A distributed service-oriented mediation tool

Colombe Hérault[1], Gaël Thomas[2] and Philippe Lalanda[3]

[1,3]*Laboratoire LSR-IMAG, 220 rue de la Chimie*
Domaine Universitaire, BP 53
F-38041 Grenoble, Cedex 9, France
Colombe.Herault@imag.fr, Philippe.Lalanda@imag.fr

[2] *Laboratoire d'Informatique de Paris 6*
8 rue du Capitaine Scott
75015 PARIS - France
Gael.thomas@lip6.fr

***Abstract*—Integration of heterogeneous information becomes again a requirement with the emergence of large-scale distributed applications such as Web-Services based Applications. Enterprise Service Buses (ESB) deals with distribution and communication, but they still do not fix all the mediation issues such as design, deployment and administration of mediators. It turns out however that current solutions are technology-oriented and beyond the scope of most programmers. In this paper, we present an approach that clearly separates the specification of the mediation operations basing on a service component model, and their execution on a distributed ESB. Model and ESB are independent of the targeted middleware used by applications. This work is made within the European-funded S4ALL project (Services For All).**

***Index Terms*— Service Oriented Architecture, Mediation, Model Oriented Approach**

I. INTRODUCTION

There is today a growing need in the industry to integrate, and then reuse, business applications developed by different organizations. Currently, applications mostly belong to business activities being part of the same overall organization. Business to business applications are however emerging: these applications call for inter-organization integration. Applications considered for integration are of very diverse natures. It can be traditional IT applications or, more recently, operational applications related to field operations. For instance, in the power field, it is of higher importance today to integrate back-end applications and control applications run on the factory floor. Seamless integration allows the development of new services for a better usage of electrical power. It also dramatically speeds up decision-making by providing executives and decision software with accurate, up-to-date, and appropriately formatted information. Finally, it allows organizations to significantly improve capital-asset management and maintenance optimization.

In most cases, data produced by the applications need treatments before their exploitation, including aggregation, transformation, quality of service enforcement and SLA enforcement. To get back to our power distribution example, monitoring the power quality in a large area requires to deal with a large amount of data, some of them being protected. A centralized server can't handle the amount of data produced: aggregation and security enforcement is needed along the path of data. Moreover, devices do not use the same data representation and alignment transformations are needed. Generally speaking, it is clear that the integration of applications requires ontological transformations. Finally, from a commercial point of view, observation and enforcement of the quality of service provided by a tier is a real requirement. Integrating heterogeneous data requires thus a layer of mediation [1][2] between data producers and data consumers or between servers and clients. A mediation element is a software entity realizing these required mediation operations (aggregation, ontological alignment, security enforcement, etc.).

Defining and executing this mediation layer is a complex and hard task without tools: developers have to define mediation elements and relation between these elements, they have to deploy or to redeploy these elements in a distributed environment and they must provide a distributed execution environment to execute the mediation layer. Partial solutions already exist with Web Service [3] and Enterprise Service Bus [4], [5] but they remain very technology-oriented and do not separate the different tasks of building a mediation layer.

In this paper, we propose a model-driven approach to construct the mediation layer independently from the targeted execution support. We present a mediation tool and a model allowing the integration of disparate information of business applications in a timely fashion. The mediation tool allows the definition, deployment and execution of mediation components. Such components are assembled in order to form coherent mediation chains. The model describes the mediation

chains independently from the execution support. It defines the relation between components by using relevant and simple concepts: components consume and produce data and a chain is constructed by binding consumers to producers. We also present in this paper a supporting execution environment based on a service-oriented approach to execute our models. More precisely, it makes use of the OSGi standard [6] in order to allow dynamic evolutions of the mediation operations and of JORAM[1], an open source implantation of the JMS standard[2]. Finally, we present an application of our mediation tool in the Web Service context.

The paper is organized as it follows. The first part introduces mediation and ESB concepts. Then we present our approach, defining a component model and mediation designing tools. We also provide an accurate execution platform that is based on OSGi and message communication. Finally, we present a simple application example and conclude.

II. MEDIATION

A. Definition

The activity of integrating disparate information sources in a timely fashion is known under the name of mediation. The mediation is a software layer that stands between clients and services. Its purpose is to enable interoperability and integration of service, in other words, to enable a client (e.g. consumer, that be may itself a service) to easily and properly use a service. Mediation has been historically used to integrate data stored in IT resources like databases, knowledge bases, file systems, digital libraries or electronic mail systems [7].

Usually, the mediation layer is organized as mediations chains that transport the request from the client to the service and the answer the other way round. The mediation chains can be decomposed into light weight components called mediators that implement simple operations.

Encapsulating the integration of distributed, heterogeneous data in dedicated software is clearly a good practice. Mediation provides a single point of interface to the different network elements implied in the communication. This reduces the number of connections needed and facilitates change management. Mediation also provides an isolation layer from hardware and software details and, if appropriately configurable, permits the quick and cost-effective development of new services. The mediator layer improves reusability and evolution of clients and services. It facilitates the integration of new clients and services. Thanks to this layer it is also possible to add, without modifying the client and service code, new QoS concerns such as security, reliability, etc. Finally, it leads to the improvement of the scalability of the whole system.

[1] http://joram.objectweb.org/
[2] http://java.sun.com/products/jms/

We have identified four main categories of mediators:

- control mediation : mediators that influence the progress of the request in the mediation chain such as routing, filtering, aggregation,
- transformation : XSLT, ontology matching, semantic validation and enrichment,
- QoS mediation : security, transaction, compensation, exception handling, fault tolerance,
- SLA enforcement : mediators that negotiate and manage the contracts between services.

Security is a major concern for several reasons. First, data produced by business or control applications are generally private and have to be protected. The second issue is even more serious: since control applications can act upon the environment, malicious use of these devices can have dramatic consequences. Security is then a non negotiable feature for most customers (application owners) and represents clearly an important brake to wide application integration.

B. Web Services and Enterprise Service Bus

Web services constitute a very promising technology to improve integration within organizations and between different organizations as well. Web service technology provides a uniform model for services, especially in the context of heterogeneous distributed environments and interoperation with external environments. It can be used to expose applications as well as software or hardware components. Once published as Web services, software elements can be accessed from any language that supports the Web service interface. Services can also be dynamically added, removed or re-configured. Web Services define only a communication protocol and the notion of repository of services, but remains basically a client/server paradigm: Web Services provide a way to expose remotely existing software entities, but does not address the problem of mediation between services.

Enterprise Service Buses (ESB) can be seen as a new class of integration tools essentially relying on the Web services technology and associated standards. First commercial ESB products were mainly described as a way to integrate existing middleware services (J2EE application servers, message-oriented brokers, etc.) or applications. To that end, they provide transformation and routing functions. These are typically mediation operations. More recently, with the advent of SOA approach (Service-oriented architecture [8]), ESBs are also presented as a way to create a service-oriented architecture.

An ESB is a software bus for services, which facilitates connectivity management. It allows services to connect to a single point, the bus, which is in charge of the processing of the communications. Interactions are realized through an

asynchronous message oriented middleware. Note that the main standard in the domain today is Sun Java Business Integration (JBI)[3].

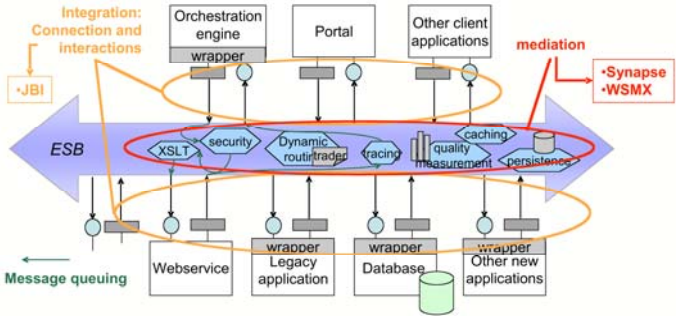

Figure 1 : ESB architecture[4]

Regarding mediation operations, commercial ESB mostly provide proprietary solutions (BEA ALSB[5] or IBM[6][7] for instance). Emerging open source products such as Apache ServiceMix[8], ObjectWeb Petals[9] or Codehaus Mule[10] tend to provide more open solutions based on JBI.

But, in all cases, the mediation solutions are very technical and technology-driven. Mediation chains are hard to build, deploy and maintain. They are also uneasy to change and to reuse. We believe that there is a clear need to consider mediation operations as first order objects and to treat them accordingly. In the remainder of this paper, we present a mediation approach based on models that takes the ESB philosophy.

III. OUR APPROACH

Our general philosophy regarding mediation is to specify the mediation operations in a component-based language, distribute these operations on a physical network, and link them with message-oriented connections. Implementing mediation via mechanisms commonly seen in ESBs requires programming reactive actions within the communication middleware. This makes it difficult for the mediation software to evolve. Without a global architectural perspective, implementing a distributed mediation application that an administrator can redeploy dynamically is problematic. Instead, we want to use loosely coupled components which help provide the flexibility and adaptability that applications need to keep up with changing requirements. We also seek to dynamically use the available resources on both the IT and operational sides, distributing operations over a network of business and operational machines to create the most effective mediation chains. The challenge is to place the mediation operations at the best locations to reduce the communication load and speed up the delivery of services.

We also believe that a tool is necessary to specify mediation chains at a sufficient level of abstraction. We have adopted a model-driven approach in order to guide the development of correct mediation chains and to be as independent as possible of the supporting runtimes.

A. *Component model*

In our model, we realize a mediation application via a set of connected components. We define a component using message-based interfaces, and each component implements a single mediation operation (such as data collection, transformation, integration, routing, and so on). Components can then be placed on different physical nodes to make up distributed mediation chains.

Figure 2 : Mediator model

A mediator is a component, exposing “in” and “out” message-oriented ports that are characterized by a name, a data type, and cardinality (see fig. 2). The name of the port is used to identify the port when binding mediators. The type of the port defines the expected data type to be received or sent on this port. Data received on “in ports” are consumed by the mediator to produce data that are sent “on out” ports. Only ports of the same type should be bound.

In order to offer both synchronous and asynchronous communications, a minimum and a maximum cardinality are defined for the in port. For all in ports, the minimum must be achieved to release the mediation task. The maximum cardinality defines the maximum number of data that the mediation task will consume. Data that have not been consumed will be used lately. For example, a mediator with two ports of cardinality (min1 = min2 = max1 = max2 = 1) has to wait for one data on each port to come in. Another mediator with two ports of cardinality (min1 = min2 = 2, max1 = 2, max2 = ∞) has to wait for at least two data for each port. If one of the two ports is ready before the other, it is necessary to wait for the second port to be ready. No more than two data of the first port will be consumed, remaining data will be consumed later.

[3] http://www.jcp.org/en/jsr/detail?id=208
[4] http://www-306.ibm.com/software/info1/websphere
[5] ttp://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/aqualogic/service_bus/
[6] http://www-306.ibm.com/software/integration/wsesb/
[7] http://www-128.ibm.com/developerworks/library/specification/ws-sca/
[8] http://servicemix.org/site/
[9] http://wiki.petals.objectweb.org/
[10] http://mule.mulesource.org/

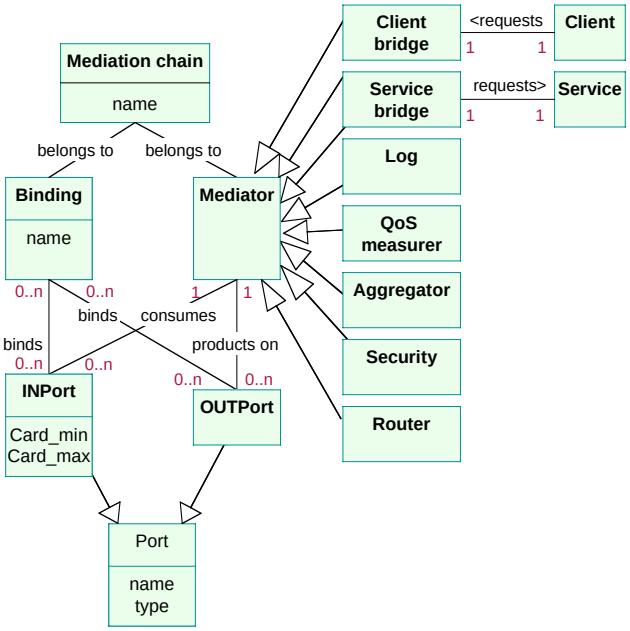

Fig. 3. Mediation meta-model

We also identified several mediation types that are recurring, for example (see fig. 3): log, QoS measurer, router, aggregator and security mediators. For each type, we can define a particular pattern that allows to generate more specific code. For example, a log mediator provides a connection to a database. The security mediator provides authoring and authentication, configurable with specific requisites. Two other mediation types are important: client and service bridges. They allow the seamless interconnection of the mediation chain with the clients and servers. The implementation of the service bridge mediator is conforming both to the mediation platform specific model and the service platform specific model (idem for the client). A web service bridge is implemented both as a mediator and a web service client sending a SOAP request.

B. *Mediation tool*

We make a clear distinction between the model of the chain and the runtime in order to deal with the complexity of design and management of mediation [9], [10]. It is then necessary to offer tools to make the right use of the runtime in accordance with the model. The different phases of the development of a mediation chain are identified and delegated to specialists. For each phase, the final aim is to provide specialists with tools facilitating their work such as graphical or code generation tools. This approach improves software productivity and quality, it consequently reduces development costs.

First, in the design phase, the mediation chain and the composing mediators are modeled in accordance with a mediation meta-model thanks to a specialized graphical editor. It has been developed thanks to Eclipse Graphical Modeling Framework[11] (GMF).

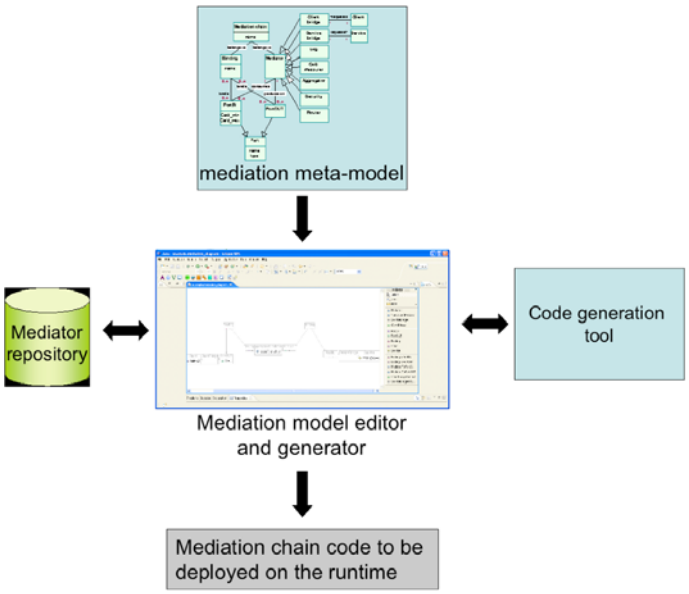

Fig. 4 : Design and Development of the mediation chain

In the development, three ways are complementary to deal with platform specific code (see fig. 4). It is possible to search for pre-existing mediators depending on ports or functionality in a mediator repository in order to improve reuse of code (this part is not fully instantiated at the moment). This code may be used straightforwardly or adapted by a domain specialist. It is also possible to use a generation code tool that generates the code of a skeleton of mediators with method to read and write on ports. It has to be filled in by a specialist of mediation. In the meta-model, specialized mediators are also available. They allow to generate more accurate code for particular purpose. For example in our tool, Web service client and service bridges are generated from a WSDL file.

Lastly the obtained mediator code is deployed, assembled and managed on the execution platform, in accordance with the mediation chain model. At the moment a deployment plan is manually written. It describes operations of migration of the mediator's code and their cardinality configuration. It also describes the bindings between mediators to allow a transparent communication over distributed machines. This deployment plan is executed by the administrator of the execution platform. In a future version of our work, the deployment plan will be built up with simple deployment rules thanks to an image of available components and gateways that is provided by the current runtime.

During execution, new clients and services join the ESB and evolve. The ESB must be distributed, flexible to support frequent adaptation of the chains. It has to support the arriving and departure of mediators, reconfiguration of the request route. We developed a platform based on the OSGi component

[11] http://www.eclipse.org/gmf/

model. OSGi components are dynamically deployed on gateway and can be reconfigured without interruption of the gateway (its concepts are close to those of IBM SCA). Communication among mediators is done through a specific MOM that provides total transparency of the distribution of mediators. Interactions with this execution environment are done thanks to a centralized administration entity that is able to build an image of the running gateways and to deploy mediators on it. Section 4 describes our ESB.

IV. EXECUTION PLATFORM

Interconnected nodes constitute the execution platform. Each node executes a set of mediators and each mediator communicates via its in and out ports with other mediators by using a Message Oriented Middleware (MOM, see Figure 5). Nodes are built over OSGi [6], which provide facilities to load and update Java code dynamically. Two modules are necessary on each node. The first one is the MOM Bridge. It allows communication between mediators. The second one is the administration module that manages the node. A third entity, called administration console, manages the set of administration modules remotely in a centralized manner. This administration console is used to deploy a mediation chain.

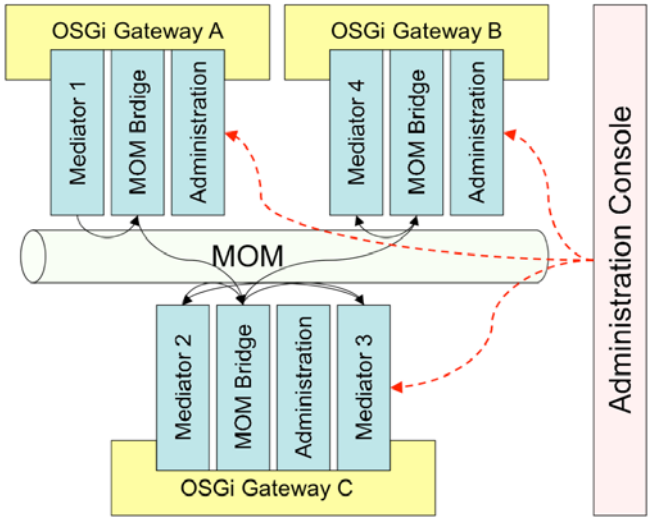

Figure 5 – Execution Platform

A. OSGi and code deployment

OSGi is a specification defined by a large consortium. The main concept of OSGi is the notion of bundle. A bundle is a deployment unit. OSGi allows dynamic loading, unloading and reloading of bundles: this property is used in our work to load and install new mediator code on a node. OSGi defines also dynamic binding between bundles by using service concepts. A service is a Java interface: each bundle contains service implementations that provide service interface. A service interface is registered in an OSGi service repository. A bundle possesses also an activator, called at startup to allocate services and to register them in the service repository.

In our work, a mediator is packaged into a bundle. When the bundle is loaded in OSGi, its activator registers a mediator factory in the service repository. This factory is then used by the administration module to allocate and configure new mediators. Configuring a mediator uses the Message Oriented Middleware module to bind in and out ports of the mediator with message topics.

B. The MOM Bridge and inter-mediators communication

A MOM defines named topics of messages. Producers send messages to a topic and all consumers of the topic receive these messages. To connect out ports to in ports, topics are created. If the name of the topic isn't specified at the model level, the name of the port is used to construct a topic name. An out port produces messages to this topic and an in port consumes messages from this topic. To improve the reusability of our mediators, the name of the topic can be defined in the mediation chain model: mediators developed by different groups can be connected even they don't use the same port name.

The MOM Bridge uses the local MOM service defined by OSGi to communicate locally: two co-located mediators communicate without using the network. For distributed communication, the MOM Bridge relies today on JORAM, an open source implementation of the JMS specification. Because Joram doesn't scale with the number of mediators, we plan to use a DHT-based algorithm for diffusion, such as Scribe [11].

C. Local administration

Each node ready to host mediators exposes remotely a local administration interface thanks to the local administration module (Figure 5.). This interface defines three sets of functions :

- An *install* method used to load a new mediator code packaged in a bundle, an *update* method to update the bundle and all the instance of the mediator, and an *uninstall* method that free all instances of the mediator and uninstall the bundle;
- An *instantiate* method used to allocate and configure a new mediator thanks to the factory provided by the bundle, and an *uninstantiate* method to free a mediator;
- An introspection method to consult the running mediators on the node and the installed bundles.
-

A local administration module participates also in a failure detector based on a two-ring algorithm [12]. This failure detector constructs a group membership of the running nodes and is used by the administration console. A unique number assigned at startup identifies a node in the membership. Our failure detector supports new node arrivals and faults of nodes. In [12], a complete performance evaluation of the algorithm shows that a correct image of a network constituted by 4000 nodes is computed in a small number of seconds.

D. Administration Console

The administration console is used to deploy complete mediation chains. It takes the chain description in entry and where will be deployed the mediators of the chain (by using the node identifier). The administration console uses the local administration modules to interact with the nodes.

The administration console can be launched at anytime and is not required during chain execution. At launching time, the administration console gets the image of the gateway present on the network by using the membership. The console asks then each gateway to know which mediator it hosts. A mediator is identified by a name (the chain name and a mediator name). The administration console takes the deployment plan and deploys the new mediators, undeploy the unused mediators and update the modified mediators (a modified mediator is a mediator with new connections, new code or new ports): the console doesn't interrupt the mediation chain, but adapt it to the new requirements. Because at bootstrap no mediators are running, all the chain is deployed.

V. Web Service Based Application Example

Our approach is built and validated in the European ITEA S4ALL project. It is demonstrated on a telecommunication scenario that is confidential and can not be developed here. However, we develop now a simple example that shows the interest of our solution.

Web services are one of the domains where mediation is required. Indeed, integrating Web Services provided by different organizations usually requires to develop many sorts of mediators. For example, constructing a classic trader, that finds the best interest rate from a set of banks, requires integration of different services built by different banks. Because the banks are independent from each other, services provided by the bank are close in term of interface, but not identical. Despite the emergence of ontologies in Web Service, such as OWL-S [13], banks will not always follow the ontology to add bank-specific possibilities. To integrate services provided by a bank, transformations are required to fit the gap between interfaces. Each bank should be accessed through a mediator that transforms its interface to a common interface.

To show the interest of mediation chains, we take a simple client/server application where the server is the bank A. Figure 6 shows the original application on the top. A client calls a web service method to know the interest rate of a bank and uses it. Because the client has a new contract with a second bank, called bank B, he wants to trade interest rate from the two banks and choose the most interesting rate. Instead of modifying its application, he can insert a mediation chain in front of the banks Web Services.

Bottom of figure 6 describes this mediation chain. A Server bank bridge (SBB), which provides the same Web Service interface as Bank A, replaces the bank A. The SBB is a mediation element. It posts the trading request to the Client Bank Bridge (CBB) of the Bank A directly and of the Bank B through a mediator called Transform that performs the ontological transformation. CBB A calls the Bank A via its Web Service interface and post response to the trader (CBB B does the same thing for Bank B). When both rates are received, the trader calculates the best rate (to simplify the example, interest rates of banks A and B are in the same format) and posts this rate to SBB. SBB sends the Web Services response the client.

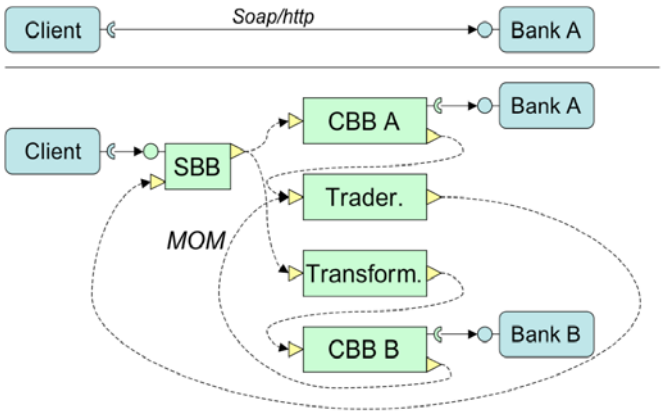

Figure 6 – Bank mediation chain

Using a mediation chain in this example has two main interests. First, from an engineering point of view, client is not modified. Second, adding new banks is now easy. SBB, CBB A and CBB B have been totally generated thanks to Bank Web Services WSDL. Most effort is done on Transform and Trader. Transform performs a simple ontological task. Trader is written by the bank specialist that has only to know how to choose the best rate.

From the model point of view, we show that building a mediation chain is relatively easy and that the programmer does not have to manage with the real infrastructure (distribution, message communication, quality of service). He describes its mediation chain and then will deploy it.

From the execution point of view, we have a very flexible system: adding new mediators (for example, to interact with a third bank) does not require an interruption. Moreover, quality of service becomes possible: the trader can wait the fastest bank and it can tolerate fault.

VI. Conclusion

Important organizations, such as government or corporate companies, seek solutions to seamlessly integrate their software services, not only at a B2B level but also at the production or transportation level with operational data. Web services provide a simple way to expose their services on the Internet. Still they do not guarantee full integration of services.

In this paper, we argue that mediation plays an important role in this integration challenge. However, it seems that there is a real need today for non-technology-centric solutions that

tends to be future proof and allows to integrate disparate ad-hoc solutions.

We show that providing a clear separation of the mediation concepts and its underlying execution environment mechanisms is a prerequisite to build durable service-based applications. It provides understandability and manageability to dynamic, distributed systems that provide complex and efficient fault tolerance protocol.

We have identified a mediation meta-model that defines simple and main elements of the mediation without preoccupation for the details of distribution, communication or other low-level protocols. It is based on the component model communication through messages. We also provide graphical and code-generation tools that allows to execute a mediation chain model on an efficient, distributed platform that allows dynamic deployment and reconfiguration (without platform stopping). This solution offers flexibility regarding distribution and life cycle management.

We are still working on improving both modelling tools and the runtime. By practicing with our platform, redundant mediators are identified and will enrich a repository of mediators and code generators in order to facilitate and fasten the job of the mediator's developer.

We also investigate on tools to (semi-)automate the choice of location for mediators in order to ease the placing mediation operation and reduce communication load. For example, Mediators such as aggregator are likely to be placed near the data sources, early reducing the amount of data on the network. To do so, we need to identify best practices in term of deployment in the mediation domain, which has not been done yet because solutions for distributed mediation are still fresh.

An important topic of research for us is how to make mediation part of the process development. We aim to integrate our solution to the APEL workflow suite [14] (graphical editor and runtime) that have been experienced in our team. It will allow the process developer to have a complete overview of the behavior of its process, while having at his disposal an understandable and accurate tool to specify mediation parts.

REFERENCES

[1] G. Wiederhold, "Mediators in the Architecture of Future Information Systems," Computer, vol. 25, no. 3, 1992, pp. 38–49.

[2] G. Wiederhold and M. Genesereth, "The Conceptual Basis for Mediation Services," IEEE Expert, vol. 12, no. 5, 1997, pp. 38–47.

[3] L. F. Cabrera, C. Kurt, D. Box, An Introduction to the Web Services Architecture and its Specifications, Version 2.0, October 2004, http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnwebsrv/html/introWSA.asp

[4] David A. Chappell. Enterprise Service Bus. O'reilly Media, 2004.

[5] 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.

[6] OSGi Alliance. "OSGi Service Platform Core Specification Release 4, http://www.osgi.org, August 2005.

[7] H. Garcia-Molina et al., "The Conceptual Basis for Mediation Services," IEEE Expert, vol. 12, no. 5, 1997, pp. 38–47.

[8] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. Commun. ACM, 46(10):24–28, 2003.

[9] Philippe Lalanda, Luc Bellissard, Roland Balter, "Asynchronous Mediation for Integrating Business and Operational Processes," *IEEE Internet Computing*, vol. 10, no. 1, pp. 56-64, Jan/Feb, 2006.

[10] Colombe Hérault, Gäel Thomas, and Philippe Lalanda, "Mediation and Enterprise Service Bus — A Position Paper,", Proceedings of the International Workshop on Mediation in Semantic Web Services, 2005.

[11] M. Castro, P. Druschel, A-M. Kermarrec and A. Rowstron, "SCRIBE: A large-scale and decentralised application-level multicast infrastructure", IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications). 2002

[12] Bertil Folliot and Gaël Thomas. Protocole de membership hautement extensible : conception est expérimentations. In Actes de la 2éme Conférence Française sur les Systèmes d'Exploitation (CFSE'02), Chapitre français de l'ACM-SIGOPS, GDR ARP, pages 25-36, April 2001

[13] Masahiro Hori, Jérôme Euzenat, Peter Patel-Schneider, "OWL Web Ontology Language XML Presentation Syntax", Note, W3C, 2003

[14] 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.