An Adaptive Enterprise Service Bus Infrastructure for Service Based Systems

Laura González, Jorge Luis Laborde, Matías Galnares,
Mauricio Fenoglio, and Raúl Ruggia

Instituto de Computación, Facultad de Ingeniería,
Universidad de la República, Uruguay
{lauragon,mgalnares,mfenoglio,ruggia}@fing.edu.uy

Abstract. Service-based systems (SBS) increasingly need adaptation capabilities to agilely respond to unexpected changes (e.g. regarding quality of service). The Enterprise Service Bus (ESB), a recognized infrastructure to support the development of SBS, provides native mediation capabilities (e.g. message transformation) which can be used to perform adaptation actions. However, the configuration of these capabilities cannot usually be performed at runtime. To deal with this limitation, Adaptive ESB Infrastructures have been proposed which leverage their mediation capabilities to deal with adaptation requirements in SBSs in an automatic and dynamic way at runtime. This paper presents a JBossESB-based implementation of an Adaptive ESB Infrastructure and demonstrates its operation by describing their main functionalities. The paper also presents an evaluation of the implemented solution.

Keywords: enterprise service bus, service-based systems, adaptation, mediation.

1 Introduction

As service-based systems (SBS) operate in an increasingly dynamic world, they need adaptation capabilities to behave correctly despite unexpected changes. This becomes especially relevant in internet-scale systems and virtual service-oriented enterprises [1], where their massively distribution leads to several challenges regarding performance, availability and security, among others [2].

A widely recognized approach to deal with highly distributed services is based on Enterprise Service Buses (ESB) [3], which are a mainstream middleware to support the implementation of SBS. Within an ESB-based infrastructure, services communicate by sending messages through the ESB. This way, messages may be processed by mediation flows, implementing integration and communication logic, which apply different mediation operations to them (e.g. transformations). Furthermore, propositions like the Integration as a Service paradigm [4], and more concretely the Internet Service Bus (ISB) [5], aim to apply these integration technologies in the context of cloud-based and internet-scale SBS. ESB products usually support a wide range of mediation capabilities including message transformation and intelligent routing [3].

A.R. Lomuscio et al. (Eds.): ICSOC 2013 Workshops, LNCS 8377, pp. 480–491, 2014.
© Springer International Publishing Switzerland 2014

However, while mediation flows can be used to react to unexpected changes (e.g. a transformation can be set up to handle a change in a service contract), they usually have to be configured at design time, in a per-service basis and in a slightly static way. This restricts the rapid responsiveness of the system and the generality of the solutions. Furthermore, in large-scale internet SBS (e.g. Internet of Things, Internet of Services) is unreasonable to assume that every time a service changes (regarding its contract or quality of service), all its clients will rapidly adapt. This motivates implementing server-side mechanisms that reduce the risks of client incompatibility.

In order to address these limitations we proposed an Adaptive ESB Infrastructure [2,6,7], which deals with adaptation requirements in SBSs by leveraging its mediation capabilities and allowing them to be configured and applied dynamically and automatically at runtime. The adaptive infrastructure is based on messaging and integration patterns commonly supported by most ESBs, so it provides a generic solution which is likely to be implemented in most of these products. Although in our previous work we developed some prototypes to validate the technical feasibility of the conceptual solution, we had neither implemented the complete adaptive infrastructure nor evaluated it regarding, for example, performance issues.

This paper describes a complete JBossESB-based implementation of the proposed adaptive infrastructure and demonstrates its operation by describing their main functionalities including the configuration of the adaptive behavior of services and the runtime adaptation capabilities according to different situations (e.g. response time degradation). The paper also presents an evaluation of the implemented solution in terms of the overhead it introduces and the resources it uses.

The rest of the paper is organized as follows. Section 2 presents the Adaptive ESB Infrastructure and its main components. Section 3 describes details on how this infrastructure was implemented on top of the JBossESB product and demonstrates its main functionalities. Section 4 presents an evaluation of the implemented solution. Section 5 presents related work. Finally, Section 6 presents conclusions and future work.

2 The Adaptive ESB Infrastructure

This section describes the Adaptive ESB Infrastructure [2,6,7] which, leveraging the mediation capabilities provided by ESBs, has the ability to address adaptation requirements in SBS in an automatic and dynamic way at runtime.

2.1 Overall Approach

In order to address adaptation requirements within the infrastructure, services are invoked through Virtual Services deployed in the ESB, following Service Virtualization patterns [8]. Also, for each supported mediation capability the infrastructure hosts an Adaptation Service which performs a specific mediation operation (e.g. a transformation). Adaptation Services are generic given that their behavior is not fully specified at design time, but it depends on run time information (e.g a Transformation Service executes a different transformation logic for each incoming message).

The general idea to achieve adaptation at runtime is to intercept all ESB messages and, if an adaptation is required for the invoked Virtual Service, drive them through Adaptation Flows. These flows, composed by Virtual and Adaptation Services, include all the required mediations steps (e.g. transformations, routing) to carry out a specific Adaptation Strategy (e.g. invoke an equivalent service).

Fig. 1 presents a general overview of the proposed solution where a client application invokes a virtual service (SRV), which virtualizes an external Web Service.

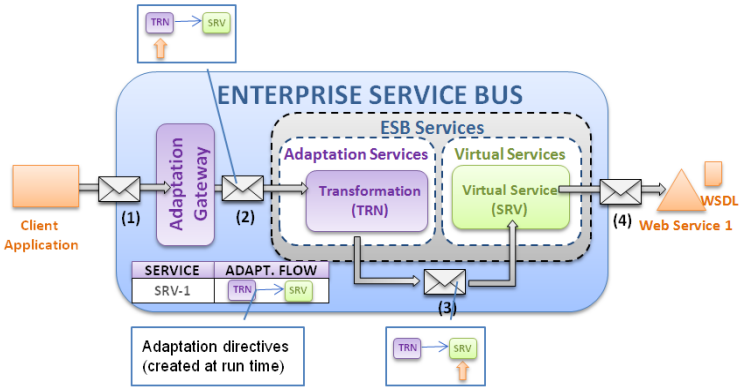

Fig. 1. Adaptive ESB Infrastructure [6]

First, the client sends a message through the ESB (1) to invoke the service SRV. The message is intercepted by an Adaptation Gateway following the Gateway pattern [8], which applies a common set of mediations to all incoming ESB messages. Given that there is an adaptation directive for SRV, the gateway attaches an Adaptation Flow to the message and routes it to the first step in the flow (2). This is supported by the itinerary-based routing pattern [3], which determines the message destination based on an itinerary included in the message itself. In this case, the itinerary (i.e. Adaptation Flow) consists of an Adaptation Service, which performs a transformation (TRN), and a Virtual Service (SRV). Thus, the message is routed to the TRN service which after performing the required transformation (specified in the message), routes the message (3) to the next step in the flow (SRV). Finally, the service SRV invokes the external Web Service (4) and, eventually, a response is returned to the client.

The infrastructure handles the conceptual elements of the S-Cube Adaptation and Monitoring Framework [9], which generalizes and broadens the state of the art in SOA adaptation. In particular, Monitoring Mechanisms refer to any mechanism to check if the actual situation corresponds to the expected one. They are used to detect Monitored Events (e.g. response time degradation), which represent the fact that there is a difference with respect to the expected system state, functionality or environment. Monitored Events trigger Adaptation Requirements (e.g. reduce response time) which represent the need of changing the underlying system, in order to remove the differences between the actual situation and the expected one. Finally, Adaptation Strategies define the possible ways to achieve these requirements (e.g. invoke an equivalent services) and they are realized by Adaptation Mechanisms (e.g. routing a request).

2.2 Logical Architecture

Fig. 2 presents the logical architecture of the solution which consists of internal ESB components and an Adaptation and Monitoring (AM) Engine.

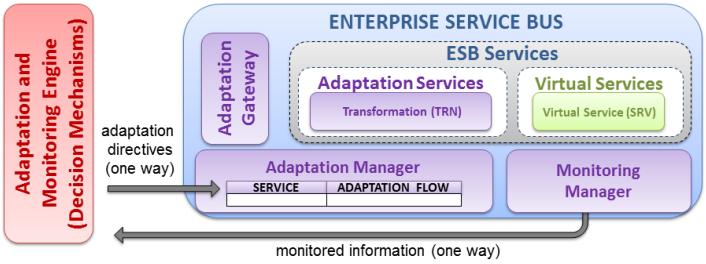

Fig. 2. Logical Architecture of the Adaptive ESB Infrastructure [6]

At runtime, the Monitoring Manager sends monitored information (e.g. the average response time of services) to the AM Engine, which is obtained by interacting with the built-in ESB Monitoring Mechanisms. When the AM Engine receives this information, it decides (based on services requirements and metadata) if an adaptation directive (implementing an Adaptation Strategy) should be created for a given service. If so, the directive (i.e. an Adaptation Flow) is sent to the Adaptation Manager which stores it, so it can be attached to all the incoming messages for the service.

2.3 Implementing Specific Adaptation Strategies

ESB products natively include different mediation capabilities (e.g. transformations) which are available as part of its execution environment. They can be configured based on specific requirements and can be combined to create mediation flows.

In the context of this solution, these mediation capabilities constitute the Adaptation Mechanisms supported by the infrastructure: transformations, routing, recipient list, aggregator, cache and delayer [6]. In particular, the transformation mechanism receives one message and returns another one, transformed according to a given transformation logic (e.g. data model transformation), and the routing mechanism dynamically determines the next service in the message path according to different factors.

In the proposed infrastructure, Adaptation Mechanisms are combined into Adaptation Flows to implement Adaptation Strategies. The infrastructure supports strategies to deal with Quality of Services (QoS) issues [6] and changes in service contracts [7] (e.g. Defer Requests, Distribute Request to Equivalent Services).

Adaptation Strategies are specified using YAWL [10]. Fig. 3 presents the load balancing strategy which was implemented using the routing and transformation mechanisms. In this case, messages are first processed by a routing mechanism which, according to a load-balancing strategy, routes the message to the invoked service (SRV-1) or to an equivalent service (SRV-2 or SRV-3). Given that services can use different data models, transformations to and from a canonical data model may be

needed. This strategy can be used, for example, to overcome a service saturation situation.

Note that although the general structure of strategies is specified at design time, their configuration is performed at runtime according to the involved service/s.

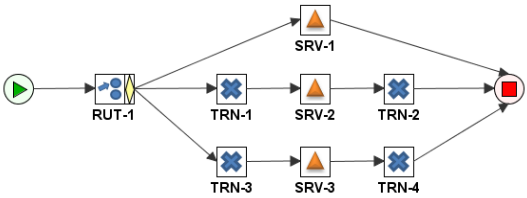

Fig. 3. Specification of the Load Balancing strategy using YAWL [6]

Fig. 4 presents all the elements, within the S-Cube Adaptation and Monitoring Framework, supported by the infrastructure which can be extended as needed.

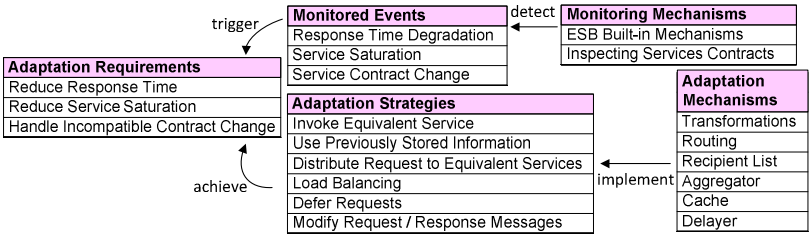

Fig. 4. Supported Adaptation and Monitoring Elements [2]

3 A JBossESB-Based Implementation

The infrastructure presented in Section 2 provides a generic solution to enhance general purpose ESB products with adaptation capabilities, by extending them with the components described in Fig. 2. This section presents a JBossESB-based implementation[1] [11] of the Adaptive ESB Infrastructure.

3.1 Internal ESB Components

This section describes the implementation of the Internal ESB Components.

Virtual Services. The implementation of Virtual Services requires a way to consume external Web Services and to expose them as Web Services through the ESB. Their implementation in JBossESB was based on the SOAP-Proxy component, natively provided by the product, which offers a direct way of implementing Virtual Services by exposing an external Web Service through the ESB.

[1] `https://code.google.com/p/esb-adaptativo/`

Adaptation Services. The implementation of Adaptation Services requires the adaptation mechanisms and a way to build ESB services based on these mechanisms.

Regarding adaptation mechanisms, JBossESB natively supports transformations, routing, recipient list and aggregator. However, given that they only support design time configuration (e.g. to use the transformation mechanism, the transformation logic has to be specified at design time in a configuration file), we enhance them to be able to obtain configuration values (e.g. the transformation logic) from the incoming messages. On the other side, the cache [12] and delayer mechanisms are not natively provided in JBossESB, but they were successfully implemented with the extensibility mechanisms provided by this product.

In order to create Adaptation Services based on the previous mechanisms, we used the native feature of JBossESB where each service is specified as an actions pipeline, in which each action performs a mediation operation (e.g. a transformation).

Itinerary-Based Routing Support. JBossESB does not natively support this pattern, so the product has to be enhanced with this capability. Our implementation uses a graph representation of an itinerary which can be included in the messages passing through the ESB. Also, Adaptation Services were built using a pipeline with two actions: the first one executes the mediation operation over the message, and the second one routes the message to the next step in the itinerary.

Adaptation Gateway. The Adaptation Gateway has to intercept all messages sent to the ESB. This component was implemented as an ESB Service leveraging the native HTTP-Gateway provided by JBossESB. In particular, an URL pattern was configured so that this service is the only entry point to the infrastructure. Fig. 5 presents the XML representation of the implementation of this component using this approach.

The Java class "GatewayAction" implements the specific logic of this component. In particular, it has to interact with the Adaptation Manager to check if there is an adaptation directive for the invoked service and, if so, it has to attach an Adaptation Flow (i.e. an itinerary) to the message. In order to specify the service to be invoked, clients has to use the wsa:To property of the WS-Addresing standard.

```
<service category="esb-adaptative" description="Gateway basado en HTTP localhost:8080/JBoss-Esb-Adaptative/http"
         invmScope="GLOBAL" name="ServiceGatewayHttp">
    <listeners>
        <http-gateway name="Http" urlPattern="/*"/>
    </listeners>
    <actions>
        <action class="org.fing.edu.uy.esbadp.action.gateway.GatewayAction" name="GatewayAction"/>
    </actions>
</service>
```

Fig. 5. Adaptation Gateway [11]

Adaptation Manager. The Adaptation Manager deals with the Adaptation Flows, received from the AM Engine, and it provides this information to the Adaptation Gateway when requested. The implementation of this component is based on an in-memory HashMap. So, given the identification of a Virtual Service, the corresponding Adaptation Flow (if any) can be obtained. As an additional feature, this component stores the history of the applied Adaptation Flows for each Virtual Service.

Monitoring Mechanisms and Manager. Monitoring mechanisms are used by the platform to check if the actual situation corresponds to the expected one.

First, we developed mechanisms which monitor Virtual Service invocations by leveraging the built-in monitoring features provided by JBossESB. In particular, JBossESB exposed monitored information as Java MBeans which can be accessed by clients with JMX support. This way, we implemented three Monitoring Mechanisms by interacting with the "MessageCounter" MBean and obtaining the values for the following attributes: messages successfully processed count, messages failed count and overall service time processed.

Also we developed mechanisms to monitor the contracts of the virtualized Web Services. They were implemented following the ideas of a tool[2] which compares two versions of a WSDL document and identifies the changes it suffered (e.g. a new operation). Also, we used the EasyWSDL[3] library which allows manipulating WSDL documents. When this mechanism is executed, it compares stored versions of WSDL documents with the current ones. This way, the infrastructure can detect which changes WSDL documents have suffered.

The Monitoring Manager interacts with the Monitoring Mechanisms to calculate Monitored Properties for each service. Our implementation supports four properties: number of invocations per time unit, ratio of successful responses, average response time and changes in service contracts. After calculating the values of the Monitored Properties, the Monitoring Manager sends them to the AM Engine.

Other Internal Components. Our implementation includes other internal components: Service Registry and Service Requirements Manager. The Service Registry allows registering services, specifying metadata for the services (e.g. XSLT transformations to and from the canonical data model of the platform) and specifying equivalent services. The Service Requirement Manager allows specifying requirements for the services which, along with the monitored information, allow detecting situations that need to be handled. Service requirements can be specified, for example, based on the monitored properties (e.g. average response time < 1000 ms).

3.2 AM Engine and Administrative Console

This section describes the implementation of the Adaptation and Monitoring Engine and the Administrative Console.

Adaptation and Monitoring Engine. The AM Engine receives monitored data from the Monitoring Manager, sends adaptation directives to the Adaptation Manager and takes the different adaptation and monitored decisions required by the infrastructure. This component also manages the Adaptation Events, Requirements and Strategies supported by the platform.

Every time the Monitoring Manager sends new values for the Monitored Properties for a given service, the AM Engine processes this information to generate, if needed,

[2] `http://www.membrane-soa.org/soa-model-doc/1.2/compare-wsdl-java-api.htm`

[3] `http://easywsdl.ow2.org/`

new adaptation directives for that service. This is done based on the monitored information, service metadata and service requirements. The AM Engine also knows which Adaptation Strategies can be used to deal with each Adaptation Requirement.

The engine also implements the required logic to select an Adaptation Strategy (currently in a random way) in case more than one is applicable to deal with a given Adaptation Requirement. As an additional feature, if more than one Adaptation Strategy has to be applied (e.g. a service has more than one Adaptation Requirement) the AM Engine combines various Adaptation Flows in a single one.

Administrative Console. The Administrative Console provides functionalities which facilitate the configuration of the adaptive infrastructure through a graphical interface. It also allows controlling and analyzing the adaptation processes generated for each service. The implementation of the Administrative Console was based in JSF 2.1 and in a set of components provided by the framework Primefaces 1.5. It is completely decoupled from the ESB and from the AM Engine: it interacts with them via JMX.

3.3 Demonstration of the Main Functionalities

This section presents the main features of the Adaptive ESB Infrastructure by means of the JBossESB-based implementation.

Fig. 6 presents the Administrative Console of the infrastructure which provides functionalities grouped in three main categories: Virtual Services management, Adaptive Server (ESB) configuration and AM Engine configuration. On the right side of the figure, three Virtual Services, virtualizing external Web Services, are listed.

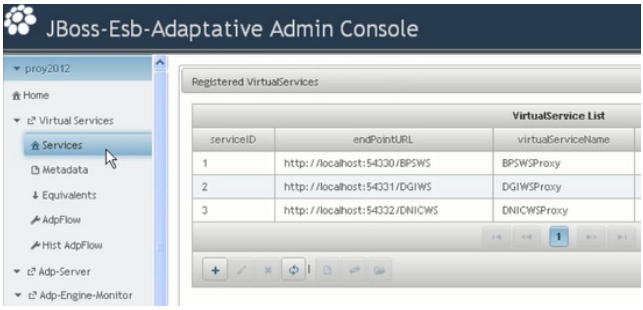

Fig. 6. Administrative Console: Virtual Services Management

The Administrative Console allows configuring basic information of Virtual Services (e.g. name, category), the transformations to and from the canonical data model of the platform and it allows associating equivalent services. It also allows configuring the adaptive behavior of services (e.g. the maximum number of invocations a service can support in a given period of time).

The adaptive infrastructure identifies and responds to different situations. For example, if a service is saturated by sending it more requests that the number it can handle, the solution detects this situation and applies a suitable Adaptation Strategy to

handle it. Also, the current Adaptation Flows implementing the selected strategies for each Virtual Service are graphically visualized through the administrative console.

Fig. 7 shows the first two levels of the current Adaptation Flow for the service DNICWS, which implements a Load Balancing strategy and expires in 283 seconds.

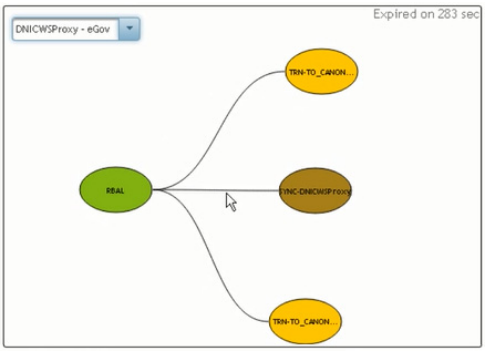

Fig. 7. Current Adaptation Flow for the Service DNICWS

Finally, the history of the applied Adaptation Flows for a given service can also be visualized through the administrative console.

4 Evaluation of the JBoss-ESB Based Implementation

This section presents an evaluation of the implemented solution. The SoapUI tool v4.5.1 was used to simulate service invocations and the VisualVM tool v1.3.5 was used to monitor resource usage. The platform was run on a desktop PC with 4GB of RAM and a dual-core processor of 3.2Ghz.

4.1 Overhead Introduced by the Infrastructure

We performed some tests in order to quantify the overhead in the invocations introduced by the infrastructure. Table 1 presents the results of the tests which were obtained based on 1200 invocations to a Web Service.

Table 1. Overhead in Invocations [11]

Adaptation Strategy	Average Time (ms)	Overhead (ms)
Direct invocation to the Service	18	N/A
Invocation through the Infrastructure	21	3
Invoke Equivalent Service	94	76
Use Previously Stored Information	6	N/A
Distribute Request to Equivalent Services	350	332
Load Balancing	129	111
Defer Requests	121	3
Modify Request / Response Messages	179	161

The first row in the table, correspond to a direct invocation to the Web Service. The second row corresponds to an invocation to the Web Service through the infrastructure but without applying any strategy. The rest of the rows correspond to invocations to the Web Service where the different Adaptation Strategies were applied. For a given strategy, the overhead value is calculated with respect to the time required for a direct invocation.

We believe that the obtained values are acceptable, given that for most of the strategies the overhead is less than 200 milliseconds.

4.2 Resource Usage

We also monitored the CPU and memory usage while the platform was operating and applying different adaptation strategies.

As presented in Fig. 8, the first interval (1) had a normal CPU usage of around 40%. However, in the second interval (2) the usage was increased because the "Distribute to Equivalent Services" strategy was applied, which requires performing various messages copies and transformations. In the third interval (3), after the execution of this strategy was finished, the CPU usage decreased to a normal use again.

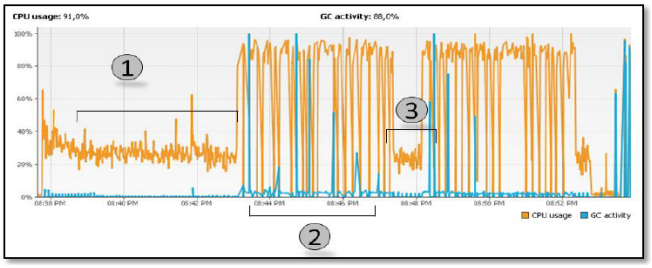

Fig. 8. CPU Usage [11]

Regarding memory usage, Fig. 9 presents similar characteristics, that is, the usage increased when the "Distribute to Equivalent Services" strategy was applied. It also allows visualizing that the solution does not present memory leaks, as the memory usage at the end of the test is very similar to the one at the beginning.

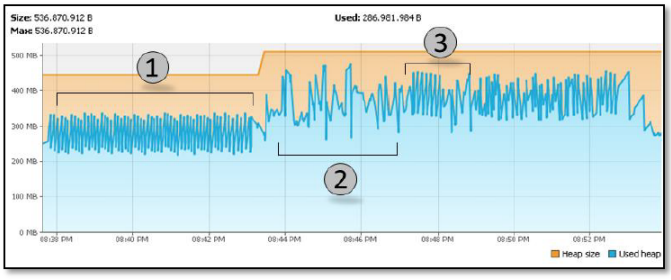

Fig. 9. Memory Usage [11]

5 Related Work

As stated in our previous work [1,6,7], dynamic adaptation in ESB has recently been addressed. Most authors propose solutions that allow applying or configuring ESB mediation capabilities in a more dynamic and automatic way. For example, a recent work [13] proposes an Adaptive ESB as part of an Adaptive SOA platform. The Adaptive ESB focuses on adapting service compositions by selecting at runtime (based on past invocations and QoS values) the services to be invoked.

Related proposals, although dealing with dynamic and automatic adaptation in ESBs, they use limited mediation capabilities, they are not extensible, and mediation flows to be followed by messages are defined at design time. In turn, our solution uses a wider and extensible range of mediation capabilities (e.g. cache and delayer). Also, it enables to select concrete mediation flows for messages at run-time, which enables to choose different strategies when an issue is detected (e.g. regarding response time). Finally, our proposal aims at enhancing general purpose ESB products with adaptation capabilities. To this, the proposed infrastructure is based on commonly supported ESB patterns and it is not restricted to the characteristics of specific ESB products.

6 Conclusions and Future Work

This paper presented a JBossESB-based implementation of an adaptive ESB infrastructure, which provides adaptation capabilities for implementing large-scale SBS. The experimental results show the feasibility of this approach through an implementation based on a general purpose ESB product.

The main contributions of this paper consist of the proof-of-concept implementation of the complete adaptive ESB solution, which provides experimental results concerning the development and the execution. Such results are crucial to better know about the behavior of the followed approach as well as to identify aspects to improve. Regarding the development, JBossESB features and other existing tools enabled a direct implementation of most of the components of the solution. However, some elements required significant ad-hoc development. Concerning execution, the overhead introduced by the solution, which is one of the most critical aspects, appears to be acceptable as preventing client execution failures. CPU and memory usage is also acceptable. However, applying adaptation strategies which require significant message copies and transformations increased this usage in a considerable way.

Beyond current results, this work aims at being a step forward for adding dynamic adaptation capabilities to Internet SBS, notably based on ESBs.

This work is currently being extended by implementing different strategies as well as by improving the overall system. The approach is being applied to implement do-main-specific integration platforms for Geographic Information Systems [14] and Bioinformatics. In addition, a research line on applying this kind of platform to Internet of Things scenarios [15] and context-aware service systems [16] is ongoing.

Future work consists in specifying and prototyping other Adaptation Strategies, addressing other Adaptation Requirements, as well as implementing the adaptive infrastructure with other ESB products. Other topic to be addressed concerns the application of Complex Event Processing mechanisms for the AM Engine.

References

1. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice-Hall (2004)
2. González, L., Ruggia, R.: Adaptive ESB Infrastructure for Service Based Systems. In: Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions. IGI Global (2012)
3. Chappell, D.: Enterprise Service Bus: Theory in Practice. O'Reilly Media (2004)
4. Buyya, R., Broberg, J., Goscinski, A.M.: Cloud Computing: Principles and Paradigms. Wiley (2011)
5. Ferguson, D.F.: The internet service bus. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, p. 5. Springer, Heidelberg (2007)
6. González, L., Ruggia, R.: Addressing QoS issues in service based systems through an adaptive ESB infrastructure. In: Proceedings of the 6th Workshop on Middleware for Service Oriented Computing - MW4SOC 2011, Lisbon, Portugal, pp. 1–7 (2011)
7. González, L., Ruggia, R.: Addressing the Dynamics of Services Contracts through an Adaptive ESB Infrastructure. In: 1st International Workshop on Adaptive Services for the Future Internet, Poznan, Poland (2011)
8. Wylie, H., Lambros, P.: Enterprise Connectivity Patterns: Implementing integration solutions with IBM's Enterprise Service Bus products
9. Kazhamiakin, R.: Adaptation and Monitoring in S-Cube: Global Vision and Roadmap. In: Proceedings of the Workshop on Monitoring, Adaptation and Beyond (MONA+), Madrid, Spain, pp. 67–76 (2009)
10. Hofstede, A.H.M., ter Aalst, W.M.P., van der Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its Support Environment. Springer (2009)
11. Laborde, J.L., Galnares, M., Fenoglio, M.: Implementación de una Plataforma ESB Adaptativa (2012), `https://esb-adaptativo.googlecode.com/`
12. Yan Fang, R., Ru, F., Zhong, T., Eoin, L., Harini, S., Banks, T., He, L.: Cache mediation pattern specification: an overview
13. Masternak, T., Psiuk, M., Radziszowski, D., Szydlo, T., Szymacha, R., Zielinski, K., Zmuda, D.: ESB-Modern SOA Infrastructure. SOA Infrastructure Tools, Concepts And Methods. Poznan University of Economics Press (2010)
14. Rienzi, B., González, L., Ruggia, R.: Towards an ESB-Based Enterprise Integration Platform for Geospatial Web Services. In: The Fifth International Conference on Advanced Geographic Information Systems, Applications, and Services, pp. 39–45 (2013)
15. González, L., Cubo, J., Brogi, A., Pimentel, E., Ruggia, R.: RunTime Verification of Behaviour Aware Mashups in the Internet of Things. Presented at the 3rd International Workshop on Adaptive Services for the Future Internet, Malaga, Spain (September 2013)
16. González, L., Ortiz, G.: An ESB based Infrastructure for Event Driven Context Aware Web Services. Presented at the 3rd International Workshop on Adaptive Services for the Future Internet, Malaga, Spain (September 2013)