

An Automatic, Bi-Directional Service Integration Gateway

Dominic Greenwood and Monique Calisti
Whitestein Technologies AG
Gotthardstrasse 50
CH-8002, Zurich
Switzerland
Phone: + 41 1 2055500
Fax: + 41 1 2055509
E-mail: {dgr, mca}@whitestein.com

ABSTRACT

There is little doubt that Web services are becoming a part of the essential fabric enabling machine-to-machine information exchange across the Web. However, there is also a perceptible progression toward the realisation that this principal means of invoking operations on remote hosts will become insufficient to their task almost as quickly as they rose to prominence. Already emerging amongst the community of adopters is a recognition that richer semantics are required to express any truly useful information about the nature of business processes and the way in which they interoperate. But ultimately, as the power to meaningfully express information improves, so does the need for automatic processes (i.e. agents), that are capable of interpreting, processing and deciding about this information. Given this, a means of connecting the realms of agents and Web services is required and so this paper reports on a proposed design for a Service Integration Gateway for seamlessly connecting Web services to agents and their services. The Gateway is a progressive work evolving from existing approaches, yet taking an innovative approach encapsulating fully transparent and automatic operation in a simple and scalable solution.

1. INTRODUCTION

From the very early days of the Internet, and particularly since the advent of the World Wide Web, it has been evident that the prevailing bias towards human-to-machine interaction would eventually be extended by technology designed to support direct machine-to-machine interaction, sometimes driven by humans, sometimes in a completely autonomous mode. In recent years this momentum has begun to manifest itself predominantly as Web services; a reworking of many of the concepts associated with RPC-style communication to account for XML and to make further use of the predominant 'port-80' to transit executable operations. The W3C defines a Web service as "... a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with

other Web-related standards." [9].

The current state of affairs is that Web service technologies are exerting an increasing impact across broad swathes of commerce and industry, not because they are based on any particularly innovative technology, but rather because they are a straightforward means of engineering machine-to-machine relationships using proven technology. In addition, Web services are observed to be approaching a point of critical inertia beyond which they will become an accepted mainstream technology. Tracing this path through the next few years we can predict that emerging technologies, such as the Semantic Web will substantiate these trends to take the basis of Web services towards more powerful modalities trading on semantic expressivity, autonomy and knowledge processing, as highlighted by Davies et al. in [4].

Against this backdrop there is also an anticipation that software agent concepts and technology will form a concrete component of this future, taking the momentum gained through widespread adoption of Web services into a new phase. Several arguments have been made to support this case, including [1] [12] and [11], but perhaps none more evocative than statements made in the Web Services Architecture specification of the W3C [9] which clearly expresses the notion that, "... *software agents are the running programs that drive Web services - both to implement them and to access them as computational resources that act on behalf of a person or organisation*".

Given this observation, identifying a means of connecting agents and Web services is the motivation and central foundation of our work. Due to evident differences, including strong vs. loose coupling and representational encodings, we have identified an approach that introduces an intermediary entity, called the *Service Integration Gateway* (SIG) which is designed to encapsulate all functionality required to connect the two realms, whilst ensuring minimal human intervention and service interruption. This paper describes the SIG and addresses the two functional perspectives; that of agents invoking Web services and the inverse case, of Web services invoking agent services. Taking the former, i.e. from the perspective of agents, Web services are simply programmatic entities that can be called upon to perform an advertised and typically unitary function. But to users, or consumers, of Web services, agents can form a powerful means of indirection by masking the Web service for purposes of redirection, aggregation, integration or administration. *Redirection* describes the case where a Web service may no longer be available for some reason, or the owner of

the Web service wishes to temporarily redirect invocations to another Web service without removing the original implementation. *Aggregation* allows several Web services to be composed into logically interconnected clusters providing patterned abstractions of behaviour that can be invoked through a single service interface. *Integration* describes the means of simply making Web services available to consumers already using, or planning to use, agent platforms for their business applications and *Administration* covers aspects of automated Web service management where the agent autonomously administers one or more Web service without necessary intervention from a human user. From the opposite perspective the architecture proposed herein closes the loop by enabling Web service clients to invoke application services exposed by agents. In this instance, the type of agent service made available is naturally restricted due to the limited expressivity of typical service invocation calls initiated by Web service clients. The intrinsic implication here is though, that applications are now able to bidirectionally seamlessly bridge the agent and Web service domains.

Three key features of the SIG are:

- *Transparency.* The SIG is designed to be operationally transparent to invoking entities, whether they be agents or Web services. For example, an agent wishing to invoke Web service sends the invocation message directly to the SIG with a property of the message content set to the name and address of the Web service. The SIG will seamlessly transform the message, issue it and transcribe any response before returning it to the invoking agent.
- *Automation.* Once operational, the SIG requires no manual intervention or configuration.
- *Integration.* The SIG encapsulates and integrates the functionality required to deliver the above features without requiring additional external resources.

2. RELATED WORK

The work reported in this paper is an attempt to identify a relatively simple, yet extensible model that brings about loosely-coupled integration of agents and Web services. As such it is strongly influenced by and indeed evolved from, several existing contributions to the field such as that reported by Lyell et al. [11] which discusses the concept of a hybrid J2EE/FIPA-compliant software agent system using Colored Petri Nets. This approach identifies two key concepts, (1) That agents should be able to expose their services as Web services for the potential use of non-agent clients and (2) that these *Web service-enabled* agents should advertise using both a UDDI registry and a FIPA Directory Facilitator (DF). Our approach also addresses these issues, but without the specific use of J2EE or special "Web service agents" to expose Web services. Instead the SIG is intended for use with any J2*E technology platform and deliberately avoids the introduction of specialized agents for handling the exposure of Web services. Instead, the SIG is an independent set of codecs that can process and translate Agent messages and Web service calls without bringing any agents or Web services within the Gateway boundary. We find that this approach is more scalable than 'one agent for one Web service', as only new entries in the SIG DF and UDDI repositories need be generated each time a new agent service or Web service is exposed - rather than creating a new agent for the purpose.

Also in [13] we are shown how automation can be driven by agent generators reasoning about the semantic descriptions of the Web

services they are configured to compose and use. However, perhaps the most directly referential work relating to the SIG architecture is that of the Gateway Model produced by the Agentcities.NET¹ Web service working group.

The recommendation produced by this group [7] details a two-part solution to constructing a bidirectional agent-to-Web service gateway. The model, intended for use by the JADE² agent system, consists of two separate implementations that (to date) have yet to be integrated into a unitary solution. One aspect, called WSDL2JADE [7] is designed as a wrapper solution for converting agent sourced ACL encoded requests into the appropriate Web service operation calls and corresponding responses back into ACL. The tool generates a JADE proxy agent for an existing Web service described by a WSDL file, making it possible to call Web services indirectly within an agent environment. The second aspect, called WSAI [7] and also implemented using JADE, allows an agent service to be published as a Web service. The implementation consists of a Web Services Agent Gateway (WSAG) and an agent Generator. The WSAG manages the transition from agents to Web services and the Generator is a support tool for generating agents that operate within the gateway providing a concrete Web service interface for a particular external agent. When a Web service invokes a SOAP call on the gateway, the gateway transforms this synchronous call into an asynchronous message communication via FIPA ACL messages to the software agent that provides the service. In its current form this solution is neither integrated, transparent nor automatic and has the additional drawback of requiring a degree of manual configuration. In addition it has limited support for extensions enabling advanced services such as proxying, composition and semantic filtering. The SIG approach takes some of the ideas generated by this project and extends them to form a model of a transparent, automatic and extensible gateway service.

Regarding some of the advanced features discussed in the latter half of this paper, we note that [12] discusses the use of agents as proxies that assist in selection of Web services according to the quality of matching criteria. The approach supports the dynamic selection of services as a path toward autonomic computing where computational resources are self-managing and self-configuring. In addition, in [1] and [14] the authors discuss the use of process description languages for enacting business processes in the contemporary workplace. They note that strict adherence to prescribed workflows implies that systems are largely unable to adapt effectively to unforeseen circumstances. The work proposes that workflow description languages be used to specify multiagent systems, specifically advancing the idea that the Business Process Execution Language for Web Services [3] can be used as a specification language for expressing the initial social order of a multiagent system, which can then intelligently adapt to changing environmental conditions.

3. ARCHITECTURAL MODEL

The SIG is a stand alone, encapsulated service that provides transparent, bidirectional transformations between FIPA compliant agent

¹Agentcities.NET is a European Commission funded 5th Framework IST project, IST-2000-28384, completed in 2003, aiming at creating an open dynamic service environment based on (FIPA compliant) agent technology and make it accessible to potential users. More information can be found at <http://www.agentcities.org/EUNET/>

²JADE is a FIPA compliant, Java-based agent development toolkit. Additional information can be found at <http://jade.tilab.com>

services and Web services that use the standard WSU³ stack. We have chosen to design a solution that complies with FIPA both due to its status as a widely adopted industry standard and also as we wish to provide the SIG as a JADE/LEAP platform service. With respect to Web services, to ensure some baseline compliance we employ the WSU stack, also due to widespread industry adoption.

3.1 Assumptions

The following assumptions were made when designing the SIG architecture:

- All agents are assumed to be FIPA compliant and capable of communicating with FIPA-ACL encoded messages.
- All Web services operate using the standard Web service stack consisting of WSDL for service descriptions, SOAP for message encoding and UDDI for directory services.
- The SIG is designed as a platform service accessible to both agents running on and external to JADE platforms, and Web service clients operating from any accessible network locations.
- The SIG is registered as an agent service in FIPA Directory Facilitators and as a Web service endpoint in UDDI directories.
- All interactions between the SIG and agents use ACL encoded FIPA-Request and FIPA-Inform performatives [6]. This simplifies the operation of the SIG and are essentially all that is needed to invoke basic request-response Web services.

3.2 Architecture

The problem as specified is to provide an automatic means of mapping the functional and representational dependencies associated with the invocation of agent services onto those associated with Web services and vice versa. Our proposed solution introduces an intermediary processor into the communication path between these service calls. This Service Integration Gateway (SIG) allows both agent and Web services to register with it and thereby publish their service descriptions to consumers outside their normal operational domain. The SIG will then intercept calls to these registered services allowing agents to invoke Web services and vice versa by transforming message encodings and creating service access endpoints. A functional overview of the SIG is described by Figure 1.

As illustrated, the SIG contains several operational components, including a FIPA compliant Directory Facilitator (DF) and a Web service stack compliant UDDI repository. These internal registries are at the heart of the SIG, maintaining records of all agent and Web service descriptions that have been registered with, and can be invoked via, the gateway. Access to these registries is provided to both agents and Web service clients by exposing appropriate interfaces providing the standard operations:

- Registration of a new service description.
- Deregistration of an existing service description.
- Modification of an existing service description.
- Searching for a registered service description.

³WSDL, SOAP and UDDI

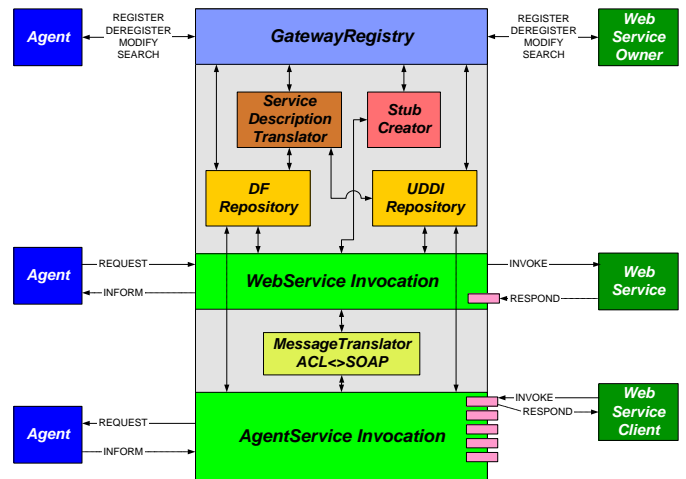


Figure 1: Functional overview of the SIG architecture entities.

The semantics of these operations are described further in Section 4.

The remaining operational components are described below:

The GatewayRegistry

This is the logical component that receives and processes incoming directory operations, thereby controlling access to the internal registries. ACL encoded service descriptions received from agents are stored in the internal DF registry and automatically translated (by the ServiceDescriptionTranslator component) into their WSDL equivalent and stored in the internal UDDI registry. The reverse case is also true for WSDL encoded service descriptions received from Web service owners. In this manner the SIG maintains a mapping of each service description in both ACL and WSDL forms. Additionally, a RegistryID tag with the same unique identity is assigned to the DF and UDDI instances of the service description to ease future lookups across both directories. In the case of malformed service descriptions an exception is raised and treated accordingly relative to the sender type (agent or Web service).

The ServiceDescriptionTranslator

Utilized by the GatewayRegistry, the ServiceDescriptionTranslator component acts as a transformation filter with two paths of operation. The first path is initiated when a request to register a new ACL encoded agent service descriptions [5] is received by the SIG. The ServiceDescriptionTranslator automatically transforms these service descriptions into equivalent WSDL Web service descriptions [15], returning the result to the GatewayRegistry from where it is registered with the internal UDDI repository. The second path is the inverse case where WSDL encoded Web service descriptions are transformed into their equivalent ACL agent-service description. This process ultimately ensures that all registered service descriptions are duplicated across both internal registries.

Mutual inconsistencies prevent the mapping between the ACL and WSDL service descriptions from being one-to-one; those description features with no direct mapping are translated as optional properties.

The StubCreator

Web services are typically exposed as service endpoints, herein defined as stubs, which are called as remote invocations. Thus whenever a new agent service (that is to be exposed to Web service clients) is registered with the SIG, the GatewayRegistry invokes the StubCreator which generates a stub to be exposed via the AgentServiceInvocation component. The stub is built from the WSDL service description of the agent service held in the SIG UDDI directory and generated by the ServiceDescriptionTranslator component as previously described. When an invocation is received from a Web service or Web service client, the stub is activated and manages reception and processing of the incoming message and any subsequent conversation context. In addition, whenever an agent invokes a Web service via the WebServiceInvocation component, if a response is expected from the Web service a temporary stub is created by the StubCreator. This stub is removed once the expected reply is received.

WebServiceInvocation

Described by Figure 2, when an agent wishes to invoke a Web service it first looks up the ACL encoded service description from the SIG's DF directory (translated from the original WSDL by the ServiceDescriptionTranslator). Using this, the agent formulates a standard FIPA-Request message containing the Web service invocation call and sends it to the WebServiceInvocation component of the SIG which is exposed as a standard FIPA message transport endpoint. This component then uses the ServiceDescriptionTranslator component once again to map the ACL encoded service description into the corresponding WSDL, stored in the internal UDDI directory. Finally, the returned WSDL description is used to create a SOAP encoded message using the MessageTranslatorComponent which contains a bi-directional codec for transforming ACL into SOAP and vice-versa. This message is then used to invoke the appropriate Web service with a return stub being exposed to receive any expected response. Any message received by this stub is translated back into ACL and returned to the calling agent using a FIPA Inform message with any reply-specific attributes from the original ACL service invocation call.

AgentServiceInvocation

This component performs the inverse of the previous process with one significant difference. For a Web service client to invoke an agent service it must be able to make a SOAP encapsulated call onto the agent service, or a stub that reflects the functionality of that agent service. To this end, the SIG creates and exposes a service stub for any agent service that registers with it. This stub is created using the corresponding WSDL description of the agent service held in the Gateway UDDI directory presenting a service binding to external Web services. When an invocation is received from a Web service, the stub first looks up the WSDL service description in the internal UDDI directory, using the ServiceDescriptionTranslator to identify the appropriate ACL encoded service-description stored in the local DF. From this a FIPA Request message is constructed containing the service call and parameter values received from the calling Web service client. The message is then sent to the agent exposing the agent service. Additionally, the SIG keeps track of all ACL messages sent in a conversation context. The receiving agent processes the ACL message and assuming the absence of errors, will invoke the correct agent service and return any expected results to the SIG as a FIPA Inform message. Inside the SIG, this message is parsed using the MessageTranslator the return value passed back to the calling Web service client through the existing agent service stub interface. If a response is expected from

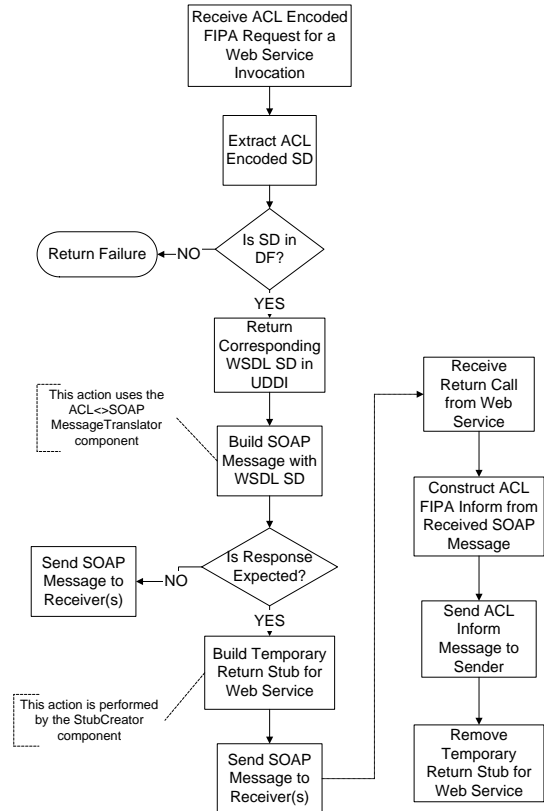


Figure 2: Web service invocation.

the agent service and none is received within a specified period, then a timeout exception can be raised.

The MessageTranslator

This contains the parsing functionality for transforming SOAP encoded messages into ACL encoded messages and vice-versa. In both directions parse-trees are constructed from the incoming message, which can then be traversed with each leaf node being translated into the corresponding encoding. This requires a schema that defines the SOAP equivalents of ACL slots and attributes, and vice versa.

4. THE SIG IN ACTION

The SIG has two principal modes of operation: an agent service invoking a Web service and a Web service invoking an agent service. Ancillary to these are the operations associated with registering services with the SIG. Additional advanced features are discussed in Section 4.2.

4.1 Standard Operation

The standard features of the SIG are described by the following operations:

SIG Initialisation

When initialising, the SIG registers with any available or pre-stated external registries in both the agent and Web service domains. From the agent perspective, the SIG is registered as a standard FIPA agent

service with an agent platform AMS⁴ and exposes the same interfaces (register, deregister, modify and search) as that of a typical FIPA compliant DF. From the Web service perspective, the SIG exposes identical functionality, but encoded as UDDI endpoints as described by the UDDI Schema [2]. From this point the SIG is treated as would a standard directory service with which to register services and search for services to invoke.

As the SIG is a gateway all requests for service (and responses from invoked services) must be sent to the SIG with the identity of the ultimate receiver encoded as a property of the request/response message. An example of this can be found in the message exchange illustrated in Figure 4.

Register a Service Description

Following the process described by Figure 3 the service descriptions of agent services and Web services are submitted to the SIG by their corresponding owners; in the agent case this is most likely to be the agent itself. As Web services are not typically capable of registering themselves, some entity acting as the service owner must submit the description. Once received by the SIG, a description is automatically translated into either ACL if received in WSDL form via a UDDI registration or the inverse if received via a DF registration. The dual descriptions are stored in the internal registries with a common RegistryID and thereby made available for discovery by external entities. Finally, if registering an agent service the stub interface used by a Web service to invoke it is exposed via the AgentServiceInvocation component of the SIG.

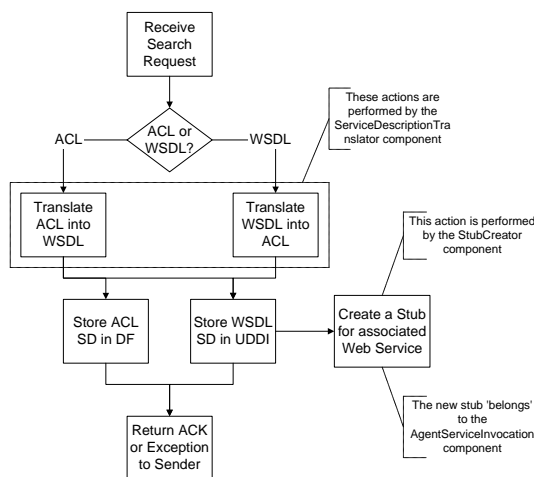


Figure 3: Registering an agent or Web service with the SIG entities.

De-register a Service Description

A de-registration request received by the SIG will remove both local copies (DF and UDDI) of the service description and any stub established for Web service sourced invocations.

Modify a Service Description

A modification request received by the SIG will replace an existing service description. This procedure can be treated in the same way as a new Registration.

Search for a Service Description

Unlike the previous operations, a search request received by the SIG will act only on the appropriate internal registry; DF in the case of agent services and UDDI in the case of Web services.

Invoke an agent service from a Web service client

Described in terms of messages that might be exchanged, Figure 4 illustrates a scenario where a Web service client initiates a SOAP message to invoke a getTime service previously published with the SIG by an agent resident in a JADE platform. We assume that this service has been successfully registered with the SIG, that a stub has been exposed and that the Web service client has searched the SIG UDDI repository for the appropriate service description. With reference to Figure 4, once the SOAP message carrying a call to invoke the service (1) is received by the SIG it is processed and a corresponding FIPA Request message (2) created and sent to the agent hosting the getTime service. The result is then sent back by the agent as a FIPA Inform message (3) to the SIG, transformed into a SOAP message (4) and returned to the initiating Web service client.

Invoke a Web service from an Agent

Similar in form to the previous case, the inverse invocation direction is initiated by an agent sending a FIPA Request message to the SIG specifying the service description and parameters of a call it wishes to make onto a Web service. Assuming that the required Web service is registered, the SIG transforms the ACL message into the corresponding SOAP encapsulated WSDL and sends this to the appropriate Web service. If a reply is expected from the Web service a stub is exposed by the SIG to receive it. In such a case, upon reception of a reply message, it is transformed into a FIPA Inform message and returned to the initiating agent.

4.2 Advanced Features

Beyond the standard features offered by the SIG we anticipate that several more advanced features will become available. These relate to ways in which the SIG (according to security constraints) can intercept and manipulate the basic service invocation calls, such as proxying, redirecting and composing. However, all of these features are intrinsically dependent on the level of semantic expressivity present in service descriptions, which in the current model is quite low. We are currently investigating the use of OWL-S⁵ [10] to add semantic depth to service descriptions and so provide more opportunities to apply reasoning to the selection and manipulation of services.

Redirection

An additional operation offered by the SIG is the ability to perform automatic fail-over redirection when a service published by the SIG is no longer unavailable. This can be managed pre-execution by notifying the SIG (via a reserved message type) of which service(s) should be substituted in the event of a failure condition. Alternatively, if no pre-selection has been made the SIG can be configured to dynamically search its internal registries for a suitable replacement service. The effectiveness of the latter solution is dependent on the level of semantic detail used for the service descriptions and is best delivered through a specialised proxy component.

⁴FIPA Agent Management System [5]

⁵The Ontology Web Language Service ontology, details available at <http://www.daml.org/services/owl-s/1.0/>

(1) Web service issues an agent service invocation call to the SIG

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
  <n:getTime xmlns:n="urn:TimeService"
    soapenv:encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding">
    <zone xsi:type="xsd:string">GMT+6</zone>
    <encoding xsi:type="xsd:string">UTC</encoding>
  </n:getTime>
</soapenv:Body>
</soapenv:Envelope>
```

(2) SIG receives call and transforms it a FIPA-Request message

```
(request
  :sender (agent-identifier
    :name sig@platform1:1099/JADE
    :addresses (sequence http://platform1:7776/acc))
  :receiver (set (agent-identifier
    :name TimeService@platform1:7774/JADE
    :addresses (sequence http://platform1:9999/acc)))
  :content
    "(action (gettime
      :properties (set
        (property invoker@www.travel.com/TimeTraveller)
        (property set GMT+6 UTC))))")
```

(3) Agent performs request and sends FIPA-Inform message to SIG

```
(inform
  :sender (agent-identifier
    :name TimeService@platform1:7774/JADE
    :addresses (sequence http://platform1:9999))
  :receiver (set (agent-identifier
    :name sig@platform1:1099/JADE
    :addresses (sequence http://platform1:7776/acc)))
  :content
    "(result
      :properties (set
        (property invoker@www.travel.com/TimeTraveller)
        ("T11:12:30Z"))")
```

(4) SIG transforms return message into SOAP response

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
  <n:getTimeResponse xmlns:n="urn:TimeService"
    soapenv:encodingStyle="http://schemas.xml
      soap.org/soap/encoding/">
    <getTimeReturn xsi:type="xsd:datetime">T11:12:30Z
    </getTimeReturn>
  </n:getTimeResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 4: Messages describing the invocation of an agent service via the SIG.

Proxying

During standard operation the SIG acts passively by exposing access endpoints via the conventional registry and invocation components. However, extending the redirection model the SIG could house an internal proxy mechanism designed to interface between services whose descriptions are held by the SIG and their invoking clients [12]. The proxy can contain rule engines or reasoners capable of playing an active role in selecting a service according to qualitative matching of application criteria with published service descriptions. To be effective the descriptions would need to be published in a representation such as DAML-S or OWL-S.

Workflow and Composition

There are several complete, ongoing and emerging initiatives that are defining extended notations for the management and operation of Web services [8]. To name but a few there are: WS-Addressing, WS-Attachments, WS-Context, WS-Coordination, WS-Federation, WS-ReliableMessaging, WS-Routing, WS-Transaction, WS-Trust, XML-DSig, XML-Encryption, XKMS, SAML, XACML, ebXML, WSBPEL and WSRP. Amongst these are several compositional notations that express the flow of control and data across a collection of Web Services whose choreography performs a workflow. Specifically, the WSBPEL⁶ specification enables businesses to employ workflow management that enacts business processes described by the language [14].

However, strict adherence to prescribed workflow makes it impossible for a system to adapt to unforeseen circumstances and it is anticipated that agent systems could act as the intelligent controllers of the workflow enactment mechanism. The SIG is well positioned to enable this, and dynamic composition of Web services, by providing connectivity between the two domains.

5. CONCLUSIONS

This article has reported on the design of an architectural model for enabling transparent, automatic connectivity between Web services and agent services. The expectation is that solutions such as this will avert any risk of an asymptotic relationship developing between these two essentially complementary technologies. The proposed architecture, termed the Service Integration Gateway (SIG) contains several logical components operating on two internal registries that keep a record of all services (both agent and Web based) that have been registered. A prototype of this model is currently under development, with a finalized version due later this year to be rendered public as a component of the JADE system in a planned future release.

In the latter part of the paper we discuss some advanced features of the SIG that could be considered as the locus of the model's true usefulness. We expect that the initial process of creating such a gateway service bridging agents and Web services is only a prelude to the potential it offers as a means to connect, compose and manipulate service populations.

6. REFERENCES

- [1] P. Buhler, J.N. Vidal, and H. Verhagen. Adaptive workflow = web services + agents. In *Proc. of the International Conference on Web Services*, pages 131–137, Las Vegas, U.S.A., July 2003. CSREA Press.

⁶The Web Services Business Process Execution Language, formerly known as BPEL4WS

- [2] OASIS Consortium. *UDDI Specifications and Schema*.
<http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm>, 2004.
- [3] OASIS Consortium. *Web Services Business Process Execution Language Specification*.
<http://www.oasis-open.org/committees/>, 2004.
- [4] N.J. Davies, D. Fensel, and M. Richardson. The future of web services. *BT Technology Journal*, 22(1):76–82, January 2004.
- [5] Foundation for Intelligent Physical Agents. *FIPA Agent Management Specification*.
<http://www.fipa.org/specs/fipa00023/>, June 2002.
- [6] Foundation for Intelligent Physical Agents. *FIPA Communicative Act Library Specification*.
<http://www.fipa.org/specs/fipa00037/>, June 2002.
- [7] Agentcities Task Force. *Integrating Web Services into Agentcities Recommendation*.
<http://www.agentcities.org/rec/00006/actf-rec-00006a.pdf>, 2003.
- [8] CBDi Forum. *The Web Services Protocol Stack*.
<http://roadmap.cbdiforum.com/reports/protocols/>, 2004.
- [9] W3C Web Service Architecture Working Group. *Web Service Architecture Recommendation*.
<http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>, 2004.
- [10] M. Laukkanen and H. Helin. Composing workflows of semantic web services. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [11] M. Lyell, L. Rosen, M. Casagni-Simkins, and D. Norris. On software agents and web services: Usage and design concepts and issues. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [12] E.M. Maximilien and M.P. Singh. Agent-based architecture for autonomic web service selection. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [13] D. Richards, S. van Splunter, F. Brazier, and M. Sabou. Composing web services using an agent factory. In *Proc. of the 1st International Workshop on Web Services and Agent Based Engineering*, Sydney, Australia, July 2003.
- [14] J.M. Vidal, P. Buhler, and C. Stahl. Multiagent systems with workflows. *IEEE Internet Computing*, 8(1):76–82, January/February 2004.
- [15] W3C. *WSDL Specifications and Schema*.
<http://www.w3.org/TR/wsdl>, 2004.