Change Management in Semantic Business Processes Modeling

Uttam Kumar Tripathi
Department of Computer Science & Engg.
Indian Institute of Technology Kanpur
Kanpur-208016 India
uttam@cse.iitk.ac.in

Knut Hinkelmann
University of Applied Sciences
Northwestern Switzerland
4600 Olten, Switzerland
knut.hinkelmann@fhnw.ch

Abstract

Keeping IT align with business is an important task in an agile business environment which is related to change management in software development. We present a methodology and system for changing SOA-based business process implementation. We distinguish two layers: At the design layer processes are modeled in the ontology-based semantic markup language for web services OWL-S. For execution the processes are translated into BPEL. At the core of our system is a central change management component. We have implemented some generic transformation functions that can be composed to realize any configuration and reconfiguration of process modeled in OWL-S. Finally, we demonstrate our approach with an example e-government.

Keywords

Service Oriented Architecture, semantic language, Business processes, Web Services, OWL-S, BPEL.

1. Introduction

In a business environment changes are an ever occurring phenomenon. For example, agility of the market, new client needs, or organizational restructuring (e.g. because of mergers, acquisitions or outsourcing) affect the way of doing business and hence do influence the design of business processes. Adaptability to changes and speed of innovation are a prerequisite for business process management. Hence, a business process management approach should be able to accommodate these changes in the model as and when they occur.

Business processes are supported by IT. In case of changes in business it must be ensured that IT is still aligned with business.

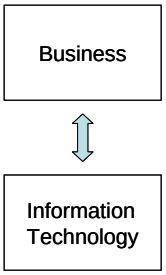

Fig 1. Aligning IT with Business

In recent years there is a trend towards service-oriented architectures for implementing business software. Instead of having a monolithic, integrated software system, modern business software is composed of reusable software components. An example of this is SAP's Netweaver technology [2,6].

A service-oriented architecture, however, is not enough to cope with the agility of business. Business semantics is required to (1) efficiently identify the components that have to be adapted because of changes in the business processes and (2) to ensure consistency of the changes.

In this work we present a methodology for aligning IT with a business process model by automatically adapting the corresponding execution environment in case a change in the business process occurs.

In accordance with most approaches for business process management we distinguish design and implementation of business processes:

- On design level we not only model the business process but also define the semantics of the enterprise services that are implemented on the implementation layer. Thus, we need an expressive modeling language with a well-defined semantics. We use OWL-S [9], OWL-based web service ontology.
- As a standard for implementing a service-oriented architecture we rely on web services. To combine web services to processes we use the business process execution language for web services BPEL [1,4] which makes it possible to define also inter-organisational services.

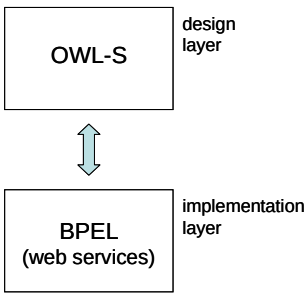

Fig 2. Process languages for design and execution

A major reason for choosing OWL-S as the process description language was that OWL-S has a mapping to Web Service Description Language WSDL [9], which are building artifacts in the implementation of process models via BPEL. This existence of grounding of service for concrete realization was a major benefit in using OWL-S.

In section 2 we briefly discuss some related work done in this area of change management for software and processes. In section 3 we describe our approach which involves the development of a central change management system and in the next section to it we describe in detail our Change Management System. In section 5 we present some of the generic process transformation functions that we have implemented to support the changes on the process specified in OWL-S. Then we discuss an example of process flow for administrative tasks involved in vehicle registration in a city and see how changes in the process model can be incorporated very efficiently by our suggested schema.

2. Related Works

Change and change handling has been an area of study in the domain of information technology for quite sometime now [8]. As mentioned in Carnegie Mellon University's Software Engineering Institute (SEI) Software Capability Maturity Model[1] (SW-CMM) [3] Software Configuration (Change) Management involves identifying the configuration of the software at given points in time, systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the software life cycle. The work products placed under software configuration management include the software products that are delivered to the customer (e.g., the software requirements document and the code) and the items that are identified with or required to create these software products. SPICE (Software Process Improvement and Capability Determination) or ISO 15504 [10] is a model for the assessment of business process using a capability determination similar to CMMI.

Workflow management systems are a special kind of middleware integrating applications involving systems and people as participants. Implementing processes using a SOA approach can be regarded as a distributed system. Our approach of issuing change transactions resulting in atomic changes in the business process is motivated from the approach followed in [7], where an efficient way of handling changes in distributed computing systems is described.

Changes in process models can be divided into two broad categories depending on the time period over which a change can be expected to occur.

- Short termed changes actually occur in scenarios where we in principle know the various kinds of changes (alternative paths) possible and hence our process model has to adapt to one of them at runtime. In [5] a methodology is described that uses business rules to cope with short-term process adaptivity.
- Long-term changes result in the need for the overall modification and evolution of processes in order to reflect the changes that have occurred. Some new activities might come-up in the business process model or some old ones might need to be removed, new conditional branches might come-up or some existing ones may get obsolete. A business process management approach should be capable to incorporating all these changes and should remain functional even after these changes have occurred.

Long-term changes are the focus of the work we present in this paper.

3. Our Approach

As we figured out during our study that there can be many reasons for changes of business process models, e.g.

- change of enterprise goals
- client needs
- technological innovations
- gratuitously long running times
- inefficient interfaces between organizational units

In the following subsections we describe a change management system to make sure that the implementation of a business process remains operational after facing these changes.

3.1 Components of the Change Management System

In our approach we have a central change management system which is responsible for managing and incorporating changes which are occurring in the business environment. This CMS (change management system) can be set into operation by two different triggers resulting in a sequence of change transactions being issued for the business process model. Once the changes have taken place on the process model they are mapped to the process existing in the execution environment.

Fig. 3 outlines the change management schema which consists of the following major building blocks:

- automatic and manual triggers
- change management system
- business process models (specified in OWL-S)
- execution environment (processes implemented in BPEL, as web services)

[1] http://www.sei.cmu.edu/cmmi/cmmi.html

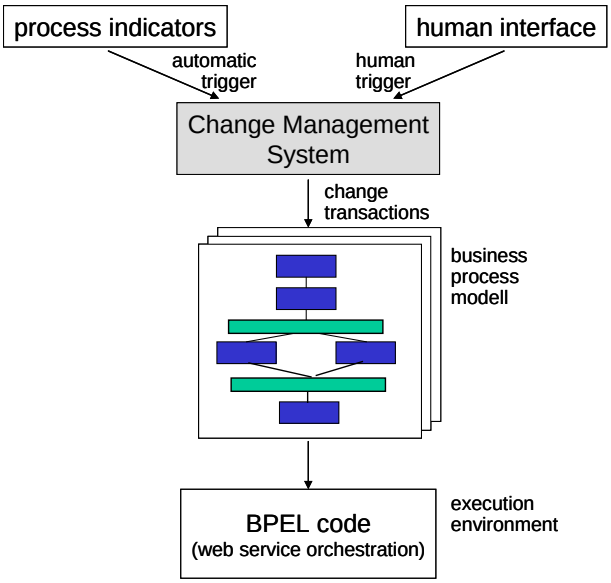

Fig. 3 The Change Management Schema

Our approach for change management is applicable for processes that are implemented in service oriented architecture. The changes managed by our approach only concern the process control structure while it does not affect the implementation of the (web) services themselves. This is in accordance with the general idea of SOA where the real implementation of the services is transparent for the application. Therefore the Web Service needed by BPEL will either pre-exist and our BPEL process will simply use it or will be created at the implementation step.

3.2 Business Process Models

In our approach business processes are modeled using OWL-S, the semantic markup language for web services. These business models are modeled to be as comprehensive and clearly outline all the details that are associated with the process model. The business process models are then implemented in BPEL with the activities being implemented as web-services.

As OWL-S provides grounding to WSDL (Web Service Definition Language) OWL-S emerged as default choice for our realization of process models. Another option would have been WSMO (Web service modeling ontology), but since grounding web services to an invocation mechanism has not yet been defined in WSMO, we decided to go with OWL-S.

3.3 Two-Step Change Procedure

Instead of directly modifying the BPEL code, however, we identify the changes on the business level which is modeled in a semantically rich ontology language OWL-S. It is obvious that once the process model gets changed the corresponding BPEL implementation needs to be redone, too. The BPEL code which will be used in the execution environment is then generated out of this modified OWL-S code (see Fig. 4).

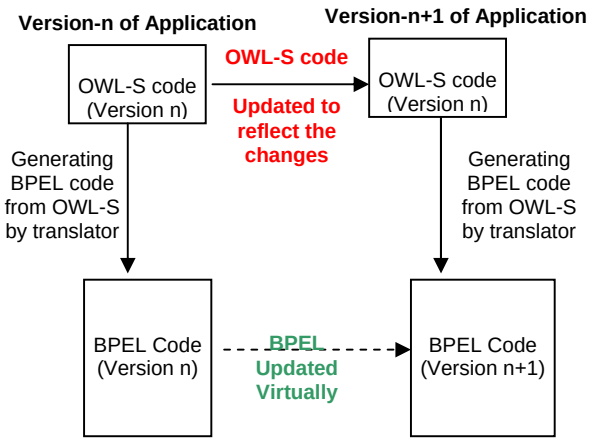

Fig. 4 Showing how an application get updated with change transaction acting on OWL-S code and BPEL code getting changed virtually

This approach is useful because for defining a process structure the details required in the OWL-S definition are sufficient. Most of the other information that is needed in BPEL is mainly implementation specific (like port information etc.). It is easy to see that by any OWL-S model can be extended in order to declaratively represent this implementation-specific information as well. Hence by making our atomic change transaction for OWL-S only, we have reduced the complexity.

In the following we concentrate on the modification of the OWL-S process, while the details of how to translate an OWL-S process to BPEL is not covered in this paper.

4. Change Management Model

As stated in the previous section in our approach we handle changes by the help of a Change management system and hence we name this solution model as change management model.

4.1 Change Triggers

Triggers are handles through which the change management system can be activated. We have foreseen two types of triggers depending on how the required change is invoked. Changes can be triggered

- automatically based on

- Measurement of key performance indicators or quality indicators, e.g. runtime, idle time or process costs.
- Mining user behavior identifying reasons for inefficiencies.

- manually by the administrator because of
  - Technology changes
  - Changes in the environment
  - Change/shift in the goals of the company

For building up the invocation system certain mapping must exist between indices and artifacts of the model, these are figured out well in advance and it remains a task of analysts. Manual invocation is provided to deal with the worst case scenario of "unforeseen" cases.

Currently we have completed the implementation invoking the change management system using the manual trigger only. The task of figuring out the quality indicators and creating the automatic invocation system based on that is a part of future work.

4.2 Change Management System

The change management system is the most important component of the overall model as once it gets invoked through any one of the triggers it issues change transaction which will be perform the desired changes in the Business process model. The change transactions that are issued are actually certain generic actions that can be used for configuration and reconfiguration of the process models.

Changes in a business process model might result in

(1) Some activities of the business process getting obsolete and hence need to be removed from the process model or
(2) New activities that have to be included in the process model or
(3) Modifying the data/control flow sequence of the process.

We implemented seven generic functions to cope with these changes:

CreateProcess generates a new process
DeleteProcess deletes the corresponding process
PutInSequence defines a sequence control construct between two services; the IDs of these processes are also given as argument of the function.
RemoveFromSequence deletes a process from a given sequence.

Apart from this we have implemented generic actions for conditional operators (If then Else, Split, Repeat Until)

On invocation a generic function results in a sequence of changes that get performed in the corresponding OWL-S code specifying the process model. We have defined two sets of generic functions in our approach one is used for configuration (creating the process model for the first time) and then another set for reconfiguration which along with the first set is used for managing the changes happening. We provide a detailed description of these generic functions and how they affect an existing OWL-S process description later on in this paper.

5. Generic transformation functions for (re)configuration of process models

For configuration of the process models at the time of their first creation and for reconfiguration at any later point in time we have defined a set of minimal generic functions. They are these functions which can be combined to perform any complex (re-)configuration action. The atomic actions adopted in our approach are such that a combination of them can be used to portray any possible change that can occur to a Business process model.

Following are the set of functions that will be used for configuration of the business process model (first creation).

- **CreateProcess:** This is used for creating a new OWL-S process. Our create action supports the passing of following information in order to it to be able to make a new process.

  - *Process ID*
  - *Input ID, ParameterType (one or more than one)*
  - *Preconditions*
  - *Effect ID*
  - *Conditional Output ID*

  All of this information will need to be passed over in the change transaction that will be issued from the change management system. Based on this information a code segment similar to the following will get generated in the OWL-S process description file.

- **PutInSequence:** This is the operation that will be used for connecting two services in a process model. Since in OWL-S the flow in logically stored as a sequence hence we use this sequence structure only to implement our generic action. Along with putInSequence we will be passing following parameters.
  - *Process to be added before or after the existing process(0 or 1)*
  - *Sequence ID*

- *Process existing in the sequence to which we need to concatenate a new process*
- *Process to be added*

- **Split and split+join:** For split operation in OWL-S a processComponentBag is created and this process bas is associated with the particular split process. Hence the processes of the bas start getting executed concurrently when the split operation is called. In our control transaction of split we need to specify the splitID and the services which are needed to be contained in the processComponentBag of that split operation.

  In OWL-S a join can exist only after a split has already occurred. This results in the process undergoing concurrent execution with barrier synchronization. Note that there can be scenario of split all and join some sub-bag.

- **If-Then-Else:** For including the control constructs which will result in generating a scenario of *If-Then-Else* for our business process model we need to pass the following information along with the *If-Then-Else* transaction.

  - *If condition*
  - *Then sequence (sequence of processes that will be executed when if condition is true)*
  - *Else sequence (sequence of processes that will be executed when if condition is false)*

- **RepeatUntil:** For implementing *RepeatUntil* we need to pass on the
  - *Condition*
  - *Process sequence to be executed while condition is true*

Note that for every conditional operator their is a unique ID associated, hence if one invokes conditional action with a ID which already exists in the process model, then simply the existing description is overwritten with the new one, thereby accommodating "partial changes" of existing conditional execution description.

For reconfiguration i.e. incorporating a change at a later point in time we will need two new generic actions apart from all those mentioned above.

- **DeleteProcess:** This is a generic action used for deleting a service from the process model. For invoking this generic action one needs to pass over
  - *Process ID*

  And the process description is removed from the model.

- **RemoveFromSequence:** For modifying the sequence flow by removing the process which is not needed anymore we use this generic action. As an argument for this action we provide
  - *Sequence ID*
  - *Process ID of the process to be removed*

6. Example using "Vehicle Registration" Case:

For understanding the change management schema that we have suggested let us consider an example scenario. Vehicle registration and monitoring in large cities is a job which requires lots of office and paper work. Let us consider the case of vehicle registration process at the RTO (Road Tax Office) in the city of Kanpur. Kanpur is an Industrial town located in the Northern India with an estimated population of 4.1 million people. Because of the high industrial activities going on in the city the traffic in the city is immense, including both motorized as well as un-motorized vehicles. As per the RTO policy of the city every motorized vehicle in the city needs to be registered with the RTO office and needs to pay some tax, which is used to build new roads and highways in and around the city. However, not all kinds of vehicles are allowed to be registered in the city as Kanpur faces very serious problem of traffic congestion because of very high density of vehicles operating in the city. Hence very heavy vehicles are denied registration and permission to operate in the city. For this purpose the RTO office uses a list which pre-exists in its database, and which provides a guideline of which kind of vehicle is allowable in the city. The overall process of vehicle registration is shown below in Fig. 5.

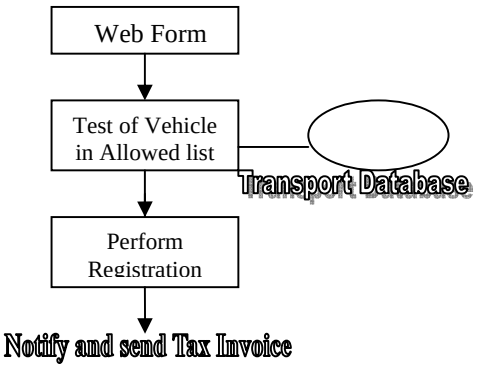

Fig. 5 Vehicle Registration process flow.

The above process can be described as an OWL-S process as follows:

<process:CompositeProcess rdf:ID="Vehicle_Registration">

```
  <process:composedOf>
   <process:Sequence>
     <process:components rdf:parseType="Collection">
        <process:AtomicProcess rdf:about="#Test_IF_InAllowedList"/>
        <process:AtomicProcess rdf:about="#PerformRegistration"/>
     </process:components>
   </process:Sequence>
 </process:composedOf>
</process:CompositeProcess>

<process:AtomicProcess rdf:ID=" Test_IF_InAllowedList ">
  <process:hasInput rdf:resource="#PersonName_In"/>
  <process:hasInput rdf:resource="#PersonAddress_In"/>
  <process:hasInput rdf:resource="#VehicleModel_In"/>
  <process:hasInput rdf:resource="#ChassisNumber_In"/>
  <process:hasInput rdf:resource="#DateOfPurchase_In"/>
  <process:hasOutput rdf:resource="#VehcileAllowed_Out"/>
</process:AtomicProcess>

<process:Input rdf:ID=" PersonName_In ">
  <process:parameterType rdf:resource="&concepts;#Name"/>
</process:Input>

<process:Input rdf:ID=" PersonAddress_In ">
  <process:parameterType rdf:resource="&concepts;#Address"/>
</process:Input>

<process:Input rdf:ID=" VehicleModel_In ">
<process:parameterType rdf:resource="&concepts#VehicleModel"/>
</process:Input>

<process:Input rdf:ID=" ChassisNumber_In ">
  <process:parameterType rdf:resource="&concepts#ChasisNumr"/>
</process:Input>

<process:Input rdf:ID=" DateOfPurchase_In ">
  <process:parameterType rdf:resource="&concepts;#Date"/>
</process:Input>

<process:UnConditionalEffect rdf:ID=" VehcileAllowed_Out ">
  <process:ceEffect rdf:resource="&concepts;# VehcileAllowed"/>
</process:UnConditionalEffect>

 <process:AtomicProcess rdf:ID=" Perform_Registration ">
  <process:hasInput rdf:resource="#VehicleAllowed_In"/>
  <process:hasOutput rdf:resource="#PayableTax_Out"/>
</process:AtomicProcess>

<process:Input rdf:ID=" VehicleAllowed_In ">
 process:parameterType rdf:resource="&concepts# VehcileAllwd "/>
</process:Input>

<process:UnConditionalEffect rdf:ID=" PayableTax_Out ">
  <process:ceEffect rdf:resource="&concepts;# PayableTax"/>
</process:UnConditionalEffect>
```

The first element represents the overall process structure. The process has ID “Vehicle_Registration”. It is composed of a sequence consisting of two atomics processes “Test_II_InAllowedList” and “PerformRegistration”. These atomic processes are then described with their input and output data (see. Fig. 5).

Let us assume that there is a change in registration policies: It was realized by the pollution control board of India that Kanpur is ranking very high in terms of air pollution. The large number of motorized vehicles operating in the city was identified as a major source for that. The RTO in a bid to lower the pollution being caused by the vehicles introduced an incentive based schema in which if a vehicle is producing the pollutants which are within the acceptable limits then the owner of that vehicle needs to pay taxes at reduced rates. The new process should look as shown in Fig. 6:

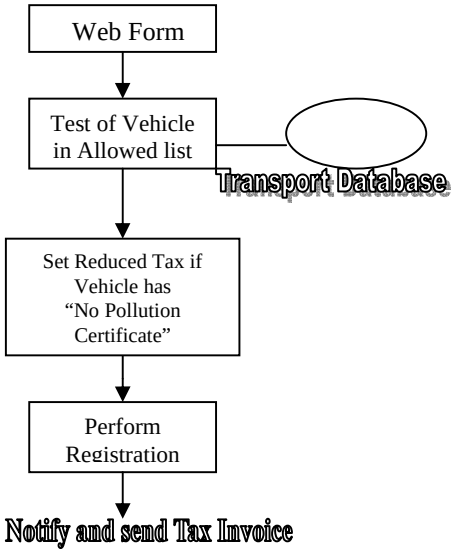

Fig. 6 Vehicle Registration process flow after introduction of new process

As is clear from the process diagram this change requires a new atomic process to be added in the flow. Rest of the process remains the same. In order to incorporate this change with the help of the generic functions described in section 5, we will need make a call of CreateProcess followed by a PutInSequence. This will result in creating a new process for checking for the “No pollution certificate” and then putting that process in the sequence just before perform registration.

In this case we will invoke our change management system and execute a CreateProcess call followed by a PutInSequence call. For create process we will specify the Process ID and the corresponding input/output then for the PutInSequence action we will specify the Process ID, the sequence to be added to, 0/1 (for before or after) and the existing service to which we need to concatenate, these sequence of atomic actions will result in modifying the semantic process description that existed earlier and hence results in generating the new process file.

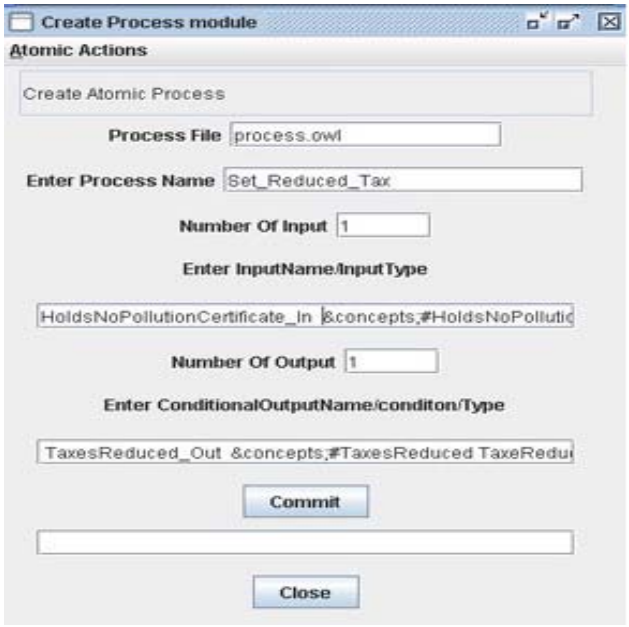

Fig 7. CMS (change Management System) executing the create process action.

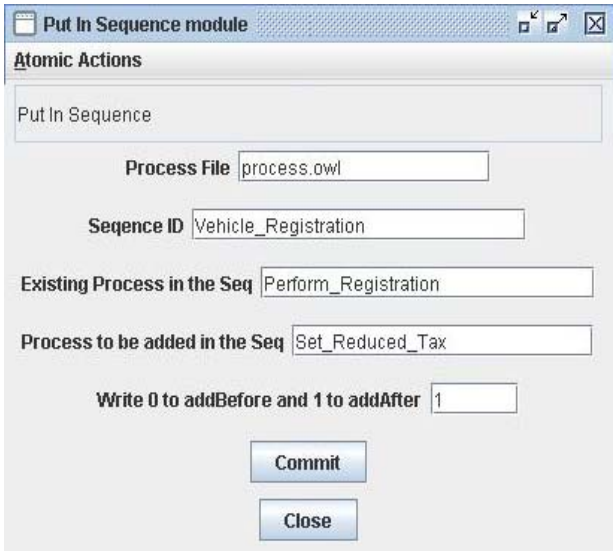

Fig 8. CMS executing the PutInSequence action.

After the changes are performed the new process file looks like:

```
<process:CompositeProcess rdf:ID="Vehicle_Registration">
 <process:composedOf>
  <process:Sequence>
   <process:components rdf:parseType="Collection">
    <process:AtomicProcess rdf:about="#Test_IF_InAllowedList"/>
    <process:AtomicProcess rdf:about="#SetReducedTax"/>
    <process:AtomicProcess rdf:about="#PerformRegistration"/>
   </process:components>
  </process:Sequence>
 </process:composedOf>
</process:CompositeProcess>
```

```
<process:AtomicProcess rdf:ID=" Test_IF_InAllowedList ">
 <process:hasInput rdf:resource="#PersonName_In"/>
 <process:hasInput rdf:resource="#PersonAddress_In"/>
 <process:hasInput rdf:resource="#VehicleModel_In"/>
 <process:hasInput rdf:resource="#ChassisNumber_In"/>
 <process:hasInput rdf:resource="#DateOfPurchase_In"/>
 <process:hasOutput rdf:resource="#VehcileAllowed_Out"/>
</process:AtomicProcess>

<process:Input rdf:ID=" PersonName_In ">
 <process:parameterType rdf:resource="&concepts;#Name"/>
</process:Input>

<process:Input rdf:ID=" PersonAddress_In ">
 <process:parameterType rdf:resource="&concepts;#Address"/>
</process:Input>

<process:Input rdf:ID=" VehicleModel_In ">
<process:parameterType rdf:resource="&concepts#VehicleModel"/>
</process:Input>

<process:Input rdf:ID=" ChassisNumber_In ">
 <process:parameterType rdf:resource="&concepts;#ChassisNum"/>
</process:Input>

<process:Input rdf:ID=" DateOfPurchase_In ">
 <process:parameterType rdf:resource="&concepts;#Date"/>
</process:Input>

<process:UnConditionalEffect rdf:ID=" VehcileAllowed_Out ">
 <process:ceEffect rdf:resource="&concepts;# VehcileAllowed"/>
</process:UnConditionalEffect>

<process:AtomicProcess rdf:ID=" Set_Reduced_Tax ">
 <process:hasInput rdf:resource="#HoldsNoPollutionCertificate_In"/>
 <process:hasOutput rdf:resource="#TaxesReduced_Out"/>
</process:AtomicProcess>

<process:Input rdf:ID=" HoldsNoPollutionCertificate_In ">
 <process:parameterType rdf:resource="&concepts;#HoldsNoPollutionCerti"/>
</process:Input>

<process:UnConditionalEffect rdf:ID=" TaxesReduced_Out ">
 <process:ceEffect rdf:resource="&concepts;#TaxesReduced"/>
</process:UnConditionalEffect>


<process:AtomicProcess rdf:ID=" Perform_Registration ">
 <process:hasInput rdf:resource="#VehicleAllowed_In"/>
 <process:hasOutput rdf:resource="#PayableTax_Out"/>
</process:AtomicProcess>

<process:Input rdf:ID=" VehicleAllowed_In ">
<process:parameterType rdf:resource="&concepts# VehcileAllwd
"/>
</process:Input>

<process:UnConditionalEffect rdf:ID=" PayableTax_Out ">
 <process:ceEffect rdf:resource="&concepts;# PayableTax"/>
</process:UnConditionalEffect>
```

Once the changes are done at the OWL-S level they are further mapped at the execution level of BPEL. In addition, a new Web Service for setting the reduced tax must be available that is invoked by the BPEL process. Once the translation of the process into BPEL is completed the Business process will again be ready and available for execution and hence serving the client requests. In this way, our change management system is able to adapt seamlessly to a change in government policy. Therefore the translation into the BPEL phase is

also an extremely important element of our approach and of the solution environment of managing changes in Business process implementations which we have implemented. As mentioned earlier the details of BPEL translation is out of the scope of this publication as we are currently in the process of developing the OWL-S to BPEL translator. The mapping phase from OWL-S to BPEL will be mostly automatic and the user would be only prompted to enter the details of the Web Services that he/she intends to modify or add. As the flow description at OWL-S level is strong enough to help us map to the final BPEL code therefore this approach has helped us save the overhead of making changes at BPEL level, which would had been larger in number.

7. Conclusion

As it is well known, automatic code generation and modification is a difficult task. As a step forward towards automated change management we presented a methodology for adapting business process implementations. Our methodology heavily relies on the fact that the processes are represented in a declarative, semantically rich modeling language based on ontology – OWL-S which itself has grounding in web services. The approach is also a contribution to the important task to keep align IT with business.

In the future work we will be working on the task of figuring out the quality indicators and creating the automatic invocation system based on that another very important task will be to figure out the optimization that can be made at the BPEL level while mapping the changes of OWL-S to BPEL so that the overhead of regenerating all the BPEL code gets eliminated.

References

[1] Andrews, T. et al.: Business Process Execution Language for Web Services, Version 1.1 ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, 2003

[2] Campbell, Scott; Mohun, Vamsi: Mastering Enterprise SOA with SAP NetWeaver and mySAP ERP, Wiley, 2006.

[3] Chrissis, Mary Beth; Konrad, Mike; Shrum, Sandy: CMMI®: Guidelines for Process Integration and Product Improvement. Addison Wesley Professional 2003

[4] Gaur, Harish; Zirn Markus (ed.): BPEL Cookbook: Best Practices for SOA-Based Integration and Composite Applications Development. Packet Publishing 2006

[5] Hinkelmann, K., Probst, F., Thönssen, B. (2006): Agile Process Management Framework and Methodology. AAAI Spring Symposium on Semantic Web Meets e-Government, Stanford University, March 2006

[6] Karch, Steffen; Heilig, Loren: SAP NetWeaver. Galileo Press, 2004.

[7] Kramer, Jeff and Magee, Jeff: The Evolving Philosophers Problem: Dynamic Change Management IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 16, NO. 11

[8] Leblang, David B. and Levine Paul H.: Software Configuration Management: Why is it needed and what should it do? ICSE SCM-4 and SCM-5 Workshops, on Software Configuration Management 1995.

[9] Martin, David (ed.): OWL-S: Semantic Markup for Web Services, http://www.daml.org/services/owl-s/1.1/overview/, 2004.

[10] van Loon, Han: Process Assessment and ISO/IEC 15504. A reference book. Springer 2004.

The work presented in this paper is part of the project FIT (Fostering self-adaptive e-government service improvement using semantic technologies), funded by the European Commission in the Information Society Technologies program (IST-2004-27090); http://www.fit-project.org