Integrating Declarative Processes and SOA: A Declarative Web Service Orchestrator

Natália Silva[1,2], **Renata Carvalho**[1], **Ricardo Lima**[1], **and Cesar Oliveira**[1]
[1]Center of Informatics, Federal University of Pernambuco, Recife, Pernambuco, Brazil
[2]C.E.S.A.R - Recife Center for Advanced Studies and Systems, Recife, Pernambuco, Brazil
Email: {ncs, rwm, rmfl, calo}@cin.ufpe.br

Abstract— *Service Oriented Architecture (SOA) is a computer model that aims at building new software by assembling independent and loosely coupled services. Traditional web service orchestration is a mechanism for combining and coordinating different web services based on a predefined pattern. However the orchestration requirements may evolve due to business needs. In business context, the declarative approach has emerged to provide flexibility by modeling* what *must be done but not* how *it must be executed through business rules. When working with such a model, the results produced depend on the users' preferences. It is therefore fundamental that orchestration mechanisms provide simple yet efficient ways to dynamically make service composition. This paper proposes a web service orchestrator for declarative processes that makes service composition at runtime. The resulting business process obey all the business rules. The composition is done as the user chooses the service to run, providing an application-aligned infrastructure that can be scaled based on the needs of each business process, since it is described using declarative strategy.*

Keywords: Service Oriented Architecture, runtime orchestration, flexible process, dynamic service composition.

1. Introduction

Service Oriented Architecture (SOA) is a software architecture that focuses on delivering functionalities through services that can be reused across an enterprise. However these services are independent, they are defined in sequences to fulfill business processes. Services operate without any context from other processes within the organization.

Not only can SOA deliver on its promises of reusability and ability (usefulness), but it can also reduce the overall cost of ownership through the standards-based approach (ease of use) [1]. Moreover, SOA provides a complete integration between data and application.

Web services are an established technology for implementing SOA. They can be composed to create a higher level services or applications. Through service composition a complex service can be created by aggregating component services available. Conventionally, the web service composition specifies what services need to be invoked, in what order, and how to handle exceptional conditions [2]. Standard interfaces (such as WSDL), protocols (such as SOAP) for describing and invoking web services, and the loose-coupling of these services are important characteristics that lead to more interoperable distributed systems [3].

The SOA orchestration mechanism uses a central process to control the execution of web services. It receives the client requests and invokes the component services. This mechanism is referred as a centralized orchestration [4].

The orchestrator behavior may evolve as the business requirements change. Hence, several approaches emerged to provide more flexibility to business process execution [5]. The declarative approach provides the desired flexibility by modeling *what* must be done but not *how* it must be executed [6]. When working with such a model, users are driven by the system to produce required results. However, the manner in which results are produced depends on the usersÕ decisions along the process execution. Since the orchestrator behavior changes whenever the user invokes a service execution, it is important to provide a simple and efficient way to modify the services composition dynamically.

This paper proposes an orchestration mechanism that makes the service composition at runtime. Instead of binding pre-modeled compositions, the proposed flexible orchestrator binds the output data of a service to the input data of another service at runtime. The composition is done as the user chooses the service to run, following a declarative approach for the business process. For the automated arrangement, coordination, management, and binding of services, the user must provide some data configuration through an intuitive XML file. Our orchestrator provides an application-aligned infrastructure that can generate various web service composition at runtime based on the needs of each business process, since it is described using the declarative strategy.

The remainder of the paper is organized as follows. We first introduce the concept of flexible process in Section 2. Some related works and their main contributions are compared to this work in Section 3. Section 4 presents our web-service orchestrator. The section shows the benefits provided by our orchestrator and details its architecture. Section 5 conducts a case study. Finally, Section 6 discuss the main conclusions we draw in this paper.

2. Flexible Processes

When the company tasks are less repetitive and predictable, workflows are not able to properly represent the possible flows of work [5]. They often are either too simple, thus unable to handle the variety of situations that occur; or they are too complex, trying to model every imagined possible situation but being hard to maintain. In both cases they may cause several problems to the company. To tackle these problems, flexible processes surged as a shift paradigm from traditional approaches [7]. The word *flexible* in this context means the process is not static, it can change or get adjusted during its execution according to different situations.

Several different implementations of tool support for managing enterprises employing flexible processes have been proposed. They can be split into two categories:

- *change-oriented*: allow the business process change at runtime;
- *declarative*: less prescriptive than workflow; it adopts declarative languages.

This paper focuses on declarative process, whose the main concept is to define the business process behavior by business rules described through a declarative language. Traditional workflows take an "inside-to-outside" strategy, where all the executions alternatives must be detailed on the process model. On the other hand, a declarative process takes an "outsite-to-inside" strategy, where the execution options are guided by constraints [8]. Adding new constraints reduces the number of execution options.

In this constraint-based approach, a process model is composed of two elements: activities and constraints. An activity is an action that updates the enterprise status and is executed by a resource. A constraint is a business rule which must be respected during the whole process execution. Thereby, the permission to execute activities is controlled by business rules, where each activity is enabled to be executed as soon as the business rules allow it.

3. Related Works

There are some tools available to support the execution of declarative processes [9] [10] [11] [12]. However none of them employs Service Oriented Computing (SOC) concepts. In all these systems, the activities are not actually executed by the tool . The user only informs when he starts and conclude/cancel each activity but its execution is not integrated to the system. The user has to manually execute the activities.

Current web service solutions are not able to execute flexible processes because their implementations provide a static execution [13]. BPEL [14], the facto standard for web services business process description, is static and not easy to adapt [15]. Hence, there is a need for service oriented solutions to be more flexible since the business policies and environments change quickly. In this context, some solutions [16][17] [18] [19] [20][21] employs SOC and intends to make BPEL descriptions more flexible or adaptable.

These solutions makes the BPEL more adaptive by allowing dynamic composition. The idea is to make the service composition during runtime. This is less prescriptive than the traditional static composition strategy. Some works [16][17] aims at improving the QoS and prevent SLA violations. They keep the composition structure and monitor the QoS parameters to decide which service to invoke from a group of possible services.

Other dynamic composition solutions [18] [19] [20][21] adapt the business process structure to reflect the current status of the process execution. Such approaches do not redeploy the process after modifying their structure.

VxBPEL [18] is an extension to the standard BPEL language that provides VariationPoint. VariationPoint is a container of possible BPEL codes available for selection at runtime.

AO4BPEL [19]is another extension that improves the business process flexibility using aspect-oriented concepts. The BPEL structure can be changed through the aspects defined. The user can activate the aspects during runtime and then the web service flow composition can change at runtime. CEVICHE [20] is a tool that employs the AO4BPEL. CEVICHEÕs users do not activate the aspects. Instead, it activates the aspects through a Complex Event Processing (CEP) engine. CEVICHE can automatically decide *when* and *how* to adapt the system by analyzing events with CEP technology.

Xiao *et al.* [21] proposes a constraint-based framework that employs process fragments. A process fragment is a portion of a process that can be reused across multiple processes. These fragments are selected and composed based on some business constraints and policies. The resulting process is a standard BPEL process, deployable on standard BPEL engines.

Another dynamic composition proposal is the SCENE service execution environment [22]. It allows the BPEL to be changed at runtime by choosing the correct service to be invoked based on business rules. These rules are used to realize the correct bindings between the BPEL engine and the services. For this purpose, there is a rule engine that makes the decisions about the services selection.

All the aforementioned works are extensions to BPEL aiming at making it more adaptive. However, none of them provide ways to execute declarative processes. Since declarative processes do not have any predefined structured, it is not possible to execute them using BPEL or its extensions.

4. A declarative Web-service orchestrator

In the absence of solutions to execute declarative business processes through web services composition, we propose a

flexible orchestrator. Our work aims at allowing users to compose services at runtime, generating a business process that respect the business rules. As in declarative processes, if the business rules do not prohibit the execution of an activity, it is enabled. The user chooses one of the enabled activities (service operation) to execute, generating a state transition. Then, the engine evaluates the process rules to determine the set of enabled transition in the next state.

To implement the orchestration mechanism, the engine must register the input/output data of the services executed. The current implementation supports values of the following types: int, float, double, String, or boolean types, or a list of any of these types.

This section presents our declarative orchestration mechanism and the architecture of our solution.

4.1 Process Definition

Before starting the process execution, the user must specify the process activities, their respective service bindings, and the business rules of the business model. Moreover, the user should provide the data bindings necessary for the execution of each activity (service operation). We created a XML-based language to express such business model and service properties. The code in 1 presents an example of a process definition using this language.

```
<process>
  <globalData>
    <variable name="list"
        type="STRING_LIST"></variable>
    <variable name="output" type="INT"
        initialValue="5"></variable>
  </globalData>
  <activities>
    <activity name="activtyA">
      <serviceBinding operation="operationName"
          wsdlUrl="serviceUrl"
          portType="portType"
          binding="binding" />
      <dataInputBinding>
        <variableBinding variableName="list"
            global="true"
            expression="xpath:/input/list" />
        <variableBinding
            variableName="localVariable"
            global="false" type="FLOAT"
            expression="xpath:/input/test" />
      </dataInputBinding>
      <dataOutputBinding>
        <variableBinding variableName="output"
            expression="//servicoResponse/return"
            />
      </dataOutputBinding>
    </activity>
  </activities>
</process>
```

XML 1: Process Definition

The main tag *process* contains all the necessary information to the process execution.The user must specify the activities and the global data the orchestrator will manage. The tags *activities* and *globalData* are used with this propose.

The *globalData* tag contains the list of global variables. These variables are public. Any activity in the process can access and modify their values during its execution. A *variable* tag contains three attributes: *name*, *type*, and *initialValue*. The *name* and *type* are required and refer to the variable name and variable data type respectively. The *initialValue* is optional and indicates the variableÕs initial value.. If it is not defined, the variable is initialized with the default value for its data type.

The *activities* tag contains a list of activities. Each one has only one attribute, that defines its name, and three properties: *serviceBinding*, *dataInputBinding*, and *dataOutputBinding*. The *serviceBinding* tag includes all the necessary information to associate this activity execution with a service invocation. Such information is described through four attributes:: *operation*, *wsdlUrl*, *portType*, and *binding*. The *wsdlUrl* informs the url where the wsdl can be accessed from; *operation* is the serviceÕs operation name; and *portType* and *binding* determine which portType and binding defined in the wsdl will be used.

The *dataInputBinding* specifies the variable binds required for each type of operation. Each operation input must be a value associated through a *variableBinding*. Thereunto, the *variableBinding* has four attributes: *variableName*, *global*, *type*, and *expression*. *VariableName* indicates the variable's name. The *global* tag is a boolean value used to distinguish global and local variables. If the value is **true**, the global variable denoted in *variableName* will be selected. Otherwise, a local variable is created with type denoted through the attribute type. The attribute expression denotes an XPATH expression. It defines the operationÕs parameter the *variableBinding* refers to. It works for simple or complex types and refers to the SOAP request message of this operation.

The SOAP message includes the current value of the referred global variable. However, when a local variable is used, the user must provide the value for the SOAP message when the referred activity is selected.

The *dataOutputBinding* is very similar to the *dataInputBinding* since it has a list of *variableBinding*. However, *dataOutputBinding* represents a global variable update through the operation response. When an operation is invoked, its return is caught by a *variableBinding* and some global variables are updated. Hence, in an output binding, all the variables are global and, because of this, the tags *global* and *type* are not necessary. Besides that, the expression attribute refers to an XPATH expression that will be used

to select a value from the SOAP response message. Then, the referred global variable value will be updated to this value.

This process definition is the user input to the system. The user must also inform the set of business rules to the rule engine. The template of business rules and their definition depends on the type of business rule engine the system will adopt.

4.2 Overview

A declarative web-service orchestrator interacts with the web-services and a rule engine. Figure 1 shows an overview of the interaction or the orchestrator and the rule engine, web services, and users.

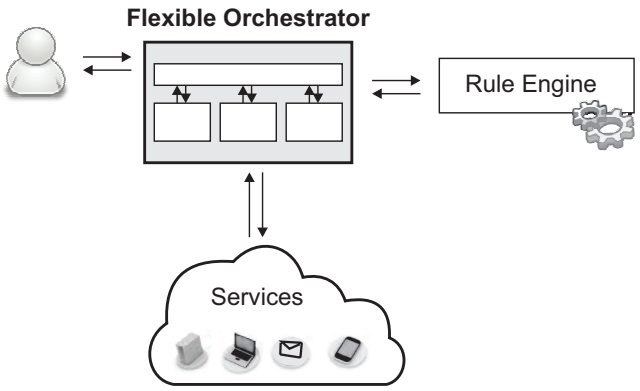

Fig. 1: Overview of the proposed web-service orchestrator.

The main component is the Flexible Orchestrator since it is responsible for interacting with all the others components. Its internal behaviors and architecture are explained in next subsection.

Through an user interface, the user can choose the next activity to be executed by the declarative web-service orchestrator. The user interface shows the enabled activities, the current states of the global data, and if the process termination is enabled or not.

When an activity is selected, the Flexible Orchestrator invokes the correct web service and then waits for the response. It notifies the rule engine whenever an activity is executed. The rule engine is responsible for checking all the business rules and updates the process instance status. Every time the process instance status changes, the Flexible Orchestrator updates the user interface with the set of activities enabled to execute.

An external system can plug its own implementation of the rule engine or adopt a available one. This rule engine must know all the rules and must listen and generate some events expected by our solution. Every time an activity is executed, the Flexible Orchestrator generates the event *DONE(activityName)* and sends it to the rule engine(s). In order to update the process instance status, we expect some events: *ENABLED(activityName)*, *DISABLED(activityName)*, *ENABLED_END()*, and *DISABLED_END()*. Hence, the rule engine must generate these events for the correct interaction with our declarative web service.

4.3 Architecture

Figure 2 presents the architecture of the proposed flexible web-service orchestrator.

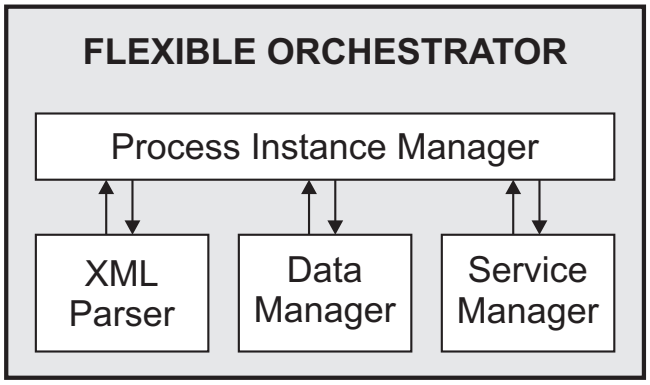

Fig. 2: The architecture of the web-service orchestrator.

Our solution contains four main components:

- **XML Parser:** reads the user input (xml) and parses it in order to make it readable by the Process Instance Manager.
- **Data Manager:** stores the global variables and controls their accesses and updates.
- **Service Manager:** invokes the service operations. This component is responsible for creating the SOAP request message, invoking the service, reading the SOAP response message, and giving the requested results to the Process Instance Manager every time an activity is chosen to be executed.
- **Process Instance Manager:** this is the component that controls the whole flow of a process instance execution. It is responsible for interacting with the other components and the user interface. When the execution starts, it interacts to the XML Parser in order to parse the user input. After receiving the activities and global variables, the execution can actually start. When the user selects an activity to be executed, the Process Instance Manager requests to the Data Manager the necessary data to the variable bindings and then the correct bindings are done. In the sequence, it forwards this information to the Service Manager and waits for its response. When the Service Manager returns the requested data, the output bindings are then made. Besides that, this component interacts with the rule engine in order to update the group of enabled activities.

5. Case Study

In order to demonstrate the use of our declarative web-service orchestrator, this section presents a case study. It consists of an example of declarative process. We demonstrate how to use our declarative orchestrator to control the execution of two different instance of this process.

A business process of a travel agency is used in this section to exemplify the usage of our orchestrator. This is a declarative process expressed using activities and rules. The travel agency can book flight tickets and/or hotels. When the travel agency faces an international transaction, it can convert currencies.

The travel agency contains three services. Each operator of a service is considered as a business process activity. Table 1 details the three services, their operations, and the input and output parameters of each operation.

Table 1: Services details.

FLIGHT SERVICE		
Operations	**Input**	**Output**
checkFlightPrice	- from - to - date - airline	- flightId - price
bookFlight	- flightID	- bookID
payFlightBooking	- bookID - value - creditCard	- paid - paymentCode
HOTEL SERVICE		
Operations	**Input**	**Output**
checkHotel	- hotelName - checkInDate - checkOutDate	- roomsAvailable
bookHotel	- hotelName - checkInDate - checkOutDate - persons	- bookID - bookValue
payHotelBooking	- bookID - value - creditCard	- paid - paymentCode
CURRENCY SERVICE		
Operations	**Input**	**Output**
convertCurrency	- value - fromCurrency - toCurrency	- newCurrency

The declarative business process for the travel agency is a set of business rules, defining constraints to control the set of activities is enabled to execute at each execution point. The process has five rules:

1) It is not possible to book a flight without checking its price before.
2) If a flight is booked, a payment for this booking must be done after that.
3) If the customer wants to book a flight, its price must be checked at least in two different airlines.
4) It is not possible to book a hotel without checking it before.
5) If a hotel is booked, a payment for this booking must be done after that.

One can notice that the currency service was not mentioned among the business rules. This means that the user can choose currency operations at any time while executing the business process, which characterize the flexibility provided by declarative processes.

After modeling the declarative business process, one must link each activity in the process to the corresponding web service. The orchestrator uses this information to perform the service bindings when an activity (service operation) is executed. The code XML 2 presents the variables used in this case study. The code only describes the XML definition for the *payFlightBooking* activity. This activity has references to local variables, whose values are provided by the user during runtime.It also has references to global variables. For example the input parameter *bookValue*, which is the return value of the activity *bookFlight*.

```
<process>
  <globalData>
    <variable name="flightID" type="STRING"/>
    <variable name="prices" type="DOUBLE_LIST"/>
    <variable name="bookID" type="STRING"/>
    <variable name="bookValue" type="DOUBLE"/>
    <variable name="paid" type="BOOLEAN"/>
    <variable name="paymentCode" type="INT"/>
    <variable name="roomsAvailable" type="INT"/>
    <variable name="newCurrency" type="DOUBLE"/>
  </globalData>
  <activities>
    ...
    <activity name="payFlightBooking">
      <serviceBinding operation="payFlightBooking"
      wsdlUrl="http://...FlightService?wsdl"
      portType="FlightServicePortType"
      binding="FlightServiceSOAP11Binding"/>
      <dataInputBinding>
        <variableBinding variableName="bookID"
        global="true"
        expression="xpath:/payFB/bookID"/>
        <variableBinding variableName="value"
        global="false" type="DOUBLE"
        expression="xpath:/payFB/value"/>
        <variableBinding variableName="creditCard"
        global="false" type="STRING"
```

```
        expression="xpath:/payFB/creditCard"/>
      </dataInputBinding>
      <dataOutputBinding>
        <variableBinding
        variableName="paid"
        expression="//payFB/result/paid"/>
        <variableBinding
        variableName="paymentCode"
        expression="//payFB/result/pCode"/>
      </dataOutputBinding>
    </activity>
    ...
  </activities>
</process>
```

XML 2: Process Definition of Case Study

In order to exemplify how the proposed orchestrator works, we will show two different executions of the travel agency business process.

Before starting the process execution, the orchestrator requests the engine the process initial state. The rule engines notifies that the activities *ckeckFlightPrice*, *checkHotel*, and *convertCurrency* are enabled. The other activities *bookFlight*, *payFlightBooking*, *bookHotel*, *convertCurrency* are disabled. The process termination is also enabled at this execution point. This happens because the process does not obligate the execution of any activity before the process termination.

5.1 First execution

Let us assume that a customer wants to buy an international flight ticket. The flight ticket is sold in dollar, but this is not the local currency. Hence, the currency service will be useful for this execution.

The *checkFlightPrice* activity is enabled at the process initial state. The travel agency employee decides to execute this service. He provides the origin and destination places, the flight date, and the airline company as input parameters. The service returns the flight identifier and add its price to the list *prices*. After the service execution, the orchestrator sends a *DONE(checkFlightPrice)* event to the rule engine. According to the rules, the set of activities enabled is not modified after executing the *checkFlightPrice* activity. Thus, the engine does not send any event back.

Afterwards, according to rule 3, the travel agency employee checks the flight price in other airline. When the rule engine receives the *DONE(checkFlightPrice)* event again, it sends the *ENABLED(bookFlight)* event back to the orchestrator, according to rules 1 and 3. One can notice that the *checkFlightPrice* activity continues to be enabled. Thus, if the travel agency employee wants to check flight price in another airline, she can repeat the operation an unlimited number of times.

The travel agency employee, along with the customer, decides to book one of the flights checked. But, before that, the customer wants to know the flight price in the local currency. For that, the employee executes the *convertCurrency* service, which does not modify the activities state. When the client authorizes, the employee books the flight. At this moment, when the rule engine receives the *DONE(bookFlight)* event, it returns *ENABLED(payFlightBooking)* and *DISABLED_END()* events. This last event was received because, according to rule 2, the activity *payFlightBooking* must necessarily be executed before the end of the process.

When the customer decides to pay its flight booking, the employee executes the payFlightBooking service. Only this activity is mandatory at the current execution point. The process termination activity cannot be enabled if this activity is not executed.

5.2 Second execution

Let us now assume that a customer wants to book a hotel in another city of the same country. For this execution, the currency service will not be used.

At the beginning, the travel agency employee check hotels in the desired city. For that, he executes *checkHotel* service informing the hotel name, and check-in and check-out dates. This service returns the number of rooms available in the hotel. After the execution of this service, and according to rule 4, the rule engine generates to the orchestrator the *ENABLED(bookHotel)* event.

The customer decides to book certain hotel, and the travel agency employee makes his booking. This action makes the engine to enable the *payHotelBooking* service. The engine also disables the process termination (through the *DISABLED_END()* event) because the *payHotelBooking* activity is now obliged to be executed before the end of the process, according to rule 5.

In another occasion, but before paying its hotel booking, the customer could decide to check other hotels. Hence, the employee executes the *checkHotel* many times in order to find a hotel that pleasure the customer. Executing the *checkHotel* service many times does not violate any rule. When he finds one, he books this hotel, and then the customer can pay for his booking. From this moment, the process execution for this customer can be finished since there is no mandatory activity pending.

One can perceive that according to the business process defined through business rules, the activities can be executed in any order and/or how many times it is necessary if this execution does not violate any rule.

6. Conclusions

This work proposes a web-service orchestrator for executing declarative business processes. This kind of business process models rely on business rules to describe the behavior of the process and to control the execution of process

instances. When working with such a model, the users are driven by the system to produce required results, while the manner in which the results are produced depends on the preferences of users.

Some of the already existent orchestrators provide a static execution, such as the ones that use the BPEL standards. Other orchestrators are more adaptive and allow dynamic composition, but they only provide runtime binding with some pre-modeled compositions. In turn, our proposed orchestrator makes service composition at runtime, binding the output data of a service to the input data of another service.

Our orchestrator receives a declarative process as input. The user also provide the service binding with the process activities, and the data bindings. For this, we defined an XML-based language to specify the business model and services properties.

Our complete solution interacts with a business rule engine. This engine receives and sends events to the orchestrator in order to check the business rules and update the process instance status.

To demonstrate our orchestrator, we showed two different executions of a same business process. The business process presented as example is declarative, and it is expressed by activities and rules. Through the different executions, it is possible to notice the flexibility to choose the order of activities executions and how our orchestrator binds data between services.

Acknowledgment

This work was partly supported by the Brazilian Research Council(CNPq[1]), grants PQ 314539/2009-3, GD 140512/2009-8, GD 142032/2010-7, and CESAR[2].

References

[1] A. Poduval, D. Todd, and H. Gaur, *Do More with SOA Integration: Best of Packt*. Livery Street Birmingham, UK: Packt Publishing, 2011.

[2] H. Kacem, W. Sellami, and A. Kacem, "A formal approach for the validation of web service orchestrations," in *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*, june 2012, pp. 42 –47.

[3] F. Rosenberg and S. Dustdar, "Towards a distributed service-oriented business rules system," in *Proceedings of the Third European Conference on Web Services*, ser. ECOWS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 14–. [Online]. Available: http://dx.doi.org/10.1109/ECOWS.2005.28

[4] X. Chen, H. Zeng, and T. Wu, "Decentralized orchestration with local centralized orchestration for composite web services," in *Proceedings of the 2010 International Conference on Parallel and Distributed Computing, Applications and Technologies*, ser. PDCAT '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 255–260. [Online]. Available: http://dx.doi.org/10.1109/PDCAT.2010.16

[5] S. Nurcan, "A survey on the flexibility requirements related to business processes and modeling artifacts," in *HICSS '08: Proceedings of the 41st Annual Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, 2008, p. 378.

[6] M. Pesic, "Constraint-based workflow management systems: Shifting control to users," Ph.D. dissertation, Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2008.

[7] W. M. P. van der Aalst and M. Pesic, "Decserflow: Towards a truly declarative service flow language," in *The Role of Business Processes in Service Oriented Architectures, 16.07. - 21.07.2006*, ser. Dagstuhl Seminar Proceedings, F. Leymann, W. Reisig, S. R. Thatte, and W. M. P. van der Aalst, Eds., vol. 06291. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[8] M. Pesic and W. M. P. van der Aalst, "A declarative approach for flexible business processes management," in *Business Process Management Workshops*, ser. Lecture Notes in Computer Science, J. Eder and S. Dustdar, Eds., vol. 4103. Springer, 2006, pp. 169–180.

[9] P. Browne, *JBoss Drools Business Rules*. Packt Publishing, 2009.

[10] E. F. Hill, *Jess in Action: Java Rule-Based Systems*. Greenwich, CT, USA: Manning Publications Co., 2003.

[11] S. Bhansali and B. N. Grosof, "Extending the sweetdeal approach for e-procurement using sweetrules and ruleml," in *Proceedings of the First international conference on Rules and Rule Markup Languages for the Semantic Web*, ser. RuleML'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 113–129. [Online]. Available: http://dx.doi.org/10.1007/11580072_10

[12] M. Pesic, H. Schonenberg, and W. van der Aalst, "Declare: Full support for loosely-structured processes," in *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, oct. 2007, p. 287.

[13] B. Orriens, J. Yang, and M. Papazoglou, "A rule driven approach for developing adaptive service oriented business collaboration," in *In: ICSOC*, 2005, pp. 61–72.

[14] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, *et al.*, "Business process execution language for web services," 2003.

[15] H. Weigand, W.-J. van den Heuvel, and M. Hiel, "Business policy compliance in service-oriented systems," *Information Systems*, vol. 36, no. 4, pp. 791 – 807, 2011, <ce:title>Selected Papers from the 2nd International Workshop on Similarity Search and Applications SISAP 2009</ce:title>. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306437910001377

[16] A. Strunk, R. Reichert, and E. Schill, "An infrastructure for supporting rebinding in bpel processes," 2009.

[17] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A framework for qos-aware binding and re-binding of composite web services," *J. Syst. Softw.*, vol. 81, no. 10, pp. 1754–1769, Oct. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2007.12.792

[18] M. Koning, C.-a. Sun, M. Sinnema, and P. Avgeriou, "Vxbpel: Supporting variability for web services in bpel," *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 258–269, Feb. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2007.12.002

[19] A. Charfi and M. Mezini, "Ao4bpel: An aspect-oriented extension to bpel," *World Wide Web*, vol. 10, no. 3, pp. 309–344, Sept. 2007. [Online]. Available: http://dx.doi.org/10.1007/s11280-006-0016-3

[20] G. Hermosillo, L. Seinturier, and L. Duchien, "Using complex event processing for dynamic business process adaptation," in *Services Computing (SCC), 2010 IEEE International Conference on*, july 2010, pp. 466 –473.

[21] Z. Xiao, D. Cao, C. You, and H. Mei, "Towards a constraint-based framework for dynamic business process adaptation," in *Proceedings of the 2011 IEEE International Conference on Services Computing*, ser. SCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 685–692. [Online]. Available: http://dx.doi.org/10.1109/SCC.2011.95

[22] M. Colombo, E. Di Nitto, and M. Mauri, "Scene: a service composition execution environment supporting dynamic changes disciplined through rules," in *Proceedings of the 4th international conference on Service-Oriented Computing*, ser. ICSOC'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 191–202. [Online]. Available: http://dx.doi.org/10.1007/11948148_16

[1] http://www.cnpq.br

[2] http://www.cesar.org.br