# Creating Context-Adaptive Business Processes

Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien

# Creating Context-Adaptive Business Processes

Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien

INRIA Lille - Nord Europe, ADAM Project-Team
Univ. Lille 1 - LIFL CNRS UMR 8022
Lille, France
firstname.lastname@inria.fr

**Abstract.** As the dynamicity of today's business environments keeps increasing, there is a need to continuously adapt business processes in order to respond to the changes in those environments and keep a competitive level. By using complex event processing, we can discover information that is relevant to our organization, which is usually hidden among the data generated in the environment, and use it to adapt the processes accordingly in order to respond to the changing conditions in an optimal way. Unfortunately, the static nature of business process definitions makes it impossible to adapt them at runtime and the redeployment of a modified process is required. By using a component-based approach, we can transform the existing business processes into dynamically bound components, adding the flexibility needed to adapt the processes at runtime. In this paper we present CEVICHE, a framework that combines the strengths of complex event processing and dynamic business process adaptation, which allows to respond to the needs of the rapidly changing environment, and its adaptation language called SBPL, an extension to BPEL which adds flexibility to business processes.

## 1 Introduction

In order to maintain a competitive level, organizations are increasingly using services to facilitate the integration of their business processes. These services are loosely-coupled instances, which enable the separation of concerns. To orchestrate them, organizations rely mainly in well known standards such as BPEL (*Business Process Execution Language*). As business processes evolve and get more complex, the data around them increases exponentially, to the point where it becomes almost impossible to find valuable information among it.

*Complex Event Processing* (CEP) is an emerging technology that can help the organizations to benefit from this data, since it allows them to find real-time relationships between different events, using elements such as timing, causality, and membership in a stream of data in order to extract relevant information [1]. CEP can be used in a wide variety of applications, like preventing theft of merchandise [2], monitoring the stock market [3], and interacting with RFID systems [4].

However, there are some occasions in which it is not enough just to be able to obtain this information from simple raw data. A better approach would be

that, once the information is found, the system could automatically adapt itself accordingly in order to respond to the presented scenario. In this case, we want to apply this approach to business processes, allowing them to be dynamically and automatically adapted, according to the information discovered using CEP in order to continue in an optimal way, and this is why we developed CEVICHE (*Complex EVent processIng for Context-adaptive processes in pervasive and Heterogeneous Environments*).

The purpose of CEVICHE is to create context-aware business processes that are able to adapt dynamically in order to respond to different unpredictable scenarios. CEVICHE relies on BPEL, since it is the most common orchestration language, it is an OASIS standard [5], and it is an execution language and not a modeling language (like BPMN). With CEVICHE the adaptation of the business process happens at runtime during the execution. The decisions of how to respond to a specific scenario are done by collecting data from different sources and transforming it into useful information, using CEP. Separating the decision making process from the core business process can help to keep a better understanding of the process, by avoiding to have cross-cutting code in the main process definition. In this paper we present the *Standard Business Process Language* (SBPL), an extension of BPEL which allows the user define the adaptation points in a business process. By using a component-oriented approach, we can define alternative processes that can be integrated into the existing business process at runtime, allowing the business process to adapt in a dynamic way.

We have already presented CEVICHE in previous work, where we used an aspect-oriented approach to monitor and maintain the Quality of Service (QoS) of a business process, by monitoring the performance and availability of the different tasks of the process, adapting the process in order to maintain a good QoS whenever a decrease of those parameters was detected [6]. In this paper we are using a component-oriented approach to show the flexibility of CEVICHE to deal with different adaptation approaches, however we mainly want to focus on the way in which this adaptation is defined by the user using CEVICHE's adaptation language, called SBPL, and the event definitions.

The rest of this paper is organized as follows. In Section 2, we use a scenario to illustrate the motivation and challenges of our proposal. Section 3 presents a background of the different domains used in this paper. In Section 4, we present the SBPL and show how events are defined in CEVICHE. Section 5 explains the CEVICHE framework and its architecture. In Section 6, we discuss our proposal and how it solves the presented challenges. Section 7 presents some of the related work. Finally, Section 8 concludes and discusses some future work.

## 2 Motivation and Challenges

In order to show how CEVICHE can help to improve the business process by adding dynamic adaptation, we will use a small health-care scenario. The goal of the process is to diagnose a patient, given her vital signs (*i.e.* temperature, blood pressure and heart-beat rate), as can be seen in Fig. 1. If the vital signs

are normal, the results are logged and the process finishes, otherwise, the process diagnoses the problem and logs the results.

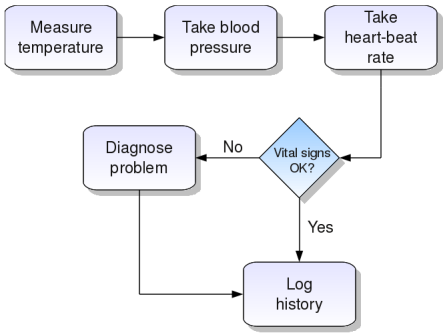


**Fig. 1.** The health-care sample process

Even though this is a very simple scenario, we can identify several limitations. For example, if the information provided by the previous steps of the process were not enough to diagnose a problem, and we needed other information to accurately diagnose it (*e.g.* saliva PH, blood oxygen), there is no way to add it to the process. We would need the user to start a new process that requests the missing information in order to complete the diagnostic.

We could add additional steps to the process, like a validation in which, if the first steps of the process do not provide enough information to diagnose the problem, then the process will continue gathering other information until the diagnostic can be completed. However, these additional steps are different for each scenario, since the information needed to diagnose each problem is not the same. Moreover, we do not want to have all the code for every case in the main process definition, since this will generate code scattering and create a very complex code that will be very hard to maintain. Finally, given the static nature of business process definitions, whenever we want to add a new problem detection to the process, or change the way in which an existing problem is being diagnosed, we would need to modify and redeploy the whole process, stopping the service while doing so and losing all the unfinished transactions.

Given these limitations, we can extract three challenges for the scenario:

1. *Situation identification.* In certain cases, the process needs to collect additional information about the patient, but we cannot anticipate what information is needed in each case.
2. *Code scattering.* Even when we can anticipate the additional information, the request for such information would have to be included in the main process definition, generating code scattering and a complex logic in the process.
3. *Process redeployment.* If we need to add a new problem to diagnose, or change the way an existing problem is being diagnosed, the whole process would need to be redeployed.

## 3 Background

In this section, we present a brief introduction to the main domains addressed in this paper: complex event processing, business process execution language, and component-oriented programming.

### 3.1 Complex Event Processing

CEP is an emerging technology for finding relationships between series of simple and independent events from different sources, using previously defined rules [1]. The CEP technology can be used, among a lot of other things, to enrich the enterprise's existing processes, by introducing rules that will allow the capture of relevant information from the different steps of their business process [7].

For example, let us consider the scenario of a retail store that keeps a record of its inventory in an existing *Enterprise Resource Planning* (ERP) system and wants to keep a live monitoring of its stocks in order to prevent shortage. To achieve this, the store installs a CEP engine that will monitor the products' movements through their life cycle in the store process by receiving and analyzing all the events generated by every change in the state. Since the objective is to monitor inventory, the CEP engine will only keep the events related to changes in the inventory and forget about the rest. By creating the necessary CEP rules, the configuration is set to specify the lowest acceptable stock of product that the store can have to avoid a shortage, *e.g.*, a 10% for normal products and a 5% for some low-demand products. Whenever a product reaches a minimum, the CEP engine alerts the managers so they can make a supply order.

In addition to that, CEP can also be used to predict unexpected situations. To complement the previous example, we can say that because of a global pandemic alert, hand sanitizers are very popular and are selling a lot more than usual. Given this demand, the store will run out of hand sanitizer before they can resupply it, even with the minimum stock alert. By adding some specialized CEP rules to analyze the frequency of sells of each product during the last 4 or 5 hours, the engine can polarize these values to know in advance (if the sells rates are kept) that it will need to resupply before the expected time, which will allow them to react in time even before it reaches the minimum level.

### 3.2 Business Process Execution Language

The *Business Process Execution Language* (BPEL) is an XML-based language for composing services, created by IBM, BEA Systems and Microsoft in 2002, and later approved as an OASIS Standard as WS-BPEL 2.0 [5]. There are two types of service composition: orchestration (execution) and choreography (control). BPEL is an orchestration language, which means that it focuses on the flow of control and data among the different services of the business process, rather than on the specification of peer-to-peer collaboration.

BPEL uses web services as a way to communicate with the different parties involved in the business process. It has two types of activities: primitive and

structured. The former refers to atomic or single activities while the latter refers to composite activities (a combination of several activities). Some instructions like `invoke`, `receive` or `assign` refer to the primitive activities, while `sequence` and `flow` are part of the structured activities and refer to the order in which the activities will execute.

In order to interact with the different parties of the business process (called `partners`), we need to define a partner link, which specifies the roles of the partner and the caller. We also need to define the different input and output variables that we will use to send information to and receive information from the service. Finally, BPEL also provides some facilities for transaction and exception handling.

### 3.3 Component-oriented Programming

Taylor *et al.* define a software component as an architectural entity that encapsulates a subset of the system's functionality and/or data, restricts access to that subset via an explicitly defined interface, and has explicitly defined dependencies on its required execution context [8].

Component-oriented programming enables programs to be constructed from prebuilt software components, which are reusable, self-contained blocks of computer code. These components have to follow certain predefined standards including interface, connections, versioning, and deployment [9]. Component-oriented programming enables the development of software by assembling independent components into a software architecture.

## 4 Events definitions and the SBPL

The CEVICHE framework uses events to identify the situations where a process adaptation is required. In this section we present how these events are defined and how they are used by CEVICHE's extension to BPEL, the *Standard Business Process Language* (SBPL), in order to trigger the business process adaptation.

### 4.1 Event definitions

Event definitions are one of the main parts of the CEVICHE framework. Since the goal of CEVICHE is to work with any CEP engine, we need to provide the user with a powerful enough event descriptor, in order to be able to use the engine's capabilities in the best possible way, after the event definitions are translated. We are also concerned about standards, since it will be easier to develop a translation plug-in for a specific CEP engine if the source were in a well documented format, and this is why we use XPath[1] expressions for our event definitions, a query language for XML defined by the World Wide Web Consortium. With XPath, the event conditions can be explicitly described and

[1] http://www.w3.org/TR/xpath20

therefore easily translated to other query languages. Besides, some CEP engines, like Esper[2], already support XPath event definitions, the translation plug-in can be very simple or even non-existent.

An example of an event definition is shown in Fig. 2. Here we define an event called Hypertension, which is triggered whenever the systolic blood pressure is above 140 *mmHg* and the diastolic blood pressure is above 90 *mmHg*. For more information about the use of XPath expressions, please refer to the specification.

```
1 <event name='Hypertension'>
2   <condition>
3     /event[@name='BloodPressure']
4     and /event/pressure[@type='Systolic'] > 140
5     and /event/pressure[@type='Diastolic'] > 90
6   </condition>
7 </event>
```

**Fig. 2.** Event definition example

### 4.2 Standard Business Process Language

In order to respond to the lack of adaptation specifications in the current standards, we created the *Standard Business Process Language* (SBPL), an extension of BPEL which allows the user to include, in the business process definitions, the adaptation points, conditions and alternative processes in order to create dynamically adaptable business processes. The syntaxis of the SBPL can be seen in Fig. 3. As an adaptation language, in SBPL we need to answer four basic questions: *Where* to adapt?, *When* to adapt it?, *How* to adapt it?, and *What* to adapt?

To answer the first adaptation question, we need to identify the exact place where we want to adapt the process, for which we use the `adaptatioPoint` tag (line 11). We can use several adaptation points throughout the business process definition. To define when adaptation that will take place, we need to specify the conditions under which such adaptation is expected, and we do so by using the `situation` tag (line 13). The first element is the `event` (line 14), which refers to the name or alias of either a simple event generated by the process or a complex event, generated by the CEP engine.

The conditions to identify these events will be defined as CEP rules and deployed in the CEP engine using the translation plug-in. By separating the event definitions from the business process definitions, we can increase the flexibility of the process adaptation, since the conditions of the event definitions can be easily managed, inserting, updating and deleting events without affecting the business process. This also helps to the separation of concerns, avoiding to mix the decision making process with the business process definition.

[2] http://esper.codehaus.org

```
1  <process>
2    <partnerLinks>
3      ...
4    </partnerLinks>
5    <variables>
6      ...
7    </variables>
8    <sequence>
9      <invoke/>
10     ...
11   <adaptationPoint>
12     <invoke/>
13     <situation>
14       <event>...</event>
15       <adaptationType>...<adaptationType>
16       <adaptationProcess>
17         <partnerLinks>...</partnerLinks>
18         <variables>...</variables>
19         <sequence>...</sequence>
20       </adaptationProcess>
21     </situation>
22     <situation>
23       ...
24     </situation>
25   </adaptationPoint>
26     <invoke/>
27     ...
28   </sequence>
29 </process>
```

**Fig. 3.** The SBPL definition

After declaring the expected event, we specify how the process should be adapted, using the `adaptationType` tag (line 15). Inspired by the aspect-oriented approach, the adaptation can be done either *before*, *after* or *around* the adaptation point [10,11]. The first two types can be understood by their name, and the third is a combination of both, meaning that additional tasks will be executed before and after the adaptation point. Also, when the *around* adaptation is used, the task marked on the adaptation point can even be excluded from the process and not executed at all.

Finally, we need to define what is to be adapted in the process, and for this we use the `alternativeProcess` tag (line 16). This tag allows the user to introduce the alternative business process that will be used to adapt the current process. When using the *around* adaptation type, a tag called *proceed* should be added in the alternative process to indicate the place where the adaptation point should be inserted. Several situations can be defined for a single adaptation point, allowing the process to be adapted in a different way according to the context information.

## 5 The CEVICHE framework

In this section we present the CEVICHE framework. We start by giving an overview of the system, then we present the framework's architecture and finally we show how the process adaptation is executed.

### 5.1 Overview and Architecture

CEVICHE is a framework that intends to facilitate the integration of CEP into existing business processes and to allow these processes to be dynamically adapted to different circumstances. With this framework we want to address mainly four issues: adaptation, dynamicity, integration of CEP into business process, and non-dependency to a specific CEP engine.

To address the first issue, adaptation, we use a component-oriented approach. We specifically use Easy BPEL[3], a library that provides a BPEL 2.0 engine to orchestrate services based on a WSDL description. With this library, each task of the business process is transformed into a component that exposes its interfaces, which are bound according to the process definition, and the whole process is exposed as a service using an *Enterprise Service Bus* (ESB). These bindings can be changed at runtime, adding or removing components to the architecture, which allows the process to be adapted dynamically, giving also a solution to our second issue.

By analyzing the current events with CEP and using context information, CEVICHE can automatically decide when and how to adapt the process, integrating CEP into business processes (third issue). To achieve this, we need to provide CEVICHE with the definition of the adaptation points for the business process and the events that will trigger that adaptation. This information is defined using the SBPL, which contains also the complete business process. In Section 4.2 we discussed the SBPL in more detail.

Finally, CEVICHE aims to be able to work with any CEP engine available. For that, as part of this framework, we use a plug-in approach, which will allow us to translate our events and conditions into the desired CEP engine. This approach allows the users to define their business processes only once and deploy them using their preferred CEP engine.

The architecture of CEVICHE is composed of four main parts: 1) a user interface to create the SBPL definitions, 2) a translation framework that separates the main business process from the adaptation conditions, 3) a translation plug-in in charge of adapting the event definitions for each CEP engine, and 4) an adaptation manager to deal with the process adaptation. CEVICHE also relies on different technologies to achieve the process adaptation, like the CEP and BPEL engines, as shown in Fig. 4.

To configure the system, the user needs to be able to specify where the process has to be adapted and the conditions that will trigger such an adaptation. In order to fulfill this need, CEVICHE requires a language that can describe the

[3] http://easybpel.petalslink.com

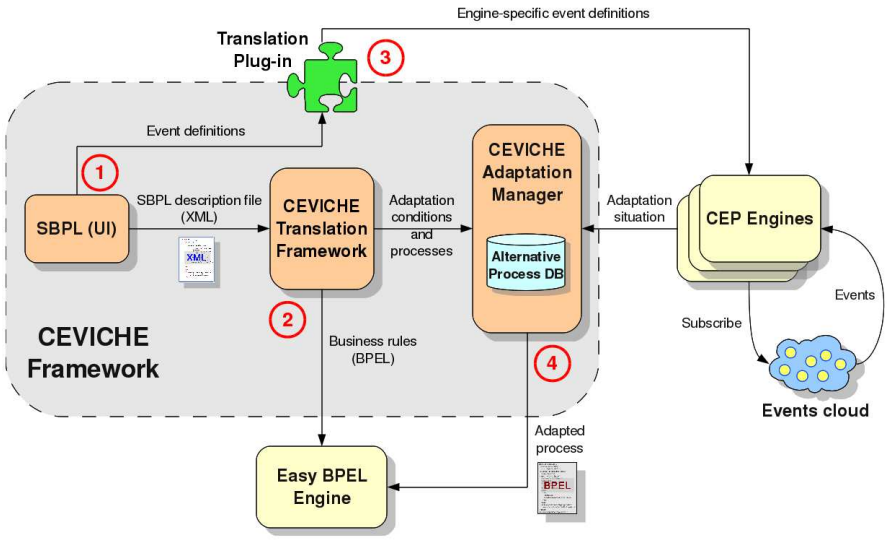


**Fig. 4.** The CEVICHE framework

business process along with all the adaptation points and conditions, and that is just what the SBPL provides.

CEVICHE needs to kinds of information: first, the event definitions that identify the special situations that require process adaptation, and second, the business process definition and adaptation points using the SBPL. Separating the decision making process from the core business process, helps to reduce the cross-cutting code and improves the maintainability of the process. The information in the process definition is sent to the translation framework, which separates the adaptation data from the core business process. The business process information is sent to the BPEL engine, while the adaptation information is sent to CEVICHE's adaptation manager.

Since there is no standard yet to define the events and CEP rules, CEVICHE uses a specialized plug-in approach to send the information in the second file in the specific CEP engine's format. This plug-in will need to be developed for each specific CEP language, unless until a standard is defined by the *Event Processing Technical Society*[4]. This way, whenever the user wants to use another CEP engine, the only thing that needs to be done is to change the plug-in, without rewriting all the specifications of the business processes. The event definitions are explained in more detail in Section 4.1.

### 5.2 Process Adaptation

Once the initial setup is ready and all the components have been properly configured, the process starts and the information begins to flow from one compo-

[4] http://www.ep-ts.com

nent to the other. First, the BPEL engine transforms the business process into components and exposes the process as a service in the ESB. When a request message is received, a new instantiation of the BPEL process is triggered. Then, the CEP engine subscribes to the different sources of events, here called the *events cloud*, which will provide the engine with the information it needs to take decisions and create complex events.

The CEP engine will gather all the events, filter the interesting ones according to the business rules and find relations that can generate complex events. When an adaptation situation is detected, the CEP engine notifies the adaptation manager, which in turn searches for the corresponding condition in the *alternative process DB* and adapts the process accordingly at runtime.

The adaptation is achieved by taking advantage of the reconfiguration capabilities of the component-oriented approach in our model. The components from the adaptation points are unbound from the process and the alternative process is bound instead. Depending on the type of adaptation (before, after or around), the adaptation point is bound again into the process in the proper place.

We can see an example of a component-based process adaptation in Fig. 5. Here, the process presented in Fig. 1 is transformed into components on the left-hand side of the figure. Then, on the right-hand side we add an alternative process using a *before* adaptation type. As can be noticed, the input bindings of the adaptation point are changed and bound to the alternative process. The output of the last component of the alternative process is then bound to the service of the adaptation point. Since the bindings in the component model can be modified at run-time, the process is dynamically adapted.

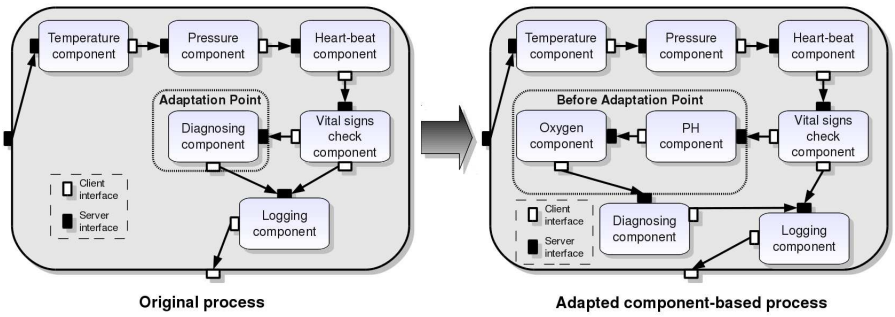


**Fig. 5.** A process adaptation example

## 6 Discussion and Validation

In this section we will show how CEVICHE can be used to deal with the challenges presented in Section 2, and discuss the advantages of our approach. In our

motivation scenario, we had mainly three challenges: 1) *Situation identification*, 2) *Code scattering*, and 3) *Process redeployment*.

*Situation identification.* In certain cases, the process needed to collect additional information about the patient, but we cannot anticipate what information was needed in each case. In CEVICHE, we use CEP rules in order to identify special situations. These situations allow us to provide the scenarios and special circumstances that we are interested in and to define how we want to respond to them. Using the information from the process and even other sources, the CEP engine matches the corresponding rules and alert the system when such a situation is found.

*Code scattering.* If we could anticipate the additional information, the request for such information would have to be included in the main process definition, generating code scattering and a complex logic in the process. To deal with code scattering, we separated the event definitions from the business process definition. By doing this, we can focus all the decision making code in a separate file which can be easily maintained without modifying or even interrupting the business process execution. The adaptation rules only need to be written once, even when the CEP engine is changed. This also helps to share the event definitions among different partners, or even a community, without limiting its use to a specific software.

*Process redeployment.* If we needed to add a new problem to diagnose, or change the way an existing problem is being diagnosed, the whole process would need to be redeployed. Given the dynamic reconfiguration of the components provided by the component-oriented model, once our business processes is transformed to bound components, these bindings can be modified at runtime, allowing the process to be adapted without redeploying it and without losing any current transactions in the process.

*Evaluation and Performance.* CEVICHE can be used to dynamically adapt business processes according to a wide number of situations. In previous work, we have shown how CEVICHE can be used to monitor and maintain the Quality of Service (QoS) of a business process by monitoring the performance and availability of the different tasks of the process, adapting the process in order to maintain a good QoS whenever a decrease of those parameters was detected [6].

As it can be expected, adding an external engine to deal with event processing, in order to discover the adaptation situations, require some additional time and resources. This overhead may vary depending on the number and the complexity of rules that need to be processed by the CEP engine. However, the overhead induced to the original process is negligible compared to the cost of the whole process, specially when dealing with Internet interactions with partners, increasing only around 1% or 2% the execution time. For the CEP engine, the overhead is insignificant, taking it less than 1 *ms* to process each event through the whole set of rules. In this case we used the Esper engine, which is an open

source stream event processing engine. The efficiency of the CEP engine to process the events is such that it exceeds over 500,000 events per second.

## 7 Related work

In this section we describe some of the works related to CEVICHE. Since our project covers several topics, we divided the related work in three sections: CEP and BPEL, business process adaptation, and BPEL context adaptation.

### 7.1 CEP and BPEL

As mentioned earlier in this paper, CEP is an emerging technology and the use of it in the business processes is a recent topic of interest and research [12,13]. An analysis of scenarios of composite event patterns comparing BPEL and BPMN is done in [12]. The authors analyze patterns of events that go from conjunction and cardinality to time relations and event consumption possibilities. The conclusion of their study is that neither BPEL nor BPMN are capable of supporting complex event scenarios in their specifications, so there is a need to integrate event pattern descriptions into the process definitions language, but they do not mention the need to adapt the process according to those complex events.

In [13], we present a service to add traceability to the RFID tagged products by using Complex Event Processing. When the RFID events are captured, they are transformed into business events that correspond to the business rules defined in the process which allows the users to have a better understanding of the status of their products, *i.e.*, the product's location and the environment it has been exposed to.

### 7.2 Business process adaptation

The need to adapt a process has been a topic of interest in the recent years and there have been different approaches that offer solutions to it [14,15,16,17]. In [14], the authors propose to deal with process adaptation by adding a web service repository that handles the web services to invoke in each case. Whenever an invocation of a web service is done, the call is intercepted and the repository is checked for changes in the process definition, before the invocation of a web service. If there have been some changes, then it examines the available web services in the repository and chooses the one that best suits the criteria, otherwise the invocation is executed as usual.

The authors in [15] use an aspect-oriented approach introducing `executable models`, which are used to represent the cross-cutting concerns. They use `open objects`, which are representations of the state of the elements in the model, to monitor the invocation of services and adapt the process by weaving the interaction with other models before (activation) and after (deactivation) the call to the service.

Another aspect oriented implementation, using the Spring .NET framework, is presented in [16]. They use a contract-based approach to assign a web service to each instance of an execution call. To achieve adaptation of the process they can change the contract at runtime to assign a new web service for the call. They can also adapt an existing implementation of a web service by using aspects to weave the new behavior.

An adaptation of the BPEL language called VxBPEL is presented in [17]. The authors insist on the need of flexibility and variability in the service-based systems and the lack of them when deploying BPEL processes. They extend the BPEL language to add new elements like `Variation Points`, which are the places where the process can be adapted and `Variants`, which define the alternative steps of the process that can be used. VxBPEL also accepts new `Variants` to be added at runtime, allowing the systems to be adapted without redeploying the process.

### 7.3 BPEL context adaptation

The work that is closer to our proposal is the one presented in [18]. The authors present a plug-in based architecture for self-adaptive processes that uses AO4BPEL. Their proposal is to have different plug-ins with a well-defined objective. Each plug-in has two types of aspects: the `monitoring aspects` that check the system to observe when an adaptation is needed and the `adaptation aspects` that handle the situations detected by the monitoring aspects. Whenever the conditions of a `monitoring aspect` are met, it uses AO4BPEL to weave the `adaptation aspects` into the process at runtime. In our approach we deal with the monitoring part using the rules deployed in the CEP engine, which detect special situations (by relating simple events) and select the aspects to be used to adapt the process.

An advantage of their work is that the monitoring aspects can be hot-deployed to their BPEL engine while with our approach the changes in the rules might not be considered at runtime, depending on the CEP engine. On the other hand, this difference also shows an advantage for our proposal, since we are not tied to a single CEP engine and we can use any CEP rules already defined for the monitoring process, while in their case we would need to create a new plug-in for each new situation we want to monitor. Also, even if we needed to restart the CEP engine in order to consider the new rules, this would not affect any active running processes, as it would if we restarted the BPEL engine.

## 8 Conclusions and future work

Process adaptation and Complex Event Processing are two topics that are creating a lot of interest in the research community, however there is still no integration of both domains. In this paper we presented CEVICHE, a framework that intends to facilitate the integration of CEP into existing business processes and to allow these processes to be dynamically adapted to different circumstances.

With CEVICHE we addressed four issues: adaptation, dynamicity, integration to business process, and non-dependency to a specific CEP engine. As part of the CEVICHE framework, we proposed the SBPL, an extension of BPEL that allows the user to include the adaptation points and conditions in order to create dynamically adaptable business processes. We also presented how users can define their own events to trigger the process adaptations. These event definitions are sent to a special plug-ins to deal with the different languages of the CEP engines, allowing the users to write their process specifications only once and deploy it in the engine they want.

With a simple scenario we showed how CEVICHE can be used to monitor the context information of a business process and adapt it dynamically to respond to it, without the need to redeploy the process and without loosing any current transactions. Using CEVICHE, we managed to respond to the three challenges on the scenario: 1) *Situation identification*, 2) *Code scattering*, and 3) *Process redeployment*.

Thanks to the modular architecture of CEVICHE, it is not strongly linked to any third party technology. In previous work, we have shown how CEVICHE can also work in an aspect-oriented approach to monitor and maintain the quality of service of a business process, showing that be used with different technologies and approaches to deal with dynamic process adaptation. This architecture also allows our work to be completely componentized in the future, facilitating the integration with other technologies and the interaction with the different parts of the architecture.

We plan to work on the definition of a RESTful architecture to leverage on the deployment of CEVICHE components and facilitate the evolution of the architecture by adding or changing components. Finally, we are creating a couple of plug-ins for some open-source CEP engines, to demonstrate the feasibility of our approach for standardizing event definitions.

## 9 Acknowledgements

This work was supported by the French Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the *Contrat de Projets Etat Region* (CPER) 2007-2013.

## References

1. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc. (2001)
2. Huber, N., Michael, K.: Minimizing Product Shrinkage across the Supply Chain using Radio Frequency Identification: a Case Study on a Major Australian Retailer. In: ICMB '07: Proceedings of the International Conference on the Management of Mobile Business, IEEE Computer Society (2007) 45

3. Mangkorntong, P.: A Domain-Driven Approach for Detecting Event Patterns in E-Markets: A Case Study in Financial Market Surveillance. VDM Verlag, Saarbrücken, Germany, Germany (2009)
4. Zang, C., Fan, Y., Liu, R.: Architecture, implementation and application of complex event processing in enterprise information systems based on RFID. Information Systems Frontiers **10**(5) (2008) 543–553
5. OASIS: OASIS Standard. Web Services Business Process Execution Language Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html` (April 2007)
6. Hermosillo, G., Seinturier, L., Duchien, L.: Using complex event processing for dynamic business process adaptation. In: SCC '10: Proceedings of the 2010 IEEE International Conference on Services Computing, IEEE Computer Society (2010)
7. Ku, T., Zhu, Y., Hu, K.: A Novel Complex Event Mining Network for Monitoring RFID-Enable Application. In: PACIIA '08: Proceedings of the 2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, IEEE Computer Society (2008) 925–929
8. Taylor, R.N., Medvidovic, N., Dashofy, E.M.: Software Architecture: Foundations, Theory and Practice. Addison-Wesley (2007)
9. Wang, A.J.A., Qian, K.: Component-Oriented Programming. Wiley-Interscience (2005)
10. Kiczales, G., Lamping, J., Mendheka, A., Maeda, C., Lopes, C.V., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In Gjessing, S., Nygaard, K., eds.: Proceedings of the European Conference on Object-Oriented Programming (ECOOP). Number 1241 in Lecture Notes in Computer Science, Springer (June 1997)
11. Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., Griswold, W.G.: An overview of aspectj. In: ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming, Springer-Verlag (2001) 327–353
12. Barros, A.P., Decker, G., Großkopf, A.: Complex events in business processes. In: BIS. (2007) 29–40
13. Hermosillo, G., Ellart, J., Seinturier, L., Duchien, L.: A Traceability Service to Facilitate RFID Adoption in the Retail Supply Chain. In: Proceedings of the 3rd International Workshop on RFID Technology - Concepts, Applications, Challenges IWRT 2009, INSTICC Press, Portugal (05 2009) 49–58
14. Lins, F.A.A., dos Santos Júnior, J.C., Rosa, N.S.: Adaptive web service composition. SIGSOFT Softw. Eng. Notes **32**(4) (2007) 6
15. Sánchez, M., Villalobos, J.: A flexible architecture to build workflows using aspect-oriented concepts. In: AOM '08: Proceedings of the 2008 AOSD workshop on Aspect-oriented modeling, ACM (2008) 25–30
16. Rahman, S.S.u., Aoumeur, N., Saake, G.: An adaptive eca-centric architecture for agile service-based business processes with compliant aspectual .net environment. In: iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services, ACM (2008) 240–247
17. Koning, M., Sun, C.a., Sinnema, M., Avgeriou, P.: Vxbpel: Supporting variability for web services in bpel. Inf. Softw. Technol. **51**(2) (2009) 258–269
18. Charfi, A., Dinkelaker, T., Mezini, M.: A plug-in architecture for self-adaptive web service compositions. In: ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services, IEEE Computer Society (2009) 35–42