Developing Software Agents Using Enterprise JavaBeans

Dejan Mitrović
Department of Mathematics and Informatics
Faculty of Sciences
University of Novi Sad
Novi Sad, Serbia
dejan@dmi.uns.ac.rs

Mirjana Ivanović
Department of Mathematics and Informatics
Faculty of Sciences
University of Novi Sad
Novi Sad, Serbia
mira@dmi.uns.ac.rs

Milan Vidaković
Faculty of Technical Sciences
University of Novi Sad
Novi Sad, Serbia
minja@uns.ac.rs

Ali Al-Dahoud
Al-Zaytoonah University of Jordan
P.O.Box 130 Amman
11733 Jordan
aldahoud@zuj.edu.jo

ABSTRACT

Enterprise JavaBeans (*EJB*s) represent one of the most widely-used server-side component architectures for developing the business logic of enterprise-scale applications. Because of their runtime properties, such as scalability, security, and transactional integrity, *Enterprise JavaBeans* are also an excellent tool for building software agents. This paper outlines an architecture for developing and deploying *EJB*-based agents. The presented approach is designed in a way that employs all the benefits of *EJB*s, while, at the same time, it hides the underlying complexity from agent developers.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Distributed Artificial Intelligence—*Multiagent systems*

General Terms

Design, Performance

Keywords

Software agents, multi-agent systems, Java EE

1. INTRODUCTION

Extensible Java EE-based Agent Framework (*XJAF*) [7] is a *FIPA*-compliant [2] multi-agent system (*MAS*). Its main tasks are to provide an efficient runtime environment for its agents, as well as to provide external clients with an easy access to the agent technology. Being implemented in *Java EE*, *XJAF* harnesses many benefits of enterprise-scale applications, such as scalability and runtime load-balancing, security, data integrity, etc.

A recent improvement of *XJAF*, named *SOA-based MAS* (*SOM*) [4], brings a high level of interoperability to the system. By offering its functionalities in form of *web services*, *SOM* can be easily used by a wide variety of external clients. Because it is a specification of web services, their functionalities and interactions, *SOM* can be implemented using many modern programming languages and platforms. The default implementation is still provided in *Java EE*, including the same set of benefits as the original *XJAF*.

SOM is accompanied by an agent-oriented programming language named *Agent Language for SOM* (*ALAS*) [5]. The two main goals of *ALAS* are to provide developers with programming constructs that hide the complexity of agent development, and to support truly heterogeneous agent mobility [6]. During the migration process, the *ALAS* source code of a mobile agent is recompiled, on-the-fly, into the executable code of the target platform. Currently supported platforms include *SOM*, *JADE*, and *PySOM*, a *Python*-based implementation of *SOM*.

This paper presents recent improvement of the *ALAS* compiler that enables the full utilization of *Enterprise JavaBeans* (*EJB*). *SOM* agents are now defined as *stateless session EJB*s, and then passed on to the enterprise application server. In this way, many advanced *EJB* functionalities, such as scalability and object pooling, can be exploited with a minimum programming effort. At the same time, the *ALAS* programming language hides this complexity from the agent developer.

The rest of the paper is organized as follows. Section 2 provides an overview of the *EJB* technology. Recent performance improvements of *SOM* and its agents are described in Section 3. Finally, general conclusions and future research directions are given in Section 4.

2. EJB TECHNOLOGY OVERVIEW

Enterprise JavaBeans (*EJB*) represent one of the most important *Java EE* technologies for developing server-side components. *EJB*s incorporate the business logic of "distributed, transactional, secure and portable" [1] enterprise applications based on *Java EE*. And yet, they are relatively simple to develop and use.

In general, there are two categories of *EJB*s (or, simply, *beans*): *message-driven*, and *session* beans. Message-driven beans are used in the context of *Java Message Service JMS*

BCI'12, September 16–20, 2012, Novi Sad, Serbia.

Local Proceedings also appeared in ISBN 978-86-7031-200-5, Faculty of Sciences, University of Novi Sad.

[3], a *Java EE* technology for asynchronous messaging. In *XJAF* and *SOM*, the agent communication infrastructure is based heavily on exactly message-driven beans and *JMS*.

Session beans can further be categorized into *singleton*, *stateless*, and *stateful*. As its name suggests, there is a single instance of a singleton bean per *Java EE* application. Concurrent access is managed by the *EJB container* and can be fine-tuned by the developer. Stateless beans maintain no conversational state between distinct invocations. They are well-suited for operations that can be executed in a single method call. A stateful bean, on the other hand, is used when the conversational state between different method calls needs to be preserved.

When it comes to the runtime efficiency of *Java EE* applications, stateless session beans offer the best performance. This is because modern enterprise application servers offer the *EJB pooling* technique. The server maintains a pool of pre-initialized stateless beans. Once a client request is made, the server selects an instance from the pool, executes the method, and then returns the bean instance back to the pool. Therefore, an instance is recycled from the pool for each new request, rather than having to be (re)allocated, used, and then deallocated. The pool size is also automatically adjusted to fit the number of concurrent requests. In this way, only a small number of stateless *EJB*s can be used to serve a large number of external clients.

To achieve the best performance, and utilize the object pooling technique, *SOM* agents are based on stateless session *EJB*s.

3. SOM AGENTS

Originally, *SOM* (and its predecessor *XJAF*) featured a single stateless session *EJB* class named *AgentHolder* [4]. When a request for a task execution is made, the system first finds an agent that can solve the required task. Then, it looks-up an instance of *AgentHolder*, passing it the reference to the created agent object (a *Plain Old Java Object* - *POJO*). The agent life-cycle is then managed by the enterprise application server.

This approach was convenient from the point of view of agent developers. They were able to develop agents as *POJO*s, without any knowledge or understanding of *EJB*s and the whole enterprise application architecture. An obvious disadvantage, however, is the extra allocation/deallocation of the agent object – the exact step *EJB* pooling avoids.

With the recent development of *ALAS*, this disadvantage can be alleviated. The language compiler has now been modified to output an *EJB* directly, and thus avoid the process of "packing" a *POJO* inside of the *AgentHolder* bean.

As an example, Listing 1 shows the source code of a simple *PingAgent* written in *ALAS*. The agent exposes a single *service* named *ping* which outputs the received parameter.

Listing 1: *ALAS* source code of *PingAgent*

```
package examples;
agent PingAgent {
  service void ping(String msg) {
    log(msg); } } // outputs the message
```

From the given source code, the *ALAS* compiler produces a stateless session *EJB* shown in Listing 2. The bean, also marked as *local-only* features a single handler for all incoming messages.

Listing 2: Auto-generated *EJB* for *PingAgent*

```
@Stateless @LocalBean
public class PingAgent implements Agent {
 public void onMessage(ACLMessage msg) {
   if (msg.getCommand().equals("Ping")) {
     // unmarshal the actual content
     Ping task = XMLMapper.unmarshallPing(
         message.getContent());
     String param = task.getMsg();
     alas.stdlib.java.common.Log.write(
         param);
     return; } } }
```

This simple example demonstrates how *ALAS* represents a solution that achieves the best runtime performance, while hiding the complexity of enterprise applications from agent developers.

4. CONCLUSIONS AND FUTURE WORK

Enterprise JavaBeans represent one of the most widely-used technologies for developing the server-side business logic of large-scale applications. A category of *EJB*s, known as stateless session *EJB*s, provide the best runtime performance due to advanced object pooling and load-balancing techniques offered by modern enterprise application servers. This paper presents a solution for developing and deploying software agents that are based exactly on stateless session *EJB*s. At the same time, the *ALAS* agent-oriented programming language hides the complexity of enterprise applications from agent developers, allowing them to focus on problem-solving tasks.

Further improvements of the system will be focused on efficient approaches and algorithms for preserving the agent state between distinct running sessions. Additionally, the existing *JMS*-based system will be replaced by a new and recommended technique of asynchronous bean invocation. The goal is to continuously improve the system's performance, while following the modern standards and technologies.

5. ACKNOWLEDGMENTS

This work is partially supported by Ministry of Education and Science of the Republic of Serbia, through project no. OI174023: "Intelligent techniques and their integration into wide-spectrum decision support."

6. REFERENCES

[1] Enterprise JavaBeans technology homepage. `http://www.oracle.com/technetwork/java/javaee/ejb/index.html`. Retrieved on July 15, 2012.

[2] FIPA homepage. `http://www.fipa.org/`. Retrieved on July 15, 2012.

[3] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout. Java Message Service (JMS) specification. `http://www.oracle.com/technetwork/java/jms/index.html`, April 2002. Retrieved on July 15, 2012.

[4] M. Ivanović, M. Vidaković, D. Mitrović, and Z. Budimac. Evolution of Extensible Java EE-Based Agent Framework. In G. Jezic, M. Kusek, N.-T. Nguyen, R. Howlett, and L. Jain, editors, *Agent and Multi-Agent Systems. Technologies and Applications*, volume 7327 of *Lecture Notes in Computer Science*, pages 444–453. Springer Berlin / Heidelberg, 2012.

[5] D. Mitrović, M. Ivanović, and M. Vidaković. Introducing ALAS: a novel agent-oriented programming language. In T. E. Simos, editor, *Proceedings of Symposium on Computer Languages, Implementations, and Tools (SCLIT 2011) held within International Conference on Numerical Analysis and Applied Mathematics (ICNAAM 2011)*, AIP Conf. Proc. 1389, pages 861–864, September 2011. ISBN 978-0-7354-0956-9.

[6] B. J. Overeinder, D. R. A. D. Groot, N. J. E. Wijngaards, and F. M. T. Brazier. Generative mobile agent migration in heterogeneous environments. *Scalable computing: practice and experience*, 7(4):89–99, 2006.

[7] M. Vidaković, B. Milosavljević, Z. Konjović, and G. Sladić. EXtensible Java EE-based agent framework and its application on distributed library catalogues. *Computer science and information systems, ComSIS*, 6(2):1–16, 2009.