Dynamic Reconfiguration Using Template Based Web Service Composition

Kristof Geebelen, Sam Michiels and Wouter Joosen
IBBT-DistriNet, Dept. Computer Science, K.U.Leuven
Celestijnenlaan 200A
3001 Leuven, Belgium
{kristof.geebelen, sam.michiels, wouter.joosen}@cs.kuleuven.be

ABSTRACT

Current workflow languages introduce limitations regarding modularity and flexibility. They are lacking support for reusability of primitive and structured activities. Designing processes often leads to duplication which makes the workflow descriptor complicated and unnecessarily large. Furthermore, due to the static character of the scripts, there is insufficient flexibility to model dynamic, evolvable and fail-safe workflows. In this paper we present a framework that allows the design of WS-BPEL processes in a modular way based on reusable templates. In addition, we introduce an extra layer on top of WS-BPEL that allows template processing based on parameter values. This layer offers support for decision logic to adapt processes dynamically. The approach is based on the "Ruby On Rails" (RoR) framework, known for adding dynamism to static web pages. The proposed solution is portable with existing WS-BPEL engines.

Categories and Subject Descriptors

H4.1 [**Information Systems Applications**]: Workflow Management

General Terms

Design, Management

Keywords

Modularization, Dynamic Reconfiguration, Service Composition, WS-BPEL

1. INTRODUCTION

Web services have become increasingly popular as a means to integrate existing applications into new environments. With a Service Oriented Architecture [11] as foundation for creating individual solutions, an integration language can be used to orchestrate existing services into more complex business solutions. At this moment, the Web Services Business Process Execution Language (WS-BPEL) [12] has profiled itself as the de-facto industry standard for orchestrating web services. However, this standard exhibits major limitations regarding modularity and flexibility.

When developing an application in a general purpose language like Java, code modules can be reused by defining classes and methods. Functionality declared in a class or method body can be reused by creating new objects or by invoking methods multiple times. WS-BPEL is used to model the behavior of processes with xml-based syntax. However, for designing and implementing such a process there is little support for concepts that foster modularization and reuse of code. Besides modularization of the functional part, it is useful to provide a means to separate concerns from the workflow specification. Examples of such concerns are logging, billing and security. Typically, we want to avoid that those concerns end up scattered across the process specification and tangled with one another.

A second limitation concerns the flexibility offered by workflow languages. The integration of heterogeneous web services, often developed by a third-party, into a new business process over the Internet may lead to unpredictable behavior. The web services used in workflows are not always controllable by the developer of the business process. To provide reliable solutions for services that are often subject to change there is need for fail-safe mechanisms that allow anticipation on unpredictable factors. Examples of irregular behavior include web services that go off-line and performance bottlenecks. WS-BPEL offers fault handlers and compensation handlers that allow anticipation to a certain degree. We argue, however, this is not sufficient to provide reliable, high quality solutions in the dynamic context of web services.

The contribution of this paper complements earlier research (Padus [8]), in which we propose an aspect-oriented approach to comply with shortcomings of WS-BPEL; We leverage on this expertise and investigate how RoR can be mapped to the domain of web services since it handles similar issues in dynamic web design. This paper proposes a template framework for designing WS-BPEL processes that allows the interpretation of templates based on parameters. The approach is easy to use for the designer and tackles the shortcomings mentioned above without reducing performance at runtime. The idea is to start from a generic master process in which designers can model functionality independent of concrete implementations. A template library is provided which contains combinations of WS-BPEL activities that fulfill common functionalities. The templates together with the generic master process are the ingredients for a controller that generates a tailored WS-BPEL process. The controller composes the WS-BPEL process by interpreting various parameters associated with the runtime environment and by selecting appropriate templates from the template library. The template framework introduces a mechanism on top of the execution layer that tackles the issues concerning flexibility and modularization before deployment. This makes the resulting WS-BPEL process compatible with existing WS-BPEL engines.

2. WEB SERVICE COMPOSITION WITH WS-BPEL

2.1 Introduction

Business Process Execution Language for Web Services (WS-BPEL) [12] provides a language for the specification of business processes and business interaction protocols. WS-BPEL is built on top of XML specifications. For example, it uses the web service description language (WSDL) [13] to describe the web service interfaces that participate in a process. An executable WS-BPEL process is defined by a control flow that consists of a combination of basic and structured activities.

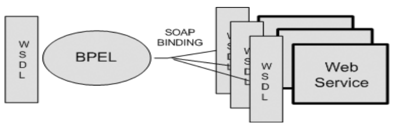


Figure 1: Overview

The Business Process Modeling Notation (BPMN) [14] is a standardized graphical notation for drawing business processes using a flow chart. The BPMN specification includes a mapping from BPMN diagrams to executable WS-BPEL processes. Processes built with a design tool can automatically be generated from the workflow diagrams.

To make a business process operational, it has to be deployed on a WS-BPEL orchestration engine (e.g. IBM [17], Oracle [16], ActiveBPEL [15]). Since WS-BPEL is standardized, processes are portable between different engines to a certain extent.

2.2 Limitations

In different research papers, current limitations of workflow languages have been discussed, including modularization and flexibility.

In an evaluation of WS-BPEL to Scientific Workflows [1] the need for adaptive and flexible workflows is identified. However, there is little support in WS-BPEL for dynamically modifying a workflow at runtime. Furthermore, the reusability of the basic and structured activities in WS-BPEL is limited. It is not possible, for instance, to re-execute an activity that is defined earlier by referring to it later [1].

IBM and SAP [8] identify a similar problem related to modularity: "The BPEL language currently does not support the explicit definition of business process "fragments" that can be invoked from within the same business process or from another business process".

Other papers [2, 5, 6, 7] focus on the need for separation of concerns in the workflow specification. WS-BPEL processes suffer from a problem that is known as the "tyranny of the dominant decomposition". A WS-BPEL process can only be decomposed according to the control flow of the process, and concerns that do not align with this decomposition end up scattered across the process specification and tangled with one another [2]. These cross-cutting concerns give rise to large and complex process definitions which are difficult to understand and maintain.

3. TEMPLATE BASED WEB SERVICE COMPOSITION

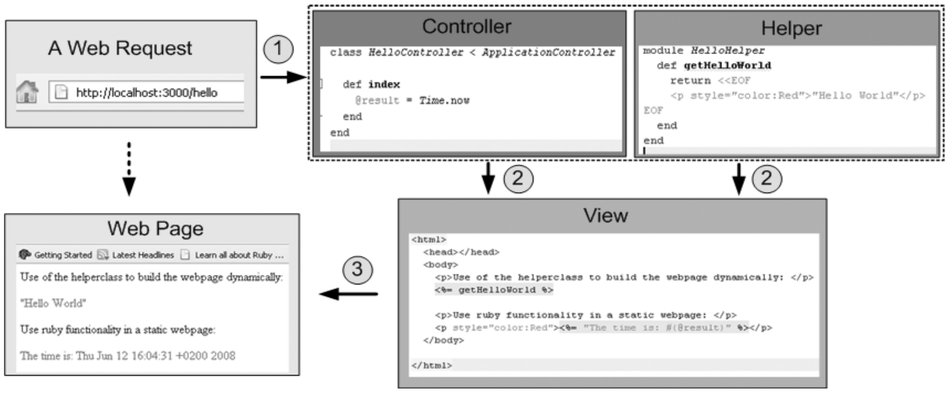
In this section we address the limitations of WS-BPEL introduced previously. First we motivate our approach by mapping the problems to the domain of dynamic web design. Then we present the basic building blocks used in the framework: the template definitions, the library, the master process and the controller. Support for modular design and dynamic configuration is discussed in detail by means of a representative example.

3.1 Approach

Our approach to create WS-BPEL processes in a modular and dynamic way is based on similar evolutions in the domain of web design. The intent of web design is to create a website that presents content to the end users in the form of web pages. To comply with today's expectations of end-users, there is growing tendency to use dynamic web pages. In contrast to static pages, where the content and layout is not changed with every request, dynamic pages adapt their content on the fly depending on the user's input.

To support dynamic web design there are a lot of products and tools on the market. One that caught our attention is the "Ruby On Rails" framework [10]. RoR is a popular open source web application framework written in the Ruby programming language. RoR is based on an architecture known as MVC (Model - View - Controller). The architecture consists of a set of design patterns that allow a clear separation between data models, user interfaces, and control logic of the application. The model is responsible for interacting with the application data. The view is the presentation layer. It defines how the application presents data. Controllers orchestrate the application; they receive events, interact with the model, and display an appropriate view to the user. Figure 2 illustrates the basic functionality provided by the framework to handle a web request: (1) Web requests are routed to the appropriate controller; The controller coordinates the interaction between the user, the views, and the model. To use application data, it can contact the model (not presented in fig. 2). The controller manages helper modules which extend the capabilities of the view templates without bulking up their code. (2) The controller instructs the view to generate an appropriate presentation. (3) The view uses the results from the controller to render the next browser screen [10].

The RoR framework allows dynamically adapting the content of a static web page expressed in html. Based on the user request, the framework calculates the values of variables that are then used to generate the web page. Furthermore, the helper class contains template definitions of html code that can be substituted in the html page. The ruby programming language is used to implement the logic that decides on which template is returned.



```
class HelloController < ApplicationController

  def index
    @result = Time.now
  end
end
```

```
module HelloHelper
  def getHelloWorld
    return <<EOF
    <p style="color:Red">"Hello World"</p>
EOF
  end
end
```

```
<html>
  <head></head>
  <body>
    <p>Use of the helperclass to build the webpage dynamically: </p>
    <%= getHelloWorld %>

    <p>Use ruby functionality in a static webpage: </p>
    <p style="color:Red"><%= "The time is: #(@result)" %></p>
  </body>

</html>
```

Use of the helperclass to build the webpage dynamically:

"Hello World"

Use ruby functionality in a static webpage:

The time is: Thu Jun 12 16:04:31 +0200 2008

Figure 2: A web request with Ruby On Rails

We extend the concept of dynamic web design to web service composition in order to comply with frequently changing environments and unpredictable behavior. The proposed template framework is based on the MVC concept used in the RoR framework. Figure 3 shows the general architecture. The idea is to start from a generic master process. This process represents a generic workflow where requirements can be modeled without being bound to specific implementations. The concrete modules of WS-BPEL activities are modeled as templates and stored in a library. (1) The controller can be triggered to regenerate the process. (2) It interprets provided parameter values (2a) and looks up the corresponding templates from the library (2b) which are integrated in the master process. (3) The result is an executable WS-BPEL process generated by the controller.

The introduction of a layer on top of the WS-BPEL execution allows dynamic composition of templates into the master process by interpreting changing parameters. This new recomposition, triggered at the template controller, happens in the modeling space and replaces the old workflow composition at the execution layer by redeployment. The mapping from the template framework illustrated in Figure 3 to the RoR example shown in Figure 2 is straightforward: Through the RoR Helper class (Template Controller), concrete templates can be selected which extend the View (Master Process) to render a specific representation (Executable BPEL Process).

In this paper, we restrict our focus to the reuse of template definitions. This maps only to a basic feature of in the RoR framework. However we motivate our approach from this viewpoint since we believe that many other features offered by RoR have the potential to be successfully mapped to the domain of web services. These include features were the execution feeds back into the model and they will be investigated in future work.

3.2 Building Blocks

We show how the template framework can be used to design a simple brokerage service. The service integrates some web services into a workflow that allows end-users to retrieve stock quotes and place market orders. For this process we want to apply the following informal policy rules:

- Users must be authenticated before using the services.
- Since there are multiple broker services available, the one with the lowest response time will be used. Reevaluation of the response time will be done every hour.
- After the invocation of a broker services, a billing scheme must be applied that bills a fixed amount to the customer.

3.2.1 *Template Definitions*

The syntax used for defining templates is based on an existing syntax for describing WS-BPEL modules [3].

A template definition represents a coherent set of basic and structured WS-BPEL activities that realizes a particular functionality. For example, Listing 1 shows a template definition that allows the invocation of a billing service.

```
<template name="GenericBilling" result="">
      <module name="billingService">
          <invoke partnerLink="billingPL" portType="bill:
                 billingPT" operation="bill"
              inputVariable="billingMsg">
      </module>
      ...
</template>
```

Listing 1: Template defining generic billing

A template consists of one or more modules. A module is a cluster of WS-BPEL activities. Different modules are used when the contained activities are scattered throughout the process to realize the functionality provided by the template. Listing 2 illustrates a template that consists of two modules. The first one is responsible for registering the start time. Next, some action will proceed for which its duration will be billed. After that action the resulting price is sent to a billing service.

```
<template name="DurationBilling(String Price)" result="">
   <include name="GenericBilling"/>
   <module name="start">
     <assign>
       <copy>
         <from expression="func:getCurrentTime()" />
         <to variable="startTime" />
       </copy>
     </assign>
   </module>
   <module name="end">
     <sequence>
       <assign>
        <copy>
         <from expression="func:calculatePrice( bpws:
               getVariableProperty('startTime'), $Price)"/>
         <to variable="billingMsg" part="price" />
        </copy>
       </assign>
       <template name="billService" />
      </sequence>
    </module>
   ...
</template>
```

Listing 2: Template defining Duration Billing

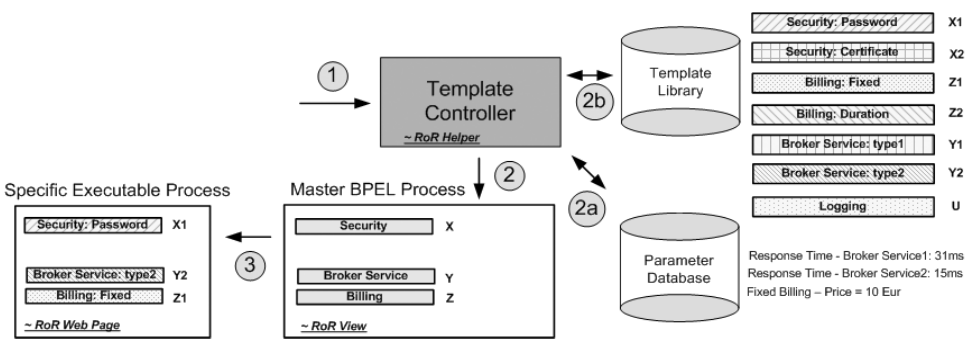


Figure 3: The template framework

Another feature illustrated in Listing 2 is the reuse of templates from within a template definition. Include statements are used to include the contents of another template file. The semantics of the statement is the same as if the contents of the included template were copied to this template.

The template definition can be compared with the signature of a method in a general programming language. Arguments can be passed in the form of strings and WS-BPEL variables. The values of the parameters will be replaced at the corresponding references within the template.

3.2.2 *Library*

In this section we zoom in on the template library available for the design of the workflow. The library contains templates for billing, security and invoking a broker service. We illustrate examples in Listing 3 and 4. Analog there are alternative templates for invoking a second broker service (Service B) and for certificate validation (Mechanism B) respectively.

Template library:

- Place Market Orders

```
<template name=”Broker1(Var order)” result=”Var status”>
  <module>
    <invoke partnerLink=”stockBroker1” portType=”sb:
          stockBroker1PT”   operation=”placeMarketOrder”
          inputVariable=”order” outputVariable=”status”>
  </module>
  ...
</template>
```

Listing 3: Service A - Stock Broker 1

- Security

```
<template name=”passwordSecurity(XpathString
      IncommingPassword)”>
  <module>
    <sequence>
      <assign>
        <copy>
          <from>IncommingPassword</from>
          <to variable=”password” part=”msg”/>
        </copy>
      </assign>
      <invoke partnerLink=”passwordSecPL” portType=”ps:
            passwordSecPT” operation=”checkPassword”
            inputVariable=”password” outputVariable=”Result
            ”/>
      <if name=”ValidateAuthenticationToken”>
        <condition>Result = false</condition>
        <throw faultName=”ps:AuthenticationFailure”
              faultVariable=”Fault” />
      </if>
    </sequence>
  </module>
  ...
</template>
```

Listing 4: Mechanism A - Password Security

3.2.3 *Master Process*

With the template library in mind we can design our master process. The master process is designed as a regular process. But instead of including all the specific details of all possible situations on which anticipation is required, it is designed on a meta-level, containing more generic template statements. Listing 5 shows a fragment of the master WS-BPEL process. Here we state that the invocation of a broker service, a security mechanism and a billing scheme are needed. Details are passed as an argument.

```
<process>
  ...
    <%= insertSecurity($BrokerServiceOperationIn.
          securityToken) %>
    ...
    <sequence name=”Sequence1”>
      <%= placeMarketOrder($BrokerServiceOperationIn.
            orderRequest) %>
      <%= insertFixedBilling() %>
        ...
    </sequence>
  ...
</process>
```

Listing 5: Fragment of master WS-BPEL process

A business process is usually designed in a workflow chart. Figure 4 illustrates a graphical representation of the workflow. The master process is modeled like a regular process except for the generic parts we use controller statements instead of concrete WS-BPEL activities. The circles in Figure 4 represent high level calls in the workflow.

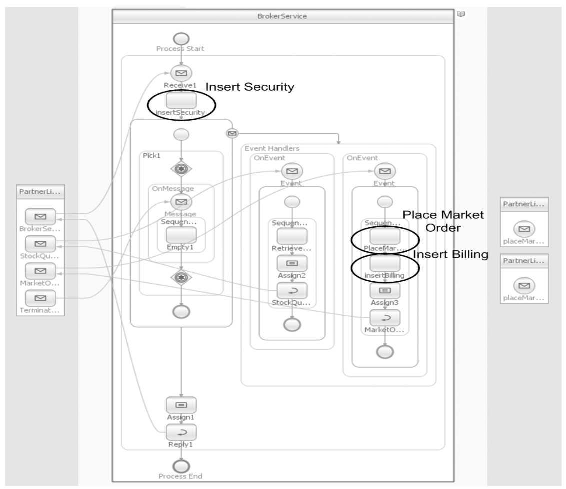


Figure 4: Master BPEL Process

3.2.4 *Controller*

The final step is to implement the logic that decides which templates to use. The decision logic on how specific processes must be generated is implemented in the template controller. The concrete implementation is shown in Listing 6. Parameters we use to decide on which template to choose are for example the response times of two broker services. Those times are calculated in the background and can be retrieved from a database. The template corresponding with the service that has the lowest response time is returned and substituted in the master process. Depending on how frequently the controller is triggered to check for a change in parameter values and generate and deploy a new version of the process, decides how quick we can anticipate on a changing environment.

```
module BrokerController
  def insertSecurity(incommingPassword)
     return templates.passwordSecurity(incommingPassword)
  end
  def placeMarketOrder(order)
    int responseTimeA = Database.getResponseTimeA()
    int responseTimeB = Database.getResponseTimeB()
    if (responseTimeA < responseTimeB)
      return templates.Broker1(order)
    else
      return templates.Broker2(order)
    end
  end
  def insertBilling
    String price = Database.getPrice()
    return templates.FixedFeeBilling(price)
  end
end
```

Listing 6: Template Controller Logic

3.2.5 *Discussion*

We illustrate the impact on an evolving application by describing a scenario that implies the need to change. Imagine that the password security mechanism currently used is no longer safe and needs to be upgraded to a system that exchanges certificates. This requirement can easily be achieved by changing the template controller logic to return the corresponding template found in the library. Since ruby is an interpreted language, changes in the implementation are applied immediately.

The lack of flexibility is handled by introducing an extra layer that allows template processing. By using templates, the framework supports not only the separation of concerns, but also modularization in a more general context. Grouping coherent pieces of WS-BPEL activities into reusable modules minimizes redundancy while developing business processes. The use of standardized templates eases the exchange of existing modules. The complete integration of the templates in the master process provides a high coupling between the modules and the parent process.

4. PROTOTYPE

For the implementation of the framework we have reused the concepts from the RoR framework. Simply by considering the master process as a static web page and the template controller as a helper class we can already generate executable processes by simulating a web request. Of course a more usable and user friendly solution requires still some efforts. Extensions to the RoR framework are needed to make the design of master processes straight forward. Integration of WS-BPEL support like easy deployment and graphical design tools are the next steps to come to a useful solution.

5. RELATED WORK

BPEL-SPE [8] introduces an extra layer on top of the WS-BPEL language that supports invocation of a business process as a sub-process of another business process. This approach focuses on modularization and reuse. One of their challenges is to allow a tight coupling between the parent and sub-process. The paper examines various flavors of sub-processes and different modes of invocation, and outlines the syntax and semantics of needed extension. The language specification, which defines the precise syntax of this extension, is not implemented yet.

Akram [1] investigates the requirements of Scientific Workflows in context of WS-BPEL and evaluates to what extend WS-BPEL is able to fulfill these requirements. Due to the complexity, unpredictability and inter-dependency of the components in a scientific workflow there is a demand for high flexibility of the underlying workflow that will be used. The identified shortcomings of WS-BPEL for Scientific Workflows were also relevant to this paper.

Casati [4] presents the eFlow platform that allows dynamic and adaptive composition of e-services. The paper discusses some interesting constructs to allow adaptive workflows: dynamic conversation selection, multi service nodes, generic nodes and dynamic process modifications. In contrast to our approach the binding of service templates is done at a late stage and requires a customized workflow engine.

Although no pointcut language is used in our approach, it is strongly related to AOP. The controller statements can be seen as annotations that allow the weaving of reusable modules into the base program (master process).

Charfi [5] and Courbis [7] both present an aspect-oriented extension to WS-BPEL allowing runtime aspect addition and removal. The approach we used to generate an executable process is done statically. This means that the templates and base-code are merged before runtime. The statically generated process needs still to be deployed on a BPEL engine before it will be executed. However, we believe that this approach has some advantages compared to runtime adaptation of the workflow. When a new WS-BPEL process is created it can usually be packaged and deployed on a running engine on the fly. This means that redeploying a new process generates only a one-time minimal extra overhead. From then on instances from the new process can be executed at full performance. Runtime adaptation would imply that each time a WS-BPEL activity is called, we need to recheck the templates during the execution of the process. Real time processes, where performance is extremely important, do not tolerate that extra overhead. So in many cases the cost of periodical redeployment will be much lower than the cost of continuous probing for changes. Only if the adaptation rate of the workflow is very high continuous probing will probably perform better. Furthermore, to allow runtime checking a custom-made BPEL engine is required. This would bind the framework to a specific implementation and limit the portability of the resulting workflows. Another possible problem with runtime adaptation that comes to our mind is a need for consistency checking. When the workflow of a process instance gets adapted while it is running it can lead to unpredictable behavior and enter an inconsistent state.

Isolating process-level concerns using Padus [2] is also based on an aspect-oriented language extension. The static weaving approach used here introduces no runtime overhead and the resulting process is compatible with existing WS-BPEL engines. The concepts from Padus can be used complementary to the ideas presented in this paper. Instead of inserting controller statements manually in the master process, we can use the rich joinpoint model, the pointcut language and the deployment descriptor of Padus to weave them into the workflow in an aspect-oriented way. In a next step, the template framework will translate them to concrete WS-BPEL activities.

6. CONCLUSIONS AND FUTURE WORK

This paper focused on the modularization and flexibility of WS-BPEL processes. We presented a framework that uses a controller to select template definitions from a library and integrate them into a master process. Our approach is based on the helper module concept from the RoR framework. The resulting process is a standard WS-BPEL process, deployable on any existing engine and executable without generating extra performance overhead at runtime.

We highlighted only one of the aspects that RoR offers to design dynamic web pages and mapped it to the WS-BPEL context. However there are other features offered by the framework that can be useful to adopt [10]:

- The Rails controller classes are the logical center of the web application. The code written allows adding application level functionality by extending the basic functionality offered by the static web pages. Similarly we could extend the basic functionality offered by WS-BPEL with ruby method invocations during the execution of the workflow. Currently most WS-BPEL engines already enable to call custom functions at runtime. However these functions are often integrated in the engine implementation and difficult to adapt. The challenge is to provide user-friendly support, integrated with the template framework which allows a WS-BPEL designer to easily extend WS-BPEL activities with custom-made ruby method calls.
- Active Record is an Object-Relational Mapping (ORM) layer supplied with Rails. This layer provides support to easily use stored data into the web application. Database support in WS-BPEL is usually limited to persistence of process instances. Extending the execution of a workflow with database access via customized functions improves its flexibility.

Another possible extension to the template framework allows real time modification of workflows based on incoming messages. This approach demands to queue and process incoming messages. If this message requires adaptation of the workflow, a new specific version is generated and deployed by the template framework. Then the queued message can be rerouted and executed by the adapted workflow. It is obvious that the extra flexibility offered by this approach is at the cost of performance. We need to study this more in detail to see if certain cases could benefit from this approach.

7. ACKNOWLEDGMENTS

We would like to acknowledge the input of Kris Verlaenen to the ideas presented in this paper. Also thanks goes to Dimitri Van Landuyt and Tom Stijnen for some interesting discussions on the topic and to the reviewers for their constructive comments.

8. REFERENCES

[1] Akram, A., Meredith, D. and Allan R.: Evaluation of BPEL to Scientific Workflows. Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06).

[2] Braem, M., Verlaenen, K., et al.: Isolating process-level concerns using Padus. Proc. of the 4th Int'l Conf. on Business Process Management (BPM 2006), Vienna, Austria, Springer (2006).

[3] Braem, M., Joncheere, N., Geebelen, K. and Verlaenen, K., Guiding aspect-oriented service composition in WS-BPEL and Padus (Demonstration), Proceedings of the 6th international conference on aspect-oriented software development, March 2007.

[4] Casati, F., Shan, M. C.: Dynamic and adaptive composition of e-services. Information system. 6(3): 143-162,2001.

[5] Charfi, A., Mezini, M.: Aspect-oriented web service composition with AO4BPEL. In Zhang, L.J., ed.: Proceedings of the 2nd European Conference on Web Services (ECOWS 2004), Erfurt, Germany, Springer-Verlag (2004) 168-182.

[6] Charfi, A., Mezini, M.: Aspect-oriented workflow languages. In the Proceedings of the 14th International Conference on Cooperative Information Systems (COOPIS'06).

[7] Courbis, C., Finkelstein, A.: Towards aspect weaving applications. In: ICSE '05: Proceedings of the 27th international conference on Software engineering, New York, ACM Press (2005) 69-77.

[8] Kloppmann, M., Rickayzen, A., et al.: WS-BPEL Extension for Sub-processes - BPEL-SPE. A Joint White Paper by IBM and SAP (2005).

[9] Suvée, D., Vanderperren, W.: JAsCo: An aspect-oriented approach tailored for component based software development. In Akÿsit, M., ed.: Proc. 2nd Int' Conf. on Aspect-Oriented Software Development (AOSD-2003), ACM Press (2003) 21-29

[10] Thomas, D.,Hansson, D.,Breedt, L. and Clark, M.: Agile Web Development with Rails, 2nd Edition

[11] Reference Architecture for Service Oriented Architecture Version 1.0, April 2008, OASIS Technical Committee, `http://www.opengroup.org/projects/soa/`.

[12] Web Services Business Process Execution Language Version 2.0, April 2007, OASIS Technical Committee, `http://docs.oasis-open.org/wsbpel/`.

[13] Web Services Description Language (WSDL) Version 2.0, June 2007, W3C Note, `http://www.w3.org/TR/2007/REC-wsdl20-20070626/`.

[14] Business Process Modeling Notation (BPMN) Version 1.1, January 2008, OMG Specification, `http://bpmn.org/`.

[15] Active Endpoints, `http://www.activevos.com/community-open-source.php` .

[16] Oracle BPEL Process Manager, `http://www.oracle.com/technology/products/ias/bpel/index.html` .

[17] IBM Business Process Execution Language for Web Services JavaTM Run Time (BPWS4J), `http://www.alphaworks.ibm.com/tech/bpws4j`.