more protocols.
- *network* - the (distributed) network can be configured to simulate/emulate network characteristics
- *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. All 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

The first type of backend is a very strict simulation environment on one machine, with a single thread of execution. This simulation environment runs each agent sequentially. Consequently, this simulation can, in principle, provide deterministic test runs, supporting evaluation of system behavior, and debugging if unexpected behavior occurs. This type of 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 allows for (pseudo-)concurrency by using multiple threads of execution. In this type of backend, agents run in parallel, each agent executing its own program. This type of simulation environment can provide some insight on how well a system is capable of handling concurrency. As the simulation environment still runs on a single machine, 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 than simulation as the application actually runs on multiple machines. Agents run in parallel, but some (artificial) synchronicity can be built into an environment to provide some control over how the application progresses. For example, agents can run in *lock-step* where multiple steps are defined and all agents synchronize on these steps. Agents execute steps in parallel, but wait until all agents have finished the current step, before proceeding to the next step. Often the step from 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 also Section V-A). Unfortunately, as the application is now distributed over multiple machines, debugging, analyzing unexpected behavior and errors, becomes more difficult.

The final type of backend provides an environment that comes 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, if a multi-agent system implicitly relies on agents being run sequentially in some order, it will only run as expected in the first backend type mentioned above, but not in the others which introduce parallelism. The algorithms must change to cope with parallelism.

IV. Prototype Implementation and AgentScape

The current implementation of the framework is written in Java. As Java is widely supported on different computer-architectures and operating systems, the framework can also be applied to different systems. Currently, two backends have been implemented, corresponding to types 2 and 4 in Section III-D: a single-machine, multi-threaded simulation environment and a multi-machine, multi-threaded emulation environment. The latter backend is built on top of the AgentScape agent platform. In addition, the framework contains functionality to enable applications to define *rounds* to run the latter backend in lock-step (type 3 in Section III-D).

The AgentScape agent platform has been designed to support the design and deployment of large scale, heterogeneous, secure, distributed agent systems [4], [5]. Within AgentScape, *agents* are active entities that reside within *locations*, communicate with each other and access *services*. Agents may migrate from one location to another. The leading principle in the design of AgentScape is to develop minimal but sufficient functionality that can be extended to incorporate new functionality or adopt (new) standards into the platform.

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 futher analysis. Creation of snapshots of the state of agents is currently being developed.

V. Usage Scenarios

This section discusses two typical usage scenarios of the simulation/emulation framework. The scenarios are from two different domains and illustrate how the framework can be used in the design, development, simulation, and emulation of distributed applications. The two domains are: distributed energy resource management and distributed auctions.

A. Distributed Auctions

Combinatorial auctions (CA) are capable of delivering (semi-)optimal solutions to resource allocation problems. Gradwell et al. [6] have compared the economic characteristics and the run-time performance of a market-based approach and a combinatorial auction, when both are applied to common data sets. The authors 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 [6].

In a follow-up study, the single-machine MDA simulation was reengineered into a distributed emulation [7]. The AgentScape multi-agent middleware has been used as the underlying platform 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. An emulated approach (AgentScape-based), 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 can be compared, while still using parallelism within each round in the emulation.

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. The experiment furthermore 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 many research projects. The use of standard simulation and development frameworks is, however, surprisingly rare. In 2007, Davidsson et al. [8] 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. A similar study on the 43 full papers published in IAT 2009 revealed similar conclusions: of the 21 papers that presented multi-agent simulation work, only five mentioned the use of a previously developed simulator or framework. The reasons why existing platforms are not used more extensively are not entirely clear.

A good selection of simulators and development frameworks does exist. Repast [9] and its successor Repast S are well-known agent-based modeling and simulation toolkits for single machine simulations. MACE3J [10] is a Java-based MAS simulation, integration and development testbed that runs on single and multi-processor environments. The Java Agent Framework (JAF) [11] and its associated simulator, Multi Agent System Simulator MASS, lets developers build and test agents out of components for standard services such as scheduling or communication. Farm [12] is a distributed environment for simulating large-scale multi-agent systems. MASH [13] is a multi-agent software/hardware simulator, intended for embedded multi-agent systems (eMAS). ProtoPeer [14] is a prototyping toolkit for developing peer-to-peer systems that supports both event-driven simulation and live network deployment. Netbed [15] is a

platform that integrates network simulation, emulation and the live deployment of wide-area distributed systems.

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 related approaches that also acknowledge the necessity of an emulation phase are not targeted to multi-agent system development. Hence, these approaches do not offer feedback on the level of multi-agent system development, but rather on "lower-level" network issues. The Simulation/Emulation framework offers feedback at the desired level as the framework is designed specifically for multi-agent system development. 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 abstract algorithms being employed, a knowledge of effects and costs of operating in a distributed environment, and an expertise in the performance requirements of the application itself. Without a good development methodology the complexity of this task can quickly become 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 are designed for a setting that does not match the settings encountered in practice.

The Simulation/Emulation framework presented in this paper enables a more integrated approach to agent application development. The framework enables incremental development, analysis, and testing using simulation and emulation to close the gap between design and real world deployment. The behavior of agent applications can be specified in a runtime-environment independent way. Different backends enable development and debugging of the application in different runtime environments: simulation or emulation. Even though the 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.

The use of 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. In the framework 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 the distributed energy resource management scenarios.

Acknowledgment

The authors thank Reinier Timmer and Evangelos Pournaras for their input on the simulation/emulation framework. This research is in part 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] IIDS, "AgentScape Agent Middleware," http://www.agentscape.org.

[5] 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.

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

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

[8] 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.

[9] 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.

[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] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.