Journal of Universal Computer Science, vol. 20, no. 8 (2014), 1071-1088
submitted: 4/4/14, accepted: 23/7/14, appeared: 1/8/14 © J.UCS

An Event-Driven Integration Platform for Context-Aware Web Services

Laura González
(Instituto de Computación, Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
lauragon@fing.edu.uy)

Guadalupe Ortiz
(UCASE Software Engineering Group, University of Cádiz, Cádiz, Spain
guadalupe.ortiz@uca.es)

Abstract: Web services are nowadays one of the preferred technologies to implement service-oriented architectures and to communicate distributed applications. On the other hand, context-awareness is highly demanded for distributed applications. However, even though there are excellent tools and frameworks for service development, getting services to be context-aware is still under investigation. In turn, an Enterprise Service Bus (ESB) is a standards-based integration platform, which provides mediation capabilities (e.g. routing, transformation). ESBs are being increasingly used in conjunction with Complex Event Processing (CEP) engines to support event-driven architectures scenarios. In this regard, this paper proposes an ESB-based integration platform which, leveraging its mediation capabilities and a CEP engine, allows the construction of context-aware web services. Concretely, CEP techniques are used to detect the complex situations that may affect services and mediation mechanisms are used to adapt service requests and responses to make them context-aware.

Keywords: web services, context-awareness, complex event processing, enterprise service bus
Categories: D.2.11, D.2.12, D.2.13

1 Introduction

Thanks to the use of XML-based protocols for interface description –Web Service Description Language (WSDL) – and message exchange –Simple Object Access Protocol (SOAP) –, among other facts, web services provide us with a loosely-coupled and platform-independent communication among distributed systems. This is why they have become an efficient solution for the implementation of distributed systems in which modularity and communication among third parties are key factors.

On the other hand, context-aware software solutions have hugely increased in popularity and are highly demanded, especially by mobile users. The great amount of devices and their continuous use clearly illustrate the importance of access not only to desktop services but also to mobile ones. It is important to mention that, even though context awareness seems to be strongly associated with mobile applications, many users start to demand desktop context-aware applications, therefore both markets are relevant for software developers.

Even though there are excellent tools and frameworks for service development, their adaptation to context has not been properly focused on to date. This is an

emerging field in which many industry and scientific communities are starting to provide their proposals. However there are not clear solutions in the scope of web services and the existing ones provide solutions with intrusive context-adaptation [Li, Sehic, & Dustdar, 2010; Q.Z. Sheng et al., 2009] or do not provide a method to integrate context detection in the adaptation platform [Gilman, Su, Davidyuk, Zhou, & Riekki, 2011]. In the past, we proposed a method for adapting services to the invoking device [Ortiz & García De Prado, 2010] as well as to adapt them to the client-specific context [Ortiz & Prado, 2010]; then in [Ortiz, Boubeta-Puig, García de Prado, & Medina-Bulo, 2012] we envisaged an architecture to tackle their adaptation to the external context. All our contributions have fostered the separation of concerns through the use of aspect-oriented techniques in order to facilitate reusing web services in different contextual situations. In [González & Ortiz, 2013], we followed through with the envisaged architecture tackling the services adaptation to external context making use of an Enterprise Service Bus (ESB) and Complex Event Processing (CEP) and specially benefitting from the use of known ESB patterns and the adaptive ESB infrastructure proposed in [González & Ruggia, 2012]. In this paper, we describe with more detail and examples the proposed solution and we discuss its main advantages and limitations. The proposed solution leverages well-known ESB mediation patterns (e.g. transformation) in order to adapt services to context transparently, not only for the final user but also for the service developer since the adaptation code is completely decoupled from the service implementation one. Secondly, the solution leverages complex event processing to analyze the events received from external sources and to detect relevant situations for the context of the service in question. Finally, a context reasoner which provides the transformations to be done depending on the context events has been provided.

The rest of the paper is organized as follows: Section 2 provides background on context-awareness, complex event processing, ESB patterns and an adaptive ESB infrastructure. Then, Section 3 explains the proposed infrastructure which makes use of the ESB and the CEP engine. Afterwards, Section 4 presents some further discussion and Section 5 outlines main related work. Finally, conclusions and future work are provided in Section 6.

2 Background

This section provides background on context-awareness, Service-oriented Architectures (SOAs) and Web Services, CEP, ESB patterns and on an Adaptive ESB. These concepts and solutions are the basis for our proposal.

2.1 Context-Awareness

Abowd et al.'s context definition in [Abowd et al., 1999][Abowd et al., 1999][Abowd et al., 1999][Abowd et al., 1999][Abowd et al., 1999] is specially well-known –page 3, section 2.2: "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*".

The term context-awareness supports the fact that the context information provided by the client, or taken from the environment, is properly used by the system so as to improve its quality; that is, using information such as location, social attributes and other information to foresee the customer necessities so that we can offer more personalized systems. Therefore, a system is context-aware if it uses the context to provide relevant information or services to the user, adapting the system behavior to the particular needs of a specific user.

A context-classification can be found at [García de Prado & Ortiz, 2011]; in this paper we will focus on the environmental context, which describes the environmental conditions of both user and services. Sensors or specific services are normally used in order to provide such kind of information as location, weather, noise, social events, etcetera. This type of context will imply adapting the information sent to the client; for instance, if the user is in a location where it is raining when searching for cultural activities, outdoors ones will be avoided.

2.2 Service-Oriented Architectures and Web Services

In recent years, SOAs have emerged as an efficient solution for the implementation of systems in which modularity and communication among third parties are key factors. This fact has led to the increasing development of distributed applications made up of reusable and sharable components (services). These components have well-defined platform-independent interfaces, which allow SOA-based systems to quickly and easily adapt to changing business conditions

Even though SOAs do not necessarily need to be built through web services, it is de facto standard. Web services are often seen as applications accessible to other applications over the Web [Alonso, Casati, Kuno, & Machiraju, 2004]. Such applications will be accessible via HTTP or SMTP, which facilitates message exchanges, and always provide an interface description (WSDL document) describing the functionality offered by the service in question. Besides, services mainly use SOAP as message format for communication. This message is composed of a mandatory body, where invocation parameters are included, and one or more optional headers.

REST, considered an alternative for Web Services implementation and mainly used from browser-based clients, describes a set of architectural principles by which data can be transmitted currently using HTTP methods. Even though REST can be quite useful for simple services, its multiple limitations suggest the use of SOAP services in many cases [Pautasso, Zimmermann, & Leymann, 2008]; this is why we focused on SOAP-based services in this paper.

2.3 Complex Event Processing and SOA 2.0

CEP [Luckham, 2002] is a technology that provides a set of techniques to help discover complex events by analyzing and correlating other basic and complex events. Therefore, CEP allows the detection of complex and meaningful events, known as situations, and inferring valuable knowledge for end users. For instance, let's suppose that we are looking for tourism activities in a city we are visiting; going to a museum is fine when is raining; however it is not the same that it is raining when

you are going to visit the museum, than it is raining and public transports have gone on strike, when you are five kilometers away from the museum.

In order to detect complex events, event queries have to continuously monitor incoming streams of simple events [Eckert, Bry, Brodt, Poppe, & Hausmann, 2011]. These queries specify situations as a combination of simple events occurring, or not occurring, over time. One approach to implement event queries is by using production rules. This approach is followed by various well-known products like Drools Fusion, which is the component of the Drools platform (http://www.jboss.org/drools/) providing CEP support.

These events will help make decisions when necessary. Currently, the integration of Event-Driven Architecture (EDA) and SOA is known as Event-Driven SOA (ED-SOA) or SOA 2.0 [Sosinsky, 2011]. SOA 2.0 will ensure that services not only exchange messages between them, but also publish events and receive event notifications from others. For this purpose, an Enterprise Service Bus (ESB) will be necessary to process, enrich and route messages between services of different applications. Further information on the integration of CEP with SOA in other scenarios can be found at [Boubeta-Puig, Ortiz, & Medina-Bulo, 2011].

2.4 ESB Patterns

ESB behavior has been characterized through different patterns. This section reviews the relevant connectivity and mediation patterns for this work.

Connectivity patterns specify high level integration styles for ESB-based solutions [Wylie & Lambros, 2009]. For example, service virtualization patterns take an existing service and deploy a new virtual service in the ESB. These patterns introduce a point of mediation in the ESB, between the client and target service, which can be used to route and transform messages, among others. Gateway patterns are used to apply a common set of mediations to all incoming and/or outgoing messages (e.g. security related mediations). Event-driven integration patterns deals with distribution of events through the ESB and the integration with CEP engines. In particular, the event distributor pattern allows the distribution of events to multiple interested parties, the event extractor pattern monitors interactions across the ESB and passes relevant events to a CEP engine, and the event reactor pattern extends the previous one by synchronically interacting with a CEP engine to be informed if the latest event has triggered a complex event.

Mediation patterns specify families of mediation operations that can be performed over messages passing through the ESB [Hérault, Thomas, & Fourier, 2005; Schmidt, Hutchison, Lambros, & Phippen, 2005]. Two commonly supported types of mediation operations are routing and transformation. Routing patterns dynamically determines the message path according to different factors. For example, the content-based routing pattern determines the message path based on its content and the itinerary-based routing determines the message destination based on an itinerary included in the message itself [Chappell, 2004]. Transformation patterns deal with the runtime transformation of messages [Hohpe, 2004]. In particular, the content transformation pattern deal with data transformation (e.g. data model transformation, data format transformation [Erl, 2009]), the content enrichment pattern consists of complementing the message content with data obtained from other sources and the content filter patterns consist of removing unimportant data items from messages.

2.5 Adaptive ESB Infrastructure

This section describes the Adaptive ESB Infrastructure proposed in [González & Ruggia, 2012], which has the goal of dynamically and automatically dealing with adaptation requirements in service based systems at runtime.

The proposed adaptive solution assumes that services communicate by sending messages through the ESB, applying service virtualization patterns. The approach to achieve adaptation at runtime is to intercept all incoming ESB messages and, if an adaptation is required for the invoked service, drive them through adaptation flows. These flows include all the mediations steps (e.g. transformations, routings) required to carry out a specific adaptation strategy (e.g. invoke an equivalent service). In order to know if an adaptation is required for a specific service, the infrastructure maintains a table with adaptation directives for each service. These directives are generated based on monitored service properties and service level requirements.

In order to show the general operation of the infrastructure, Figure 1 presents an example of an adaptation flow which consists of applying a transformation before invoking the target service.

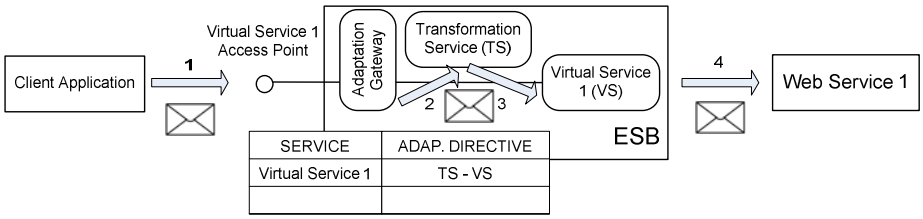

Figure 1: Adaptive ESB Infrastructure

First, the client sends a message through the ESB (1) to invoke the target service. The message is intercepted by an adaptation gateway following the gateway pattern. Given that there is an adaptation directive for the invoked service, the adaptation gateway attaches an adaptation flow to the message and routes it to the first step in the flow (2), following the itinerary-based routing pattern. After the required transformation (also specified in the message) is performed, the message is routed to the next step in the flow (3) which finally invokes the target web service (4).

3 The Proposed Integration Platform

This section presents the proposed solution which, leveraging ESB and CEP capabilities, allows the construction of context-aware web services. The examples in this section are an adaptation of the ones presented in [Kapitsaki, Prezerakos, Tselikas, & Venieris, 2009].

3.1 General Description

The approach consists of applying service virtualization patterns to build and expose context-aware services through the ESB, based on services which are not necessarily

context-aware. The context-aware adaptation logic is executed within the ESB, using its mediation capabilities, and it is automatically generated according to the situations in which the invoking users are. These situations are detected, thanks to their previous definition in the CEP engine, as complex events that are triggered based on the contextual data arriving to the ESB from different sources. Figure 2 presents a high level view of the proposed architecture.

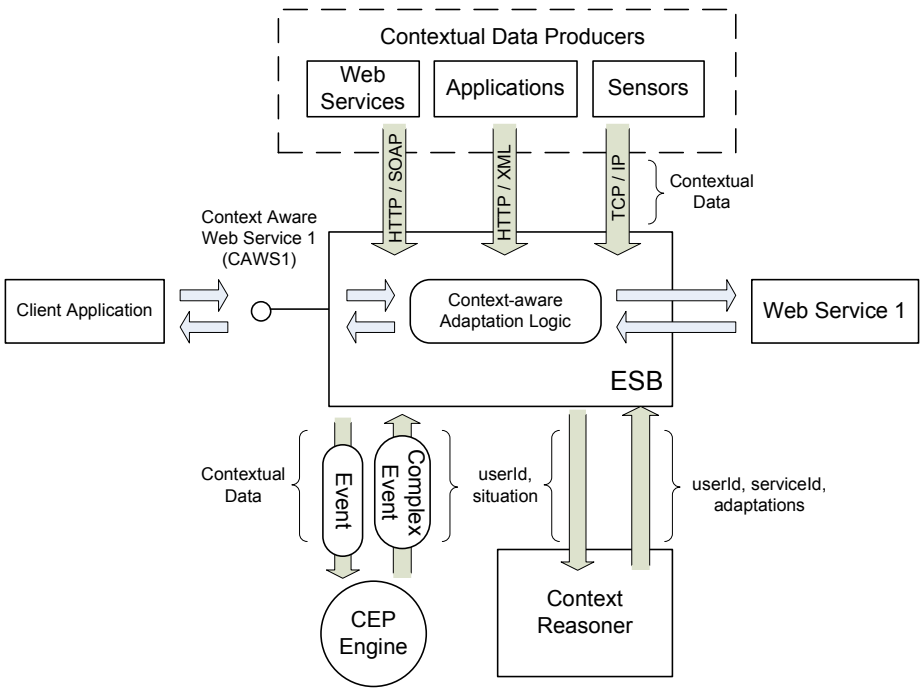

Figure 2: General Architecture

Contextual data producers provide different type of contextual data to the ESB in the form of events. For example, a mobile device with GPS support can provide information regarding the geographic location of a user. Also, sensors can monitor the environment to provide information concerning temperature and rain, among others.
The CEP Engine receives contextual data through the ESB (event distributor pattern) and, based on rules deployed in the engine, detects the complex situations in which users are. For example, based on the geographic location of a user and the weather conditions in a city, the engine can detect that a user is in a place where it is raining.

The Context Reasoner receives the situations in which users are and automatically generates the required adaptations for each configured service. For instance, if a service returns the attractions in a specific city and the user is in a city where it is raining, an adaptation can consist of not returning the attractions which involve outdoor activities. To this end, the situations which affect each service and the adaptations to be applied in each case have to be configured in the Context Reasoner.

Finally, the ESB receives the required adaptations for each pair (user, service) and, when an invocation arrives, applies these adaptations leveraging the ESB mediation capabilities. Following with the previous example, the content filter pattern can be used to take out from the service response (i.e. a SOAP message) the attractions which involve outdoors activities.

As a summary, Figure 3 presents the main conceptual elements of the proposed solution and the previously mentioned examples.

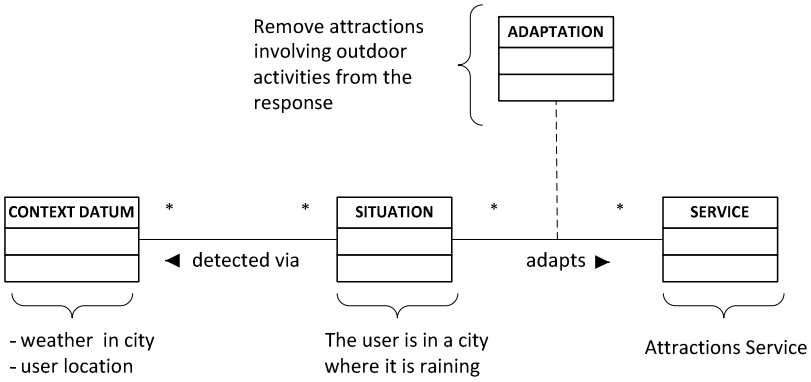

Figure 3: Main Conceptual Elements of the Proposed Solution

3.2 Receiving Contextual Data

Contextual data are obtained from the providers as events, configuring the built-in connectivity capabilities that ESBs offer. The received events are then passed to the CEP Engine. Table 1 presents different types of contextual data that can be received in the context of the examples described in the previous section.

Contextual Data	Description	Structure	Example
Weather in City	It specifies the weather conditions in a city.	[city, temp, rain?]	(Madrid, 17°C, true)
User Location	It specifies the current geographic coordinates of a user.	[userId, lat, long]	(jsmith, 40.41, -3.71)
User Agent	It specifies the current user agent of a user.	[userId, userAgent]	(jsmith, Mozilla/5.0....)
Language Preference	It specifies the language preference of a user.	[userId, language]	(jsmith, EN)

Table 1: Examples of Contextual Data

For instance, the weather information can be obtained by configuring connectors in the ESB in order to periodically invoke a web service or to consume an RSS feed. Also, the user location can be obtained by configuring an HTTP endpoint in the ESB which receives events from an application running on the mobile devices of users and

which has access to the device GPS capabilities. Finally, the user agent can be obtained from the requests performed by the users to the platform.

3.3 Detecting Situations

Situations are detected as complex events by the CEP Engine. In order to specify when a user is in a particular situation, a set of rules or patterns have to be deployed within the engine for each situation. These rules can be based on: the received contextual data, other detected situations and utility functions. Table 2 shows some situations that can be detected using the examples of contextual data presented in section 3.2 and the required elements in order to detect them.

Situation	Description	Elements to Detect the Situation
InCity	The user is in a specific city.	Contextual Data: User Location Utility Function: getCityFromCoords (lat, long)
InCityRaining	The user is in a city where it is raining.	Contextual Data: Weather in City Situations: InCity
UsingMobileDevice	The user is using a mobile device.	Contextual Data: User Agent Utility Function: isMobile(userAgent)
LanguagePreference	The user prefers a specific language.	Contextual Data: Language Preference

Table 2: Examples of Situations

The specification of the complex events to detect these situations can be performed using different languages according to the specific CEP Engine that is used. In the following examples, the Drools Rule Language (DRL), the language leveraged by the Drools Fusion platform, is used to illustrate the proposal. Figure 4 presents a DRL Rule to trigger the InCity situation. Concretely, whenever user location data are received, the utility function getCityFromCoords is used to get the current city of the user and an InCity event with this information is triggered.

```
rule "User in City"
    when
        $usrLoc : UserLocation()
    then
        InCity usrCity = new InCity();
        usrCity.setCity(getCityFromCoords($usrLoc.getLat(), $usrLoc.getLon()));
        usrCity.setUserId($usrLoc.getUserId());
        usrCity.setTimestamp($usrLoc.getTimestamp());
        // trigger the situation InCity
        insert (usrCity);
end
```

Figure 4: DRL Rule to Trigger the InCity Situation

Similarly, Figure 5 presents a DRL Rule to trigger the InCityRaining situation. In this case the event is triggered when it is detected that a user is in a specific city and it is raining in the named city.

```
rule "User in City Raining"
    when
        $userInCity : InCity()
        CityWeather (city == $userInCity.city, raining == true)
    then
        InCityRaining usrCityRain = new InCityRaining();
        usrCityRain.setUserId($userInCity.getUserId());
        usrCityRain.setTimestamp($userInCity.getTimestamp());
        // trigger the situation InCityRaining
        insert (usrCityRain);
end
```

Figure 5: DRL Rule to Trigger the Situation InCityRaining

As mentioned above, although the examples are based on DRL and Drools Fusion, other languages and CEP Engines can be used to specify and detect these situations.

3.4 Configuring Context-aware Adaptations

In order to adapt services according to the situations in which the invoking users are, the following data have to be configured in the Context Reasoner: i) the situations which affect each service; ii) the adaptations to be applied in each case; iii) the moment to apply these adaptations (i.e. in the service request or response).

As an example, consider an Attractions service with an operation named getAttractions which optionally receives the name of a city and returns a list of attractions. For each attraction the following data is returned: name, short description, long description and a Boolean value indicating if the attraction involves outdoor activities. Table 3 presents some adaptations that can be configured for this service, according to the situations presented in Section 3.3, and the required mediation patterns to implement these adaptations.

Situation	Adaptation / Moment	Pattern
InCity	Add to the request the optional parameter to specify the city.	Content Enrichment
InCityRaining	Remove from the response the attractions involving outdoor activities.	Content Filter
UsingMobileDevice	Remove from the response the long description of the returned attractions.	Content Filter
LanguagePreference	Translate the short description of the attractions in the response.	Content Transformation

Table 3: Configuration of Adaptations

In order to implement the adaptations, the mediation patterns which are used in each one of them have to be configured with all the required information. For instance, in the first adaptation of Table 3, an XSLT transformation can be specified

for the content enrichment pattern so that SOAP request messages can be transformed by adding the city parameter. Note that these adaptations are not going to be completely specified until a specific situation is detected for a user, i.e., in the aforesaid XSLT transformation the concrete city is not specified at this time.

An ESB can provide different implementations for a given mediation pattern. For instance, the content transformation pattern is usually implemented through XSLT transformations or template engines. Also, ESBs are designed to be easily extended, for example, with new mediation patterns and new implementations for them. In this way, if a complex transformation logic is required (e.g. language translation) the proper implementation (e.g. using an external translation service) can be set up in the ESB using its extensibility mechanisms.

Finally, the adaptations for each service have to be prioritized so that the ESB knows which one applies first, in case more than one has to be performed.

3.5 Adapting Services within the ESB

After configuring the adaptations for each service, when the Context Reasoner receives information regarding the situation of a user, it can automatically generate the required adaptation logic to be performed for the pair (user, service) and communicate it to the ESB. For example, assuming that the Context Reasoner was informed that the users "jsmith" and "awright" are in Madrid and London, respectively, Table 4 presents two different concrete adaptations, to be sent to the ESB, resulting from the first adaptation of Table 3.

Service / Operation	User	Adaptation
AttractionsService / getAttractions	jsmith	XSLT transformation adding the Madrid value, for the city parameter, in the SOAP request.
AttractionsService / getAttractions	awright	XSLT transformation adding the London value, for the city parameter, in the SOAP request.

Table 4: Adaptations to be Applied in the ESB

This way, users receive different results from the Attractions service according to their specific situation, in this case, the city where they currently are.

In order to dynamically apply these adaptations within the ESB, the Adaptive ESB Infrastructure presented in Section 2.5 is used. Concretely, the information which is sent to the ESB is treated as adaptation directives to be applied to the invocations to the Attractions service coming from the specified users. To this end, the Adaptive ESB Infrastructure was enhanced to consider users information in the requests and to apply adaptation directives. Figure 6 presents how the Adaptive ESB Infrastructure, with the aforementioned enhancements, is used to apply the context-aware adaptation logic presented in Table 4.

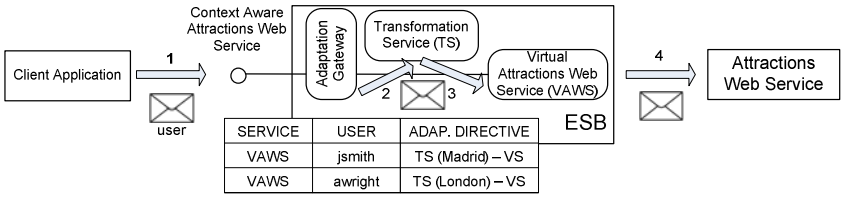

Figure 6: Applying Context-aware Adaptations

4 Further Discussion

In this section we discuss the use of other mediation patterns to perform context-aware adaptations, as well as the advantages, disadvantages and limitations of the proposed solution.

4.1 Using other Mediation Patterns

The examples presented in Section 3 only use transformation patterns (e.g. content filter, content enrichment) to perform the context-aware adaptations. However, ESB products provide a rich set of mediation capabilities which can also be used to perform adaptations.

In particular, routing mediation patterns can be used to route a service request to other service/s in order to provide a more suitable response according to the situation of the invoking user. Following with the example presented in section 3.4, if there is a new service which only provides information regarding attractions in Madrid and it was detected that a user is in Madrid, the requests from this user can be adapted by routing them to this new service. Also, if a user is in a situation (e.g. an emergency situation) which requires a higher level of quality of service (e.g. regarding availability), requests from this user can be routed to equivalent services which fulfill these quality of service requirements.

Other mediation patterns that can be used to adapt to quality of service requirements are: cache, recipient list and aggregator [González & Ruggia, 2011]. For instance, the cache mediation pattern [Rao et al., 2006] receives a request and returns a response which was previously stored and returned for that request. This can be used, for example, to deal with response time or availability requirements.

4.2 Advantages, Disadvantages and Limitations of the Solution

The main advantage of the proposed solution is that given that the context-aware adaptation logic is performed within the ESB, it is completely decoupled from services and their clients. This way, services and applications developers do not have to deal with these matters and they can delegate this kind of features to the integration platform. Also, the solution promotes a clear separation between business logic and context-adaptation logic which can facilitate the maintainability of this kind of software solutions. In the past we also made a proposal in which the adaptation to context was developed through the use of aspect-oriented programming [Ortiz et al.,

2012]; however we considered that, if the ESB can proceed at runtime with the adaptation in a completely decoupled way – so that services implementation do not have to be modified– then the use of dynamic AOP could introduce a greater overhead. In any case, in our close future work we are developing two implementations of a common case-study –with ESB adaptation and with AOP-based adaptation– in order to evaluate and compare the efficiency of both solutions.

On the other side, the main disadvantage of the proposal is the overhead that it introduces in service invocations. The execution of some adaptations can be time consuming and may degrade the response time perceived by the end user. Regarding this point, in previous work [González, Laborde, Galnares, Fenoglio, & Ruggia, 2013] we performed tests in order to quantify the overhead in the invocations introduced by the Adaptive ESB Infrastructure described in Section 2.5. The result of these tests, which were obtained based on 1200 invocations to a Web Service, have shown that the adaptations which involve transformations are the ones which introduce more overhead (332 milliseconds). However, applying the cache mediation pattern leads to lower response times than invoking the target service. In summary, we believe that the obtained values are acceptable, given that for most of the adaptations the overhead introduced by the platform is less than 200 milliseconds.

Lastly, the main limitations of the solution concern dealing with advanced security characteristics, handling changes in web services contracts and the required time to detect a situation and build a suitable adaptation. Regarding security, if web services require exchanging encrypted or digitally signed information (e.g. using WS-Security mechanisms), some adaptation actions (i.e. transformations) may not be possible. With respect to changes in web services contracts, given that the specification of some adaptations (e.g. transformations using XSLT) are specified based on the web service technical contract (e.g. WSDL message declarations), if a web service contract changes a transformation may become invalid to perform a specific adaptation; please bear in mind that this is not only applicable to the adaptation logic but contract changes may imply invalid invocations in general. Finally, given the event-based mechanism to detect situations, some adaptations may not be available on time for a specific request when the situation of a user changes. Even though we use real time engines for event detection, this could happen if the time required for detecting a situation and building a suitable adaptation for it, takes longer than the arrival of a new request from the user.

5 Related Work

In this section we will highlight the main research on the integration of CEP and SOA, the use of complex-event processing for context-awareness and approaches for context-aware service implementation.

Several works on CEP and SOA integration in different domains can be found in the literature; in the following lines we summarize some representative ones. For instance, Taher et al. [Taher, Fauvet, Dumas, & Benslimane, 2008] propose the adaptation of interactions of web service messages between incompatible interfaces. In this regard, they develop an architecture that integrates a CEP engine and input/output adapters for SOAP messages. Input adapters receive messages sent by web services, transform them into the appropriate representation to be manipulated by

the CEP engine and send them to the latter. Similarly, output adapters receive events from the engine, transform them to SOAP messages and then they are sent to web services.

There are some approaches which use CEP for monitoring such as the one from Xu et al. [Xu, Wolf, Stojanovic, & Happel, 2010], where CEP is used in the Ambient Assisted Living (AAL) domain. They propose its use to detect events in AAL for being able to take real time actions. The paper from Li et al. [Li et al., 2010] is also worth a special mention. They provide an adaptive approach to context provisioning and automatic generation of actions. The latter definitely bears similarities with our proposal, however we focus on non-intrusive service result adaptation rather than action taking.

Most of the work found in the context adaptation area specially focuses on client side adaptation. We can mention, for instance, the paper from Laakko and Hiltunen [Laakko & Hiltunen, 2005] where content adaptation is done through a proxy. They focus on adapting XHTML (Extensible Hypertext Markup Language) with XHTML MP (XHTML mobile profile) and WML (Wireless Markup Language). Another example is URICA [Mohomed, Cai, Chavoshi, & de Lara, 2006]: a technique for automatic content adaptation for mobile devices presented by Mohomed et al. The system can learn through interaction with the user, identifying the most relevant context for the latter. It is very interesting work, but it overheads the client computation.

The paper from Gilman et al [Gilman et al., 2011] also deserves special mention. They provide a framework for adapting services to context through a complex architecture composed of several components, among them a context-reasoner, context discoverer and observers, handlers and managers. There are also systems based on multi-agents, such as the one presented by Fraile et al [Fraile, De Paz, Bajo, De Paz, & Corchado, 2012], which uses them for implementing context-aware computing for home care. Sheng et al [Quan Z. Sheng & Benatallah, 2005] proposes ContextUML, a modeling language for context-aware model-driven web services. Several years later they improved their proposal supplying [Q.Z. Sheng et al., 2009] a platform for developing context-aware web services. This platform, named ContextServ, is based on ContextUML and provides an integrated environment where developers can specify and deploy context-aware services as well as generating BPEL code. The main drawback of this proposal is that the way in which context is modelled is rather complex; it requires a high learning curve and it does not seem to be intuitive for a software developer. Follow-ups on the project are more focused on BPEL compositions [Yahyaoui, Mourad, Almulla, Yao, & Sheng, 2012], or user personalization [Yu, Han, Sheng, & Gunarso, 2012]. In any case, none of these works takes advantage of the use of the ESB and CEP, which leverages the context-aware system usability and maintenance.

We also mention some other related approaches. Strobbe et al also makes use of ontologies in [Strobbe, Van Laere, Dhoedt, De Turck, & Demeester, 2011] to provide a Context-Aware Service Platform for adding and abstracting context information. This proposal is not particularly useful for web services and only focuses on context management but not on context adaptation. Not only is the environmental context relevant, but also specific user preferences. In this sense we can mention, for instance, the paper from Bao et al [Bao, Cao, Chen, Tian, & Xiong, 2012], where techniques to

model and learn personalized contexts are provided. Since we also take care of personal context in our approach, this proposal could be used in conjunction with our adaptation process. A thorough analysis of context-awareness related work can be found in [García de Prado & Ortiz, 2011].

To sum up, our proposal mainly differs from others in benefiting from the advantages of the use of CEP and an ESB to adapt services to context information in a decoupled way, where the context can be automatically detected through real time events.

6 Conclusions and Future Work

Even though context-awareness is an important capability in current distributed systems, solutions to provide context-aware web services are still under investigation. This paper addresses this issue by proposing an ESB-based integration platform which, leveraging its mediation capabilities and a CEP engine, allows the construction of context-aware web services. CEP techniques are used to detect complex situations that may affect services and mediation mechanisms are used to adapt services requests and responses to make them context-aware.

The proposed solution has several advantages. Firstly, given that the adaptation logic is performed at the ESB, services are adapted to context in a transparent way without affecting clients or services. Secondly, the use of CEP techniques allows to process different type of information with high throughput and detecting complex situations to adapt services. Finally, the design of the proposed architecture is based on commonly supported ESB and CEP patterns so it is likely to be implemented in a wide range of products supporting these characteristics.

On the other hand, several aspects remain to be addressed or analyzed. For instance, this paper only focuses in some ESB mediation capabilities. Therefore a further analysis is required to know how other more advanced mediation capabilities can be leveraged to adapt services to context within the ESB. There is also the need to analyze how advanced web services features (e.g. security, transactions support, asynchronous invocations) may affect the proposed solution. For example, if SOAP messages are encrypted or digitally signed some of the adaptation mechanisms may not be applied. Finally, even though proofs of concepts have been performed with the different technologies over which the proposed solution is based (i.e. ESB and CEP), a more complete prototype has to be developed in order to evaluate the solution as a whole, regarding functional and non-functional aspects. Two implementations of a common case study will be developed – one based on ESB adaptations and the other on aspect-oriented adaptations – and will be evaluated with the aim of proving which the optimal solution is. Future work will mainly concentrate on these aspects.

Acknowledgements

G. Ortiz acknowledges the support from Ministerio de Ciencia e Innovación (TIN2011-27242)

References

[Abowd, Dey, Brown, Davies, Smith, & Steggles, 1999]. Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. Towards a Better Understanding of Context and Context-Awareness [pp. 304–307]. London, UK: Springer-Verlag. (1999) DOI:10.1007/3-540-48157-5_29

[Alonso, Casati, Kuno, & Machiraju, 2004]. Alonso, G., Casati, F., Kuno, H., & Machiraju, V. *Web services: concepts, architectures, and applications*. Berlin ; New York: Springer. (2004)

[Bao, Cao, Chen, Tian, & Xiong, 2012]. Bao, T., Cao, H., Chen, E., Tian, J., & Xiong, H. An unsupervised approach to modeling personalized contexts of mobile users. *Knowledge and Information Systems*, *31*[2], 345–370. (2012) DOI:10.1007/s10115-011-0417-1

[Boubeta-Puig, Ortiz, & Medina-Bulo, 2011]. Boubeta-Puig, J., Ortiz, G., & Medina-Bulo, I. An Approach of Early Disease Detection using CEP and SOA. In *Service Computation 2011, The Third International Conferences on Advanced Service Computing* [pp. 143–148]. (2011) Retrieved from http://www.thinkmind.org/index.php?view=article&articleid=service_computation_2011_6_30_10134

[Chappell, 2004]. Chappell, D. A. *Enterprise service bus* [1st ed.]. Sebastopol, Calif: O'Reilly. (2004)

[Eckert, Bry, Brodt, Poppe, & Hausmann, 2011]. Eckert, M., Bry, F., Brodt, S., Poppe, O., & Hausmann, S. A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed. In S. Helmer, A. Poulovassilis, & F. Xhafa [Eds.], *Reasoning in Event-Based Distributed Systems* [Vol. 347, pp. 47–70]. Berlin, Heidelberg: Springer Berlin Heidelberg. (2011) Retrieved from http://link.springer.com/10.1007/978-3-642-19724-6_3

[Erl, 2009]. Erl, T. *SOA design patterns* [1st ed.]. Upper Saddle River, NJ: Prentice Hall. (2009)

[Fraile, De Paz, Bajo, De Paz, & Corchado, 2012]. Fraile, J. A., De Paz, J., Bajo, J., De Paz, J. F., & Corchado, J. M. Combining case-based reasoning systems and support vector regression to evaluate the atmosphere–ocean interaction. *Knowledge and Information Systems*, *30*[1], 155–177. (2012) DOI:10.1007/s10115-010-0368-y

[García de Prado & Ortiz, 2011]. García de Prado, A., & Ortiz, G. Context-Aware Services: A Survey on Current Proposals. In *The Third International Conferences on Advanced Service Computing* [pp. 104–109]. Rome, Italy. (2011) Retrieved from http://www.thinkmind.org/index.php?view=article&articleid=service_computation_2011_5_10_10018

[Gilman, Su, Davidyuk, Zhou, & Riekki, 2011]. Gilman, E., Su, X., Davidyuk, O., Zhou, J., & Riekki, J. Perception framework for supporting development of context-aware web services. *International Journal of Pervasive Computing and Communications*, *7*[4], 339–364. (2011) DOI:10.1108/17427371111189665

[González, Laborde, Galnares, Fenoglio, & Ruggia, 2013]. González, L., Laborde, J., Galnares, M., Fenoglio, M., & Ruggia, R. An Adaptive Enterprise Service Bus Infrastructure for Service Based Systems. Presented at the 1st Workshop on Pervasive Analytical Service Clouds for the Enterprise and Beyond, Berlin, Germany. (2013)

[González & Ortiz, 2013]. González, L., & Ortiz, G. An ESB-Based Infrastructure for Event-Driven Context-Aware Web Services. In C. Canal & M. Villari [Eds.], *Advances in Service-Oriented and Cloud Computing* [Vol. 393, pp. 360–369]. Springer Berlin Heidelberg. (2013) Retrieved from http://link.springer.com/chapter/10.1007%2F978-3-642-45364-9_29

[González & Ruggia, 2011]. González, L., & Ruggia, R. Addressing QoS issues in service based systems through an adaptive ESB infrastructure [pp. 1–7]. ACM Press. (2011) DOI:10.1145/2093185.2093189

[González & Ruggia, 2012]. González, L., & Ruggia, R. Adaptive ESB Infrastructure for Service Based Systems. In G. Ortiz & J. Cubo [Eds.], *Adaptive Web Services for Modular and Reusable Software Development: Tactics and Solutions*. IGI Global. (2012) Retrieved from http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-4666-2089-6

[Hérault, Thomas, & Fourier, 2005]. Hérault, C., Thomas, G., & Fourier, U. J. Mediation and Enterprise Service Bus: A position paper. In *Proceedings of the First International Workshop on Mediation in Semantic Web Services*. (2005)

[Hohpe, 2004]. Hohpe, G. *Enterprise integration patterns: designing, building, and deploying messaging solutions*. Boston: Addison-Wesley. (2004)

[Kapitsaki, Prezerakos, Tselikas, & Venieris, 2009]. Kapitsaki, G. M., Prezerakos, G. N., Tselikas, N. D., & Venieris, I. S. Context-aware service engineering: A survey. *J. Syst. Softw.*, *82*[8], 1285–1297. (2009) DOI:10.1016/j.jss.2009.02.026

[Laakko & Hiltunen, 2005]. Laakko, T., & Hiltunen, T. Adapting Web Content to Mobile User Agents. *IEEE Internet Computing*, *9*[2], 46–53. (2005) DOI:10.1109/MIC.2005.29

[Li, Sehic, & Dustdar, 2010]. Li, F., Sehic, S., & Dustdar, S. COPAL: An adaptive approach to context provisioning [pp. 286 –293]. (2010) DOI:10.1109/WIMOB.2010.5645051

[Luckham, 2002]. Luckham, D. C. *The power of events: an introduction to complex event processing in distributed enterprise systems*. Addison-Wesley. (2002)

[Mohomed, Cai, Chavoshi, & de Lara, 2006]. Mohomed, I., Cai, J. C., Chavoshi, S., & de Lara, E. Context-aware interactive content adaptation. In *Proceedings of the 4th international conference on Mobile systems, applications and services - MobiSys 2006* [p. 42]. Uppsala, Sweden. (2006) DOI:10.1145/1134680.1134686

[Ortiz, Boubeta-Puig, García de Prado, & Medina-Bulo, 2012]. Ortiz, G., Boubeta-Puig, J., García de Prado, A., & Medina-Bulo, I. Towards Event-Driven Context-Aware Web Services. In *Adaptive Web Services for Modular and reusable Software Development: Tactics and Solutions* [pp. 148–159. DOI: 10.4018/978–1–4666–2089–6.ch005]. IGI Global. (2012)

[Ortiz & García De Prado, 2010]. Ortiz, G., & García De Prado, A. Improving device-aware Web services and their mobile clients through an aspect-oriented, model-driven approach. *Information and Software Technology*, *52*[10], 1080–1093. (2010) DOI:10.1016/j.infsof.2010.05.002

[Ortiz & Prado, 2010]. Ortiz, G., & Prado, A. G. de. Web Service Adaptation: A Unified Approach versus Multiple Methodologies for Different Scenarios. In *2010 Fifth International Conference on Internet and Web Applications and Services* [pp. 569–572]. Barcelona, Spain. (2010) DOI:10.1109/ICIW.2010.90

[Pautasso, Zimmermann, & Leymann, 2008]. Pautasso, C., Zimmermann, O., & Leymann, F. Restful web services vs. "big"' web services: making the right architectural decision [p. 805]. ACM Press. (2008) DOI:10.1145/1367497.1367606

[Rao, Fang, Tian, Lane, Srinivasan, Banks, & Lei, 2006]. Rao, Y. F., Fang, R., Tian, Z., Lane, E., Srinivasan, H., Banks, T., & Lei, H. Cache mediation pattern specification: an overview. (2006) Retrieved from http://www.ibm.com/developerworks/webservices/library/ws-soa-cachemed/?S_TACT=105AGX52&S_CMP=cn-a-ws

[Schmidt, Hutchison, Lambros, & Phippen, 2005]. Schmidt, M.-T., Hutchison, B., Lambros, P., & Phippen, R. The Enterprise Service Bus: Making service-oriented architecture real. *IBM Systems Journal*, *44*[4], 781–797. (2005) DOI:10.1147/sj.444.0781

[Quan Z. Sheng & Benatallah, 2005]. Sheng, Q. Z., & Benatallah, B. ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In *International Conference On Mobile Business* [pp. 206–212]. (2005) DOI:10.1109/ICMB.2005.33

[Q.Z. Sheng, Pohlenz, Yu, Wong, Ngu, & Maamar, 2009]. Sheng, Q. Z., Pohlenz, S., Yu, J., Wong, H. S., Ngu, A. H. H., & Maamar, Z. ContextServ: A platform for rapid and flexible development of context-aware Web services. In *International Conference on Software Engineering* [pp. 619–622]. (2009) DOI:10.1109/ICSE.2009.5070570

[Sosinsky, 2011]. Sosinsky, B. *Cloud Computing Bible*. John Wiley & Sons. (2011)

[Strobbe, Van Laere, Dhoedt, De Turck, & Demeester, 2011]. Strobbe, M., Van Laere, O., Dhoedt, B., De Turck, F., & Demeester, P. Hybrid reasoning technique for improving context-aware applications. *Knowledge and Information Systems*, *31*[3], 581–616. (2011) DOI:10.1007/s10115-011-0411-7

[Taher, Fauvet, Dumas, & Benslimane, 2008]. Taher, Y., Fauvet, M.-C., Dumas, M., & Benslimane, D. Using CEP technology to adapt messages exchanged by web services [pp. 1231–1232]. New York, NY, USA: ACM. (2008) DOI:10.1145/1367497.1367741

[Wylie & Lambros, 2009]. Wylie, H. M., & Lambros, P. Enterprise Connectivity Patterns: Implementing integration solutions with IBM's Enterprise Service Bus products. (2009) Retrieved from https://www.ibm.com/developerworks/library/ws-enterpriseconnectivitypatterns/

[Xu, Wolf, Stojanovic, & Happel, 2010]. Xu, Y., Wolf, P., Stojanovic, N., & Happel, H.-J. Semantic-based Complex Event Processing in the AAL Domain. In A. Polleres & H. Chen [Eds.], *ISWC Posters&Demos* [Vol. 658]. CEUR-WS.org. (2010) Retrieved from http://dblp.uni-trier.de/db/conf/semweb/pd2010.html#XuWSH10

[Yahyaoui, Mourad, Almulla, Yao, & Sheng, 2012]. Yahyaoui, H., Mourad, A., Almulla, M., Yao, L., & Sheng, Q. Z. A synergy between context-aware policies and AOP to achieve highly adaptable Web services. *Service Oriented Computing and Applications*, *6*[4], 379–392. (2012) DOI:10.1007/s11761-012-0113-3

[Yu, Han, Sheng, & Gunarso, 2012]. Yu, J., Han, J., Sheng, Q. Z., & Gunarso, S. O. PerCAS: An Approach to Enabling Dynamic and Personalized Adaptation for Context-Aware Services. In C. Liu, H. Ludwig, F. Toumani, & Q. Yu [Eds.], *Service-Oriented Computing* [Vol. 7636, pp. 173–190]. Berlin, Heidelberg: Springer Berlin Heidelberg. (2012) Retrieved from http://www.springerlink.com/index/10.1007/978-3-642-34321-6_12