Web Service Access Management for Integration with Agent Systems

B.J. Overeinder, P.D. Verkaik*, and F.M.T. Brazier
Department of Computer Science, VU University Amsterdam
de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{bjo,patrick,frances}@cs.vu.nl

ABSTRACT

The agent paradigm includes the notion that agents interact with services. This paper identifies the need for controlled access to such services, from the perspective of agent systems (and not as is generally the case by web service providers). Mediating between web service requests from (virtual) organizations of agents, the web service gateway proposed regulates (i.e., monitors and controls) web service access according to the SLAs and organizational policies that are in effect. In addition to a model for web service access regulation, an implementation of a middleware component for web services access regulation based on SOAP and described in WSDL is presented.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multiagent systems*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-base services*

General Terms

Design, Management

Keywords

Web service interaction and integration, Web service management, agent middleware

1. INTRODUCTION

In service-oriented architectures, the consumer and middleware can vary from high-performance computing jobs in Grid environments to autonomous mobile processes in agent middleware. In both situations, shared resources (web services) are used by applications and the supporting middleware to manage individual web service usage.

Integration of web services with agent systems (multi-agent systems and/or agent platforms) requires not only semantic ontology integration [3, 5] but also management integration: regulating web services access and usage from the perspective of an agent system runtime environment. The Web Services Distributed Management (WSDM) specification addresses the integration and management of complex heterogeneous systems [11]. This standard specifies a common protocol for web service providers and their consumers. At a higher level, the application level, service level agreement (SLA) frameworks are being proposed to regulate service access from the perspective of the provider [13].

Service level agreements negotiated between individual agent platforms (e.g., collection of hosts in same IP range) and web service providers, should also be monitored and regulated by the agent platform. Individual and organization-wide agent (application) web access should be managed according to these SLAs. Consider, for example, a web service that (explicitly or implicitly) imposes a limit on the number of requests per second or the number of simultaneous TCP connections, from any one site. The (virtual) organization responsible for the site on which an agent runs, may in turn, impose explicit local policies to deal with this limitation, regulating local access. This local policy is instrumented in the agent platform.

To this purpose this paper proposes a generic extension to agent platforms, called the *web service gateway*, for controlled web service access and usage. It supports integrated management infrastructure, for example using service level agreements, to control agent access to, and usage of, web services according to organizational policies. Semantic ontology integration is explicitly not further addressed in this paper. Complementary results like Greenwood *et al.* [5] can be used to include semantic enhancement of the web service gateway.

The following section of the paper presents the general model of agent–web service interaction and the design requirements. Section 3 presents an architecture of the web service gateway, and Section 4 describes the integration of the web service gateway and SLA-based management in the AgentScape agent platform. Related work is discussed in Section 5, before closing the paper with discussion and conclusions.

2. A MODEL FOR WEB SERVICE ACCESS MANAGEMENT

The design objective of the web service gateway is transparent management of web service access. For transparent access management to external services, a number of well-known approaches exist, such as packet sniffing/filtering (used in, for example, firewalls) or proxies/gateways,[1] and sometimes combinations of these

*Current affiliation Department of Computer Science and Engineering, University of California, San Diego, CA.

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

[1]The web service gateway is comparable to a typical proxy in many web-oriented architectures, but as in this paper various manage-

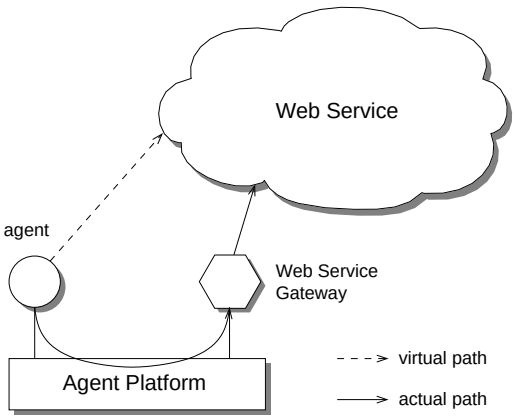

Figure 1: The web server gateway model.

approaches.

Agent systems (or more specifically agent runtime support platforms) typically run at user-level, without any special system administration permissions. Approaches incorporating techniques like packet sniffing/filtering, which require access to network devices, are not viable in this situation. Alternatively, a gateway can be incorporated with the agent platform at user-level. Another advantage of the gateway approach is that it allows for incorporating third-party solutions for semantic ontology integration [5, 14]. In the current model without semantic ontology integration, customers (agents) of web services use the known methods and standards for web service discovery (e.g., UDDI or other directory services), interface definitions using WSDL, and message exchange using SOAP. The web service gateway monitors and controls access between agents and web service providers: consumers and web service providers follow standard SOAP/WSDL practice.

By delegating access and usage enforcement to the agent platform (by means of a web service gateway), specific management policies and decisions can be realized at the appropriate management level. Agents are known by the platform in the context of, e.g., an application, their credentials, history, etc., and the platform management system can make policy decisions in the context of this information. If access and usage enforcement is realized outside the platform (e.g., a generic proxy) this context information will not be available for policy management decisions.

Figure 1 depicts virtual web service access. An agent uses standard SOAP message exchange to access a web service. The message, however, goes via the middleware[2] (or directly), to the web service gateway. The web service gateway applies its organization's management policies to individual web service requests. If in line with these policies, requests are forwarded to the web services requested. Replies to web service requests are similarly returned via the web service gateway to the consumer agent.

In many situations, resources (hosts, services, etc.) are aggregated into virtual organizations (e.g., a site, or one or more institutes and companies that collaborate and share resources). By placing the gateway function in a separate middleware service (rather than, e.g., integrating it into the middleware), the gateway is able to monitor resource usage on a virtual organization-scale rather than on a consumer basis.

A web service gateway facilitates specification and enforcement of a virtual organization's policies with respect to web service access. To this purpose a gateway monitors and controls all web service network traffic. Note that if scalability of web service access is an issue, web service gateways can be replicated: these gateways can be dedicated to specific web service providers and/or to specific local consumers.

ment tasks are incorporated in the functionality, the term web service gateway is preferred over proxy.

[2]The terms middleware and agent platform are used interchangeably. The term middleware is preferred in the context of the software architecture.

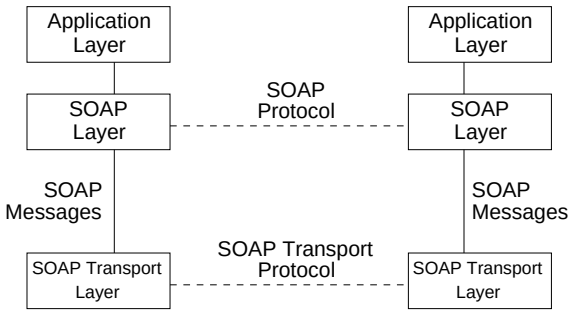

Figure 2: SOAP communication layers.

3. MIDDLEWARE AND WEB SERVICE GATEWAY ARCHITECTURE

This section describes a web service gateway architecture for agent middleware platforms. Based on the SOAP protocol and WSDL documents, an architecture is proposed to transparently intercept agent–web service interaction to monitor usage and to enforce policies.

3.1 Web Services: SOAP and WSDL

This section gives a brief overview of the basic technology that defines web services: SOAP and WSDL. These concepts are used in the design and implementation of the web service gateway in Section 3.2

3.1.1 SOAP Protocol

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment [17]. It uses XML technologies to define an extensible messaging framework with a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.

SOAP communication (see Fig. 2) consists of two layers:

- SOAP messages.
- A transport protocol that carries SOAP messages.

To send SOAP messages, SOAP must be bound to a transport protocol. The SOAP 1.2 specification specifies an HTTP binding for SOAP. However, SOAP may be bound to a variety of transport protocols. The web service gateway extension supports this option.

3.1.2 WSDL Model

The middleware web service gateway extension largely follows the model and terminology of WSDL [16], briefly summarized below, focusing on the SOAP binding of WSDL.

WSDL is an XML format for describing a web service as a set of *ports* operating on messages. In WSDL, a port is composed of the following elements:

- A set of operations (a *port type*, similar to what is commonly known as an interface).
- A protocol. WSDL can be used in conjunction with various protocols, but is, in this paper, restricted to SOAP 1.2. As described in Section 3.1.1, the SOAP protocol must be bound to some underlying SOAP transport protocol. SOAP, together with its transport protocol, constitutes the protocol to which the WSDL port is bound.
- A network address: a URL whose scheme matches the transport protocol to which SOAP is bound.

A *web service* in WSDL is defined as a set of one or more (related) ports. This allows for a web service to be distributed or replicated over multiple servers, with each server represented by a port.

Each port contains an independent SOAP protocol stack, consisting of a SOAP layer on top of a SOAP transport layer. Different ports can use different transport layers and different network addresses. Both layers of a port are (syntactically) fully specified by the WSDL document.

3.2 Design of the Web Service Gateway Architecture

The gateway architecture allows an agent application running on an agent middleware to act as a web service client to a SOAP-based web service described by a WSDL document. The web service gateway not only provides the agent application with initial access to a web service, but also acts as an intermediary for all subsequent communication between the agent application and the web service.

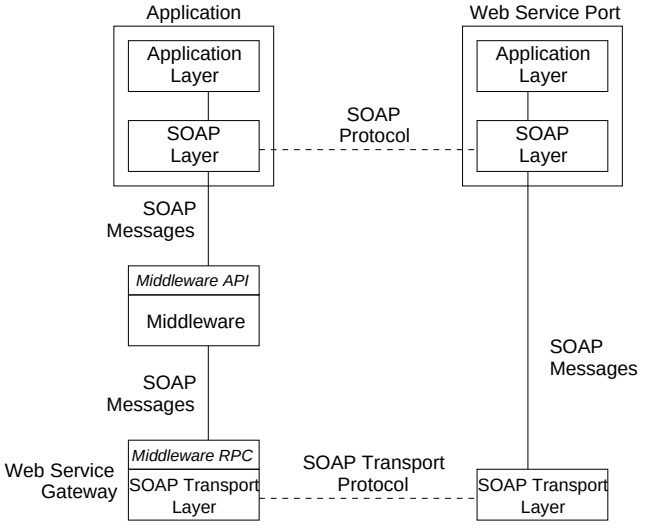

Figure 3: The SOAP gateway approach.

Being a web service consumer, an agent exchanges SOAP messages with a web service port. According to the SOAP standard, these messages are carried by a SOAP transport layer (e.g., HTTP). However, the transport layer is maintained by the web service gateway of the middleware platform, rather than the agent. The agent therefore passes any SOAP messages it wishes to send to the port (through the middleware hosting the agent) to the web service gateway. Similarly, the web service gateway passes any SOAP messages it receives from the port to the correct agent (through the middleware hosting the agent).

Agents and middleware communicate through the middleware API as required by the middleware architecture. The middleware layer and the web service gateway communicate through the middleware RPC protocol. The middleware and the web service gateway do not need to modify SOAP messages for the purpose of routing the messages to the correct port. However, the web service gateway may inspect and modify SOAP messages as part of its policy management function.

As shown in Fig. 3, both network layers that are specified in a WSDL document, namely SOAP and the SOAP transport, are implemented by separate components, namely the agent application and the web service gateway (respectively). It follows that both components must have access to the information specified in the WSDL document.

4. INTEGRATION OF THE WEB SERVICE GATEWAY IN MIDDLEWARE

The SOAP gateway approach presented in Section 3.2 includes an architecture that enforces agent application–web service interaction via the agent middleware. The following section presents the interface to the web service gateway and its functionality. Next, the mechanisms designed to parse and generate SOAP messages that are routed through the middleware are described. The last section describes a prototype implementation of these mechanisms and the SLA-based management infrastructure in the AgentScape agent middleware.

4.1 Middleware Web Service Interface

The web service API consists of two calls: (i) `requestWSDLAccess()` call allows for requests for access to specific web service, with initial parameters, (ii) `sendSOAPRequest()` call to send a SOAP request to a web service port, and as a result to receive a SOAP response from the same port.

As a result of a request call the web service gateway creates a *lease* which defines the context (i.e., the implications of the site's policies) in which an application accesses the web service's ports. This mechanism can be integrated with existing management frameworks in the middleware or with, for example, the WS-Agreements framework [1, 9]. The current design assumes that the web service gateway is responsible for the leases and that a new lease is created for each single web service. The use of leases in relation with SLAs is discussed in Section 4.4.

The API is made available to the agent application it hosts as part of the agent platform's API, and to the agent platform middleware through the middleware RPC.

4.1.1 Web Service Access Request

The `requestWSDLAccess()` is used by an application to request access to a specific web services from a web service gateway. The web service gateway can either either decline the request, or create a lease which defines the context in which an application interacts with the web service, i.e., to enforce the constraints imposed by the hosting middleware of the interaction between an agent application and a specific web service. Multiple leases may be requested for the same web service.

The web service gateway returns a lease ID to the application that has requested access. To prevent applications from accessing each other's leases, the lease ID is defined relative to the application ID. The middleware checks the application ID of a requesting

agent application before forwarding an invocation to the web service gateway.

As indicated above a `requestWSDLAccess()` call specifies a WSDL document and the name of the service requested. (Note that a WSDL document may contain multiple service descriptions.) The web service gateway extracts SOAP transport layer information (i.e., the transport protocol identifier and the network address of each port) from the WSDL description of the web service.

Some transport protocols, such as HTTP/1.1, have the notion of a *session*. For these protocols the web service gateway (i) can avoid the overhead of setting up a new session for each SOAP message exchange, and instead reuse a transport session for multiple SOAP message exchanges destined for the same web service port; and (ii) can conserve resources by dropping transport sessions (e.g., when they remain unused for an extended period of time) and multiplex SOAP messages belonging to different leases (possibly belonging to different applications) over a single transport session. However, certain web services maintain client state and associate client state with transport sessions making this impossible:

- A web service may expect a transport session to be associated with a single consumer agent. As a result, the web service gateway may not let multiple applications (or leases) share transport sessions to these web services.
- A web service may require a transport session to be maintained while a consumer agent issues a number of SOAP messages to a specific web service. As a result, a web service gateway will avoid dropping a transport session service whilst a consumer agent application is using the session (or while the lease exists).

These requirements are not described in WSDL documents, and so cannot be independently determined by the web service gateway. The solution chosen is to let an application pass a `exclTransport` parameter to `requestWSDLAccess()` that requests that for the new lease the web service gateway:

- does not share the transport sessions of this lease with other leases;
- does not drop the transport sessions of the lease while the lease exists;
- uses at most one transport session per port.

The web service gateway may decline to grant a lease with this parameter set, and may signal an error to the application at a later point if it initially granted `exclTransport` but was unable to sustain the corresponding requirements.

4.1.2 Send/Receive SOAP Requests

After an application has acquired a lease, it may send and receive SOAP messages to and from a port in the web service identified by the lease. To this purpose, it calls `sendSOAPRequest()`, to communicate the port name, the lease ID, and a SOAP request message. In addition, the middleware supplies the application ID of the calling application to the web service gateway. (Recall that the lease ID is defined relative to the application ID.) The web service gateway sends the SOAP request to the specified port and blocks the call until the SOAP response comes back from the port. The SOAP response returned by the port is passed back to the application.

Note that there is no explicit parameter that specifies which operation inside the port is to be invoked, as the operation name is part of the SOAP request.

4.1.3 Extensions

There are a number of improvements that can be made to the implementation of the web service gateway extension including:

- One-way message passing. The SOAP binding of WSDL supports two communication patterns: *one-way* (a client sends a message to a web service port) and *request-response* (a client sends a message to a web service port and the web service port sends a correlated message to the client). In the current implementation, the web service gateway only supports request-response. The extension can be improved by adding the one-way pattern. Note that WSDL recognizes two additional patterns which are not supported by WSDL's default SOAP binding: *solicit-response* (a web service port sends a message to a client and the client sends a correlated message to the web service port), and *notification* (a web service port sends a message to a client).
- Preventing a consumer agent from cheating by requesting multiple leases for the same web service. Detecting multiple requests is the first step, deciding whether the agent application is cheating is another. Given the current web services API there may be legitimate reasons for an agent to request multiple leases. For example, a consumer may wish to present itself to a web service as multiple clients for legitimate reasons, in which case it would need to request a separate lease for each client.

4.2 Consumer Application Utility

Communicating with a web service port through the middleware API as described above, requires a consumer agent application to be capable of generating and parsing SOAP messages. A number of existing utilities can be used for this purpose, an example of which is the open-source *Axis* utility.[3]

Apache Axis (eXtensible Interaction System) is a platform for creating and deploying web services applications. Essentially, Axis is an open-source SOAP engine and provides a modular, flexible, and extensible framework for constructing SOAP processors such as clients, servers, gateways, etc.

This section describes a technique that enables an agent application running on an agent platform to employ the Axis utility in conjunction with the web service gateway extension. Note that the web service gateway extension does not depend on the use of Axis by the application developer. Alternative SOAP utilities that are sufficiently flexible can use a technique similar to the one described in this section.

Given a WSDL document, Axis generates a stub implementation (currently in Java and C++) that communicates with the web service ports specified in the WSDL document. The Axis stub manages any SOAP transport sessions that are required, and implements each SOAP operation as a Java/C++ method which:

- marshalls the Java/C++ parameters of the method into a SOAP request message;
- sends the SOAP request message to the correct port using the transport protocol and network address specified for the port;
- receives the corresponding SOAP response message from the port;
- unmarshalls the SOAP response message and returns its contents as a return value of the Java/C++ method.

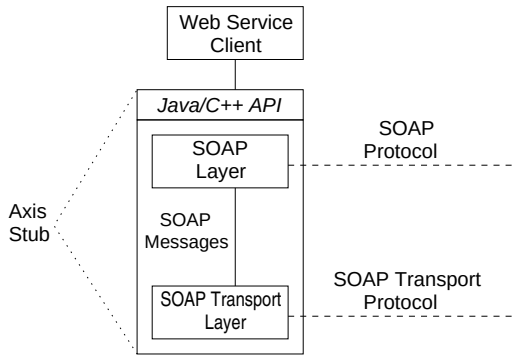

Figure 4: A SOAP stub as implemented by Axis.

Figure 4 summarizes the functionality of an Axis stub.

An agent application cannot use an Axis stub as described above, as, by default, the stub attempts to communicate with a web service port using the transport and network address specified in the WSDL document. However, in the context of the proposed middleware architecture, an application is required to access the port through the middleware API.

Axis provides the functionality needed to solve this problem as (i) the set of transports that Axis recognizes (e.g., HTTP) can be extended with additional transports; and (ii) although by default an Axis stub bases its transports and network addresses on those specified in the WSDL document from which the stub was generated, Axis allows the client to specify an alternative transport and network address at runtime.

Using Axis' pluggable architecture, the above problem is solved by providing the stub with a middleware-specific SOAP transport implementation that makes invocations to `sendSOAPRequest()` in the middleware API, rather than performing network communication. (It therefore ignores the transport layer specification in the WSDL document.) This is illustrated in Fig. 5. Note that the new middleware-specific SOAP transport protocol is called *mwrpc*.

In addition, the application developer is provided with *WSStubber*, a utility that performs the following functions:

- It makes the `requestWSDLAccess()` call in the middleware API (described in Section 4.1) on behalf of the application.
- Given an Axis stub implementation generated from a WSDL document, WSStubber instantiates an Axis stub, and modifies it to use the mwrpc transport rather than the transports specified by the WSDL. In addition, WSStubber sets up a number of parameters in the stub that ensure that the mwrpc transport implementation receives the following information:
  - A reference to the middleware API. The mwrpc implementation requires access to the API in order to invoke `sendSOAPRequest()`
  - The lease ID, to be passed to `sendSOAPRequest()`.
  - The port name, to be passed to `sendSOAPRequest()`.

4.3 Prototype Implementation in AgentScape Middleware

The web service gateway model and architecture are implemented and integrated in the AgentScape middleware. AgentScape

[3]Apache Axis, `http://ws.apache.org/axis`.

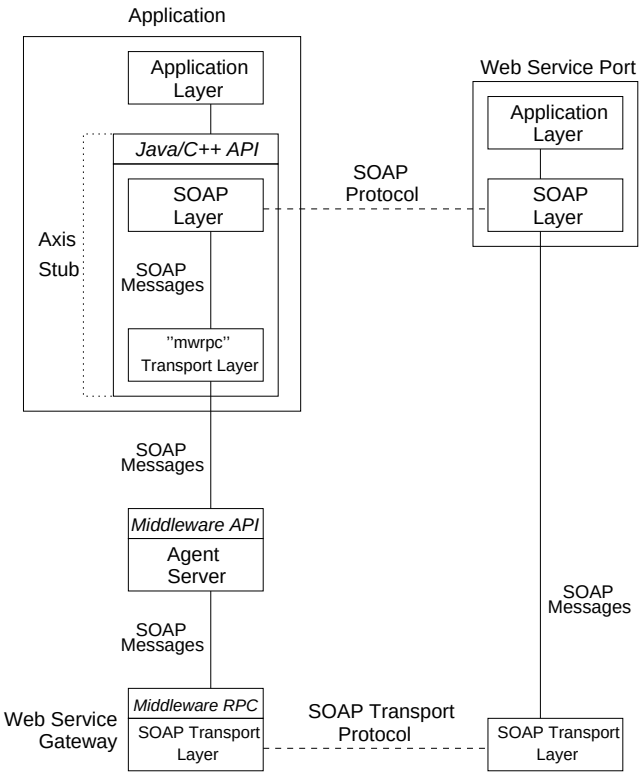

Figure 5: A SOAP stub part of a web service consumer application.

is an open, scalable platform for mobile agents (autonomous processes) [12]. In AgentScape, virtual organizations are called *locations*. An AgentScape location consists of one or more hosts running the AgentScape middleware, typically within a single administrative domain.

The AgentScape middleware consists of two layers: (i) the kernel and (ii) the middleware services. The *kernel* provides low-level secure communication between middleware processes, and facilities secure agent mobility. The *middleware services* provide higher-level middleware functionality to agents. For example, *agent servers* provide a run-time environment for agents, *host managers* are responsible for managing the middleware components running on their hosts, a *location managers* that manage AgentScape locations, and a *location service* for resolving contact addresses of agents. Figure 6 shows an example of an AgentScape location. The web service gateway has been implemented as a middleware service.

4.4 SLA-Based Management Architecture

The objective of the management infrastructure is that it must honor the autonomy of agents and resources (i.e., hosts and services). An approach to accomplish this autonomy of agents and resources, is negotiation to establish the terms of conditions of an agreement specifying the access and usage of a resource. 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, the so-called service level agreement (see Fig. 7).

Contract negotiation within AgentScape takes place at two lev-

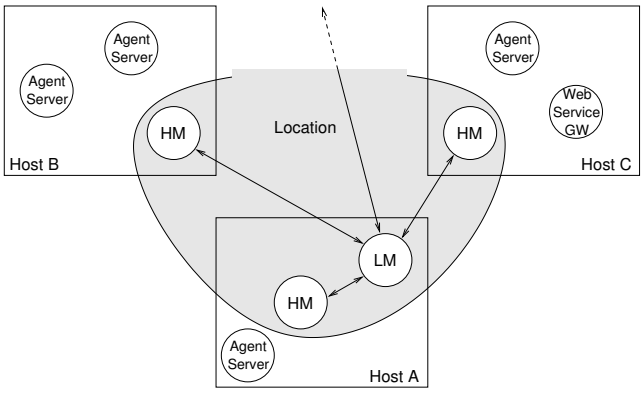

Figure 6: Management components within an AgentScape location. (LM = location manager, HM = host manager, GW = web service gateway)

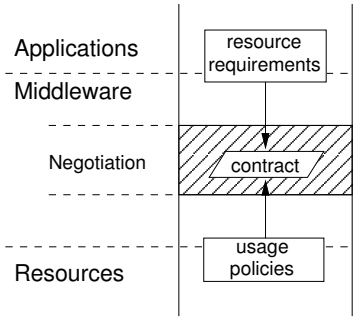

Figure 7: Abstract resource negotiation and contract model.

els: 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 of AgentScape: localizing and limiting the number of interactions required.

The *Web Services Agreement Specification* (WS-Agreement) [1] defines the format used to specify contract descriptions and negotiation interactions. 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. A detailed description of the WS-Agreement based management infrastructure can be found in the article of Mobach *et al.* [9].

In the web service gateway scenario, the *leases* that are granted to agents are the result of a WS-Agreement based negotiation. The negotiation of an agent with a web service gateway determines the access, usage, and QoS terms according to the policies at local and organization level. The web service gateway enforces the agreements by monitoring and managing the individual leases.

5. RELATED WORK

The agent paradigm assumes interaction between agents and services, of which web services are specific instances. This paper addresses how this interaction can be structured. Integration of web services with agent applications can be considered at two levels: semantic level and management level.

Independently, both Greenwood *et al.* [4] and Shafiq *et al.* [14] introduce a web service gateway as a (FIPA compliant) solution to web service integration, connecting agents and web services transparently. The operation is fully automatic, allowing web services to invoke agent services and vice versa by translating message encodings and service descriptions between the two technologies. The gateways provides service directory transformation (directory facilitator–UDDI), service description transformation (agent description–WSDL), and communication protocol transformation (ACL–SOAP). An enhanced version of the web service gateway of Greenwood *et al.* [5] allows agents and web clients to invoke atomic services and composition patterns. WS2JDADE [10] provides a similar framework for agent–web service integration, but also includes facilities to deploy and control web services as agent services at runtime for deployment flexibility and active service discovery.

Soto presents a web service based message transport service (WSMTS) [15]. The message transport service enables agents to interact through the web with web services and other agents. WSMTS is a FIPA compliant communication framework where messages are grounded using web services standards (e.g., SOAP, WS-Addressing, etc.). The WSMTS not only allows for web service accessibility as with the other proposals, but provides the possibility to perform complex interaction patterns using SOAP between agents and other agents or Web Services.

The difference with the the web service gateway presented in this paper, is that this paper focuses on management of web service interaction. Mechanisms are presented that (implicitly) enforce agents to interact with web services through the middleware, so that access and usage of web sites can be managed on a site basis, using SLA-based leases.

The only translation addressed in our approach is the ACL–SOAP translations, as described in Section 4. The other two translations (directory facilitator–UDDI and agent description–WSDL) could be be adopted from the contributions described above, to create a new web service gateway with combined functionality.

Dickinson and Wooldridge [3] present a different architecture for integrating web services with BDI agents. Service descriptions are extended with meta-knowledge with which agents can reason about individual web services in relation to specific goals, to devise plans to reach these goals. Agents can dynamically create web service bindings via the standard WSDL service descriptions, without further control or management.

Although not often cited in agent literature, the expected Quality of Service (QoS) is relevant to agent application developers. Service level agreements between a service provider and a consumer are negotiated to assure that consumers receive the service they expect and have paid for [6]. Dan *et al.* [2] present a framework for differentiated levels of service through automated management and SLAs. Machiraju *et al.* [8] introduce an architecture for federated service management, which targets management of web services that interact across administrative domains (and involves multiple stakeholders).

The potential of web service gateways for consumer side web service management has, however, not yet fully explored. Sahai *et al.* [13] have also identified this problem: customer side measurement has been neglected—relying solely on measurements on the

server side. In their article, they propose an automated and distributed SLA monitoring engine to obtain measurements from multiple sites to guarantee SLAs. The approach is applicable to web service architectures in general, and is, in itself, a subsystem of substantial size and complexity, whereas our approach is lightweight with a small footprint (and subsequently, targeted to management problem in a specific agent systems setting).

Web Service Offerings Language (WSOL) is a language for the formal specification of classes of service, various constraints (functional, Quality-of-Service, access rights, etc.), and management statements (prices, penalties, and responsibilities). Ma *et al.* [7] designed a Web Service Offerings Infrastructure (WSOI) by extending Apache Axis. The infrastructure allows for perform different monitoring activities for different WSOL service offerings. Similar to the approach presented in this paper, their solution compares to other existing solutions as less general (or powerful), but also less complex. The difference with our approach is that the web service gateway management is based on SLA negotiation, while selection and manipulation of service offerings are simpler but faster than SLA negotiations.

6. CONCLUSIONS

The web service gateway model and architecture presented in this paper manages web service access at the level of agent platform middleware, supporting monitoring and web service access control at the level of individual agents. SLAs between agents (i.e., the organizations they represent) and service providers must be obeyed by both parties. The web service gateway and SLA-based negotiation infrastructure showed to be a flexible approach which allows for the inclusion of other components, for example, for semantic ontology integration.

The web service gateway presented in this paper manages SLAs from the consumer side, the agent side. For example, if web service access is shared in an organization, multiple agents can make requests to services. However, in aggregate these requests must comply with the SLA to which they have agreed, enforced by the web service gateway.

The interoperability of the web service gateway with web service management frameworks as described by Dan *et al.* [2] and Machiraju *et al.* [8] will be further studied. Also the inclusion of the other two transormation (directory facilitator–UDDI and agent description–WSDL) is future work. Finally, a more in-depth study into the web services offerings language (WSOL) and the WS-Agreements will be performed.

Acknowledgements

The authors thank dr. Michel Oey for his valuable contributions. This research is supported by the NLnet Foundation, `http://www.nlnet.nl`.

7. REFERENCES

[1] A. Andrieux et al. Web services agreement specification (WS-Agreement) (draft). `https://forge.gridforum.org/projects/graap-wg`, 2007.

[2] A. Dan, D. Davis, R. Keamey, A. Keller, and R. King. Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, 43(1):136–158, 2004.

[3] I. Dickinson and M. Wooldridge. Agents are not (just) web services: Considering BDI agents and web services. In *Proceedings of the Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005)*, Utrecht, The Netherlands, July 2005.

[4] D. Greenwood and M. Calisti. Engineering web service–agent integration. In *Proceedings of the International Conference on Systems, Man and Cybernetics (SMC 2004)*, volume 2, pages 1918–1925, The Hague, The Netherlands, Oct. 2004.

[5] D. Greenwood, J. Nagy, and M. Calisti. Semantic enhancement of a web service integration gateway. In *Proceedings of the Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005)*, Utrecht, The Netherlands, July 2005.

[6] L.-J. Jin, V. Machiraju, and A. Sahai. Analysis on service level agreement of web services. Technical Report HPL-2002-180, HP Laboratories, Palo Alto, CA, June 2002.

[7] W. Ma, V. Tosic, B. Esfandiari, and B. Pagurek. Extending Apache Axis for monitoring of web services offerings. In *Proceedings of the IEEE EEE05 International Workshop on Business Services Networks*, Hong Kong, China, Mar. 2005.

[8] V. Machiraju, A. Sahai, and A. van Moorsel. Web services management network: An overlay network for federated service management. In *IFIP/IEEE Eighth International Symposium on Integrated Network Management*, pages 351–364, Mar. 2003.

[9] D. Mobach, B. Overeinder, and F. Brazier. A WS-Agreement based resource negotiation framework for mobile agents. *Scalable Computing: Practice and Experience*, 7(1):23–36, Mar. 2006.

[10] X. T. Nguyen and R. Kowalczyk. WS2JADE: Integrating web service with JADE agents. In *Service-Oriented Computing: Agents, Semantics, and Engineering*, volume 4504 of *Lecture Notes in Computer Science*, pages 147–159. Springer, Berlin, Germany, 2007.

[11] OASIS. Web services distributed management. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm`, 2005.

[12] B. J. Overeinder and F. M. T. Brazier. Scalable middleware environment for agent-based Internet applications. In *Proceedings of the Workshop on State-of-the-Art in Scientific Computing (PARA'04)*, pages 675–679, Copenhagen, Denmark, June 2004.

[13] A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, and F. Casati. Automated SLA monitoring for web services. In *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems*, pages 28–41, Montreal, Canada, Oct. 2002.

[14] M. Shafiq, Y. Ding, and D. Fensel. Bridging multi agent systems and web services: Towards interoperability between software agents and semantic web services. In *Proceedings of the 10th IEEE International Conference on Enterprise Distributed Object Computing (EDOC'06)*, pages 85–96, Hong Kong, China, Oct. 2006.

[15] E. L. Soto. Agent communication using web services, a new FIPA message transport service for JADE. In *Multiagent Systems Technologies*, volume 4687 of *Lecture Notes in Computer Science*, pages 73–84. Springer, Berlin, Germany, 2007.

[16] W3C. Web services description language (WSDL) 1.1. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`, Mar. 2001.

[17] W3C. SOAP Version 1.2 Part 1: Messaging framework. `http://www.w3.org/TR/2003/REC-soap12-part1-20030624/`, June 2003.