Using Agent Technology as a Support for an Enterprise Service Bus

Paola Mordacci, Agostino Poggi, Carmelo Giovanni Tiso, Paola Turci

***Abstract* — The research in distributed artificial intelligence has been addressing for several years the problem of designing and building coordinated and collaborative intelligent multi-agent systems. This interesting and advanced work can be fruitfully exploited in the area of service-oriented computing if agent technology is appropriately engineered and integrated with the key technologies in this field.**

To support this claim, in this paper we show how the agent technology integrated in an Enterprise Service Bus allows the conception and realization of real flexible, adaptive intelligent service-oriented systems.

***Index Terms* — Multi-agent systems, service oriented architecture, ESB, rule engine.**

I. Introduction

Multi-agent systems and service-oriented computing are still evolving towards a complete maturity. The quite wide spreading of the service-orientation design paradigm and of the related technologies is having a twofold influence on the evolution of agent technology.

On the one hand, several researchers belonging to the agent community are convinced that this technical area is a natural environment in which the agent technology features can be leveraged to obtain significant advantages. It is plain, in fact, that service-oriented technologies cannot provide by themselves the autonomy and social and proactive capabilities of agents. Agents, taking advantage of their social ability, exhibit a flexible coordination that makes them able to both cooperate in the achievement of a global goal and compete in the distribution of resources and tasks.

On the other hand, one of the requirements for the success of multi-agent systems is that they have to guarantee an easy integration with other widely used industrial technologies.

Driven by such motivations, a number of research works have been undertaken with the aim of tackling the problem of integrating service-oriented technologies with multi-agent systems.

The aim of this paper is in a slight different direction, that is we try to cope with the issue of adding collaboration and coordination capabilities in a service-oriented architecture. In particular, we have considered an enterprise service bus (ESB) - a software infrastructure that facilitates the realization of SOA systems by acting as a middleware through which a set of services are made available - and we have shown how the integration of agent technology in an ESB may be considered very promising.

The brief survey of the literature in the area of service-oriented technologies, reported in the background section, has the aim of showing the scenario in which the agent possibly contribution should be set and at the same time to give a short preamble acting as a motivation and rationale of the research work that we have done. Section 3 gives an overview of the related research. Section 4 deals with the integration of JADE agents in ServiceMix, an open source ESB based on JBI specifications. Section 5 describes a simple but realistic application which shows how the powerful synergism between agent and service-oriented technologies could be very promising. The paper ends by drawing some conclusions around the results of the work done.

II. Background

The growing demand for high-levels of interoperability by organizations that want applications to have broader reach, have stimulated the rapid growth of novel standards, technologies and paradigms with the aim of giving an answer to such problem. The most appropriate response to this need seems a service-oriented architecture (SOA), i.e. a system assembled from a loosely coupled collection of services and in particular of Web services - the integration technology preferred by organizations implementing SOA (Dustdar & Schreiner, 2005),

For the sake of clarity, it is necessary to say that there is no one recognized definition of SOA, however a baseline of concepts and principles and a strategic vision have emerged and collectively characterize the service-oriented design paradigm as an approach to defining integration architectures based on the concept of service.

The last outcome of the SOA movement has been the ESB, an infrastructure that can be used as a backbone upon which to build service-oriented applications. As with SOA, there has been no industry agreed on the definition of ESB so far. It is

P. Mordacci is a student at DII, University of Parma, Viale Usberti 181A, 43100, Parma, Italy (e-mail: paola.mordacci@studenti.unipr.it).

A. Poggi is with DII, University of Parma, Viale Usberti 181A, 43100, Parma, Italy (phone: +39 0521 905728; e-mail: poggi@ce.unipr.it).

C. G. Tiso is a student at DII, University of Parma, Viale Usberti 181A, 43100, Parma, Italy (e-mail: carmelo.giovanni.tiso @studenti.unipr.it).

P. Turci is with DII, University of Parma, Viale Usberti 181A, 43100, Parma, Italy (phone: +39 0521 905708; e-mail: turci@ce.unipr.it).

still a controversial issue if it is a pattern, a product or an architectural component. According to the authoritative Gartner's definition: "an ESB is a new architecture that exploits Web services, messaging middleware, intelligent routing and transformation". Anyway, one thing seems to be unquestionable; using an ESB is the quickest and most cost-effective way to address the challenge of the enterprise application integration. Several middleware vendors provide or have on their roadmap an ESB.

However, a still open problem is that the information and the research activities in this area are quite fragmented (Papazoglou et al., 2006). What clearly emerges is that the subject of service-oriented applications turns out to be vast and enormously complex and more work needs to be done in order to realize real flexible, adaptive intelligent service-oriented systems. As a matter of fact, current SOA implementations are still restricted in their application context to being an in-house solution for companies (Domingue, 2008).

In the attempt to delineate an effective solution, some researchers have envisaged as strategic the integration of SOA with both semantic and Web technologies (Domingue, 2008). Others have turned their eyes towards the agent technology, as an interesting means for the realization of more effective and reliable service-oriented systems and for SOA to be successful on a worldwide scale. Clearly, such technologies are not competitive but complementary and therefore someone else has been developing systems which integrate SOA, semantic Web and multi-agent systems (Negri, 2006).

The agents ability of operating in dynamic and uncertain environments allows coping with the usual problems of failures or unavailability of services and the consequent need of finding substitute services and/or back tracking the system in a state where execute an alternative workflow. Moreover, the capabilities of some kinds of agent of learning from their experience make them able to improve their performance over the time avoiding untrusted and unreliable providers and reusing successful solutions. These remarkable agents' features have mainly driven the research activities of the agent community in the area of service-oriented computing so far, that is agents as a valuable support for realizing Web services composition.

Other interesting features, which can be the main ingredients for automatic cooperation between enterprise services, are the agents' capabilities of collaborating and coordinating themselves. In a business environment, an example would be a broker that has frequently to seek providers as well as buyers dynamically, to collaborate with them and finally to coordinate the interactions with and between them in order to achieve its goals. An intelligent service-oriented infrastructure could do it automatically or semi-automatically, within the defined constraints. These further agents' characteristics have been considered in the research work presented in this paper.

III. Related Work

The problem of realizing service-oriented applications exploiting agent technology has mainly concerned so far three fundamental issues: (i) the management of the interactions between agents and Web services; (ii) the execution of a workflow or more in general of a plan that describes how Web services interact; (iii) the discovery of the Web services that perform the tasks required in the plan.

In other words, to date the efforts have been mainly devoted to provide an effective solution to the problem of Web services composition.

In this context, the first issue that the agent community has had to cope with was the mapping between the different semantic levels of the two paradigms or patterns of communication (Greenwood & Calisti, 2004; Nguyen, 2005; Shafiq et al., 2005).

The next step has been the implementation of prototypes of agent based frameworks coping with the static and dynamic composition of Web services, through the use of workflow technologies (Buhler & Vidal, 2005). In fact, once the infrastructure, enabling a bi-directional connectivity between the two technologies, is in place an agent can play the role of the orchestrator of dynamic Web service compositions. Even if the current multi-agent solutions, aiming at realizing an effective agent-based service composition, are still in a preliminary phase and certainly need to be improved (Savarimuthu et al., 2005), a lot of researchers and software developers are really interested in giving a significant contribution in this direction.

As far as the automatic cooperation between enterprise services is concerned, to the best of the authors' knowledge, there are very few ongoing studies. The most significant is the one reported in (Paschke et al, 2007). In this work, the authors combine the ideas of multi-agent systems, distributed rule management systems and service-oriented and event-driven architectures. The work is mainly focused on the design and implementation of a pragmatic layer above the syntactic and semantic layers. Taking advantage of this layer, individual agents can form virtual organizations with common negotiation and coordination patterns. An enterprise service bus is integrated as a communication middleware platform and provides a highly scalable and flexible application messaging framework to communicate synchronously and also asynchronously with external services and internal agents. To date, the authors have developed an interesting methodology and an architectural design but they have only outlined a possible implementation of the system.

IV. Integration of JADE in Servicemix

Apache ServiceMix (ServiceMix) is an open source ESB released under the Apache license, based on the Java Business Integration (JBI) standard (JBI, 2005). These two factors, open source and standard-based, allow for low entry cost, maximum flexibility, reuse and investment protection.

ServiceMix is lightweight and easily embeddable, integrates

Spring support and can be run at the edge of the network (inside a client or server) as a standalone ESB provider or as a service within another ESB.

ServiceMix includes a complete JBI container supporting all parts of the JBI specification: allows plug-in services (as configuration of JBI components) which can be combined to create a SOA; provides a Normalized Message Router (NMR), as the backbone of all communication (based on the exchange of normalized messages having an xml content) between services within the ESB; supports the four JBI message exchange pattern (i.e. protocols defining the messages exchanged between a service consumer and a service provider, involved in a service invocation); includes full support for the JBI deployment units with hot-deployment of JBI components.

ServiceMix distribution already includes many JBI components that provide support for some of the most common protocols and engines. JBI components are deployed to the JBI container and simply run in memory waiting for configuration. In fact, in order to make use of these components as an application developer, one has to provide a configuration for each component he/she intends to use.

ServiceMix therefore strongly encourages the "configure don't code" integration approach, that allows a faster and simpler (but less flexible..) integration.

Configurations are implementation specific but the packaging is defined by the JBI specifications. Each component configuration must be packaged as a Service Unit (SU) and each SU must be wrapped in a Service Assembly (SA). These are simply ZIP/JAR files that contain an XML descriptor.

According to the JBI specification, JBI components come in two flavours: Binding Component (BC) and Service Engine (SE).

ServiceMix BCs are used to communicate outside the JBI environment by means of a lot of supported remote protocols: HTTP/S, HTTP+SOAP, JMS, FTP, SMTP, XMPP, etc.

BCs are responsible for normalizing incoming (relative to JBI environment) messages and denormalizing outcoming messages.

ServiceMix SEs provide some type of logic inside the JBI environment. Some examples of SE components in ServiceMix include: rules engines, BPEL engines, XSLT engines, Plain Old Java Object (POJO), annotated POJO, schema validation of documents, support for Enterprise Integration Pattern, etc.

According to the "configure don't code" integration approach, to integrate JADE agents in an ESB is opportune to select the most appropriate ServiceMix component to configure, in order to obtain a service satisfying our goal. The basic idea is to build a service, acting as a proxy between the NMR and the agent community, that can interact with both services deployed in the JBI environment and JADE agents belonging to the agent community. By means of this proxy, services deployed in ServiceMix ESB can access capabilities offered by agents. Services can send normalized messages to the proxy service that can map them in ACL messages and forward them to a specific agent within the community. On the other hand, the proxy can receive requests from the agent community, map them in normalized messages and forward them to a specific service exposed in the bus. To make this possible, each proxy needs to be in relation to an associated agent that represents a proxy gate towards the agent community

A possible solution is based on the ServiceMix jsr181 component: a JBI SE exposing POJO as services on the bus.

When a normalized message, addressed to one of such services, arrives from the bus, an appropriate method of the POJO will be called, passing the service invocation parameters (specified as an xml content of the message) as method parameters. The request is forwarded in some way within such method to its associated agent (further details will be explained below). We discarded this option because the marshalling of the normalized message content is handled automatically and in a quite rigid manner

The chosen solution is based on the ServiceMix bean component, since this is a very flexible component that can receive messages from the NMR and process them in any way it likes. The bean component gives the developer the freedom to create any type of message handling but as a counterpart she/he has to hand coded all the way.

In order to configure this component one has to define a simple XML file, where the service name, and a Java class representing the bean are mainly defined. These two artefacts have to be packaged as a SU.

When a normalized message, addressed to such a service, arrives from the bus, an appropriate method of the bean will be called, passing the message as a method parameter. The message will be processed and the requested actions will be taken. Using JBI API the bean, in turn, can send the message towards other services.

The proxy service implementation is therefore accomplished by configuring the ServiceMix bean component, instantiating within the bean constructor its associated JADE agent that, using JADE API, will be executed within a new secondary container. It is therefore necessary that first JADE main container is executed, then the bean is initialized. The connection between the two entities is handled by means of references; the bean holds a reference to the agent and the agent holds a reference to the bean.

Within a bean method, invoked subsequently to the reception of a message by the proxy service, it is so possible to call an appropriate agent method that, in turn, can interact with whatever agent belonging to same or other FIPA communities.

Similarly, any agent can send an ACL message to the proxy agent that by means of the proxy service can interact with services exposed in the bus.

V. On-line Book Selling

The framework has been experimented in the realization of an online book selling application where there are N book sellers and one broker. The broker is responsible for providing its users with elementary and aggregated information collected through the collaboration with sellers.

In Figure 1, the full system architecture (with N=2) is shown.

The sellers and the broker are distributed in the network. Each entity, i.e. the coupling agent-service, has an associated ServiceMix JBI container in which a number of JBI components are deployed. The blue rectangles represent services as configuration of a JBI component. The particular services, that is the sellers or the broker, act as a proxy between the NMR and the agent community. The agent associated to each proxy (i.e. respectively agent seller or agent broker) is linked to its proxy by a dotted line. A seller maintains book data within a relational database that is handled by an agent seller or possibly another agent of the community. Finally, in the figure it is also highlighted how all interactions between services within a JBI container are mediated by the NMR.

Each seller entity provides its users with the following features: (i) an easy support to obtain the complete book list offered by the seller; (ii) the possibility, if some constraints are met, to obtain a discounted price of a specified book; (iii) the opportunity to add a new book to the seller's book database.

Users can express a request for a seller book list by means of a web interface: the http request is received by the http-book-list service (a configuration of ServiceMix BC servicemix-http).

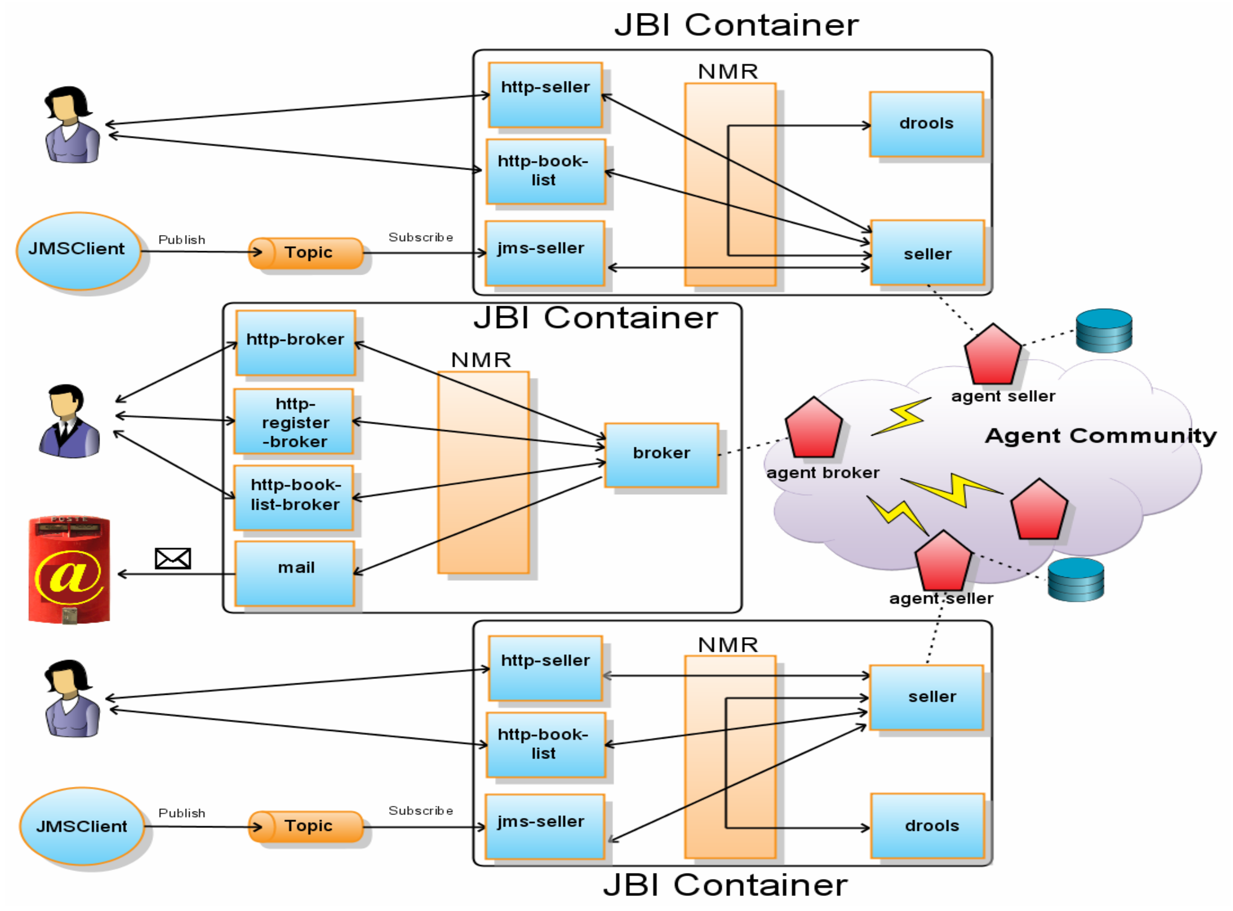

Figure 1- On-line book selling system architecture

This service maps the received http request in a message normalized content (an XML document), that is sent to the seller service (as already mentioned, a configuration of ServiceMix SE servicemix-bean).

This proxy forwards the request (mapped in an ACL message) to the agent seller, which fulfils the request or delegates the agent acting as a database handler.

The agent seller returns the response, an XML document containing the book list, to the proxy that, in turn, maps such response in a normalized message, then returned to the http-book-list service. At last, this service maps the received message in a proper HTTP response. Once the user has obtained the seller book list, he/she can request the final price (i.e. the price after the discount) of a selected book, by means of a web interface. It is necessary to specify, besides the book title, the user nickname and the desired quantity, since in this simple application the final price of a book is determined by a set of business rules depending on this information.

The HTTP request, containing the user specified parameters, is received by the http-seller service (a configuration of ServiceMix BC servicemix-http).

Similarly to the http-book-list service, this service maps the received http request in a message normalized content, that is sent to the seller service. The seller service asks its linked agent seller the book base price that subsequently will be forwarded, together with user's data, to the drools service, a configuration of ServiceMix SE servicemix-drools.

Drools is a business rule management system with a forward chaining inference based rules engine, more correctly known as a production rule system, based on an enhanced implementation of the Rete algorithm. Drools is a JBoss open source project compliant to the JSR-94 standard for business rules engine.

A rule engine is mainly based on two concepts: a set of rules (the actual logic) and assertions (facts accessed by rules). Rules are coded in a drl file, following the Drools syntax. Assertions can be passed through Java.

To configure a servicemix-drools is necessary to define, besides an xbean.xml file, a drl file containing the business rules. Each normalized message received by a drools service is processed by the drools service as an assertion. A drl file can access JBIHelper class methods to interact with NMR (e.g. it is possible to create normalized message and send them to a specified destination).

In such way, the drools service can answer the request received by a seller service replying with a message containing the discount, determinate according to the codified business rules.
To add a new book to the seller's book database, an external application, that publishes a JMS message (containing new book data) on a specified JMS Topic, is used. Such message is withdrawn by jms-seller service (a configuration of ServiceMix BC servicemix-jms) representing a JMS TopicSubscriber. This service maps the received message in a new message addressed to the seller service that will forward it to the seller agent or through it indirectly to the agent acting as book database handler.

The broker entity provides his users with the following features: (a) a support to obtain the complete book list, as the union of seller book lists; (b) the possibility to know the seller that offers a selected book at the best final price; (c) subscription to broker's newsletters; (d) notification by e-mail of new books.

Cooperating with the agent sellers in accordance with the FIPA protocols, the agent broker can collect the information necessary to provide the aforementioned functionalities.

Users can request the complete book list by means of a web interface: the http request is received by the http-book-list-broker service (a configuration of ServiceMix BC servicemix-http). Similarly to the seller case, this service maps the received http request in a normalized message, that is sent to the broker service (a configuration of ServiceMix SE servicemix-bean: the broker proxy).

This proxy forwards the request to the broker agent, that will interact with every seller in order to collect the various lists. The broker agent builds the union of the seller book lists, returns it to the proxy and so on until a proper HTTP response is sent to the requester.

Once the user has obtained the complete list, he/she can ask the broker which seller offers the selected book at the best final price. For the same reasons explained above in the seller case, the user has to specify, besides the book title, the user nickname and the desired quantity.

The HTTP request is received by the http-broker service and then sent to the broker service. This proxy forwards the request to the agent broker, which collects information from every agent seller, offering the requested book. Once agent broker receives a response from each seller or a timeout is reached, returns the best received price, through the proxy, to the user.

An http-register-broker service has been made available to allow users to register with broker's newsletter. Users need to compile an html form specifying their name and e-mail.

As usual, this service maps the received http request in a normalized message, which is sent to the broker service. Such service will forward it to the agent broker, that consecutively will delegate the agent responsible for handling users' data to add the new user to the database.

Finally, when agent seller receives notification about the insertion of a new book, it sends an ACL message containing the new book data to the agent broker and to the agent in charge of handling the book database. By means of the broker service, the agent broker, once it has collected user data from the agent responsible for handling user database, sends a normalized message to the mail service (a configuration of ServiceMix BC servicemix-mail), responsible for sending an e-mail to each registered users

VI. Conclusion

In this paper, we have addressed the integration of multi-agent systems in an ESB, highlighting the benefits that both communities can achieve from this solution. On the one hand, multi-agent systems are able to interact with the key emerging technologies in the area of service-oriented computing. On the other hand, the interesting and advanced work carried out in several years by the agent community can be fruitfully exploited in the area of service-oriented computing.

Finally, we have tried to prove, by means of a simple but realistic application, how the powerful synergism between these technologies could be very promising.

Besides the significant role that collaboration and coordination capabilities play in our application, another important point that clearly emerges from our application is that the agent technology can make successful the exploitation of the ESB technology, based on JBI specification, on a worldwide scale. To date, the ESB technology does not provide a support for the federation of distributed JBI container; each JBI container can interoperate with the others as with remote consumers or providers, that is by means of communication protocols supported by binding components.

REFERENCES

[1] Buhler P.A., Vidal, J.M. (2005). Towards Adaptive Workflow Enactment Using Multiagent Systems. Information Technology and Management, 6(1):61-87

[2] Greenwood D. & Calisti M. (2004). Engineering Web Service-Agent Integration. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pages 1918–1925. The Hague, Netherlands.

[3] Domingue J. & Fensel D. (2008) Toward a Service Web: Integrating the Semantic Web and Service Orientation, IEEE Intelligent Systems January/February, 2008.

[4] Dustdar, S., & Schreiner, W. (2005). A survey on web services composition, Int. J. Web and Grid Services, Vol. 1, No. 1, pp.1–30

[5] JBI Java Business Integration 1.0, Final Release August, 2005. available from http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html

[6] Negri A., Poggi A., Tomaiuolo M., Turci P. (2006). Agents for e-Business Applications, In AAMAS ’06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems. (pp. 907-914). Hakodate, Japan. ACM Press

[7] Nguyen X. T. (2005). Demonstration of WS2JADE. In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pages 135–136. Utrecht, The Netherlands

[8] Papazoglou1 M., Traverso P., Dustdar S., Leymann F. (2006). Service-Oriented Computing Research Roadmap.

[9] Paschke, A., Boley, H., Kozlenkov, A., Craig, B. (2007). Rule Responder: RuleML Based Agents for Distributed Collaboration on the Pragmatic Web. 2nd International Conference on the Pragmatic Web Oct 22-23, 2007, Tilburg, The Netherlands.

[10] Savarimuthu B. T. R., Purvis M., Purvis M. & Cranefield S. (2005). Integrating Web Services with Agent Based Workflow Management System (WfMS). In WI '05: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence. (pp. 471 – 474). Washington, DC. IEEE Computer Society.

[11] Servicemix, available from http://servicemix.apache.org/

[12] Shafiq M. O., Ali A., Ahmad H. F., Suguri H. (2005). AgentWeb Gateway - a Middleware for Dynamic Integration of Multi Agent System and Web Services Framework. In Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, pages 267–270, Washington, DC. IEEE Computer Society