Automatic Adaptation of BPEL Processes Using Semantic Rules: Design and Development of a Loan Approval System*

Feyza Merve Işık Bülent Taştan
Pınar Yolum
Department of Computer Engineering, Boğaziçi University,
TR-34342 Bebek, Istanbul, Turkey
{feyza.isik, bulent.tastan, pinar.yolum}@boun.edu.tr

Abstract

Dynamic service composition is an important challenge for many applications. This paper considers dynamic composition where the parties involved and the process they execute changes based on context. In such a setting, creating compositions from scratch is costly. Instead, we advocate automatic adaptation of existing compositions to fit the requested service demands.

Our approach starts from existing BPEL processes. The details and constraints on the environment are expressed by semantic rules. We automatically decide which services are necessary to complete a desired service in a given BPEL process and which services can be removed from the process by using a reasoning engine. Based on this information, our flow generator modifies existing BPEL processes to derive effective variations. The newly generated BPEL process is then executed. We study this approach in the context of a loan approval system and show how the modifications can be done on example scenarios.

1 Introduction

Web services are software that provide functionalities, such as business processes, over the Web [11]. Web services can describe their functionalities with standard languages, can be searched through public registries, such as UDDI directories, and interact with other Web services using standard protocols.

Web services are especially useful when they can provide coordinated services. Service composition refers to the interoperation of autonomous and heterogeneous Web services. When a particular service request cannot be handled by a single Web service, a number of Web services should be selected and combined to satisfy this particular request. Since, in real-life, most requested business processes are complicated, composition of various services with different capabilities becomes crucial.

Various approaches to composition exist. The simplest approach assumes that the service request and possible Web services are available to achieve the service request before execution starts, in other words all staff is done at design time. Hence, these approaches build a static workflow from the available services to fulfill the desired service. Business Process Execution Language for Web Services (BPEL4WS) is an important language to describe such static workflows [2]. While such a composition is easy to build; it lacks dynamism needed in many scenarios. Static workflows are not sensitive to changes in the domains that we consider. More specifically, the requested service as well as the available service may not be available after the execution starts. Another composition type is automated composition that most of the composition efforts are taken for this type. In automated composition, it's difficult to select suitable involving services and build the workflow for desired service. Each time, workflow is constructed from scratch.

Rule-based adaptive workflows are known as one of semi-automated composition types. The main underlying idea of these techniques is that based on some rules that are received from the requester or the environment, the existing workflows can automatically modify themselves to satisfy the rules. Thus, even though there is an initial workflow, the automatic modification allows the workflow to adapt to dynamic changes in the environment. Without such an automatic adaptation,

*This research has been partially supported by Boğaziçi University Research Fund under grant BAP06A103 and The Scientific and Technological Research Council of Turkey by a CAREER Award under grant 105E073.

1-4244-0832-6/07/$20.00 ©2007 IEEE.

the workflow would have to be changed manually at compile time. This is clearly not acceptable in dynamic situations.

Another advantage of adaptable workflows is that it supports process-oriented structure. In other words, sometimes—based on the current situation—a process does not have to execute all of its tasks. If the workflow can adapt itself, then it can decide not to execute these unnecessary tasks. Such an adaptation would improve the efficiency considerably, because the most suitable workflow would be generated at run time based on the request.

Accordingly, this paper develops an approach for automatic adaptation of BPEL processes at run time. The adaptation is triggered by semantic rules that are defined to describe a specific domain. In other words, we propose to represent a process generically at compile time and to automatically modify it at run time with the leading of given semantic rules.

We use the loan approval domain as an example. Put roughly, customers contact banks (more precisely banks' Web services) to request loans for different amounts. Factors such as gender, age, occupation, and so on play in to the bank's decision for approving a loan. To gather detailed information about the customer, the bank contacts other organizations' Web services, such as the social security offices or the tax offices. However, for different customers different Web services are relevant. For example, for a citizen under the age of 20, contacting a retirement agency will not be of any use and hence should be skipped. We represent such rules declaratively in Semantic Web Rule Language (SWRL) [12]. Our adaptation approach starts with generic BPEL processes to achieve loan approval. Then, it considers the declarative, domain rules to decide which Web services that are part of the generic workflow are necessary and which need to be removed. This reasoning is done through a reasoning engine that can infer conclusions from the given rules. Following this reasoning, our adaptation algorithm modifies the generic BPEL process by removing unnecessary Web services and comes up with a modified BPEL process. Finally, this process is executed instead of the generic BPEL process. We examine two realistic loan requests to show how our adaptive approach works.

The rest of this paper is organized as follows: Section 2 describes our service-oriented architecture for loan approval system. Section 3 explains our generation of adaptive processes from existing processes. Section 4 depicts how our system can be used by walking through two examples and Section 5 compares our work with other relevant approaches.

2 Service-Oriented Approach for Loan Approval

We envision a bank Web service that receives loan requests from customers. A customer makes a loan request with her personal information and the amount of the loan that she needs. The bank performs a risk assessment based on the provided details as well as other necessary information gathered from other Web services. That is, the bank contacts other Web services to find required information to make a decision. The gathered information is used to give a grade to the customer. Based on the grade assigned by the bank's Web service, the loan is approved or rejected.

For our concrete implementation, we consider a typical loan approval system in Turkey. The relevant parties from which a bank can retrieve information are the following:

- TC Kimlik (Identification Agency): TC Kimlik has the personal information of the people, such as social security number, name, gender, age, and so on.
- Occupation Offices: Emekli Sandigi (National Pensions Agency), SSK (Social Security Agency), and Bagkur (Private Pensions Office) can provide past and present salary information of customers. A customer can have a relation with only one of these organizations based on her occupation.
- Educational Agencies: MEB and YOK are two educational agencies. MEB can provide information about a customer's high school and YOK can provide information about her undergraduate and graduate university education.
- Tapu Kadastro (Land Registry Agency): Tapu Kadastro has information related to immovable and the immovable income of people. For example, if a person owns a house, this is registered in Tapu Kadastro.
- Central Bank: This is a government bank that contains information related to deposits, security and interest amount of people.

Each organization mentioned above is represented as an individual Web Service, such as BankService, TCKimlikService, EmekliSandigiService, and so on. They are composed into a BPEL process to provide a loan approval service, which needs the capability of each service to make a decision at the end of the process. Figure 1 depicts this loan approval process in BPEL workflow form.

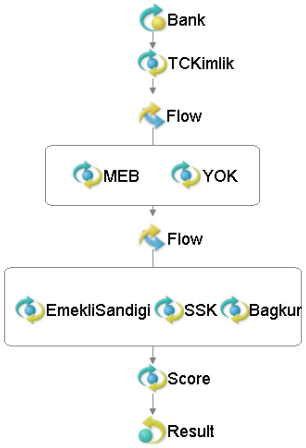

Figure 1. BPEL Process for Loan Approval

```
<BANK>
    <Finance metric="0.26"/>
        <Salary metric="0.08"/>
        <ImmovableIncome metric="0.64"/>
        <SecurityInterestIncome metric="0.28"/>
    </Finance>

    <Assurance metric="0.56"/>
        <Guarantor metric="0.59"/>
        <Immovable metric="0.25"/>
        <Deposit metric="0.16"/>
    </Assurance>

    <Occupation metric="0.12"/>
        <EmekliSandigi metric="0.45"/>
        <SSK metric="0.35"/>
        <Bagkur metric="0.20"/>
    </Occupation>

    <PersonalInfo metric="0.06"/>
        <MaritalStatus metric="0.28"/>
        <Education metric="0.64"/>
        <Gender metric="0.08"/>
    </PersonalInfo>
</BANK>
```

Figure 2. Bank's preferences file

To reach a decision on loan request, the BankService needs a standardized method. This method should reflect the weighted metrics of all criteria that are considered during the loan approval process. Note that different banks will have different criteria to evaluate applications and may possibly alter these criteria as time goes on. However, the BankService should be capable of performing loan approval process successfully even when the parameter values of the rules change without any changes in the system architecture.

Analytic Hierarchy Process (AHP) is used for calculating a score based on the information gathered from different sources by giving appropriate weights to these sources. Each bank has its own categories of criteria and each applies the rules in different ways. In this scenario, the main categories are finance, assurance, occupation and personal information are used as the main criteria [1]. Each category includes some subcategories. For example, finance contains salary, immovable income, security and interest income. The importance of each subcategory and main categories are calculated according to bilateral comparisons between each other. For instance, a bank may decide that finance is three times more important than assurance and occupation is five times more important than finance. Each customer is given a grade on individual subcategories and the weights of the individual subcategories and categories are used to assign an overall grade to the customer. This overall grade is compared with the limit score of the bank to approve or reject the loan. Each bank specifies its own weights for different subcategories in an external XML file as shown in Figure 2. For our purposes, any standard weighted criteria resolution algorithm would have been sufficient. We preferred AHP because of its usage by human loan experts in many banks in Turkey.

At the very beginning of the process, the customer provides her TCKimlikNumber (i.e., Turkish Personal Identity Number) and the requested loan amount to the BankService. The BankService carries out the actual process with consultation of necessary parties and then makes a decision. The BankService firsts asks the TCKimlikService for the validation of the identification of the customer and learns the metrics for education, gender and occupation type. The occupation type is important since based on the occupation one of Emekli Sandigi or SSK or Bagkur is consulted. The salary metric is found out by interactions with one of these occupation agencies. After determining of salary metric, the immovable metric is obtained from TapuKadastroService. If there exists an immovable, the income of this immovable is asked from the same service; otherwise this step is passed. The CentralBankService has the deposit, security, and interest metrics of the customer. Accordingly, these are obtained from that service. Finally, the BankService makes the score computations using the AHP parameter metrics.

3 Automatic Adaptation of BPEL Processes

In the previous section we have studied how a loan approval process can be modeled in a static way as composite BPEL process and how it can be executed in conjunction with bank preferences information. In this section, our focus is on automatically modifying the loan approval BPEL process based on external rules about the domain.

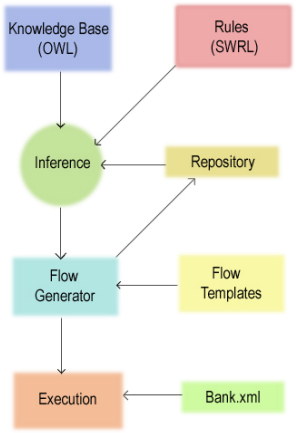

Figure 3. Automatic adaptation framework

Figure 3 depicts our framework for automatically modifying a process by using rules. There are three parts in our framework: decision part, process generation part, and process execution part. The decision part decides which service in the predefined BPEL process needs to stay and which services need to go. The process generation generates a new BPEL process based on this information and finally the process execution part executes the newly generated process for the customer's loan application.

3.1 Decision Making

The decision part consists of the knowledge-base of the domain ontology and a decision maker that makes the inference. A knowledge-base is required to represent the information that is specific to a particular domain. All concepts are defined in the domain ontology and then can be used to make inferences.

For the loan approval case, domain knowledge can be divided into two parts: the personal information of the customer that is used in inference process and the modifications that can be applied to the existing process. The concepts related to personal information are basically customer, gender, age, education and occupation type. The properties related to these concepts are relationships such as hasAge, hasOccupation, and so on, which denote different characteristics of a person.

The concepts that are related to the process are the process parts and the actions that the flow generator can apply to the process. The relevant property of this set of classes is hasValue, which can be either true or false. This property of a process part can be used as a flag for execution that shows whether the part can be removed or not. This value is set after the rules are applied. For example, if an instance of deposit class has the value false for the hasValue property then it means flow generator will remove the deposit node from the workflow.

The domain knowledge is represented in Web Ontology Languages (OWL) [8] to make it compatible with current Semantic Web efforts. Figure 4 is an example instance of Person class in the domain ontology. This is the seed information related to the customer's application for the loan approval process.

```
Person
    hasTCKimlikNo    = "2100555333"
    hasFirstname     = "Murat"
    hasSurname       = "YILMAZ"
    hasOccupation    = "None"
    hasAge           = 25
    hasEducation     = "HighSchool"
    hasMaritalStatus = "Single"
    hasGender        = "Man"
```

Figure 4. Sample Person ontology instance

This domain knowledge is used within the rules to make inferences. The rules are constructed using Semantic Web Rule Language (SWRL) [12] in the Java Expert Shell System (Jess) [5] plug-in of Protégé [10] framework. Jess is a reasoner for semantic web that support SWRL rules for a given domain ontology within the Java code using Protégé's source code. A bridge exists between Protégé and Jess, which connects the domain knowledge part and the reasoner. Accordingly, our ontology and the SWRL rules are reasoned with Jess to make inferences. The following rules are samples from loan approval ontology:

- Rule Age-1 states that if the customer is less than 25 years old, then visiting Tapu KadastroService may be meaningless. We assume that this rule is

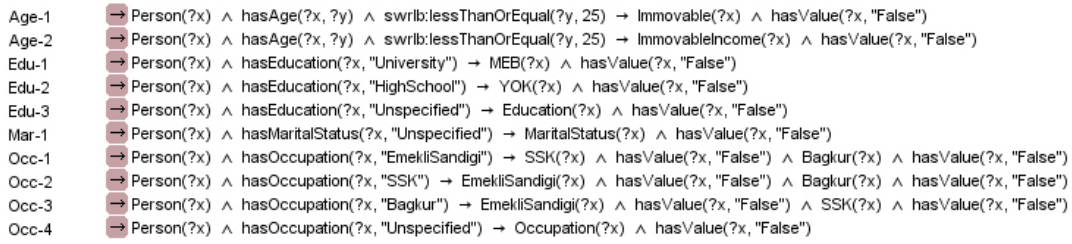

Figure 5. The SWRL rules for inferencing

specified by the bank. Then the flow generator removes the node that asks the immovable metric from the TapuKadastroService.

- Rule Age-2 states that if the customer is less than 25 years old and if rule Age-1 concludes that this customer cannot have an immovable, then visiting the TapuKadastroService to learn the customer's immovable income metric is meaningless. Thus, this node is also removed from the workflow template.

- Rule Edu-1 states that if the customer hasEducation University, then MEB node would have a value false for the hasValue property. That is, since the customer has a university education then there is no need to retrieve information about her high school.

- Rule Edu-2 states that if the customer hasEducation PrimarySchool as learned from TCKimlikNo, there is no need to visit the YOK service. It is obvious that including MEB service in the workflow is enough for this customer. Hence, YOK node becomes false.

- Rule Edu-3 states that if the education of the customer is undefined, visiting any service related to education, YOK or MEB, is meaningless, instead the default metric specified by the bank for education is used for this customer in this case. Hence, values for both nodes are set to false.

- Rule Occ-1 states that if the customer is a member of Emekli Sandigi, then she cannot be a member of SSK or Bagkur. In this case, the values for SSK and Bagkur nodes are set to false.

- Rule Occ-2 states that if the customer is a member of SSK, then she cannot be a member of Emekli Sandigi or Bagkur. In this case, the values for Emekli Sandigi and Bagkur nodes are set to false.

- Rule Occ-3 states that if the customer is a member of Bagkur, then she cannot be a member of SSK or Emekli Sandigi. In this case, the values for SSK and Emekli Sandigi nodes are set to false.

- Rule Occ-4 states that if the occupation type of the customer is undefined, visiting any service related to occupation type and salary, Bagkur, SSK or Emekli Sandigi, is meaningless, instead the default metric specified by the bank for salary is used for this customer in this case. Values of all the nodes related to these services are set to false.

The decision process is performed as follows: The ontology and the rules of the bank should be available at the beginning of the process. Whenever an application is submitted, then the bank obtains the seed information for the customer from TCKimlikService via her TCKimlikID. At this time, decision mechanism loads the domain ontology that is represented in OWL and SWRL rules. An individual of person is created in the ontology for the current customer with her personal information. The reasoning engine executes the rules defined by the bank, and determines how the workflow would be modified. If a node has value false at the end of the inference process, then it is removed from the workflow by the flow generator. The flow generator is informed about the nodes those have value false. Now, flow generator is aware of which nodes should be removed from the workflow.

3.2 Flow Generation

The second part of the framework is the flow generation. The flow generation consists of a library of existing flow templates, a flow generator that generates new flows on demand, and a repository that saves modified flows. A workflow repository is a useful container that stores the different workflows that were generated and executed before. If the needed flow is part of the repository, it can be used directly from the repository;

there is no need to generate a new flow. However, if the workflow is not present in the repository, then we need to retrieve an existing workflow from the flow template pool and adapt it based on decisions of the inference mechanism.

A flow template pool includes workflows with different complexities. Here the complexity means the variations of the components in workflow such as nodes, computational operations, messages and partners. Template selection depends on two criteria. First is the amount of the loan that the customer requests. If the amount is very high then a dense process template is used because the detailed information is required about the customer: nothing should be missed in this case. Here, we are inspired by real life situations. When the requested loan amount is high, the number and seniority of the bank officials that oversee the process increase. Another criterion is the decision made by the inference. If the decision of the inference mechanism requires removing many nodes, using the dense BPEL process, which has all possible nodes and paths, would be an inefficient way of flow generation. Instead, using sparse flow would be suitable in this case. The number of workflows in the template pool depends on the domain requirements. In our loan approval process, two templates is enough: sparse and dense BPEL4WS processes.

After a template is chosen from the template pool, the flow generator removes the necessary nodes. The nodes to be deleted from the BPEL template are specified by the inference mechanism as described above. The flow generator reads the BPEL template and deletes the lines including these nodes. These nodes can be any of "receive", "reply" or "invoke" activity in BPEL. When a node in the BPEL process is removed, then its input and output variables are retrieved, because these variables are not needed anymore and they have to be deleted as well. (All of the variables used in the process are described at the top of the BPEL template.) Since the variables are removed, the assignment operations for these variables must also be deleted from the "assign" activities in the template. We know that these variables are not needed for the inference, so it is also guaranteed that the values stored in these variables are not used in any part of the BPEL process. Finally, any "copy" operation that uses these variables are also removed. The "Partner Links" are not deleted, because when they are not used in the process they can still remain in the process without disturbing the flow.

When the deletion is finished, BPEL process is created. The flow generator places this newly created BPEL file into a new folder. Similarly, it places the WSDL files into another folder and creates a wsdlCatalog xml file, which stores the paths to the WSDL files. Then a Process Deployment Descriptor (.pdd) file is created. Each BPEL process has its own deployment descriptor. This file describes the partner links and WSDLs which are involved in the BPEL process. Since the partner links are not changed during node removal operations, all the partner links those described in the BPEL template are all included in this descriptor file. After the creation of BPEL process, a deployment archive file (.bpr) is built by using the .pdd file, WSDLs of the involved services, wsdlCatalog.xml file and this newly created BPEL process. Deployment archive is sent to the execution part of the process.

ActiveBPEL Engine[3] is used to execute the workflows. Execution part of the framework is trivial; it starts after a .bpr archive file is located in the deployment folder of BPEL engine. BPEL engine deploys the .bpr archive file, the verification that the archive to be deployed should be done. After deployment process, process is ready to serve for the current customer's loan request.

4 Evaluation

We illustrate how the loan approval service can automatically modify a generic workflow using two scenarios.

4.1. Scenario 1

The loan requester is a student who wants to receive some loan from the banking system for his new job idea. The system takes only two inputs: TCKimlikNo (Turkish Personal Identity Number) and the amount of the requested loan.

He is connected to the system by bank's Web service (i.e., BankService). As the first step, the system connects to the TCKimlikService to retrieve his personal information. With the data retrieved from the TCKimlikService, an instance of Person class is created. After the personal information is retrieved, his preferences are sent to the reasoning part of the system. We apply the rules in Figure 5 to derive the values of service nodes. From the list given in Section 3.1, rules Edu-2, Occ-4, Age-1, and Age-2 are applicable for this customer.

We assume that the repository is empty and that we need to generate a new process from available process templates. As was mentioned, we consider two BPEL process templates: one for higher loan requests and one for lower requests. Assume that the requested amount here is low and the template that we start with is that shown in Figure 1.

Based on the above four rules, YOK, EmekliSandigi, SSK and Bagkur nodes have value false and thus need to be removed. The nodes as well as the variables and assignments used for those services are also deleted. Then a WSDL, wsdlCatalog, .pdd file and generated BPEL are archived into a .bpr file. This file is stored in the repository. One entry is added to the repository with removed nodes and the name of the .bpr. The .bpr file is sent to BPEL engine to be deployed. Figure 6 shows the resulting BPEL process.

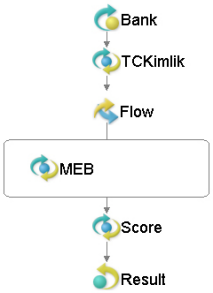

Figure 6. Low class BPEL process after the nodes removed

During execution, each Web service in the process is invoked one by one and grades are given by each service according to the credentials returned by the services. Finally, these grades are weighted with the bank's preferences as declared in bank.xml file. The overall score is used to decide whether the requested loan will be given.

4.2 Scenario 2

Next, we consider a female who applies for a loan of $20,000 from our banking system. Let's assume that this information is returned from TCKimlikService: She is 35 years old, has a high-school degree, is married and is associated with SSK. Again, by applying the rules, we derive that YOK, Emekli Sandigi, and Bagkur service will have false values and thus need to be removed.

Since our customer requests $20,000, it is classified as high (critical) and high BPEL template is considered to be used (Figure 7). Let us again assume that matching processes do not exist in the repository. The flow generator starts with the dense BPEL template and deletes the listed nodes and their related variables. The resulting process is shown in Figure 8. The new

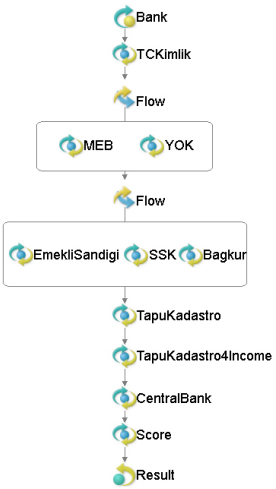

Figure 7. High class BPEL template

BPEL process is sent to the repository for possible future use and to be executed for immediate usage.

5 Discussion

Static Web service composition techniques are not adequate to satisfy many realistic service composition requirements. To achieve a dynamic composed process at run time, we developed an approach to automatically modify existing BPEL processes using semantic rules. As a running example, we used a loan approval system that is composed of several services. First, we showed how such a system can be organized as a service-oriented system. Then, we created a framework that separates the dynamic adaptation and workflow execution of a composition.

In the adaptation part, customer information is represented as instances of concepts in an ontology. Next, predefined semantic rules are applied to these instances to find out which services are relevant to the specific customers. The results of this reasoning are fed into a flow generator that starts with BPEL process templates and tailors them according to the results of the reasoning engine. As a result, since the BPEL process is designed according to the customer's specific situation, only the necessary services are run and necessary information are gathered. After the modified BPEL process is generated, it is executed in a BPEL engine by calling the necessary Web services in the process.

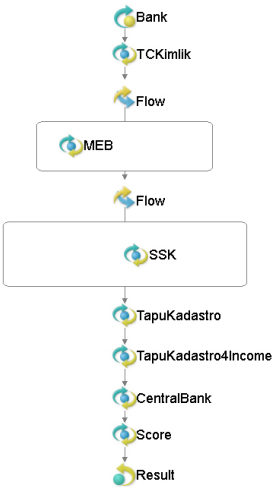

Figure 8. High class BPEL process after the nodes removed

Next, we review some relevant literature. In Rule-Driven Service Composition [7], the rules are used for matching relevant web services in composition. Contrary to our approach, these rules are not used for generating the workflow. Müller and Rahm [6] use the modification technique to deal with exceptions. However, our purpose here is to make the workflow modification before the business execution. Further, their proposed technique is not applied to web services.

Charfi and Mezini propose a composition with BPEL using an aspect manager [4]. The aspect manager interprets the aspect-oriented BPEL and adds in-between activities to the actual BPEL process. After some manipulations, the process is run in BPEL Runtime. While their framework adds some dynamism to the activities, their process manipulations are not done automatically as we have done here. Further, the process logic and BPEL logic are not separated well. However in our approach, BPEL logic and process logic are separate. The BPEL process is not manipulated manually; instead an intelligent BPEL generator is used to create the actual process with the given rules.

OWL-S is a service ontology that provides semantic descriptions to enable dynamic composition [9]. By itself, OWL-S based composition techniques use purely dynamic composition methods, such as planning to achieve a workflow. In our case, starting from scratch at every loan request would have been unnecessary and meaningless, since the involved Web services are mostly known. Hence, only making the necessary modifications is enough to achieve the desired process for the current customer.

References

[1] Murat Atan, Ufuk Maden, and Ebru Akyıldız. Analitik hierarsi sureci (AHS) kullanimi ile bir bankada kredi taleplerinin degerlendirilmesi. Technical report, Gazi University, 2004.

[2] BPEL. Web services business process execution language version 2.0, 2006.

[3] Active BPEL. Active bpel web site: http://www.activebpel.org, 2006.

[4] Anis Charfi and Mira Mezini. Aspect-oriented web service composition with ao4bpel. In L.-J. Zhang and M. Jeckle, editors, *ECOWS, LNCS 3250*, pages 168–182. Springer-Verlag, 2004.

[5] Ernest J. Friedman-Hill. *Jess in Action: Rule-Based Systems in Java*. Manning, CT, 2003.

[6] R. Müller and E. Rahm. Rule-based dynamic modification of workflows in a medical domain. In A.P. Buchmann, editor, *Proceedings of BTW*, pages 429–448, March 1999.

[7] Bart Orriens, Jian Yang, and Mike. P. Papazoglou. A framework for business rule driven service composition. In B. Benatallah and M.-C. Shan, editors, *Proceedings of TES, LNCS 2819*, pages 14–27, 2003.

[8] OWL-DL. Owl description logic, 2006.

[9] OWL-S. Semantic markup for web services, 2005.

[10] The Protégé ontology editor and knowledge acquistion system. http://protege.stanford.edu, 2000.

[11] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, Chichester, UK, 2005.

[12] SWRL. A Semantic Web Rule Language Combining OWL and RuleML, 2004.