Scalable Middleware Environment for Agent-Based Internet Applications]

Benno J. Overeinder[*] and Frances M.T. Brazier

Department of Computer Science, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{bjo,frances}@cs.vu.nl

Abstract. The AgentScape middleware is designed to support deployment of agent-based applications on Internet-scale distributed systems. With the design of AgentScape, three dimensions of scalability are considered: size of distributed system, geographical distance between resources, and number of administrative domains. This paper reports on the AgentScape design requirements and decisions, its architecture, and its components.

1 Introduction

Agent-based Internet applications are by design autonomous, self-planning, and often self-coordinating distributed applications. They rely on the availability of aggregated resources and services on the Internet to perform complex tasks. However, current technology restricts developers options for large-scale deployment: there are problems with the heterogeneity of both computational resources and services, fault tolerance, management of the distributed applications, and geographical distance with implied latency and bandwidth limitation.

To facilitate agent-based applications, a middleware infrastructure is needed to support mobility, security, fault tolerance, distributed resource and service management, and interaction with services. Such middleware makes it possible for agents to perform their tasks, to communicate, to migrate, etc.; but also implements security mechanisms to, for example, sandbox agents to prevent malicious code harm the local machine, or vice versa, protect an agent from tampering by a malicious host.

The AgentScape middleware infrastructure is designed to this purpose. Its multi-layered design provides minimal but sufficient support at each level. The paper presents the AgentScape design and implementation, with a discussion on decisions made, and concludes with future directions.

2 Scalable Multi-Layered Design of Agent Middleware

A number of high-level agent concepts are first introduced to define their role in the paper. The requirements for scalable agent middleware are then discussed, followed by the current software architecture of the AgentScape middleware [6].

[*] This research is supported by the NLnet Foundation, http://www.nlnet.nl/.

2.1 Concepts

Within AgentScape, *locations* exist, *agents* are active entities, and *services* are external software systems accessed by agents hosted by AgentScape middleware. Agents in AgentScape are defined according to the weak notion of agency [7]: (*i*) autonomy: agents control their own processes; (*ii*) social ability: ability to communicate and cooperate; (*iii*) reactiveness: ability to receive information and respond; (*iv*) pro-activeness: ability to take the initiative.

Agents may communicate with other agents and may access services. Agents may migrate from one location to another location. Agents may create and delete agents; agents can start and stop service access. All operations of agents are modulo authorization and security precautions, e.g., an agent is allowed to start a service if it has the appropriate credentials (ownership, authorization, access to resources, etc.).

Mechanisms are provided to make external services available to agents hosted by AgentScape middleware. For example, external services are wrapped and presented as Web services, using SOAP/WSDL generated dynamic interfaces.

A location is a “place” in which agents and services can reside. (see also Fig. 1). More precisely stated, agents and services are supported by agent servers and (Web) service gateways (respectively) which belong to a location. From the perspective of agents, agent servers give exclusive access to the AgentScape middleware. Similarly, service gateways provide access to services external to AgentScape.

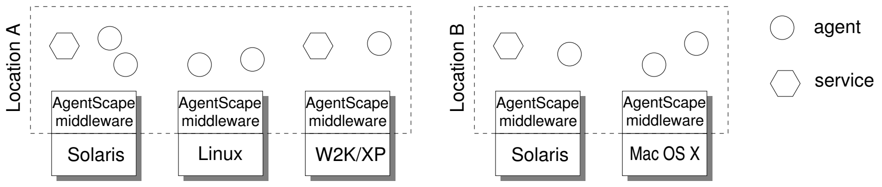

Fig. 1. Conceptual model of the AgentScape middleware environment.

2.2 Requirements

Successful deployment of Internet applications has to take dimensions of scale into account [5]. Applications and systems on Internet-scale networks have to include mechanisms to deal with: (*i*) number of entities in a distributed system: agents (create, delete, migration), resources (add and delete resources to/from distributed platform, allocate resources), lookup service (resolve identifiers of entities to contact address, etc.); (*ii*) geographical size: the distance between different resources (communication latency and bandwidth); (*iii*) number of administrative domains: security mechanisms for authentication and authorization.

The design goals set in the AgentScape project are to provide efficient support for distributed systems that scale along the dimensions outlined above. A distinction is made between agent application level support and agent middleware mechanisms. At the agent application level, functionality to develop and implement scalable applications must be made available. The application programming interface must reflect this, e.g., to allow for latency hiding and mobility, or give access to scalable services in the middleware layer. The agent middleware's task is to implement the functionality provided by the application programming interface and the available services. This implies, amongst others, support for asynchronous communication mechanisms for latency hiding, include scalable services for name resolution (lookup service), and an effective management system that scales with the number of agents and resources.

2.3 Software Architecture

The leading principle in the design of the AgentScape middleware is to develop a minimal but sufficient open agent platform that can be extended to incorporate new functionality or adopt (new) standards into the platform. The multiple code base requirement, e.g., supporting agents and services developed in various languages, makes that language specific solutions or mechanisms cannot be used.

This design principle resulted in a layered agent middleware, with a small *middleware kernel* implementing basic mechanisms and high-level *middleware services* implementing agent platform specific functionality and policies (see Fig. 2). This approach simplifies the design of the AgentScape kernel and makes the kernel less vulnerable to errors or improper functioning. A minimal set of middleware services are agent servers, host managers, and location managers. The current, more extensive set includes a lookup service and a web service gateway.

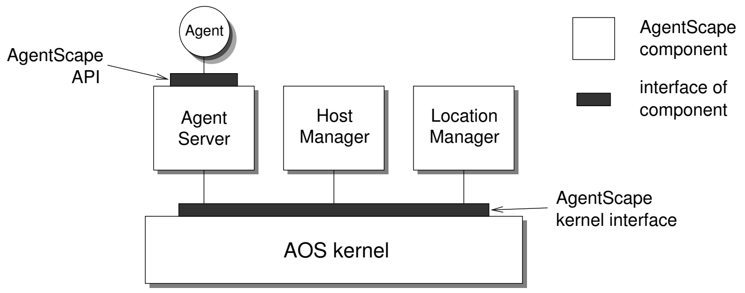

Fig. 2. The AgentScape software architecture.

Minimal Set of Services
Middleware services can interact *only* with the local middleware kernel. That is, all interprocess communication between agent servers and web service gateways is exclusively via their local middleware kernel. The kernel either directly handles the transaction (local operation) or forwards the request messages to the destination AgentScape kernel (remote operation).

The agent server gives an agent access to the AgentScape middleware layer (see also Fig. 2). Multiple code base support in AgentScape is realized by providing different agent servers per code base. For security considerations, it is important to note that an agent is "sandboxed" by an agent server. For Java agents this is realized by the JVM, for Python (or other interpreted scripting languages like Safe-Tcl) by the interpreter, and for C or C++ (binary code) agents are "jailed".

A location manager is the coordinating entity in an AgentScape location (thus managing one or more hosts in one location). The host manager manages and coordinates activities on a host. The host manager acts as the local representative of the location manager, but is also responsible for local (at the host) resource access and management. The policies and mechanisms of the location and host manager infrastructure are based on negotiation and service level agreements [4].

Additional Services
Extensibility and interoperability with other systems are realized by additional middleware services. The web service gateway is a middleware service that provides controlled (by AgentScape middleware) access to SOAP/WSDL web services. Agents can obtain WSDL descriptions in various ways, e.g., via UDDI and can generate a stub from the WSDL document to access a web service. However, stub generation using the Axis WSDL2Java tool is slightly modified so that calls on the web service interface are directed to the AgentScape web service gateway. If access is authorized, the web service gateway performs the operation on the web service and routes the results transparently back to the agent that issued the call.

3 Implementation and Experiences

A number of implementation alternatives have been tested and evaluated, and development still evolves: the AgentScape project aims to provide both a scalable, robust agent platform, and a research vehicle to test and evaluate new ideas. The AgentScape middleware has been tested on different Linux distributions (Debian 3.1, Fedora Core 1, and Mandrake 9.2.1) and Microsoft Windows 2000 (service pack 4).

The current kernel is implemented in the Python programming language, allowing for rapid prototyping. All following middleware services available in the current AgentScape release have been implemented in Java: Java agent server, web service gateway, host manager, and a location manager. The current lookup service, implemented in Python, is a centralized service that basically does its job, but is not scalable and secure. A next generation lookup service is under development: a distributed peer-to-peer lookup service.

4 Related Work

Over the last few years, a number of agent platforms have been designed and implemented. Each design has its own set of design objectives and implementation decisions.

JADE is FIPA compliant agent platform [1]. In JADE, platform and containers are similar concepts as location and hosts in AgentScape. A JADE platform is a distributed agent platform (location), and with one or more containers (hosts). A special front end container listens for incoming messages from other platforms. The AgentScape location manager fulfills a similar function. The front end container is also involved in locating agents on other JADE platforms. In AgentScape this is a different (distributed) service. Mobility in JADE is limited to one platform (intra-platform), whereas in AgentScape, agents can migrate to any location.

Cougaar is a Java-based agent architecture that provides a survivable base on on which to deploy large-scale, robust distributed multi-agent systems [2]. The design goals are scalability, robustness, security, and modularity. Cougaar is not standards' compliant, and messages are encoded using Java object serialization. The lack of standards' compliance was in part due to Cougaar's complex planning language, which does not easily fit into the ACL format, and Cougaar's research focus of a highly scalable and robust system as opposed to interoperability.

DIET is an agent platform that addresses present limitations in terms of adaptability and scalability [3]. It provides an environment for an open, robust, adaptive and scalable agent ecosystem. The minimalistic "less is more" design approach of DIET is similar to AgentScape. The platform is implemented in Java. Mobility is implemented by state transfer only, so no code is migrated. Agents can only migrate if their classes are already in the local classpath of the target machine. Security is not specifically addressed in DIET.

5 Summary and Future Directions

The design rationale for scalability in AgentScape lies in the three dimensions as defined in Section 2.2. The integrated approach to solve the scalability problem is a unique signature of the AgentScape middleware. The small but extensible core of AgentScape allows for interoperability with other open systems.

Future directions in the AgentScape project are a new middleware kernel developed in Java and completion of the implementation of security model. Agent servers for binary agents (C and C++) and Python agents are being considered. A new decentralized lookup service based on distributed hash tables is also being developed.

References

1. F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software: Practice and Experience*, 31(2):103–128, 2001.
2. A. Helsinger, M. Thome, and T. Wright. Cougaar: A scalable, distributed multi-agent architecture. In *Proceedings of the International Conference on Systems, Man and Cybernetics (IEEE SMC 2004)*, The Hague, The Netherlands, October 2004.
3. C. Hoile, F. Wang, E. Bonsma, and P. Marrow. Core specification and experiments in DIET: A decentralised ecosystem-inspired mobile agent system. In *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2002)*, pages 623–630, Bologna, Italy, July 2002.
4. D.G.A. Mobach, B.J. Overeinder, O. Marin, and F.M.T. Brazier. Lease-based decentralized resource management in open multi-agent systems. In *Proceedings of the 18th International FLAIRS Conference*, Clearwater Beach, FL, May 2005.
5. B.C. Neuman. Scale in distributed systems. In T. Casavant and M. Singhal, editors, *Readings in Distributed Computing Systems*, pages 463–489. IEEE Computer Society Press, Los Alamitos, CA, 1994.
6. N.J.E. Wijngaards, B.J. Overeinder, M. van Steen, and F.M.T. Brazier. Supporting Internet-scale multi-agent systems. *Data Knowledge Engineering*, 41(2–3):229–245, 2002.
7. M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.