A Framework for Developing Agent-Based Distributed Applications*

Michel Oey Sander van Splunter Elth Ogston Martijn Warnier Frances M.T. Brazier

Delft University of Technology
Delft, The Netherlands

Extended Abstract

Introduction

The development of agent applications in large-scale, distributed, open environments, such as the energy domain, crisis management, transportation, and sensor-networks, is a challenge. This paper proposes a framework to support incremental development and deployment of distributed agent applications. The main motivation for 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 and latencies, concurrency, asynchronous communication, dead locks, live locks, non-determinism, distributed nature, security, replication, fault tolerance, etc. The trajectory from design to real-world deployment contains four phases: (i) *Design*, (ii) *Simulation*, (iii) *Emulation*, and (iv) *Deployment*.

Design In the first phase, a design or model of a distributed multi-agent system is made, including models of each of the individual agents and interactions between them. For convenience, this phase is assumed to include all the steps necessary 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.

Simulation 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.

To study different aspects of an application in isolation, the configuration of the simulation environment can be changed. Each configuration tests one or more specific versions of an application, each potentially focusing on a different aspect, all within a controlled environment. For example, to test an application's communication protocol, first a faultless network and in a later stadium network failures can be simulated. 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, such as logging, snapshots and sometimes the ability to suspend a running application.

Unfortunately, simulation also 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 simulator uses to simulate distributed, open environments may not be sufficiently accurate. Models of non-determinism in distributed environments, for example, are not often realistic. The following phase is needed to include realistic characteristics of distributed open environments and bridge the gap between simulation and real-life deployment of applications.

*The full version of this paper is published in the Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT-10), 2010.

Emulation 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 networked machines, testing the application in a real distributed setting with network latencies and race-conditions. Often the computer systems used (e.g., a small cluster on a LAN) for (distributed) emulation are controlled to support debugging.

Debugging facilities in a distributed environment become more important, but are often more difficult to implement. However, in emulation, the underlying agent platform can provide support for logging, distributed measurements, 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.

Deployment The final phase in system development is deployment in the intended open environment. Whereas during emulation machines were under control of the developers, after deployment, the machines typically are not. 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.

The Framework

To support transitioning between these incremental phases, a framework is defined focusing on simulation and emulation. Its purpose is to streamline the development of large-scale, distributed multi-agent system applications, but also benefits distributed applications in general. The framework provides a 'fixed' interface for multi-agent system design, supporting analysis of, for example, different algorithms and communication patterns, during system design. The interface hides details of the underlying runtime environment, supporting the transition between simulation and emulation as often as needed.

The architecture of the framework consists of three layers. The top layer is the (distributed) multi-agent system 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 environment.

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: (i) *single machine, single thread*, (ii) *single machine, multiple threads*, (iii) *multiple machines, multiple threads, lock-step*, and (iv) *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 two backends run on a single machine, and therefore, are closer to simulation. The last two backends provide a distributed environment and come closer to emulation. In 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. Lock-step provides some control over how an application progresses.

In short, the framework facilitates simulation and emulation for incremental development, including analysis, and testing. This framework closes the gap between design and real world deployment. The behavior of agent applications can be specified in a runtime-environment independent way. Different backends provide an application different runtime environments for development and debugging. Even though algorithms may have to be adapted when changing runtime environments to handle multi-threading, 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 has been developed in Java, including two backends. One backend provides a simulation environment (type ii), and the other provides an emulation environment (type iv) running on AgentScape, a full-fledged agent platform*.

*http://www.agentscape.org