Mediation and Enterprise Service Bus
A position paper

Colombe Hérault, Gaël Thomas, and Philippe Lalanda

Université Joseph Fourier,
Laboratoire Logiciels Systèmes Réseaux, Équipe Adele
F-38041 Grenoble Cedex 9, France
firstname.name@imag.fr

Abstract. Enterprise Service Buses (ESB) are becoming standard to allow communication between Web Services. Different techniques and tools have been proposed to implement and to deploy mediators within ESBs. It turns out however that current solutions are very technology-oriented and beyond the scope of most programmers. In this position paper, we present an approach that clearly separates the specification of the mediation operations and their execution on an ESB. This work is made within the European-funded S4ALL project (Services For All).

1 Introduction

The integration of business activities is a long standing problem that has been tackled with different approaches (asynchronous middleware, Enterprise Integration application, etc.). The integration issue is perceived today through a new angle with the need to integrate distant applications available on the Internet. This raises challenging new problems related to communication over a public network, security and of course interoperability.

Service-oriented architectures constitute a very promising approach to integrate Internet applications: they actually provide the level of flexibility and scalability required to build industrial e-applications. However, service-oriented computing is today essentially technology-driven. Most available platforms focus on the technology allowing to publish and compose services and to make them communicate (i.e. SOAP [1], WSDL [2], UDDI [3], etc.). Different services may manipulate similar data or services under very different formats. The problem of data and interfaces heterogeneity is not directly addressed and left to the e-applications programmers.

As a remedy to this issue, several research areas are explored, including work about the semantic Web and ontologies[1] [4]. In this paper, we focus on an emerging middleware called Enterprise Service Bus (ESB [5] [6]). ESBs are providing technological solutions to intercept messages between Web Services and to translate or route them to help the integration of business applications.

[1] http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

In this paper, we argue that this technological solutions must come with models and tools allowing developers to describe the mediation operations at a higher level of abstraction. This work is carried out within the European ITEA project S4ALL (see www.itea-office.org), which aims at creating tools and models to build services using data from many heterogeneous sources from IT servers to industrial devices.

The paper is organized as it follows. The next section describes the notion of mediation and generalizes this notion to different non-functional behaviors. The section 3 presents a toy application as part of the European S4ALL project. The section 4 presents the global vision that we plan to implement. Finally, the section 5 concludes this paper.

2 From mediation to ESB

The concept of mediation is primarily an answer to the lack of interoperability between clients and data sources in Information Systems. Early mediation solutions evolved in order to also enhance the global quality of service provided by large scope of database systems. More recently the ESB (Enterprise Service Bus) concept has emerged in the context of B2B (Business to Business [7]). The ESB incorporates the concept of mediation to facilitate the design of application based on Webservices.

2.1 Mediation

Definitions As defined by G. Wiederhold in [8, 9], mediation is *"a layer of intelligent middleware services in information systems, linking data resources and application programs"*. the integration of various data resources (databases, Webservices or devices) and application programs (Webservices, enterprise applications, etc) raise a number of issues, essentially due to heterogeneity (see fig. 1).

The mediation layer is made of many mediators that are light weight components (e.g. independent black boxes that can be composed) and are composed to form mediation chains between client applications and data sources. G. Wiederhold defines a mediator as *"software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications. It should be small and simple, so that it can be maintained by one expert or, at most, a small and coherent group of experts"* [8].

Mediators capabilities Mediation includes tasks such as transformation and synthesis of data that can go from basic format translation in order to match a particular standard, to more sophisticated analysis using ontology or expert knowledge, adding new values to the data. Synthesis may be done over multiple data sources, having heterogeneous data type. For example, one may have to apply a currency conversion between services from different countries or may want to extract the global evolution of the activity of a plan over a week period

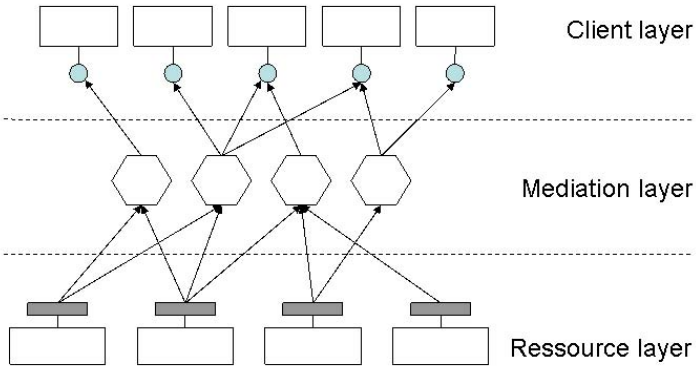

Fig. 1. Mediation layer

from a database containing daily information.
The aim of mediation is thus to abstract data and extract the domain knowledge [8] in order to ease the decision making by providing operations such as :

- selection upon different data;
- transformation from a data type to an other;
- integration of different data sources;
- selection of data source when there are many data sources available;
- resolution of inconsistent data.

Mediation added value The mediation code can be seen as code that is particular to a specific exchange between a client and a data source. Motivations to dissociate mediation code and business code are obviously to increase the separation of concerns and to decrease the coupling between client and data sources. Mediators are seen as elements which are not really part of the client, nor part of the data source but much more of the "binding" between them. Mediation improves:

- reusability : you may reuse a particular mediator in several mediation chains, for example an XML validation service.
- evolution of code : when a new client or a new data source appears, or even when there is a simple modification of one of them, it is not necessary to modify the code its interlocutors; only a new mediation element is added to the mediation chain.
- scalability : as mediation allows to integrate new clients and data sources progressively.

Using mediation concepts makes it easier to integrate code that enterprises do not want to modify, because of the cost (legacy code). It also avoids to product a client or a data source that would be able to interact with only one specific interlocutor.

Mediator patterns In the service based application life cycle, a new business arises : the mediator chain designer. Its task consists in finding appropriate mediators and detail their sequence. From [10] and [6], a basic mediator pattern list can be established :

- *examiners* modify the content of the request (e.g. validation, authentication, authorization, or monitoring);
- *transformation mediators* modify the content of the request (e.g. data type mapping and enhancement);
- *transcoder mediators* modify the format of the request but not its content. They allow requests to go through different transport protocol to interact;
- *cache mediators* stock results of already executed requests in order to save time and resources;
- *routers* chose the service they are giving the request to, depending on the content of the request. *Discovery mediators* do the same thing using besides a trader to dynamically chose the right service.
- *operator mediators* (e.g comparator, union, intersection, combination or aggregation);
- *clone mediators* dispatch a unique request to several services.

Related works Most interesting solutions in data mediation field are based on ontologies mapping. Indeed the heterogeneity of data leads to the need for semantic information. In order to deal with it, applications have been enhanced with ontologies. But it leads to the multiplication of ontologies in information systems regrouping many applications. Solutions such as WSMX[2] and TSIMMIS[3] [11], FOAM[4] or [12] provide abstractions and tools to generate mediators implementing the mappings between ontologies.

2.2 Mediation in ESB

Initially used for the integration of heterogeneous data store (databases, files, etc), the concept of mediation takes a new breath with SOA and Webservices. Mediation has thus become an essential part of ESBs (Enterprise Service Buses) (see fig. 2 inspired from [5] and IBM/SOA[5]).
An ESB is actually a middleware providing integration facilities built on top of industrial standards such as XML, SOAP, WSDL, WS-Addressing, WS-Policy, WS-Security and WS-ReliableMessaging, J2EE Connector Architecture [13]. Besides mediation functionalities, the ESB provides:

- a trading service in order to find appropriate services;
- communication service (mostly asynchronous with MOM and publish/subscribe);
- orchestration service (based on BPEL [14]).

[2] http://www.wsmo.org/TR/d13/d13.3/v0.2/
[3] http://www-db.stanford.edu/tsimmis/tsimmis.html
[4] http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/
[5] http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb

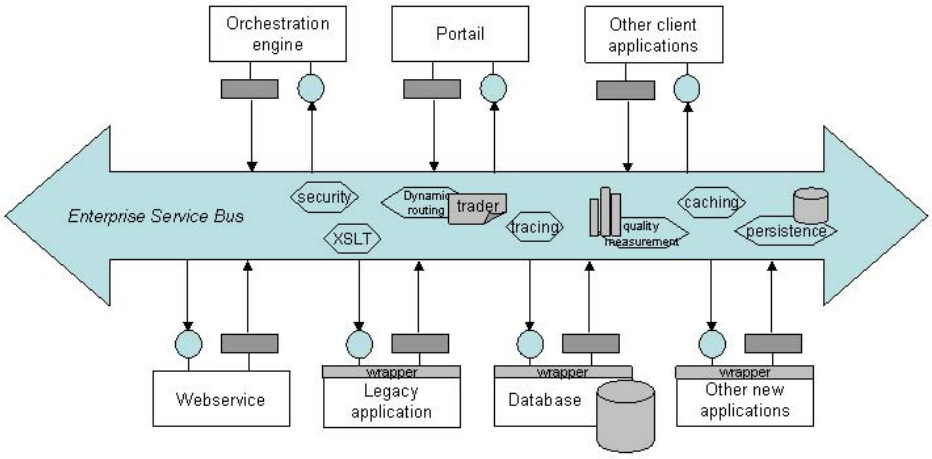

Fig. 2. Enterprise Service Bus

In ESB, the term mediation has a larger acception. In addition to transformation functionalities, mediation also includes :

- Security (e.g. cryptography, authorization, etc) which is a major preoccupation when different companies, using heterogeneous security systems need to interact.
- Dynamic routing and dispatch of requests potentially to multiple receivers in order to perform load balancing or to respond to failure of a data source for instance.
- Other non-functional actions related to QoS management such as incomplete data management, quality measurement, tracing, caching, or failure detection and recovery.

In the context of the ESBs, there are two different approaches to implement mediators. Mediators can be *ad hoc* pieces of code that intercept requests and process them. Mediators are dispatched over the network, some elements being closer to the client side (such as transformation to proprietary data format) other to data source side (such as synthesis, reducing network congestion). The second approach is to implement mediators as Webservices. Then, they are integrated to the service-based application as any other service. In any case, their requirements are to be available, reliable and easily maintainable. In the architecture, there are no constraints on whom the mediation components belong to or on their implementation.

2.3 limits of ESB

Despite these new approaches, there is no unique definition of ESB. ESB solutions are closer to commercial products packaging ad-hoc tools, than to a structured architectural layer. First ESBs were proprietary tools (Sonic ESB[6], Fiorano ESB[7], Cape Clear[8], PolarLake Integration Suite[9], etc). They were using proprietary solutions that were managed only at implementation level, such as the mediations in WebSphere [15]. Now these companies make an effort to capitalize there work into projects such as :

- projects directly working on mediation solution such as Apache Synapse. Synapse is a really recent project that emphasizes the role of mediation in SOA solutions;
- projects focusing on a larger scope that mediation such as new Open-source ESBs (e.g. Celtix[10] and Petals[11] from Objectweb, Mule[12] and ServiceMix[13]) from CodeHaus, OpenESB[14] from Sun). These projects provide code and tools (container, communication service, trader, etc) to build specific well-suited ESB. For the moment, they do not really focus on modeling the mediation. In the ObjectWeb consortium, researchers working on ESBs capitalize their results the ESB Initiative (ESBi[15]).
- a project that does not relate directly to mediation seems to overcome the ESB domain at the moment : the Sun Java Enterprise Service Bus API[16] (JBI). This API mostly defines a standardized container for services. It may have an impact on mediation because it also standardizes the exchanges between services (the sequence and format).

ESBs do not provide sufficient software engineering abstractions to give a high level comprehension of the architecture and the interactions of this layer. It focuses on development and administration and leads to lack of maintainability, whether it is a dynamic or a statical administration of the system.

3 Projet S4ALL

3.1 Project summary

The S4ALL project has been active since July 2005. It is funded by the European Community and brings together major industrial actors interested in delivering

[6] http://www.sonicsoftware.com/products/sonic_esb/
[7] http://www.fiorano.com/products/esb_key_for_bca.htm
[8] http://www.capeclear.com/products/ccESB4ws.shtml
[9] http://www.polarlake.com/en/html/resources/esb/
[10] http://celtix.objectweb.org/
[11] http://petals.objectweb.org/
[12] http://mule.codehaus.org/
[13] http://servicemix.org/
[14] https://open-esb.dev.java.net/
[15] https://wiki.objectweb.org/ESBi/
[16] http://www.jcp.org/aboutJava/communityprocess/edr/jsr208/

new services to their customers, including Alcatel, Nokia, Schneider Electric. The high level objectives of S4ALL are the following :

- To study the process of service creation, taking into account end-users, manufacturers and service providers requirements and to implement service creation and customization tools for different audience and supporting environments, that are the professionals, the end-users on PC and the end-users on mobile devices,
- To specify and implement the appropriate service execution infrastructure. In particular, the partners will focuse on the delivery of open sources OSGi platforms, a J2EE server and an Enterprise Service Bus.
- To demonstrate selected vertical applications in the telco and industrial fields illustrating all aspects developed within the project.

Applications that have been provided by the telco (Alcatel and Nokia) and by Schneider Electric (a world leader in power distribution) cannot be disclosed for the moment. In the following section we thus present a toy example (dealing with supermarkets and bakeries) that exhibits the main characteristics of the industrial applications studied in the project.

3.2 An example to illustrate the problem

The purpose of this section is to present a simple example and show how mediation techniques can be used in the Webservice context.
Figure 3 presents a simple scenario where a cybermarket provides a web service to buy different kinds of products, including bread. The cybermarket actually buy its bread from two different bakeries. Relationships between the cybermarket and the bakeries are implemented with Webservices. As it can be expected, data exchanged and interfaces used by these two bakeries aren't the same. In this example, the cybermarket wants to find the n least expensive bread sorts. Bakery 1 can directly answer to this question, but bakery 2 can only send the list of all its prices.

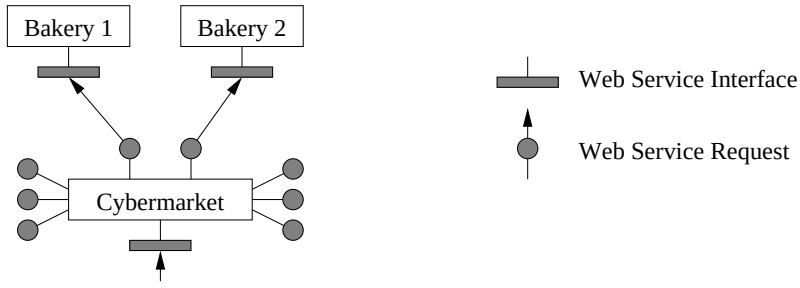

Fig. 3. A simple scenario

Using an ESB approach, the two bakeries are integrated using proxies that act as mediation elements. Two proxies are used in the cybermarket to encapsulate the calls to the bakeries and the cybermarket. With workflow languages, we can model our application but we can't separate the proxies from the application model: the proxy are inserted in the application model either like Web Services or by using language extensions.None of these two solutions is satisfying: by mixing at the same level the mediation and the process, an application designer can't understand the logical of the application (the model of the application) because there is no difference between the high level description of the application (what the application does) and the implementation of this description (how this application is implemented). The consequence of this lack of separation between the model and the implementation are principally:

- A lack of reusability of the application. Indeed, if a bakery server changes the designer should change the functional description of the application, although the model of the application doesn't change. In the same way, the mediation chain to access a bakery can't be reused in another application because it is mixed in the functional description of the application.
- A difficult design of the application because the application designer should simultaneously design the model of the application and the way it is implemented.

To improve these two aspects, our first goal is to provide a complete separation between the application model and the implementation view of this application. The second requirement of our work is to reuse the ESB tools : indeed, our purpose is not to re-engineer the already existing execution models, but the separation between these models and implementation aspects like mediation. By generalizing the notion of mediation to other non functional aspects of the application, like the quality of service, we are also improving the separation between the application model and its implementation.

3.3 An solution based on mediation

Figure 4 presents an architecture including mediation. A first mediator is used to aggregate the different bakeries. The aggregator gives the n lowest bread prices by mixing the data received from the two bakeries. Then, two mediators are used to hide the heterogeneity between the bakeries. Bakery 1 translator sends a request to find the n least expensive breads from bakery 1 to save bandwidth. Bakery 2 translator sends a request to find all the prices and gives only the n least expensive prices.

On the cybermarket side, a unique data format and a unique interface is then used to question the bakeries. By using mediation, the server (the bakeries) and the client (the cybermarket) don't have to be modified to incorporate new non-functional properties.

This solution based on mediation shows the complexity of the mediation chain. Indeed, we are only managing two bakeries and the chain remains simple.

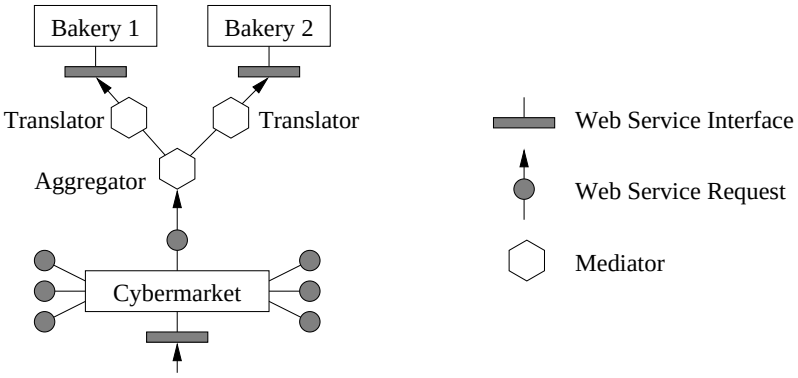

Fig. 4. A simple scenario with mediation

To really construct mediation chains, we should separate what does a mediation element from how it is connected: this separation between the chain and the implementation of mediation elements improve also the reusability. A mediation element can be reused in other application and we are separating the work of designing a mediation element from the work of composing these mediation elements. The architecture of the mediation chain also allows the insertion of other non functional element, like monitoring, security or verification elements: as explained before, these elements shouldn't appear in the application model, but in the mediation chain.

One of the possibility introduced by modeling the mediation chain between an executable model and the services that we plan to implement is the distribution of such mediation element in a network of hosts. Indeed, for load balancing, proximity or bandwidth reasons, it is more efficient to execute the mediation elements in a network of machine instead of using a single machine. A language to deploy the chain in a real network (a distributed ESB in our case) is thus necessary.

The last requirement of our mediation architecture is the dynamic adaptation of a mediation chain. Indeed, by separating the executable model of the application from its implementation, we doesn't have to stop the executable model to change how a service call is made. In our bakery scenario, dynamic adaptation allow, for example, the insertion of new bakeries to provide bread in the cybermarket without interrupting it. This possibility avoids the interruption of service time during a reconfiguration.

4 Our approach

4.1 A first experience

The purpose of this position paper is to promote the idea that it is important to specify the mediation operations in an abstract fashion, decoupled from the

execution environment (the ESBs in our case). To do so, our goal is to extend a mediation tool developed by the Scalagent company [16]. This tool allows to :

- describe mediation chains with an ADL (Architecture Definition languages) where mediation operations are performed by software components
- describe the execution environment and the way the mediation components have to be deployed on it
- automate the deployment and administration of the code installed on the network

This tool has been used successfully to develop (and industrialize) e-services in the domain of power distribution [17]. In this context, mediation is used between applications run on a J2EE infrastructure and OSGi-based gateways (see www.osgi.org). We are now exploring, in the S4All project, the way to extend this tool to deal with the connection of applications and Web services through ESBs and to automate the code generation and to allow dynamic reconfiguration.
The reminder of this section describes our new proposition. It presents the architectural and deployment views that we plan to model and implement.

4.2 Decoupling mediation

Figure 5 gives an overview of our model. First, we have defined a Platform Independent Model (PIM)[17] [18][19] in order to specify mediation chains independently of the ESBs and of the execution process: the model of the application and the implementation are separated, and the implementation of the mediation chain between the execution process and the services is modeled in a high level language. Our purpose is to increase understandability and reusability. This model includes the following elements:

- Service: any data source (Webservice, equipment, etc);
- Client: any application (Webservice or other) requesting a Service;
- Mediators: a component that is able to receive 1 to n pieces of data and send 1 to n pieces of data; it can be seen as a binding between Clients (1..n) and Services (1..n);
- Mediator component binding: link between Mediator components.

Through a dedicated tool, it is possible to assemble these elements to form mediation chains. The implementation of mediation elements themselves aren't modeled. Only the links between these elements are described in a high level language. A Mediation chain is close to the concept of "parternlink" in BPEL [14] but there are two main differences that come from the lack of integration of mediation in BPEL:

[17] http://www.omg.org/mda/

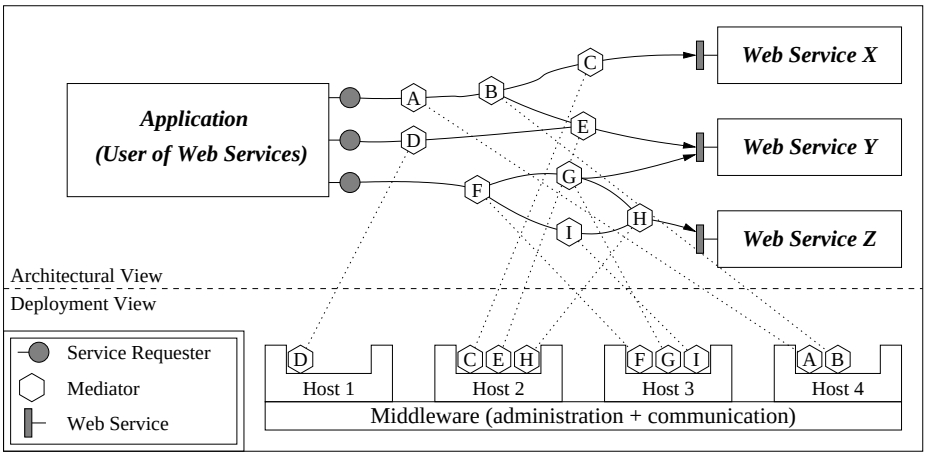

Fig. 5. Global architecture

- the data type is not necessarily the same on the client side of the link and on the server side because transformations are done on data;
- the Mediator chain is much more complex than the parterlink; it allows to describe processings done on data and the binding (e.g . synchronous, asynchronous, etc).

Another model is used to describe the execution environment, that is the ESB. In the current version of the mediation suite, the execution environment is provided by Scalagent under the form of an adapted version of the open source Joram middleware[18]. The challenge of the current project is to allow the execution of the mediation chains on an ESB. To meet the needs of the mediation suite, the ESB has to able to load dynamically new mediators and to take decision during a reconfiguration. This execution environment should also provide a middleware layer to interconnect the different execution environment. Figure 6 summarizes this execution environment. The middleware will be used to manage the monitoring, but also to interconnect the different mediators. It is important to note that the ESBs presented in the previous sections meet these technical requirements.
To automatically deploy and manage the set of mediators on ESBs, a specific language has to be developed. This language gives information about the location of mediators. This language will only describe where are physically located the mediators. This language has to remain very simple: it simply does the correspondence between a symbolic name of a mediator (used also in the ADL) and a host. We believe that is should be based on XML because. Through an extensible XML Schema, it allows to construct structured files to describe the mediation chain, that may be shared between the mediation actors, using a com-

[18] www.objectweb.org

mon vocabulary and grammar.
We also plan to extend ESBs functions in order to improve dynamism. To do

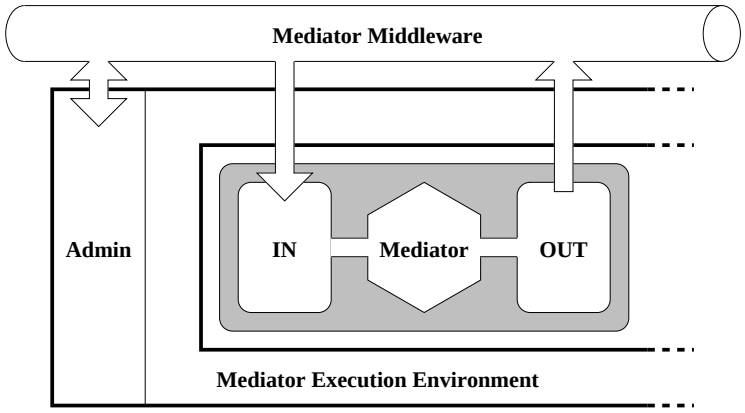

Fig. 6. The Mediation Middleware

so, we will use OSGi which provides a good solution to dynamically load and unload code. OSGi provides also a good solution to dynamically link the mediator with the execution environment. Inside the execution environment, a remotely administrable part should be planned. This part will communicate through the middleware.

5 Conclusion

Today's solutions for Web Service orchestration do not integrate heterogeneity of data type. At runtime, mediation operations have to be added to the system in order to ensure interoperability between Web Services. But mediation solutions are very technology-oriented. They do not provide abstractions that would facilitate the design and the administration of mediation chains. Dealing with distribution over heterogeneous networks and appearance of new Web Services becomes a hard task.
We propose then a higher abstraction model for mediation. Our model defines two levels: (i) a Platform Independant Model that provides an architectural view of the mediation chain. It defines the elements of the chain (Service, Client, Mediator component and Mediator component) that may be mapped automatically to the second level (ii) a framework that should provide a deployment language, a runtime environment and tools to dynamically administrate and reconfigure the platform by generating the communication code.
The architecture and the model presented in this paper meet the requirements of services integration: (i) the execution process is independent from the mediation

chain model to improve the reusability of the two elements, (ii) the mediation chain model separates the implementation of the mediation elements from their bindings, also to improve the reusability of this two elements, (iii) a mediation chain is projected on a distributed ESB to balance the load, (iv) a mediation chain can be updated during the execution of the execution process to avoid interruption of services during reconfigurations.
Our work builds on top of the mediation suite provided by the Scalagent company that has been used successfully in several device-oriented domain. The challenge is to replace the proprietary execution environment by ESBs (commercial or open source) while keeping the abstract, platform independent description of mediation chains. We are also exploring new ways to express business and mediation code. A promising approach is to use BPEL, that allows to describe Webs Services choreography, or another choreography language like APEL [20].
Another objective is to extend the ESBs execution capabilities in order to make them more dynamic regarding administration and configuration. To do so, we plan to integrate OSGi as an execution platform for the asynchronous middleware runtimes.

References

1. W3C. Simple object access protocol (soap) 1.2. Technical report, W3C, June 2003.
2. W3C. Web services description language (wsdl) 1.1. Technical report, March 2001.
3. OASIS. Uddi executive overview: Enabling service-oriented architecture. Technical report, OASIS, 2004.
4. T.R. Gruber. A translation approach to portable ontology specifications. *Academic Press*, 1993.
5. David A. Chappell. *Enterprise Service Bus*. O'reilly Media, 2004.
6. M.-T Schmidt, B. Hutchinson, P. Lambros, and R. Phippen. The enterprise service bus: Making service-oriented architecture real. *IBM System Journal*, 44(4):781, 2005.
7. C. Bussler. *B2B Integration*. Springer-Verlag, June 2003.
8. Gio Wiederhold. Mediators in the architecture of future information systems. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 185–196, San Francisco, CA, USA, 1997. Morgan Kaufmann.
9. Gio Wiederhold and Michael Genesereth. The conceptual basis for mediation services. *IEEE Expert: Intelligent Systems and Their Applications*, 12(5):38–47, 1997.
10. John Todd, Christian Och, Roger King, Richard Osborne, Jr. William J. McIver, Nathan Getrich, and Brian Temple. Building mediators from components. In *DOA '99: Proceedings of the International Symposium on Distributed Objects and Applications*, page 352, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0182-6.
11. J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 483, San Jose, California, June 1995.

12. F. Scharffe and J. de Bruijn. A language to specify mappings between ontologies. In *IEEE SITIS'05*, Yaoundé, Cameroon, November 27th - December 1st 2005.
13. Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, March 2005.
14. Microsoft SAP AG Siebel Systems IBM BEA Systems. Business process execution language for web services version 1.1. Technical report, July 2002.
15. Rachel Reinitz and Andre Tost. Building an enterprise service bus with websphere application server v6 – part 4– http://www-128.ibm.com/developerworks/websphere/techjournal/0505_reinitz/0505_reinitz.html. IBM WebSphere Developer Technical Journal, May 2005.
16. Philippe Lalanda, Luc Bellissard, and Roland Balter. An asynchronous mediation suite to integrate business and operational processes. In *IEEE Internet Computing*, January-February 2006.
17. Philippe Lalanda. E-services infrastructure in power distribution. In *IEEE Internet Computing*, May-June 2005.
18. J. Miller and J. Mukerji. Model driven architecture (mda) omg tc document ormsc/2001-07-01. Technical report, Object Management Group (OMG), July 2001.
19. Frankel David. *Model Driven Architecture*. John Wiley and Sons Ltd, January 2004. ISBN 0471319201.
20. Jacky Estublier, S. Dami, and M. Amiour. Apel: a graphical yet executable formalism for process modeling. In *Automated Software Engineering, ASE journal*, volume 5, 1998.