

Enterprise Service Facilitation within Agent Environments

Edward Curry, Desmond Chambers and Gerard Lyons
Department of Information Technology,
National University of Ireland, Galway, Ireland
{edward.curry, des.chambers, gerard.lyons}@nuigalway.ie

ABSTRACT

A prerequisite of participating in an enterprise system is the ability to cope with the rigorous demands experienced within the system. In order to cope with these demands, a number of infrastructure support services are available to assist developers in their creation. A key obstacle to the widespread deployment of agent technology is the relative immaturity of agent technology with regard to its infrastructure. This paper presents a solution to the problem by offering enterprise-level infrastructure services to agent platforms in an agent friendly manner. The proposed solution uses Service-Agent Gateways (SAG) to offer these services within an agent environment. This paper describes the SAG design pattern and presents an implementation of the pattern that offers the functionality of Enterprise Message Services (EMS) to an agent environment. The Java Message Service (JMS)-Agent Gateway enhances the acceptability of agent platforms within business environments, moving them a step closer to full-scale participation in the digital enterprise.

KEY WORDS

Agent Infrastructure, Software Agents, Software Engineering, Java Message Service (JMS)

1. Introduction

Large-scale enterprise-level software systems experience vigorous demands for dynamic flexible deployments, 24/7 reliability, high throughput performance and security while maintaining a high Quality-of-Service (QoS). Such requirements increase the load on a system's infrastructure. In order to cope with these circumstances a number of services are available to assist in the development of large-scale software systems. These services, generally provided by domain experts such as Sun or IBM, relieve the burden on the application developer of writing a number of complex infrastructure services needed by the system; these services include persistence, distribution, transactions, load balancing, clustering, etc.

At present, no agent platform implementation has reached the quality of commercial software applications [1]. A central theme of our research is to facilitate the use of these infrastructure services within an agent platform to allow participation within an enterprise system. Most infrastructure services are provided through frameworks and application platforms such as Java 2 Enterprise Edition, .NET, CORBA, Tibco, BEA WebLogic, IBM WebSphere, etc. In order to achieve this objective we must enable the use of such services in an agent-oriented manner, providing a gateway to bridge the differences between the paradigms.

This work is conducted within the scope of the Virtual Logistics multi-Agent Broker (V-LAB) research project at the Department of Information Technology, National University of Ireland, Galway. V-LAB uses the Java Agent DEvelopment (JADE) platform as its agent environment. JADE [2], a Foundation for Intelligent Physical Agents (FIPA) compliant software framework, is designed to assist the development of agent applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. JADE simplifies the development of a software agent while ensuring standard compliance through a comprehensive set of system services and agents. The first step in this work was to identify a generic reusable method of packaging a service for easy use from within an agent-environment. The Service-Agent Gateway (SAG) is a derivative of the Gateway design pattern that achieves this 'repackaging'. As a proof of concept, a specific SAG has been developed to provide Enterprise Message Services (EMS) to JADE agent platforms.

Development has completed on the 1.0 version of the Java Messaging Service-Agent Gateway (JMS-Agent Gateway). This has been released under the LGPL open source license, available for download from the JADE website or from <http://ecrg.it.nuigalway.ie/jade>. The remainder of this paper gives an overview of the SAG design pattern and covers the JMS-Agent Gateway architectural design, implementation, service ontology, and future development plans.

2. Enterprise Services

As enterprises continue to deploy distributed applications on ever increasing scales, transcending geographical, organisational, and traditional commercial boundaries, the demands placed upon their infrastructure increase exponentially. The process of building dynamic, highly flexible, cohesive enterprise-class distributed systems can be simplified by utilising infrastructure support services developed by domain experts such as IBM, Sun, BEA, and Oracle. Such services provide the foundations that distributed enterprise systems build upon.

Support services are available for a number of system concerns such as transactions, messaging, concurrency support, clustering/fault-tolerance, persistence, etc. Normally provided as part of an application platform or framework, these services are accessed through a propriety interface (specific to the application server or framework such as Weblogic Clustering) or a standardized interface (agreed industry standard such as Enterprise Java Beans (EJB)) from standard bodies such as the Java Community Process (JCP), Object Management Group (OMG) World Wide Web Consortium (W3C), etc. One such standard specification, the Java Message Service (JMS) Application Protocol Interface (API) [3] specification, provides a set of semantics that describe the interface and general behaviour of an Enterprise Message Service (EMS). The goal of the JMS specification is to provide a universal way to interact with multiple heterogeneous EMS in a consistent manner. The utilisation of such specifications are vital to preventing the tight coupling of an application to a particular service implementation, such 'vendor lock-in' restricts the ability of system to change service implementation. An application developed using the JMS API is able to plug-in any JMS compatible EMS, this allows the developer to choose the best EMS implementation for the application's requirements and to easily change the implementation of the EMS if new, or altered application requirements dictate.

3. Gateways: Interoperability between Disparate Systems

The gateway design pattern [4] is a fundamental technique used to bridge the differences between disparate systems. Gateways act as an intermediate program/layer that allow an external system to interact with a system in its internal (native) format.

Ideally, a software system should encapsulate interactions with a foreign system(s). Gateways are an effective mechanism of encapsulating such interaction. Gateways are commonly used when a consumer needs to be isolated from explicit knowledge of the foreign system and needs to access foreign data transparently in a friendly and local manner. A gateway can offer a system in one or more native environments. A systematic illustration of the gateway design pattern is presented in Figure 1.

1. A native representation or "proxy" of the external system is created within the local environment.
2. The local proxy implementation, when invoked by the consuming client, passes the invocation to the gateway.
3. The gateway translates any data or interaction from the local format to the equivalent format or action of the external system. (Data transformations can be as simple as renaming a field or converting a data type; they could also involve more complex structural and semantic transformations).
4. The gateway invokes the external system to perform the requested action. The gateway does not contain any data or implementation; it simply acts as a mechanism to obtain the action/data from the external system.
5. Once the operation on the external system is complete, it returns the result, if any, to the gateway.
6. The gateway then translates the result from the foreign format into the appropriate local format.
7. The results are returned to the proxy and then to the local consuming client.

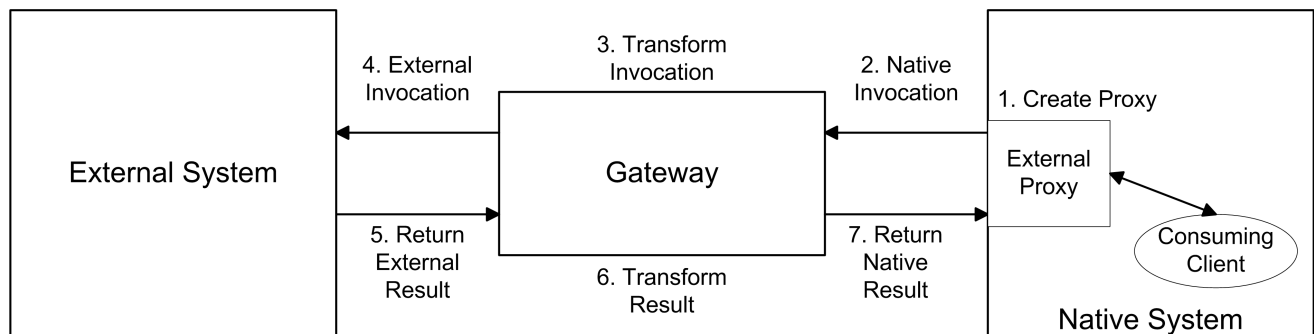


Figure 1. Gateway design patter

The gateway pattern has the obvious benefit of allowing clients access to external systems through native objects within their local environment. A number of drawbacks also exist with the overhead of running a gateway. Gateways add an additional level of complexity to the system; the effort needed to create the support infrastructure may be over-kill in a simple system. Nevertheless, the gateway design pattern offers a powerful and flexible solution to interoperability among disparate systems. Alternatives to the gateway pattern include the Remote Façade [4], Mapper [4] and Proxy [5] design patterns.

4. Service-Agent Gateway

Typically, an enterprise service will define a contract, such as the JMS API, that all service consumers must conform to in order to access the service. These contracts define the requirements for communicating with the service, such as communication protocols and message definitions. If an application is to use the service, it must fulfil its responsibilities as defined in the contract. Complying with a service contract requires you to potentially implement a number of mechanisms to perform interactions with the service, such as authentication, message marshalling, encryption, service configuration, connection management, and transactional responsibilities. A service gateway encapsulates the code that implements the consumer portion of the service contract, acting as a proxy, to provide a clean interaction with the service. This encapsulation of the service contract reduces the effort required to maintain updates to the contract.

The Service-Agent Gateway (SAG) is a specific instance of the gateway pattern that has been adapted for use with agent-oriented systems. The service-agent gateway adapts an external service to work in an agent friendly manner using FIPA interaction standards. The gateway presents interactions with the service in terms of its operation rather than in terms of communication, infrastructure, and security protocols. This process adapts the consumer’s calling semantics into the calling semantics of the services contract.

4.1 Architecture

The role of the service-agent gateway, as illustrated in Figure 2, is to provide a bridge over which non-agent based services can be offered to agents in an agent-oriented style. The gateway is responsible for receiving agent requests for the service through its service facilitator (proxy) agent; this facilitator agent exists within the local agent environment. The service facilitator agent is responsible for fulfilling requests from its consuming agent. Interactions with the facilitator are carried-out using FIPA Interaction Protocols (IP) and a specific service-agent ontology defined to describe the service offered by the gateway. The service facilitator agent may register with the platform directory facilitator to offer its services to agents within the platform.

When the service facilitator agent receives a request, it forwards it to the gateway. The gateway converts the agent’s request into the relevant corresponding service action. The gateway is responsible for implementing the interaction with the external service, setting up any required infrastructure such as connections, security, transactions, etc. When the call to the external service returns/completes (both synchronous and asynchronous interactions are supported) the gateway transforms any results back into an agent format. The transformed result is passed back to the service facilitator agent for delivery to the consuming agent in the local environment.

As a proof of concept for the SAG design pattern, a specific SAG has been developed to provide enterprise message services via the JMS specification. This JMS-Agent gateway allows agents to access any JMS-compatible EMS implementation such as WebSphereMQ [6] (formulary MQSeries), TIBCO [7] and OpenJMS [8].

5. Java Message Service-Agent Gateway

Previous work on integrating an EMS with an agent platform focused on the implementation of an EMS enabled Message Transport Protocol (MTP) for the JADE platform. This JMS-MTP [9] allowed inter-platform messages to be sent transparently over an EMS.

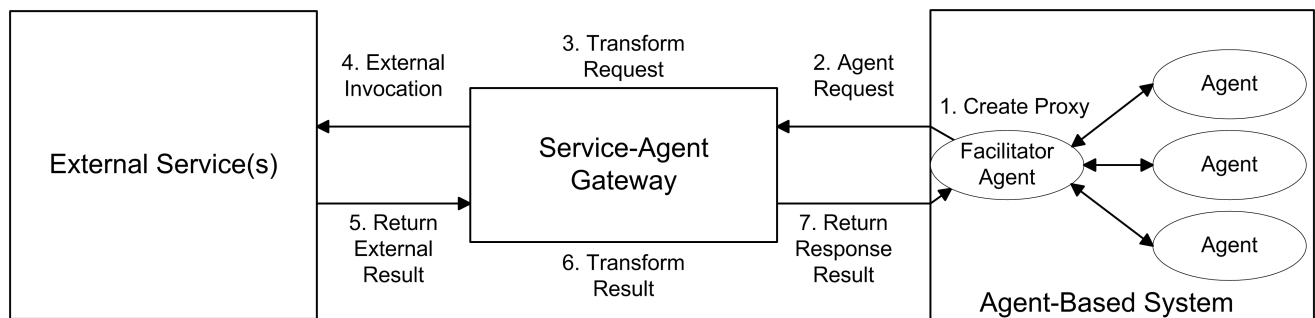


Figure 1. Service-Agent Gateway

The JMS-MTP is a flexible solution for interconnecting agent platforms as it leverages the benefits of EMS in the areas of decoupling, scalability, system availability, and cohesiveness. This reduces the effort required to add and remove participants from an interconnected group of agent platforms.

This platform-level solution does not allow individual agents to interact directly with the EMS. Agents are limited in their ability to utilise the messaging models and features present within an EMS.

5.1 Message Models

One of the major benefits of utilising an EMS emanate from its messaging models. Two main message models are commonly available, the point-to-point and publish/subscribe models. These models center on the exchange of messages through a channel (queue). A typical system will utilize a mix of these models to achieve different messaging objectives.

The point-to-point messaging model provides a straightforward asynchronous exchange of messages between participants. In this model, messages from producing clients are routed to consuming clients via a First-In First-Out (FIFO) queue, as messages are consumed they are removed from the head of the queue. Each message is delivered only once to only one receiver. The model allows multiple receivers to connect to the queue but only one of the receivers will consume the message; commonly referred to as ‘once-and-once-only’ message delivery.

The publish/subscribe messaging model is a very powerful mechanism used to disseminate information between anonymous message consumers and producers. This one-to-many and many-to-many distribution mechanism allows a single producer to send a message to one user or potentially hundreds of thousands of consumers. Clients producing messages “publish” to a specific topic or channel, these topics are then “subscribed” to by clients wishing to consume messages. The service routes the messages to consumers based on the topics to which they have subscribed as being interested in. This messaging model can assist in the construction of wide-scale information dissemination systems such as news services (current affairs, sports, financial, entertainment, etc), stock tracking information, classified ads, bulletin boards, and even process control systems.

5.2 Gateway Overview

The role of the JMS-Agent gateway is to provide a bridge over which non-agent EMS are accessible to agents in an agent-oriented fashion. The gateway is responsible for receiving agent requests for the JMS provider through its JMS facilitator agent; it must then convert this request into the JMS action corresponding to

the agent request. The gateway is responsible for making the connection to the JMS provider to perform subscription and message processing tasks on behalf of its clients.

The JMS facilitator agent is responsible for fulfilling requests from its consuming agents; possible requests include subscription administration (adding, removing), publishing messages to JMS destinations, and forwarding messages to consuming agents based on their subscription requests. Interactions with the facilitator are carried-out using the JMS-Agent Gateway ontology defined in the following section. Publishing requests made of the facilitator by client agents follow the FIPA Request interaction protocol [10] and agent subscription requests follow the FIPA Subscribe [11] interaction protocol.

The gateway design uses the JMS API; this enables any JMS compatible EMS to interact the gateway. Platform administrators are able to choose the most appropriate EMS implementation for their agent platform, and to easily change this implementation if required.

5.3 Ontology

The ontology used with the JMS-Agent Gateway changes the task-oriented JMS API (connect to service, configure connection, find topic, subscribe, send message etc) into a goal-oriented interaction (subscribe, send message). The tasks or functions of the ontology derive from the main interactions of the API; the task of setting up low-level support infrastructure is now the responsibility of the service gateway. The configuration of this low-level infrastructure may still be directed by the consuming agent via embedded settings within the ontology, such settings include provider connection information, message persistence, filters, message time-to-live, etc. As an illustration of this transformation, the following code sample implements the *PublishMessage* task of the ontology.

The *PublishMessage* task of the ontology directs the JMS facilitator agent to publish the messages contained within the *JmsMessage* frame to the destination defined in the *ProviderInfo* frame. These frames contain all the required information needed to publish the message, allowing the agent to direct the facilitator’s actions when processing the message. Agents are now able to interact with an EMS using this high-level ontology, moving the burden of implementing low-level service contracts to the gateway. The ontology also supports tasks for subscription requests and message receiving; a full list of the ontology’s frames and tasks is available from <http://ecrg.it.nuigalway.ie/jade>.

When transforming a JMS interaction into the equivalent ontology format the payload of a message is unaltered. Currently the gateway supports messages of type `javax.jms.TextMessage` and `javax.jms.ObjectMessage`, such message formats can be used to carry FIPA Agent Communication Language (ACL) messages with different FIPA Content Languages

```

// Setup Initial Context
properties.put(Context.INITIAL_CONTEXT_FACTORY, providerICF);
properties.put(Context.PROVIDER_URL, providerURL);
Context context = new InitialContext(properties);

// Create Connection
QueueConnectionFactory qcf = (QueueConnectionFactory)
context.lookup("JmsQueueConnectionFactory");

Connection conn = qcf.createQueueConnection(username, password);
conn.start();

// Create Session and Queue
Session session = conn.createQueueSession(false, QueueSession.AUTO_ACKNOWLEDGE);
Queue que = session.createQueue(destination);

// Create and configure MessageProducer
MessageProducer msgProd = session.createSender(que);
msgProd.setTimeToLive(5000000);
msgProd.setPriority(5);
msgProd.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

// Create and configure message
Message message = session.createTextMessage();
message.setText("Test Payload");
message.setStringProperty("Foo", "Bar");

// Send Message
msgProd.send(message);

// Cleanup
conn.close();

```

Table 1. Frame representation of a JMS Message

Frame Ontology	JmsMessage ECRG-JMS			
Parameter	Description	Presence	Type	Reserved Values
destination	Message Destination	Mandatory	String	
priority	Message Priority.	Mandatory	Integer	0 – 9
timeToLive	Time-to-Live value of message	Mandatory	Integer	0
deliveryMode	Message delivery mode	Mandatory	Boolean	Persistent / Non_Persistent
payload	Message Payload	Optional	String	
properties	Message Properties	Optional	Property	

Table 2. Task to Publish a message to a destination

Task	PublishMessage
Ontology	ECRG-JMS
Supported by	JMS Facilitator Agent
Description	The execution of this task requests for the accompanying JMS message to be published.
Frames	ProviderInfo, JmsMessage
Dependency	The task has no dependency and can be executed at any time.

(CL) such as the Semantic Language (SL) [12] and the Resource Description Framework (RDF) [13]. Potentially, a message payload could undergo a transformation into an ACL format, however such transformations are application specific and would require an application or agent-specific gateway, the goal of this work is to provide a generic JMS-agent gateway.

The approach used to develop the JMS-gateway ontology could also be used to develop an ontology for a number of other services such as the Java Naming and Directory Interface (JNDI) [14], transactional services/monitors, accessing web services, Enterprise Application Integration tools and servers, Enterprise Resource Planning Systems, Java Connector Architecture adaptors [15], etc.

6. Future Plans

A number of enhancements for the JMS-Agent Gateway are possible, increased support for JMS message formats and message property formats are planned. Potential new features include support for the transformation of incoming message payloads into ACL/CL and the reverse transformation of out-going messages. Such transformations are application/domain specific and could be performed using an eXtensible Mark-up Language (XML) transformation technology such as eXtensible Stylesheet Language: Transformations (XSLT).

The next stage of our research is to focus on the definition of tools and techniques to further enable a streamlined integration of infrastructure services and agent-based systems. Current investigations include work on a tool to automatically generate (partial) service-agent gateways based on the API specification of the service and its consumer contract. This tool can potentially generate the ontology, facilitator agent, and various utility methods for consuming agents.

7. Conclusions

This paper describes work on developing Service-Agent Gateways (SAG); such work forms the necessary next step in integrating agent-based systems into enterprise applications.

A key obstacle to the widespread deployment of agent technology is the relative immaturity of the technology with regard to enterprise capabilities such as scalability, federation, persistence, transactions, security, deployment lifecycle, management, messaging, and integration with legacy systems [1]. We believe the solution to this problem is to utilise the enterprise-level infrastructure services employed in current systems. To achieve this, we purpose the Service-Agent Gateway design pattern to offer these services to an agent environment. This pattern has been used to offer the functionality of Enterprise Message Services (EMS) to the JADE agent platform.

The JMS-Agent Gateway opens the world of EMS and their messaging models to agent-based systems. Such services are used to build highly reliable, scalable, and flexible distributed applications. EMS provides two powerful messaging models for the dissemination of information. Agents are now able to utilise these models, via the gateway, in an agent-oriented manner. With direct access to these models, agents can now easily interact with a variety of information dissemination services including news services (current affairs, sports, financial, entertainment, etc), stock tracking information, classified ads, bulletin boards, etc. The Service-Agent Gateway design pattern enables a streamlined integration of an enterprise service with an agent platform, enhancing their acceptability within business environments and full-scale participation in the digital enterprise.

8. Acknowledgements

The support of the Informatics Research Initiative of Enterprise Ireland is gratefully acknowledged.

References

- [1] Cowan, D. and M. Griss, Making Software Agent Technology Available to Enterprise Applications. HP Labs Technical Reports, 2002.
- [2] Bellifemine, F., A. Poggi, and G. Rimassa. JADE - A FIPA-compliant agent framework. in PAAM. 1999. London.
- [3] Hapner, M., et al., Java Message Service Specification v1.1. 2002, Sun Microsystems, Inc.
- [4] Fowler, M., Patterns of Enterprise Application Architecture. 2002: Addison-Wesley. 560.
- [5] Gamma, E., et al., Design Patterns: Elements of Reusable Object-Oriented Software. 1995: Addison-Wesley.
- [6] Gilman, L. and R. Schreiber, Distributed Computing with IBM MQSeries. 1996, New York: John Wiley.
- [7] Skeen, D. An Information Bus Architecture for Large-Scale, Decision-Support Environments. in USENIX Winter Conference. 1992.
- [8] OpenJMS, ExoLab Group.
- [9] Curry, E., D. Chambers, and G. Lyons. A JMS Message Transport Protocol for the JADE Platform. in IEEE/WIC International Conference on Intelligent Agent Technology. 2003. Halifax, Canada: IEEE Press.
- [10] FIPA, Request Interaction Protocol Specification. 2002.
- [11] FIPA, Subscribe Interaction Protocol Specification. 2002.
- [12] FIPA, SL Content Language Specification. 2002.
- [13] FIPA, RDF Content Language Specification. 2001.
- [14] Java Naming and Directory Interface API and SPI Specifications. 1999, Sun Microsystems.
- [15] J2EE Connector Architecture Specification. 2003, Sun-Microsystems.