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

A distributed management architecture is proposed for Internet-scale, open, distributed agent middleware systems. The management architecture presented supports the autonomy of both agents and middleware resources, incorporating an agent-initiated contract negotiation model for resource allocation and access. A leasing mechanism infrastructure designed and implemented for this purpose is presented.

Introduction

Mobile agents in Internet-scale, open distributed systems need to be able to access resources at different locations. These resources are heterogeneous in many different ways: different types of hardware, running different operating systems, connected by different types of networks, administered by different owners. Different systems have different access policies, and different interfaces. Agent platforms are designed to mediate between agents and resources, providing a uniform interface for agents to access these resources.

Managing agent resource access within the middleware is not a straightforward task. Resource requirements and usage conditions need to be specified by both agents and the systems on which they are hosted. A framework should be uniform and standardized, allowing heterogeneous agent applications to access distributed resources in a uniform and coherent manner.

This paper presents a contract negotiation model, in which agents acquire time-limited access to (possibly distributed) resources, specified in contracts, called leases. A middleware management architecture is responsible for distributing and enforcing these leases. A management architecture based on leases has been implemented in AgentScape, a framework for supporting large-scale multi-agent platforms.

Design Objectives and Requirements

The design of a management system for Internet-scale, open distributed mobile agent systems begins with a number of design objectives and requirements (Wijngaards *et al.* 2002; Overeinder & Brazier 2004). A distributed agent platform consists of numerous hosts at geographically distributed locations, each capable of supporting a large number of possibly mobile agents each with their own *requirements* with respect to the resources they need to access. These requirements can vary from functional (for example, the availability of a specific service, database, library) to hardware related (for example, required CPU type or performance, memory size, or available bandwidth). From the agent perspective these requirements are the basis for agent placement upon its creation, and agent migration.

From the perspective of the resource providers other restrictions may hold. The owner of a resource specifies the conditions under which an agent is allowed to make use of one or more specific resources, as policies. For example, a policy can define that only agents from affiliated research institutes will be given access to a host's resources, or that all agents given permission to access a system's resources will only be allowed to use a certain fraction of the available bandwidth (e.g., all agents are assigned at most 1 MB/s). A management system compares each agent's requirements with its resource policies to decide whether an agent is to be accepted on a system, and if so, the extent to which resources will be made available.

A management mechanism infrastructure must be scalable in both number of agents and the number of hosts and it must honour the autonomy of agents and resources. A central solution will not work. As a result, a resource negotiation infrastructure needs to provide mechanisms to negotiate the terms of resource usage between the agents and the resources locally, and to finalize the negotiation in a contract specifying the agreed resources usage at this level (see Fig. 1).

AgentScape Management Architecture

The AgentScape (Overeinder & Brazier 2004) management architecture reflects the design objectives of a scalable infrastructure for contract negotiation between agent requirements and resource policies. The management infrastructure is decentralized and is organized around the concepts of hosts and locations.

An AgentScape *location* is an aggregation of hosts within an administrative domain. Each host is represented by a *host manager*. A host manager is responsible for implementing the resource negotiation functionality on a host, enabling

*Current affiliation LIP6, Paris, olivier.marin@lip6.fr

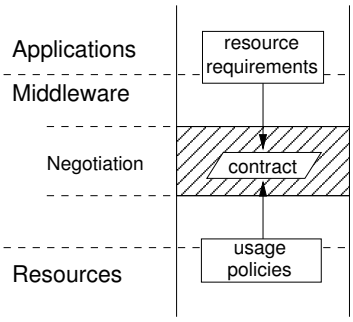

Figure 1: Abstract resource negotiation and contract model.

the leasing of resources made available by the other middleware services running on the host, e.g., agent servers and a web service gateway. A location is represented by a *location manager*. A location manager implements the resource management functionality on a location-wide level. Hosts can dynamically join and leave a location. Host managers inform a location manager about their availability via a heartbeat mechanism. Figure 2 shows an example of an AgentScape location.

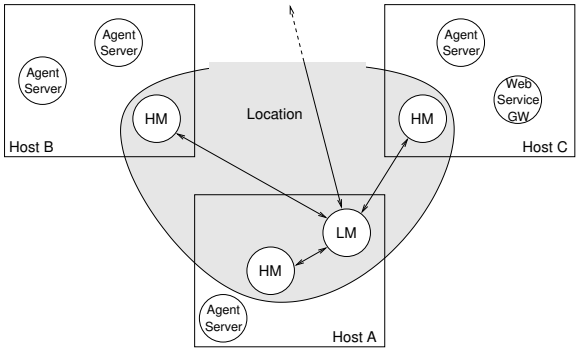

Figure 2: Management components within an AgentScape location. (LM = location manager, HM = host manager, GW = gateway)

Contract negotiation within AgentScape takes place at two levels: between agents and location managers, and between location managers and host managers within a location. The two tier contract negotiation model is in line with the scalable middleware architecture design objectives: agents are not concerned with finding and negotiating with particular hosts within a location, and location managers only engage in negotiation with hosts than can support the requested resources. A standardized format for contract specification provides different resource owners with a means to express usage conditions in a common format, and allows agent applications to access heterogeneous resources in a coherent manner.

Contracts issued to agents are time-based, and are referred to as *leases* in the remainder of the text. Restricting the duration of a resource contract defers responsibility of resource (re-)acquisition to the agents, and allows the management architecture to re-evaluate contract content upon each new request.

AgentScape's Lease Model

The AgentScape leasing infrastructure (i) provides agents with the ability to request resource access, and (ii) allows the middleware to express resource usage conditions and limitations in the form of leases. Agents are responsible for requesting and renewing time-constrained leases to access middleware resources. Within the model, a location manager is responsible for:

- Maintaining information about the resources available on the hosts within the location.
- Enabling agent applications to acquire resource access by negotiation.
- Negotiating with the individual hosts to acquire lease proposals for agent applications.
- Maintaining information about the created leases within the location.
- Enforcing location-wide resource usage and access policies.

Host Managers are responsible for:

- Enabling location managers to acquire lease offers for host resources by negotiation.
- Enforcing resource usage and access policies on the host.
- Monitoring and controlling resource usage on the host, according to the created leases.

The *Web Services Agreement Specification* (WS-Agreement) (Andrieux *et al.* 2004) defines the format used to specify lease descriptions and lease interactions.[1] The specification defines an XML-based language for specifying agreements between resource providers and consumers, and a protocol for establishing these agreements. Agreement *terms* are used to describe the (levels of) service negotiated. Two types of terms are distinguished for agreement specifications: (i) Service Description Terms, describing the services to be delivered under the agreement, and (ii) Guarantee Terms, expressing the assurances on service quality (e.g., minimum bounds) for the services described in the service description terms. The specification of domain-specific term languages is explicitly left open. The AgentScape management architecture defines these terms for resources in the AgentScape middleware.

The WS-Agreement interaction model defines that consumers can create agreements *offers* based on available agreements *templates*, which, if accepted, result in new agreements (these offers are called requests in the context of this paper). Agreement status can be monitored at run-time, to monitor agreement compliance. The following list shows the lease related calls which are based on the web service port types specified in the WS-Agreement specification:

[1]This specification is currently under development by the Global Grid Forum's Grid Resource Allocation and Agreement Protocol Working Group.

- *requestTemplates(): template-list*
  Request the available lease templates.
- *requestLease(LeaseOffer): lease*
  Request a lease based on the supplied lease offer.
- *acceptLease(LeaseID)*
  Accept a lease.
- *requestLeaseStatus(LeaseID): lease*
  Request the current status of a lease. Returns a lease document, including the current status of each term.

To enable an agent to contact locations and engage in lease negotiation, the four calls are available to agents via the AgentScape API. Each call has been extended with an argument allowing agents to send requests to a specific location.

- *requestTemplates(LocationID)*
- *requestLease(LocationID, LeaseOffer)*
- *acceptLease(LocationID, LeaseID)*
- *requestLeaseStatus(LocationID, LeaseID)*

Resource Leasing Scenario

This section discusses an agent migration scenario demonstrating the use of resource leases within AgentScape, using the WS-Agreement specification language introduced above. In this scenario, an agent has obtained a list of potential destination locations from a local directory service, and engages into negotiation with a location to which it wishes to migrate and to obtain leases for the resources it requires.

As a first step, the agent requests the target location's lease templates from the target location (see Figure 3).

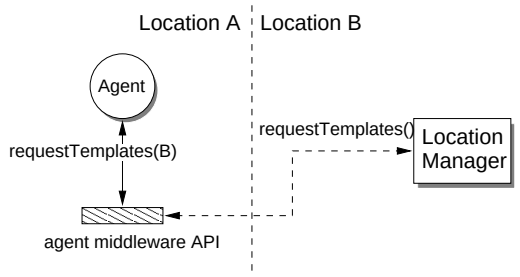

Figure 3: An agent sends a template request to location B.

The location manager receives the template request, and returns the available templates to the agent. These templates are based on templates of the hosts in the location, and are gathered in advance by contacting the host managers and requesting their templates. The agent receives a template, specifying which resources are available for leasing at the target location, and possible constraints (see Example 1).

The resources available for leasing at the target location in this scenario are: `time-to-live`, representing the number of seconds an agent may reside at the location, and `communication`, representing the communication bandwidth an agent may use for agent-agent communication.

The agent creates a lease request based on the template, specifying the required resources and associated quantities,

```
<wsag:Template>
  <wsag:Name>Template1</wsag:Name>
  <wsag:Context/>
  <wsag:Terms/>
  <wsag:CreationConstraints>
    <wsag:Item>
      <wsag:Location>//wsag:ServiceDescriptionTerm//
agentscape:timeToLive</wsag:Location>
      <xs:maxInclusive xs:value="5000">
      <!-- max value of 5000 (seconds) -->
    </wsag:Item>
    </wsag:Item>
      <wsag:Location>//wsag:ServiceDescriptionTerm//
agentscape:communication</wsag:Location>
      <!-- term allowed, no further constraints -->
    </wsag:Item>
  </wsag:CreationConstraints>
</wsag:Template>
```

Example 1: A lease template.

and sends it to the target location. In this scenario, the agent requests a time-to-live on the location of 2000 seconds, and a minimum communication bandwidth of 50 Kilobytes per second (see Example 2).

```
<wsag:AgreementOffer>
  <wsag:Name>Offer1</wsag:name>
  <wsag:Context>
    <wsag:AgreementInitiator>
      agentX
    </wsag:AgreementInitiator>
    <wsag:TemplateName>Template1</wsag:TemplateName>
  </wsag:Context>
  <wsag:Terms>
  <wsag:All>
    <wsag:ServiceDescriptionTerm wsag:Name="TimeToLive"
                        wsag:ServiceName="LocationY">
      <!-- requirement: 2000 seconds running time-->
      <agentscape:timeToLive>2000</agentscape:timeToLive>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceDescriptionTerm wsag:Name="Communication"
                        wsag:ServiceName="LocationY">
      <!-- requirement: 50 Kb/s bandwidth -->
      <agentscape:communication>
        <agentscape:minBandWidth>
          51200
        </agentscape:minBandWidth>
      </agentscape:communication>
    </wsag:ServiceDescriptionTerm>
  </wsag:All>
  </wsag:Terms>
</wsag:AgreementOffer>
```

Example 2: Agent lease request.

The location manager of the target location receives the request, and determines which hosts to contact for the required leases, i.e., which hosts support leasing of both resources. The location manager sends these hosts a lease request based on the initial request of the agent (see Fig. 4).

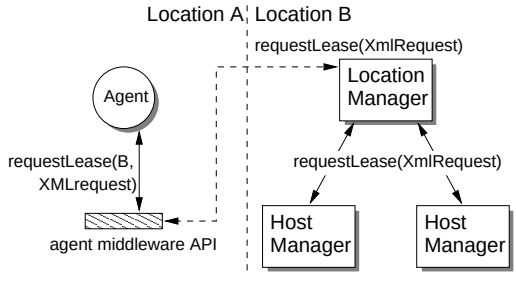

Figure 4: A lease request is sent to location B. Multiple hosts are sent a lease request by the location manager.

The two hosts each receive and evaluate the lease request. Monitoring information is examined to determine current resource availability, and management policies are checked to determine if the request may be processed. In this scenario, the two hosts each respond with a lease proposal, but the hosts cannot meet the required communication bandwidth. Instead, Host 1 proposes a lower value of 15 KB/s for the minimum communication bandwidth (see Example 3). Host 2 offers a value of 30 KB/s.

```
<-- Host 1 Terms -->
<wsag:Terms>
  <wsag:All>
    <wsag:ServiceDescriptionTerm wsag:Name="TimeToLive"
                          wsag:ServiceName="LocationY">
      <agentscape:timeToLive>2000</agentscape:timeToLive>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceDescriptionTerm wsag:Name="Communication"
                          wsag:ServiceName="LocationY">
      <agentscape:communication>
        <agentscape:minBandWidth>
          15360
        </agentscape:minBandWidth>
      </agentscape:communication>
    </wsag:ServiceDescriptionTerm>
  </wsag:All>
</wsag:Terms>

<-- Host 2 Terms -->
<wsag:Terms>
  <wsag:All>
    <wsag:ServiceDescriptionTerm wsag:Name="TimeToLive"
                          wsag:ServiceName="LocationY">
      <agentscape:timeToLive>2000</agentscape:timeToLive>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceDescriptionTerm wsag:Name="Communication"
                          wsag:ServiceName="LocationY">
      <agentscape:communication>
        <agentscape:minBandWidth>
          30720
        </agentscape:minBandWidth>
      </agentscape:communication>
    </wsag:ServiceDescriptionTerm>
  </wsag:All>
</wsag:Terms>
...
```

Example 3: Terms proposed by Hosts 1 and 2.

The location manager receives the lease proposals of the hosts, and determines, based on the request by the agent as well as policy information, which lease proposal it accepts. In this example, Host 2 is selected as its offer is closest to the request made by the agent. The hosts are informed of the decision, and the accepted lease is used to create a lease which is sent back to the agent (see Example 4). The offer sent to the agent is time-limited, to ensure that if the agent does not accept the offer, resources are freed when the time-period expires. In this scenario, the agent accepts the lease, and migrates to the location.

Communication Leasing Experiments

To test the application of the leasing model in the AgentScape architecture, a small-scale experiment has been performed. In this experiment, communication is managed by requesting and granting leases for communication bandwidth. The aim of the experiment is to show the effectiveness of the leasing mechanism in realizing quality of service policies for different types of communication.

The leasing policy defined in the experiment limits the total bandwidth used by the agent server, and determines the individual (per agent) bandwidth allocation. Having re-

```
<wsag:Terms>
  <wsag:All>
    <wsag:ServiceDescriptionTerm wsag:Name="TimeToLive"
                          wsag:ServiceName="LocationY">
      <agentscape:timeToLive>2000</agentscape:timeToLive>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceDescriptionTerm wsag:Name="Communication"
                          wsag:ServiceName="LocationY">
      <agentscape:communication>
        <agentscape:minBandWidth>
          30720
</agentscape:minBandWidth>
      </agentscape:communication>
    </wsag:ServiceDescriptionTerm>
  </wsag:All>
</wsag:Terms>
```

Example 4: Final lease accepted by agent.

ceived a lease request from an agent, the management system tries to fulfill the request. If a 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 fewer agents access the same resource, the assigned fraction increases dynamically. The policy of fair ratio has great similarities with the Shortest Remaining Processing Time (SRPT) scheduling policy which has been known to be optimal for minimizing the mean response time (Bansal & Harchol-Balter 2001).

The experiment includes a small number of agents with differentiated behaviour, to demonstrate the influence of the lease-based interaction model in a comprehensible and controlled manner. Four agents on an agent server at one AgentScape location, communicate with four other agents running at another AgentScape location on a 1-1 basis. The agents have different communication profiles, materialized in different message sizes (payload) that are sent to their counterpart agents. After receiving a message, a counterpart agent sends back an acknowledgment, allowing an agent to determine average response time. 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.

Limited bandwidth experiment

The first experiment series is without leasing. The results of this experiment are reference results where no arbitration or quality of service management is implemented.

The outgoing bandwidth of 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 specified payload, and measures the average response time.

Figure 5 shows the average results of ten repeated exper-

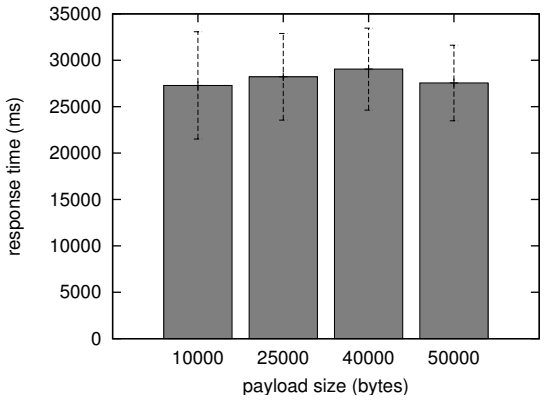

Figure 5: Average response times (ms) with agent server restricted to 100 000 bytes/second.

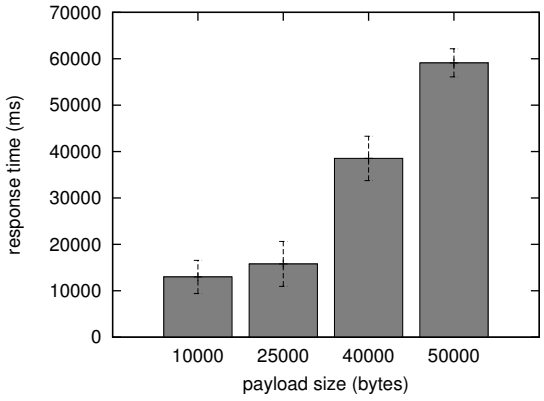

Figure 6: Average response times (ms) with leasing.

iments. The bars in the figure are the standard deviation of the average values. All four agents measured approximately the same response time. Agents sending relatively small messages of 10 000 bytes experience the same response delay as agents sending large messages of 50 000 bytes.

Bandwidth leasing experiment

In the leasing experiment, agents first obtain a lease before sending messages to the other agents. At the beginning of the experiment, each of the four agents requests a lease for the communication resource, including the desired bandwidth. 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 is fixed at 2 seconds per lease, and the maximum total amount which can be leased was set to 100 000 bytes per second. When the expiration time of a lease has been reached, new messages are denied. An agent may always request a new lease.

Figure 6 depicts the average results of ten experiments. As shown, the use of leases allows the agents with 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.

Related Work

This section discusses 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 (Gray *et al.* 2002) multi-agent system, agents acquire computational resource access rights through a market. In the NOMADS (Suri *et al.* 2000) multi-agent system fine-grained resource control of disk and network access (rate and quantity) is achieved using mechanisms provided by the Java-compatible Virtual Machine (Aroma). The Java-based J-SEAL2 mobile agent kernel (Binder, Hulaas, & Villaz 2001) provides a hierarchical resource control model, allowing for the control of CPU usage, memory, and the number of threads and subprocesses. Both NOMADS and J-SEAL2 offer resource control mechanisms, but do not specify an architecture to enable agents to interact with the management system. In the Ajanta mobile agent system (Tripathi *et al.* 1999), a dynamic proxy-based approach is used to control access to application-defined resources by agents. The FIPA agent platform specification (Poslad, Buckle, & Hadingham 2000) does not address management of resources, neither does the FIPA compliant JADE framework (Bellifemine, Poggi, & Rimassa 2001).

Traditionally, leases are used in distributed file/object systems for maintaining cached file consistency. More recently, the concept of leasing has been used in the area of distributed application frameworks, for example in Jini (Waldo 1999), where leases are used for distributed garbage collection. In the Jini framework, 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 include a negotiation model or protocol specification. In the SHARP (Fu *et al.* 2003) 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. The goals of the SHARP architecture and the AgentScape management architecture are similar in nature, with the AgentScape management architecture being more oriented towards the agent middleware domain.

In theoretical agent research, much work has been performed in the area of contracting for agents. Well-defined negotiation frameworks, such as (Sandholm & Lesser 1995), support agents with limited computational abilities in dynamic and real-time environments. To limit the resources required for negotiation, different types are distin-

guished, including variable levels of commitment, e.g. low-commitment negotiation with several agents simultaneously. The resource management framework presented in this paper currently uses a straightforward negotiation protocol, but could benefit from this and other theoretical frameworks offering robust and flexible negotiation facilities.

Discussion and Future Work

This paper presents an agent-initiated contract negotiation model that uses a lease-based mechanism infrastructure to match agent resource requirements with resource usage policies. Agents are responsible for obtaining and keeping required resource access, while lease expiration times ensure that ultimate control over resource access is kept in the hands of the management system.

The leasing infrastructure is a subsystem of a larger management architecture within the AgentScape framework, making resource access regulation, such as: accounting, security management, and performance management possible.

As the proposed management infrastructure is intended for deployment in large-scale distributed environments, fault-tolerance is an important aspect for the components of the management system. In particular, the location manager in the management architecture should be protected against failure, as failure could lead to the unavailability of resources within an entire AgentScape location. Failures of host manager processes are less problematic, as this only leads to the unavailability of resources on a single host. An approach to this problem currently being researched involves replication (either active or passive) of location managers, enabling a failing location manager to be replaced by a replica residing on another host within the location.

The leasing infrastructure can also be used for non-agent applications, e.g., for resource allocation in Grid platforms. Static agents are responsible for contract negotiation for resource allocation and access. Upon completion of the negotiation phase, a parallel job is sent to the remote resource(s) in the Grid platform for execution. In this scenario, the static agents can be considered to be part of the Grid management system.

The current foci of this research are: Specification of (more complex) resource usage policies, and contract negotiation between agents and resources.

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

Andrieux, A.; Czajkowski, K.; Dan, A.; Keahey, K.; Ludwig, H.; Pruyne, J.; Rofrano, J.; Tuecke, S.; and Xu, M. 2004. Web Services Agreement Specification WS-Agreement (draft).

Bansal, N., and Harchol-Balter, M. 2001. Analysis of SRPT Scheduling: Investigating Unfairness. In *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 279–290.

Bellifemine, F.; Poggi, A.; and Rimassa, G. 2001. Developing Multi-agent Systems with a FIPA-compliant Agent Framework. *Software – Practice and Experience* 31(2):103–128.

Binder, W.; Hulaas, J. G.; and Villaz, A. 2001. Portable Resource Control in the J-SEAL2 Mobile Agent System. In *Proceedings of the fifth international conference on Autonomous agents*, 222–223.

Fu, Y.; Chase, J.; Chun, B.; Schwab, S.; and Vahdat, A. 2003. SHARP: An Architecture for Secure Resource Peering. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 133–148.

Gray, R. S.; Cybenko, G.; Kotz, D.; Peterson, R. A.; and Rus, D. 2002. D'Agents: Applications and performance of a mobile-agent system. *Software— Practice and Experience* 32(6):543–573.

Overeinder, B. J., and Brazier, F. M. T. 2004. Scalable Middleware Environment for Agent-Based Internet Applications. In *Proceedings of the Workshop on State-of-the-Art in Scientific Computing (PARA'04)*.

Poslad, S.; Buckle, P.; and Hadingham, R. 2000. The FIPA-OS Agent Platform: Open Source for Open Standards. In *Fifth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, 355–368.

Sandholm, T., and Lesser, V. 1995. Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. In Lesser, V., ed., *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, 328–335. San Francisco, CA, USA: The MIT Press: Cambridge, MA, USA.

Suri, N.; Bradshaw, J. M.; Breedy, M. R.; Groth, P. T.; Hill, G. A.; and Jeffers, R. 2000. Strong Mobility and Fine-grained Resource Control in NOMADS. In *Proceedings of the Second Int'l Symp. on Agent Systems and Applications and Fourth Int'l Symp. on Mobile Agents (ASA/MA2000)*, volume 1882 of *Lecture Notes in Computer Science*, 2–15. Zurich, Switzerland: Springer-Verlag.

Tripathi, A. R.; Karnik, N. M.; Vora, M. K.; Ahmed, T.; and Singh, R. D. 1999. Mobile Agent Programming in Ajanta. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, 190–197.

Waldo, J. 1999. The Jini Architecture for Network-centric Computing. *Communications of the ACM* 42(7):76–82.

Wijngaards, N. J. E.; Overeinder, B. J.; van Steen, M.; and Brazier, F. M. T. 2002. Supporting Internet-Scale Multi-Agent Systems. *Data and Knowledge Engineering* 41(2-3):229–245.