Lease-based Decentralized Resource Management in Open Multi-Agent Systems

D.G.A. Mobach, B.J. Overeinder, O. Marin*, and F.M.T. Brazier
IIDS Group, Department of Computer Science
Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{mobach,bjo,omarin,frances}@cs.vu.nl

Abstract

Internet-scale, open, distributed mobile agent systems need to be supported by agent platforms that are able to support large numbers of heterogeneous agents and resources. Interfaces must be well-defined: for distributed agent applications to be able to access resources and for resource providers to be able to specify access and usage policies. This paper presents a distributed management architecture that implements agent-initiated resource access and in which leases play a central role.

1. Introduction

Internet-scale, open, distributed mobile agent systems need to be supported by agent platforms that are able to support large numbers of heterogeneous agents and resources. Agents themselves are responsible for obtaining and maintaining access to resources in such systems. This paper presents a contract-based approach to resource access, in which agents negotiate contracts with resource providers to acquire time-limited access to resources. A middleware management architecture is responsible for distributing and enforcing the contracts issued, so-called leases. This management architecture has been implemented and tested in AgentScape, a framework for supporting large-scale multi-agent platforms.

2. Management in AgentScape

This section briefly introduces AgentScape and resource management within AgentScape.

2.1. AgentScape

AgentScape [14] is a distributed middleware framework designed to support large-scale mobile agent systems. The design objectives of AgentScape are to provide a scalable, secure platform for multi-agent systems, supporting multiple code bases, capable of running on different operating systems, and interoperability with other agent platforms. At the base of the middleware is the AgentScape kernel, that supports interaction between AgentScape middleware processes and different operating systems. Agents are supported by agent servers, and services are made available through service access providers. Additionally, each host runs a host manager, which enforces management policies and performs management tasks.

An AgentScape location consists of a number of kernels running on the participating hosts, and a number of agent servers, service access providers, and host managers running on these hosts. Additionally, a location manager runs within the location, to apply location-wide management policies, and to perform management tasks covering multiple hosts or involving remote locations. Locations interact with each other when needed, e.g., to enable agent communication, service interaction, and agent migration.

Agents interact with the middleware through an interface to an agent server. This interface supplies a number of calls allowing agents to: send/receive messages to/from other agents; perform life cycle operations (kill/suspend/migrate) on itself; creating new agents; interact with services.

2.2. Resource Management in AgentScape

Large-scale multi-agent platforms consist of numerous hosts spread across the globe, which in turn support large numbers of agents. Not only are agents created and removed frequently throughout such a system, agents also have the ability to move from one place to the next when performing their tasks. In addition to this dynamic agent population, complexity is added by the fact that hosts supporting
the agents are heterogeneous in terms of hardware and operating systems. A management system suited for this domain should be able to handle a diverse and constantly changing agent population, as well as a broad range of different resources available to the agents.

**Design Objectives**

To prevent a management system from becoming a bottleneck for the system it is managing, and to ensure that the management system is able to handle a large number of agents, a management architecture should be built to scale. For this reason AgentScape’s management architecture is strongly decentralized: there is no global management. Each location is responsible for managing its own resources. Management communication between locations is limited. Basic resource information is exchanged if necessary, and direct contact between location managers only takes place when needed (for example, for agent migration).

To allow a management system to perform tasks ranging from lightweight management on the one hand, to intensive complex management on the other hand, a flexible architecture is necessary. Management overhead should be related to the complexity of the performed management tasks. Also, to allow the future addition of functionality or management model changes, a management architecture should be modular in nature, with clear interface definitions between the components. AgentScape’s management components offer clear separation of location-wide management functionality from local host management functionality and management mechanisms, to allow for modification and changes on each of these areas separately.

Other important elements of management system design are the security and fault-tolerance of both the management system itself, and that of the managed system. If an agent management system is not supported by a sound security model for authentication, authorization, communication and migration, the system could be at risk. The management system implements fine-grain authorization mechanisms currently being implemented in the AgentScape middleware.

**AgentScape’s Management Architecture**

The overall goal of the AgentScape management architecture is the provision of management facilities for security, fault-tolerance, performance, configuration, and accounting. These facilities include support for specifications of policies and mechanisms to enforce policies. From an operational perspective, the management mechanisms provide the basis upon which policies can be specified.

One of the most important mechanisms used by management tasks is resource access and usage regulation. This mechanism allows a management system to implement policies limiting agent access to resources, for example, to ensure fair use of a resource, to limit resource access to specific agents according to specific security policies, or to enable detailed accounting of resource usage by agents.

In AgentScape, agents only have access to resources provided within the location in which they are running. As a consequence, resource usage policies are at most defined and applied location-wide. Figure 1 shows a schematic view of three hosts within a location. Each host runs a number of AgentScape middleware components, including a host manager process. One of the hosts also runs the location manager process.

The characteristics of resources differ, for example, communication related resources (bandwidth, delay, etc.), agent server configurations, agent life-cycle related resources (time-to-live, suspending options, etc.), and external resources made available to agents through service access providers.

In AgentScape, agents engage in contract negotiations with resource providers through the middleware (see Fig. 2). The contracts issued are called leases.

The resulting contracts issued to agents are time-based, and are therefore referred to as leases in the remainder of
the text. Restricting the duration of a resource contract defers responsibility of resource (re-)acquisition to resource consumers, and allows the management architecture to re-evaluate contract content upon each new request. The following section introduces the AgentScape resource leasing infrastructure for enforcing management policies regarding resource access regulation.

3. AgentScape’s Lease Model

The lease descriptions and lease interactions are specified in the description language and negotiation interactions of the Web Services Agreement Specification (WS-Agreement) [1]. This specification is currently under development by the Global Grid Forum’s Grid Resource Allocation and Agreement Protocol Working Group. The specification defines a language for specifying agreements between resource providers and consumers, and a protocol for establishing these agreements. The use of this specification gives agents a uniform and well-defined interaction model and enables agents to be more expressive in presenting resource requirements.

The AgentScape leasing infrastructure (i) provides agents with the ability to express resource requirements and request resource access to the underlying middleware, and (ii) allows the middleware to express resource usage conditions and limitations to the agents using the resources. As such, the leasing infrastructure is used to control resource usage by agents within a location. Leases are valid for a period of time and agents are responsible for requesting and renewing leases when access to middleware resources is required.

The leasing protocol is based on the protocol described in the WS-Agreement specification mentioned earlier. As a first step, an agent requests a lease template from an AgentScape location. Each template contains information about which resources can be leased under which conditions. An agent can then use the template as a basis for a lease request, which it then, in turn, sends to the target location. The location manager of the target location receives the lease request. Location-wide leasing policies determine to which host managers the lease request is forwarded. Each host manager determines which lease it offers, depending on its own leasing policies. The leases proposed by the hosts are then compared by the location manager, a choice of the most appropriate host is made, and a description of this lease is returned to the agent. The agent can then choose to accept the offer if the proposed resources are within acceptable boundaries. Figure 3 shows the leasing interaction.

4. Communication Resource Leasing

An experiment has been performed to examine the applicability of leases as a method for managing access to a typical agent platform resource: Agent communication. The experiment demonstrates the use of leases to express and apply fine-grained resource access policies to enable communication management on a per-agent basis. The experiment is done with a relatively small number of agents with differentiated behavior, to explore the influence of the lease-based interaction model on the performance of the system.

In this experiment, communication is managed by requesting and granting leases for communication bandwidth. In AgentScape, agents are sandboxed in an agent server, and can send and receive messages only through the AgentScape interface supplied by the agent server to the agent. The agent server receives messages sent by the agents it hosts, and ensures that the messages are delivered to the agent servers that host the destination agents.

4.1. Experiment Description

The leasing policy defined in the experiment limits the total bandwidth used by the agent server, and determines the individual (per agent) bandwidth allocation. With a lease request from an agent, the policy tries to fulfill the requested amount, with the restriction that there is no oversubscription, i.e., that the total requested amount does not exceed the agent server limit. If the resource is oversubscribed, lease requests are granted a ‘fair’ fraction of the requested resource amount. A fair fraction implies that small consumers are granted a larger fraction of their requested amount than large resource consumers. The policy of fair ratio adapts to the dynamical change in number of agents.
As more agents access the same resource, the granted fraction to the resource decreases for all individual agents; and vice versa, as less agents access the same resource, the assigned fraction increases dynamically. The policy of fair ratio is similar to Shortest Remaining Processing Time (SRPT) scheduling policy which has been known to be optimal for minimizing mean response time [2].

The experiment is done with four agents running on the same agent server of one AgentScape location, coupled to four other agents running at another agent server at another AgentScape location. The agents have different communication profiles, expressed in the different message sizes (payload) used. After receiving a message, an agent sends back an acknowledgment to the message sending agent, allowing the message sending agent to determine average response times. This average response time is used as a metric to quantify the effect of communication resource leasing.

The experiments have been conducted on two Linux PCs (Pentium III, 1.2 GHz), connected by switched FastEthernet (100 Mb/s). The experiments were run on a non-dedicated infrastructure. The PCs and network are under normal laboratory activity.

4.2. Limited Bandwidth Experiment

In the first experiment series, the agents did not use leasing. The results of this experiment can be considered as reference results where no arbitration or quality of service management is implemented.

The agent server hosting the sending agents was limited to approximately 100 000 bytes per second, and the four sending agents were assigned message payload sizes of respectively 10 000, 25 000, 40 000, and 50 000 bytes. In the experiment, each agent sends 100 messages using the payload specified for each of the agents, and measures the average response time. To introduce concurrency between the agents and guarantee some random ordering of the messages sent by the agents, each agent uses an exponential distribution with a mean of 1 second to generate random waiting time intervals between each message.

Figure 4 shows the average results of ten repeated experiments. The bars in the figure are the standard deviation of the average values. All four agents measured approximately the same response time. Agents sending relative small messages of 10 000 bytes experience the same response delay as agents sending large messages of 50 000 bytes.

4.3. Bandwidth Leasing Experiment

In the leasing experiment, agents first obtain a lease before sending messages to their counterpart agent. At the beginning of the experiment, each of the four agents requests a lease for the communication resource. In this request, the desired bandwidth is indicated by the agent. In the experiment, each agent uses its specified payload size as the desired amount, thus 10 000, . . . , 50 000 bytes/second respectively.

The lease module in the agent server applies the policy, and either assigns the amount desired by the agent to the lease, or a fair fraction as dictated by the policy when the communication resource is oversubscribed. In this experiment, lease duration was fixed at 2 seconds per lease, and the maximum total amount which could be leased was set to 100 000 bytes per second. When the expiration time of a lease is reached, any new messages arriving using this lease are denied. This results in an agent having to request a new lease.

Figure 5 depicts the average results of 10 experiments. As can be seen, the use of leases allows the agents with

![Figure 4. Average response times (ms) with agent server restricted to 100 000 bytes/second.](image)

![Figure 5. Average response times (ms) with leasing](image)
smaller payload settings to obtain better response times, at the expense of a longer response time for the agents with larger payloads. Effectively, agents with smaller resource demands experience better quality of service than agents putting heavy demands on available resources. In this experiment, enforcement of the communication limit was implemented using a simple buffering mechanism. Additionally, in this experiment, a fairly simple lease policy was used. A more effective policy could be applied, for example by incorporating information about the resource usage history of agents with regard to the desired resource.

5. Related Work

Controlling resource access by agents is also addressed in other agent platforms. This section gives a brief overview of a number of current agent platforms and their resource management facilities, and discusses the use of the leasing concept in other distributed architectures.

In the D’Agents [7] multi-agent system, market-based resource control is achieved by having agents acquire computational resource access rights through a market. Resource control in the NOMADS [8] multi-agent system is based upon a Java-compatible Virtual Machine (Aroma), which implements mechanisms for fine-grained resource control of disk and network resources [11]. In NOMADS, dynamically adaptable limits can be set on the rate and quantity of resources used by agents. The Java-based J-SEAL2 mobile agent kernel [4] provides a hierarchical resource control model, allowing for the control of CPU usage, memory, and the number of threads and subprocesses. Java byte-code rewriting techniques are used to ensure portability and transparency. Both NOMADS and J-SEAL2 concentrate on offering resource control mechanisms, but do not specify an architecture to enable agents to interact with the management system. In the Ajanta mobile agent system [12], a dynamic proxy-based approach is used to control access to application-defined resources by agents. Each agent is given a customized proxy to access a resource, which implements access restrictions to the proxied resource. The FIPA [10] and JADE [3] agent platform specifications do not specify a resource interaction model to enable management of resource access by agents running on the platform.

Traditionally, leases are used in distributed file/object systems to maintain cached file consistency. Clients acquire leases for objects, which guarantee that a client is notified of any updates to the object during the period for which the lease is valid. Responsibility for renewing the lease is left up to the client, thus eliminating the bookkeeping overhead this would otherwise cause for the management subsystem. Extensions of this lease model have been made, such as volume leases [15], which enable issuing of leases for groups of objects to further reduce lease overhead, asynchronous leases [5], which introduce the notion of logical time instead of real-time, and cooperative leases [9], in which the notification of object updates is generalized, and leases can be issued to groups of clients.

The concept of leasing is also used in the area of distributed application frameworks, for example in Jini [13]. Leases are used within the Java-based Jini framework for distributed garbage collection: Clients lease resource access (such as for example service registration within a lookup service). The acquired lease allows a client to make of use of that resource for a limited time-period. When a lease expires, and no explicit renewal is requested by the client (for example because of network failure), the associated resource is made available for other clients, preventing unnecessary resource allocation. The Jini specification does not cover a negotiation model or protocol specification. In the SHARP [6] architecture, tickets (soft resource claims) can be redeemed by resource consumers for leases (hard resource claims), which guarantee access to a resource. Ticket holders can delegate resources to other principals by issuing new tickets.

6. Discussion and future work

Leasing can be used to control resource usage for resources that require regulation, in an agent-driven manner. It gives agents the ability to explicitly express resource requirements. The lease mechanism gives a management system responsible for regulating resource access a flexible way to allocate resource access. The location-centered management architecture described in this paper uses leasing to apply management policies in the form of leases and leasing policies. Agents are responsible for obtaining and keeping required resource access, while lease expiry times ensure that ultimate control over resource access is kept in the hands of the management system.

The leasing architecture is a subsystem of a larger management architecture within AgentScape, and can be used as a method for uniformly applying management policies to enable higher level management tasks in functional areas requiring resource access regulation, such as: accounting, security management, and performance management.

Acknowledgments

This research is supported by the NLnet Foundation, http://www.nlnet.nl/. The authors thank Etienne Posthumus, Niek Wijngaards, and Patrick Verkaik for their valuable discussions and comments.
References


