Transforming Functional Requirements from UML into BPEL to Efficiently Develop SOA-based Systems

Anisha Vemulapalli, Nary Subramanian
Department of Computer Science, The University of Texas at Tyler, Texas, USA
avemulapalli@patriots.uttyler.edu, nsubramanian@uttyler.edu

Abstract. The intended behavior of any system such as services, tasks or functions can be captured by functional requirements of the system. As our dependence on online services has grown steadily, the web applications are being developed employing the SOA. BPEL4WS provides a means for expressing functional requirements of an SOA-based system by providing constructs to capture business goals and objectives for the system. In this paper we propose an approach for transforming user-centered requirements captured using UML into a corresponding BPEL specification, where the business processes are captured by means of use-cases from which UML sequence diagrams and activity diagrams are extracted. Subsequently these UML models are mapped to BPEL specifications that capture the essence of the initial business requirements to develop the SOA-based system by employing CASE tools. A student housing system is used as a case study to illustrate this approach and the system is validated using NetBeans.

Key words: Service-Oriented Architecture (SOA), Business Process Execution Language for Web Services (BPEL4WS or BPEL), Unified modeling Language (UML), Visual Paradigm for UML (VP – UML).

1. Introduction

Analyzing the business domain and capturing the user requirements is the primary step to develop a web application. The user requirements can be captured by user-stories or requirements-gathering meetings. Technically, the user requirements are classified into functional requirements and non-functional requirements. The intended behavior of the system which user requires the system to perform is captured by functional requirements. Good software design can be achieved by analysis of functional requirements. UML is a general-purpose modeling language that models real-world objects. Use cases are a means to typically capture functional requirements in UML. Use case diagrams are used to capture the actors, use cases and the relationships between them but, not the flow of control of the business process which is the key aspect of SOA. Sequence diagrams and activity diagrams capture the flow of control in UML.

The web applications developed employing SOA is a combination of BPEL4WS and a collection of loosely coupled web services with each web service accomplishing partial function of the overall system. The interactions between multiple web services can be defined using BPEL4WS or BPEL. BPEL is an XML (Extensible Markup Language)-based description language for web services. The essence of the initial business requirements are captured by mapping the UML models to the BPEL specifications. These BPEL specifications can be used to develop SOA based systems. Figure 1 depicts the process of transforming UML diagrams into BPEL for developing a SOA-based system.

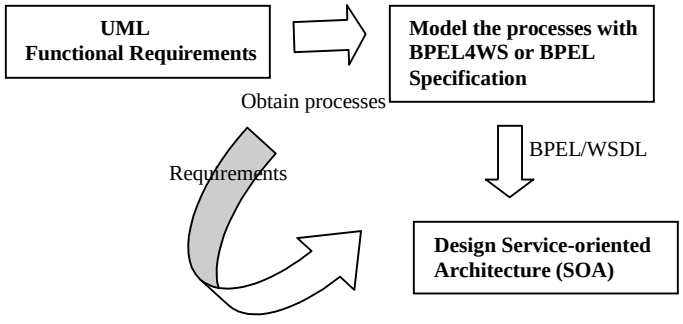

Figure 1: A process of developing SOA-based system using BPEL specification derived from UML Functional Requirements

In this paper we focus our attention on transformation of functional requirements from UML into BPEL. In our approach, the business processes that the software application should satisfy are captured by means of use-cases from which UML sequence diagrams and activity diagrams are extracted. However, the transformation from UML to BPEL is not a direct transformation. The UML model is first exported into an

XMI (XML Metadata Interchange) document which uses XML, and in the second step the BPEL, WSDL and XSD files are generated from the obtained transformation of XMI from UML. Transformation of functional requirements from UML into BPEL must be consistent with the requirements captured from user-stories. We define the mapping rules and the support tools for the transformation. This transformation generates Web Services Definition Language (WSDL) artifacts, XML Schema Definition (XSD) Language artifacts, and a BPEL file.

Related Works

We can find some related works in the literature. Mantell explores a method for translating UML activity diagrams into bpel4ws 1.0 specification, and Li Zhang and Wei Jiang from Beihang University proposes a MDA-based approach to develop work-flow oriented web applications by modeling and analyzing the business requirements, and modeling the transformation of business process into BPEL through a well-defined mapping. In this paper we focus on transforming the functional requirements captured in UML into BPEL specifications which allow us to efficiently develop SOA-based systems. Mantell proposes a methodology using Rational Rose software from IBM to transform UML to BPEL. In this paper, we propose a more convenient and different approach in which, we make use of VP – UML for constructing UML diagrams and transforming the diagrams to an XMI document, and later, we make use of NetBeans to use the created XMI document and generate the BPEL code. The main contribution of this paper is to facilitate a transformation approach, starting from capturing the functional requirements of the system being developed; for that purpose UML diagrams are very useful, as they capture the behavior of the system and the communication between objects in a more appropriate way. Then, we see how we can translate those diagrams into BPEL specifications.

Overview:

The paper is structured as follows. In the next section of this paper, why UML diagrams are used is introduced in more detail. Section 3 introduces BPEL and business processes in more detail. Section 4 proposes an approach to transform the captured functional requirements from UML models into BPEL specifications. Section 5 elaborates the transformation by presenting a case study of a web application - student housing system including the validation of mapping rules and transformation tools. Finally, the conclusions and future work are presented in Section 6.

2. Why UML Diagrams

UML is an object-oriented analysis and design language for visualizing and constructing the artifacts and developing business models of a system from Object Management Group (OMG). UML uses graphical notations to describe the system providing abstraction by suppressing the lower level details. Graphical models are easy to understand, and if necessary re-implementation of design can be done with a minimum knowledge. There are twelve diagrams in UML divided into three categories. Structural diagrams capture the elements of a system. Interaction diagrams capture the interactions among objects; these diagrams are subset of behavior diagrams. Behavioral diagrams capture the behavioral features of a system.

The architecture of the application is driven by functional requirements. Use cases have become a traditional way to capture the functional requirements providing interactions between actors and, the system being developed. UML Use case diagrams give the overview of the system being developed, show the actors and use cases in the system. The instance of a use case is called a scenario. Decomposing the system into small scenarios increases the readability of the system by the user. These scenarios can be captured by constructing sequence diagrams which show the behavior of a use case and the sequence of communication between objects. Also, use case models provide a systematic way for developing activity diagrams which describe the overall flow of control of the software being developed making use of the information sources like use cases and actors from which activities are derived.

3. BPEL4WS or BPEL

The BPEL4WS or BPEL is a XML-based language which uses a combination of web services to enable task-sharing. BPEL is originated from WSFL (Web Services Flow Language) and XLANG (An extension of WSDL-Web Services Description Language) supporting a combination of graph oriented processes and structural constructs for processes, and hence can be used to describe the interfaces for business processes. Process oriented approach is very important to SOA which can be achieved by combining BPEL4WS with web services. BPEL interacts with its partners to provide for business process behavior based in web services. WSDL defines service model on top of which the BPEL model is layered which is defined by BPEL. Figure 2 describes the relation mapping between the BPEL process and the WSDL. The interactions are modeled as a flow consisting of sequence of activities - beginning, a defined behavior and an end. BPEL contains the following components.

BPEL designer. This includes a graphical user interface to define a business process without giving any technical details.

Process flow template. The flow logic of the business process is captured by process flow template which is generated from BPEL designer at design time.

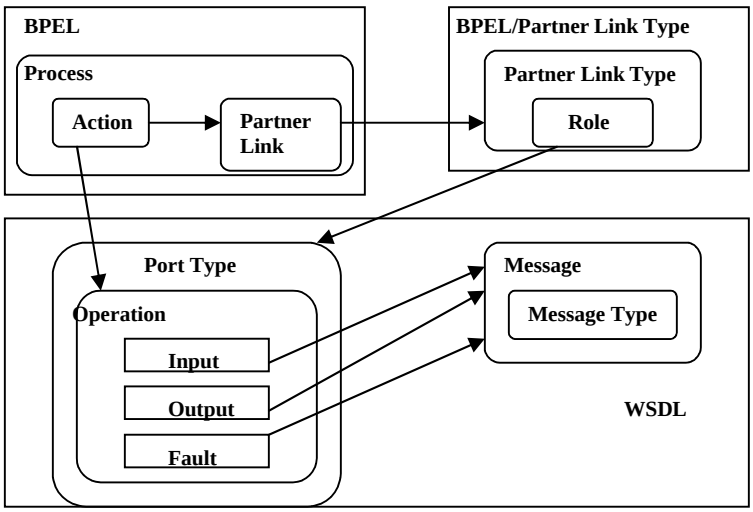

Figure 2: Relation mapping between the BPEL process and the WSDL

BPEL Engine. The generated process flow template is executed to make it compatible to BPEL standard. BPEL engine handles errors, maps the data content and invokes the web services.

To capture the process information BPEL provides the following elements:

Partner Link Types. The relationship between two services and their interaction is represented by the Partner Link Types defining the roles played by each of the services.

Process. The process section includes variables, partner Link types, fault handlers and event handlers. Two types of business processes are supported by BPEL, the executable processes and the abstract processes business protocols). The actual behavior of a participant in a business interaction is modeled by the execution processes.

Activities. BPEL defines two types of activities, basic activities and the structured activities. The basic activities include invoke, assign, receive and reply. Invoke activity calls the web services; it requires input and output variables. The partner link type, port type and the operation are specified by the receive activity. The response to the receive activity is sent by reply activity. The data to the variables is copied by assign

activity for further processing. The basic activities are composed by structured activities. Sequence, switch, while, flow and pick are the activities that are included in structured activities.

4. Modeling Approach and Validation Tools

Initially, the functional requirements of the system are captured from user stories. These requirements are graphically represented with the UML use case diagrams which can be generated from UML-based modeling tools by identifying the use cases and actors. Use cases represent the business processes in BPEL. Hence the web services and the user of the BPEL processes are modeled as actors as they interact with the use cases. Visual Paradigm for UML 7.0 Enterprise Edition is one of the UML-based modeling tools used to generate the UML diagrams. Use cases define a collection of scenarios. The UML activity diagrams capture the actual activity that a particular scenario does without including any technical details. In the first step, activities are derived from use cases and later the order of actions is identified. Then the required classes' entity classes, boundary classes and the control classes are identified for the system. Using these classes, the sequence of actions that the system should perform for a scenario is captured graphically from UML sequence diagrams. The captured sequence diagrams and activity diagrams are then transformed to XMI document with the VP – UML by exporting the UML Diagrams to XMI document, as XMI is generally used to enable meta-model exchanges. The XMI document is then transformed into BPEL executable code by importing the XMI file and generating the BPEL using the CASE tool NetBeans.

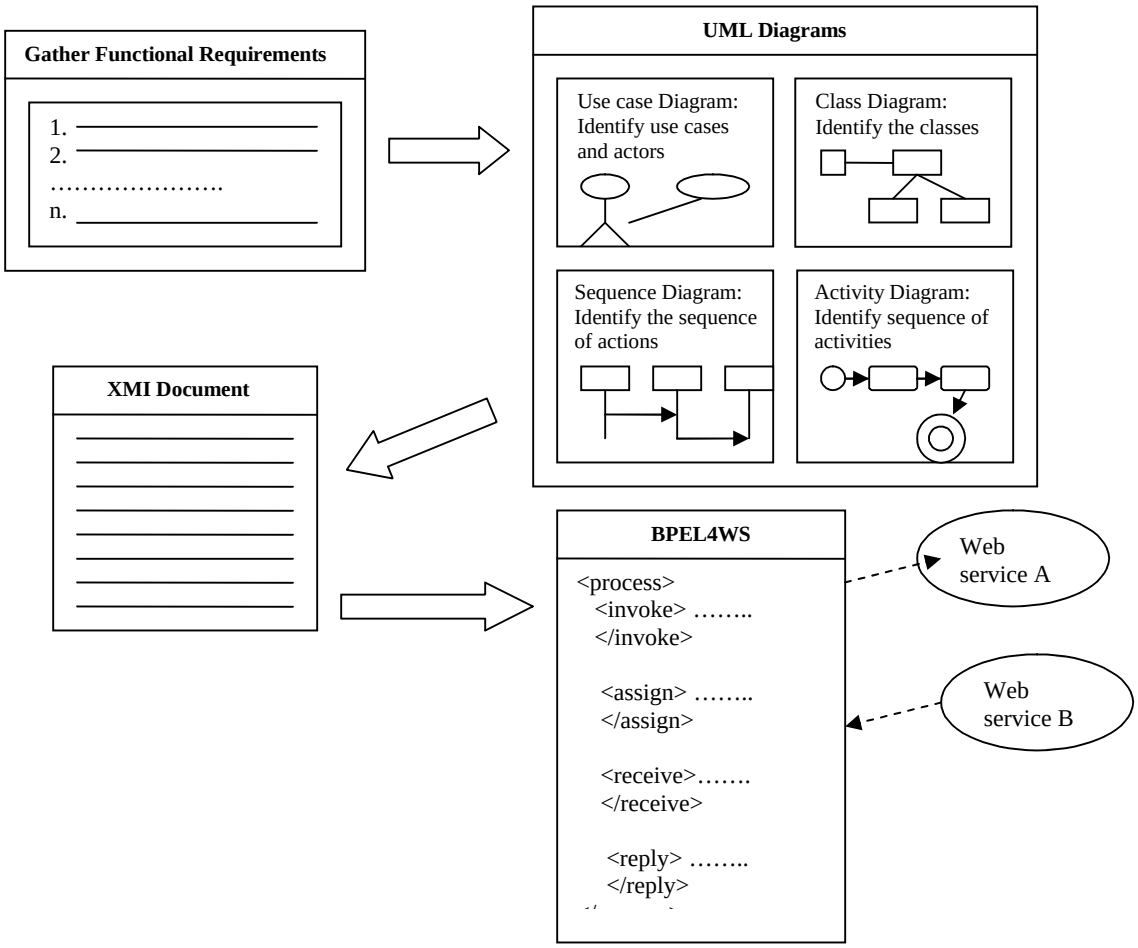

Figure 3: The methodology for transforming UML Functional Requirements into BPEL Specification

The transformation includes WSDL artifacts, XSD artifacts and a BPEL file. NetBeans enables the user to import the XMI file from the system and generate the corresponding BPEL code accordingly. Figure 3 describes the methodology for transforming UML Functional Requirements into a BPEL Specification.

VP – UML is an environment where UML diagrams can be developed. But, the UML modeling tool must be integrated with the NetBeans in order to generate the BPEL specification. Once the integration is successful, XMI documents can be generated from the VP – UML and can be imported into NetBeans to generate WSDL artifacts, XSD artifacts and BPEL file. The mapping between the UML components ant the BPEL transformations is listed in the section below.

4.1. Mapping UML Components to BPEL Transformations

The transformation mechanism from UML into BPEL can be described in four steps as follows:

Step 1: Gather the functional requirements.
Step 2: Transform the functional requirements into UML Diagrams
Step 3: Export the UML diagrams into a XMI (XML Metadata Interchange) document.
Step 4: Transform the XMI document into BPEL file.

During the transformation from UML to BPEL the UML components are transformed to executable code in BPEL. Figure 4 shows the corresponding mapping between UML components and BPEL transformations.

UML Component	BPEL Transformation
Actor	Partners represent the participants in the interaction.
Class and objects	Business process
Attribute	Variables include variable name, message type and element.
Activity Graph	Activity represents the basic behavior of a web service.
Messages	Includes partner link type, role, port type and message (receive or reply or request)

Figure 4: Mapping of UML components to BPEL transformations

5. Student Housing System Case Study

Student Housing System, a common web application is considered as a case study for illustration of how functional requirements are transformed from UML diagrams into BPEL executable code for business analysts. The transformation mechanism of BPEL executable code from the UML diagrams is illustrated with the case study in a sep-by-step manner.

Step-1: In the first step, the system that is being developed is described in a single paragraph and the functional requirements of what the system has to do are noted. Later, the use cases and actors are identified from the list of the functional requirements to figure the business model which is usually captured by the UML use case diagram.

System Description:

The Student first goes to the website and applies for the lease using his student email and password. On receiving the lease application, the student identification number (SID) is verified with the list of Student id's in the Blacklisted Students by the staff member. If the SID is not found in the blacklist then the availability of the room according to the selected floor plan in the lease application is verified. If the selected floor plan is available, the student is sent an approval letter mentioning the amount of deposit he needs to send for hold of the room. The student has to sign the approval letter and send the deposit amount to the lease office and can occupy the room according to his convenience.

Functional Requirements:

1. To login to the system by the student or the staff member or the administrator.
2. To apply for lease by the student.
3. To verify whether the student is among the blacklisted students.
4. To estimate the available rooms according to the requested floor plan.
5. To maintain the student records and room records.
6. To update the records when the tenant pays the monthly rent.
7. To keep a record of the complaints given by tenants.

Step 2: From the functional requirements a business model is developed to understand what the system has to do. According to the system description the system has 'staff member', 'administrator', and 'student' as the actors. Once the student signs the lease he becomes a 'tenant', hence 'tenant' is yet another actor that is to be included in the system. The actor 'tenant' is included just for understanding purpose. In later stages of this paper, 'tenant' is not taken into account.

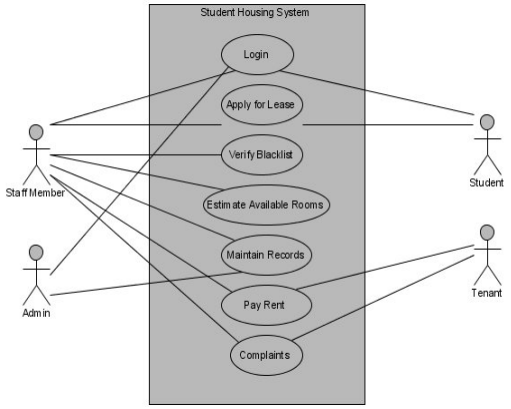

Figure 5: Use case Diagram generated from VP – UML

From the functional requirements of the system, 'login', 'apply for lease', 'verify blacklist', 'estimate available rooms', 'pay rent', 'complaints' and 'maintain records' are identified as the use cases for the system. The maintain records use case includes maintaining both the 'student records' and the 'room records'. Figure 5 depicts the generation of final UML use case model using VP – UML with four actors and seven use cases.

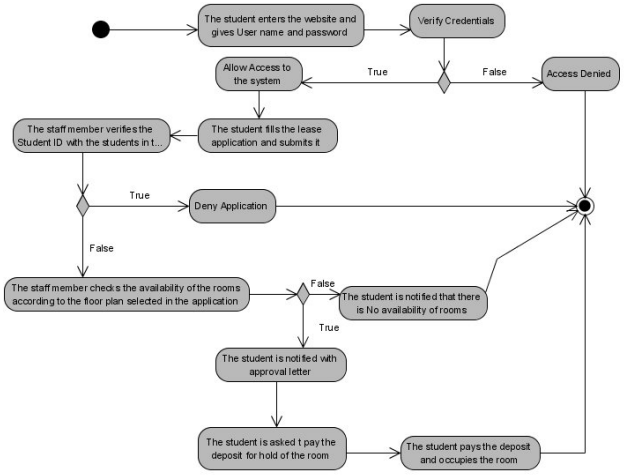

Figure 6: Activity Diagram in VP – UML for Lease approval process

After capturing the activity diagram, the classes are identified for the scenario from the use cases. A 'Records' class is identified as a boundary class, modeling the control behavior of 'Records' use case with 'create student', 'modify student', 'delete student', 'create room', 'update room', 'delete room', 'create staff', 'modify staff', and 'delete staff' as methods. A 'userinterface' is identified as a boundary class modeling the interaction between system's surroundings and its inner workings. The 'staffmember' is the entity classes which exchange the information with the 'records' class, and the 'student' class is yet another entity class interacting with the system. The sequence of actions in a scenario is captured by means of a sequence diagram. The identified classes are used to generate the sequence diagram. Figure 7 depicts the sequence diagram for the Lease approval scenario.

In the first step, the 'student' fills the application and clicks the submit button. In the second step, the 'staffmember' checks if there are any new applications. In the third step, 'verifyBlacklist' method is called to check if the 'student' is in blacklist. In the fourth step, if the 'student' is found in blacklist the 'approvalStatus' is sent as 'denial 'to the student. In the fifth step, if the 'student' is not found in the blacklist, the message is displayed to the 'staffmember'. In the sixth step, the 'staffmember' checks for the availability of the floor plan through 'chckAvailability' method. If the room is available a message is displayed to the 'staffmember' in seventh step. Later an 'approvalStatus' message is sent as 'accept' to the 'userinterface' by the 'staffmember' in the eighth step. In the ninth step, 'student' verifies his application status.

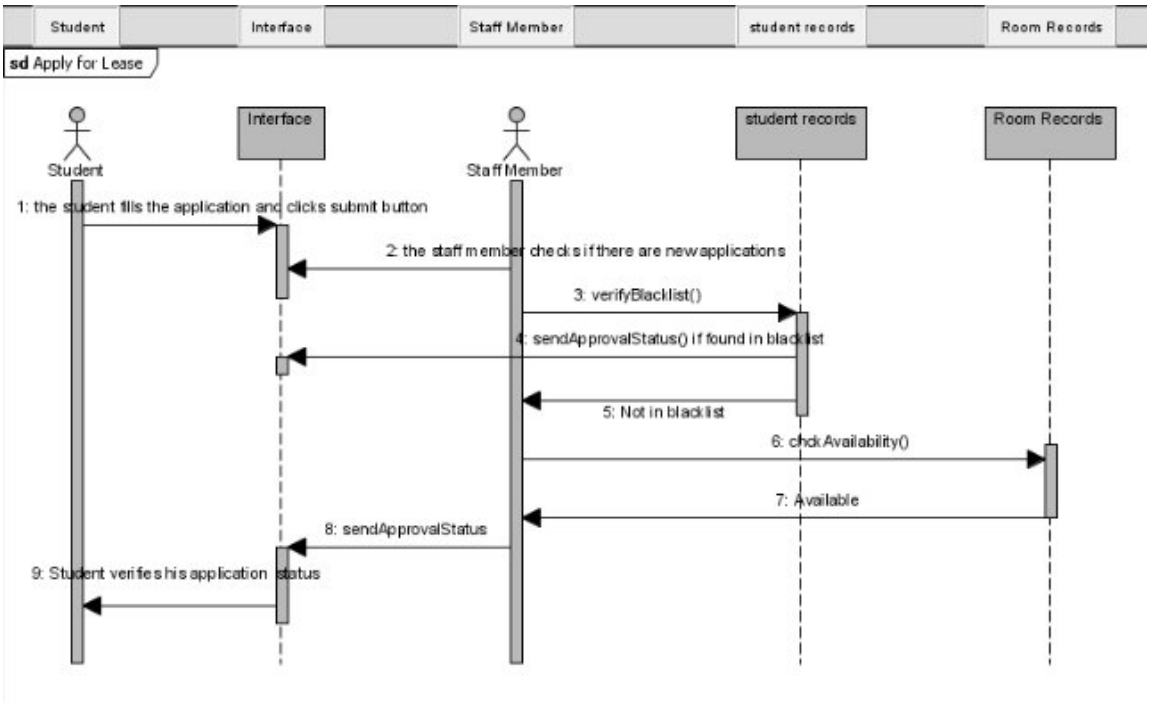

Figure 7: Sequence Diagram in VP – UML for Lease approval business process

Step 3: The generated UML diagrams are exported to XMI to NetBeans from VP – UML. VP – UML provides a direct means to export the generated UML diagrams into an XMI document by selecting *File->Export->XMI…* from main menu. An *Export XMI dialog box* is displayed on the screen. We have to locate the XMI file and click OK to export the project to the specified path in the dialog box. The UML diagrams and models are exported as an XMI file from the VP – UML project upon finishing. XMI does not include any visual information; its main purpose is to enable meta-model exchanges among the UML based modeling tools. Figure 8 shows how the UML diagrams can be exported to an XMI document in VP – UML.

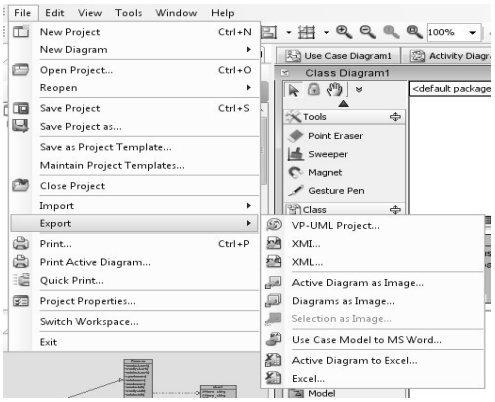

Figure 8: Exporting UML diagrams into XMI document in VP – UML

Step 4: The obtained XMI document is then transformed into BPEL file using NetBeans. Once the NetBeans is integrated with VP – UML the XMI document generated from the step 3 can be imported directly into the NetBeans environment which is later transformed into BPEL executable code as shown in Figure 9. This can be done by selecting *File->SDE EE-NB Import->XMI…* from the main menu in NetBeans. An *Import XMI dialog box* is displayed on the screen. We have to locate the XMI file and click OK to import the XMI file into the NetBeans. The imported XMI file can be found along with the list in the projects window. On right clicking on the imported XMI file, *Generate BPEL* can be found. Upon clicking generate BPEL in the window, the BPEL generation dialog is displayed and upon completion, BPEL file can be found in the projects window.

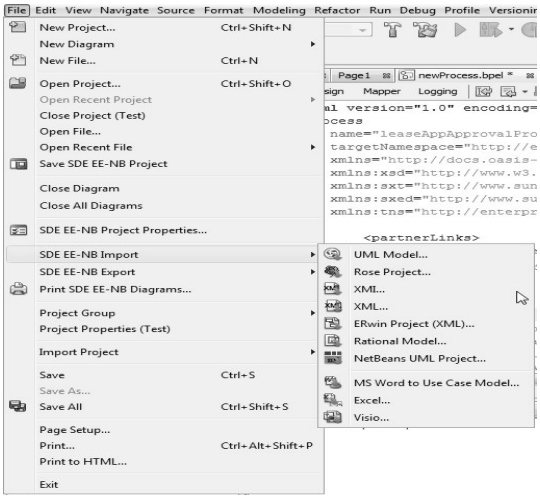

Figure 9: Transformation of XMI document into BPEL executable Code

The files are usually displayed in the Design view in NetBeans. To view the BPEL specification we need to switch to Source view from Design view. The BPEL specification generated from NetBeans is shown in Figure 10. As we have discussed earlier, the use cases in UML are depicted by the business processes in BPEL. From the BPEL specification, we can observe that the actors ‘student’, ‘staffmember’, and ‘Admin’ as the partner links ‘student’, ‘staffmember’ and ‘Admin’ accordingly. Initially, the ‘student’ applies for the lease by submitting the lease application online. This can be observed as an “applicationsubmissionmessage” in the BPEL specification. Then the student id is verified with the ‘blacklisted students’ using a “verifyblacklist” variable. If the ‘student’ is not in the blacklist, the requested room is checked for availability with a “chckRoomAvailability” message. If the requested floor plan is free, the ‘staffmember’ sends the ‘approval information’ as a lease approval to the student.

```
<?xml version="1.0" encoding="UTF-8"?>
<process
   name="leaseAppApprovalProcess"
   targetNamespace="http://enterprise.netbeans.org/bpel/StudentHousingSystem/SHS"
         <partnerLinks>
            <partnerLink name="student" partnerLinkType="appL:applyForLease"
        partnerRole="applyLeaseRole" myRole="applyLeaseCallRole">
             </partnerLink>
            <partnerLink name="staffmember" partnerLinkType="aprL:approveLease"
           partnerRole="approveLeaseRole" myRole="approveLeaseCallRole">

           </partnerLink>
           <partnerLink name="Admin" partnerLinkType="mrec:maintainRecords"
           partnerRole="maintainRecordsRole" myRole="maintainRecCallRole">
              </partnerLink>
      </partnerLinks>
<variables>
   <variable name="request"
   messageType="appL:applicationSubmissionMessage"/>
   <variable name="verifyBlacklist"
   messageType="aprL:blacklistVerificationMessage"/>
   <variable name="chckRoomAvailability"
   messageType="aprL:checkRoomAvailabilityMessage"/>
</variables>
<flow>
   <receive name="receive" partnerLink="student"
   operation="approve" variable="request" createInstance="yes" portType="appL:leaseApprovalPT">
      <source linkName="receive-to-response"
      transitionCondition="bpws:getVariableData('request', 'studentId')='studentId'"/>
      <source linkName="receive-to-approve"
      transitionCondition="bpws:getVariableData('request', 'studentID') !='studentId'"/>
   </receive>
   <invoke name="invokeStaffMember" partnerLink="staffMember"
   portType="aprL:blacklistVerifiationPT" operation="verify" inputVariable="request"
   outputVariable="blacklistVerification">
      <target linkName="receive-to-response"/>
      <source linkName="response-to-setMessage"
        transitionCondition="bpws:getVariableData('blacklistVerification', 'blacklistId')='studentId'"/>
        <source linkName="response-to-approve"
        transitionCondition="bpws:getVariableData('blacklistVerification', 'blacklistId')!='studentId'"/>
   </invoke>
   <invoke name="invokeStaffMember" partnerLink="staffMember"
   portType="aprL:checkAvailabiltyPT" operation="verify" inputVariable="request"
   outputVariable="RoomAvailability">
      <target linkName="receive-to-response"/>
      <source linkName="response-to-setMessage"
        transitionCondition="bpws:getVariableData('roomAvailability', 'occupancy')!='free'"/>
        <source linkName="response-to-approve"
        transitionCondition="bpws:getVariableData('roomAvailability', 'ocupancy')!='free'"/>
   </invoke>
   <assign name="assign">
   <target linkName="response-to-setMessage"/>
   <source linkName="setMessage-to-reply"/>
   <copy>
      <from expression="'yes'"/>
      <to variable="approvalInfo" part="accept"/>
   </copy>
   </assign>
   <reply name="reply" partnerLink="Student" portType="aprL:leaseApprovalPT"
   operation="approve" variable="approvalInfo">
      <target linkName="setMessage-to-reply"/>
      <target linkName="approve-to-reply"/>    </reply>
</flow>
</process>
```

Figure 10: BPEL specification for the Lease approval process

6. Conclusion

In this paper, we propose an approach for transforming the functional requirements captured from UML diagrams into BPEL specifications to develop SOA-based systems. In our approach, we develop business requirements models and, using the VP – UML and NetBeans initially the UML diagrams are exported into an XMI document and later the generated XMI document is transformed to build BPEL file. The approach is validated using a student housing system. Our future work will focus on the analysis of the business process models considering the non-functional requirements – security, adaptability, reliability, performance and usability and, these non-functional requirements are validated using the NFR-Framework [11]. We also plan to evaluate the consistency of business requirements captured by user-centered techniques with BPEL descriptions in future.

7. Acknowledgements

The author wishes to thank the anonymous reviewers of this paper for their detailed and thorough comments which helped us significantly improve this paper.

References:

1. Mantell, K.: From UML to BPEL, https://www.ibm.com/developerworks/webservices/library/ws-uml2bpel/
2. Li Zhang, Wei Jiang.: Transforming Business Requirements into BPEL: a MDA-Based Approach to Web Application Development. In: IEEE International Workshop on Semantic Computing and systems, pp.61–66. (2008)
3. Cambronero, M.E., Diaz, G., Pardo, J.J., Valero, V.: In: Second International Conference on Internet and Web Applications and Services, pp. 24–24. (2007)
4. Korherr, B., List, B.: Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL. In: 2nd International Workshop on best practices of UML (BP-UML '06). LNCS, vol. 4231, pp. 7–18. Springer, Heidelberg (2006)
5. Kurahata, H., Fuji, T., Miyamoto, T., Kumagai, S.: A UML Simulator for Behavioral Validation of Systems Based on SOA. In: International conference on Next Generation Web Services Practices, pp. 3–10. (2006)
6. Business processes in a Web services world, http://www.ibm.com/developerworks/webservices/library/ws-bpelwp/#code1
7. Web Services Business Process Execution Language Version 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html
8. Transformation to SOA: Part 4, How Web services transform from UML to BPEL, http://www.ibm.com/developerworks/rational/library/08/0318_pattathe/index.html
9. Transformation to SOA: Part 3, UML to SOA, http://www.ibm.com/developerworks/rational/library/08/0115_gorelik/
10. Mathew, B., Juric, M., Sarang, P.: Business Process Execution Language for Web Services 2nd Edition. PACKT (2006)
11. Chung, L., Nixon, A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic, Boston (2000)