Transformation of Business Process Models BPMN 2.0 into Components of the Java Business Platform

Narayan Debnath  
Department of Computer Science  
Winona State University  
Winona MN 55987 – USA  
debnath@winona.edu

Carlos Alejandro Martinez  
Departamento de Ingeniería en Sistemas  
Universidad Tecnológica Nacional-FRM  
Mendoza, Argentina  
carlos.martinez@frm.utn.edu.ar

Fabio Zorzán  
Departamento de Computación  
Universidad Nacional de Río Cuarto  
Córdoba, Argentina  
fzorzan@dc.exa.unrc.edu.ar

Daniel Riesco  
Departamento de Informática  
Facultad de Ciencias Físico Matemáticas y Naturales  
Universidad Nacional de San Luis  
Ejército de los Andes 950 – C.P. 5700 – San Luis Argentina  
driesco@uns.edu.ar

German Montejano  
Departamento de Informática  
Facultad de Ciencias Físico Matemáticas y Naturales  
Universidad Nacional de San Luis  
Ejército de los Andes 950 – C.P. 5700 – San Luis Argentina  
gmonte@uns.edu.ar

Abstract—The continuous improvement of organizations involves redesigning, improving or introducing new Business Processes effectively and efficiently while maintaining integration with the computer systems of the organization. In this article, we propose the creation of components of the Java EE 6 business platform from technical business processes modeled with Business Process Model Notation (BPMN) 2.0. This generation was achieved by performing three transformations in the context of Directed Model Architecture (MDA). First, a technical model BPMN 2.0 was transformed into an UML class model. Then the Unified Modeling Language (UML) model, was transformed into a UML model with Java Platform, Enterprise Edition (Java EE) profiles. Finally the last generated model was transformed through Meta Object Facility Script (MOFScript), into Java EE components. The transformations are performed with Query View Transformations (QVT) Relations and MOFScript. This work contributes with transformations for generating Java EE business components related to business processes, so it helps improving the development productivity and reducing design errors.

Keywords—MDA, BPMN, UML, QVT, Java EE, Ejb3.1, JPA, MOFScript

I. INTRODUCTION

From the continuous improvement of organizations comes the need for redesigning, improving and creating process taking into account their development and implementation through information technology. A common problem of modeling business processes is the challenge of achieving consistency with the software systems. One way to achieve this consistency is using a design of processes following a structure with different levels of abstraction ranging from modeling processes at a descriptive level to a technical level.

In this work we base on the BPMN-Framework [1] for the technical modeling in BPMN 2.0.

For the automation of business processes specified at technical or executable level [2] an fundamental alternative arises as the Directed Model Architecture (MDA) and its related standards Meta Object Facility (MOF) [3], QVT [4], Object Constraint Language (OCL) [5] and XML Metadata Interchange (XMI) [6]. MDA is a framework for software development defined by the Object Management Group (OMG) [7]. The model transformation is an essential part of MDA. It is the automatic generation of an input model to an objective model following transformation rules. Among the general purpose languages for model transformation, the one we use in this paper is the standard specified by the OMG QVT with Relations language.

The UML modeling language [8] is a general purpose language for modeling systems. A UML profile allows extending its syntax and semantics to adapt the language to different domains and platforms.

The Business Process Model Notation (BPMN) v.2.0 [9] is an OMG standard which aims at providing a notation easy to understand and communicate with users ranging from business analysts to software engineers. It is widely adopted by companies for purposes of modeling the basic processes, detailed analysis of performance, requirements specification and executable design. The BPMN 2.0 adds new elements of performance-oriented design, providing a theoretical definition of what makes a model be executable [10]. The introduction of a MOF metamodel enables the exchange, interoperability and execution of models.

The Java EE business components consist of the Enterprise Beans and Java Persistence API (JPA) [11]. Enterprise JavaBeans (EJB) 3.1 [12] is a standard defined by Java Community Process implemented by different manufacturers;
it is a server-side component that represents the business platform in a platform-based architecture of an Enterprise application in Java EE technology.

EJB 3.1 offers great advantages for the development of distributed applications and Enterprise consisting basically of a web platform, business platform and data. EJBs are displayed on application servers (Java EE servers) providing advantages to applications such as security, load balancing, transactions and clustering. For server components to be accessible and manageable, they are put under the control of an application server [13].

JPA is a framework based on Plain Old Java Object (POJO) [14] which allows the object-relational mapping in Java applications. JPA is widely used in conjunction with EJB through tools like hibernate and toplink. JPA is a specification that is part of Java EE is defined by the Java Community Process.

This paper introduces a contribution aimed at achieving the goal of maintaining consistency between the models of technical processes and Java EE business components, so as to allow improving the efficiency and effectiveness of the organization in implementing all or part of automated processes. This makes it possible to improve development productivity and reduce design errors.

To achieve this objective, 3 transformations are defined under the Directed Model Architecture (MDA): First a technical model BPMN 2 is transformed into a UML class diagram (platform-independent model). Then, the UML diagram is transformed into a UML diagram with stereotypes of EJB 3.1-2 JPA profile (platform-dependent model). Finally, the latter model is transformed into a Java source code that constitutes the elements of the Java EE business platform. The first two transformations are performed with Relations language that is part of Query/Views/Transformations and the latter with MOFScript.

This article is structured as follows: Section 2 analyzes related works. Section 3 discusses the metamodels and profiles used. Section 4 explains the transformations. Section 5, 6 and 7 detail the transformation rules with an example. And finally, Section 8 presents conclusions.

II. RELATED WORK

Related works focus on the direct transformation of BPMN 1.0 elements into object-oriented diagrams of the domain as class diagrams and use cases.

Paper [15] proposes a set of transformations from a BPMN 1.0 diagram prepared by a business analyst into a UML activity diagram, and from the activity diagram into use cases and UML class of analysis. Our work improves this by creating CIM-PIM transformations that allow the generation of UML class models directly from a BPMN 2.0 diagram. Transformations PIM-PSM-Code and profiles are added that allow users to modify the UML class model into diagrams and java source code with greater detail.

Meanwhile Paper [16] presents a method for transforming the Activity Diagrams into a class diagram. This work provides a way to associate classes taking into account the input-output flow of process diagram activities. Then, refinements are added to obtain a class diagram with detailed relationships. They do not specify the relationship of the transformations through a language. The differences with our work are similar to the previous work with the addition that we create our transformations through QVT-Relations.

Paper [17] has the objective to transform a BPMN model into a SoaML services model using QVT language. Our proposal follows different objectives but has generation of service points and their relationship with participants in common with it. Our proposal states its improvement by direct transformation of BPMN 2.0 Service Task artifacts into service points provided by Java interfaces.

Paper [18] identifies the use cases from business processes. For this it uses the concept of "Step" [19]. Step consists of following a series of steps to detect the set of tasks that can be performed by an actor without interruption. While the objectives of that paper are very different from ours, the concept of Step is used later in this article.

In general the fundamental differences with what is proposed in this article are: first, different purposes are followed and, second, these proposals do not benefit from the new features provided by the new BPMN 2.0 standard for the specification of executable technical models.

III. METAMODELS

MOF [3] is the language proposed by the OMG to create metamodels. The metamodels presented in this article are made using Eclipse Modeling Framework (EMF). EMF is an implementation of a central subset of the API of MOF [20]. The equivalent of MOF in EMF is called Ecore.

A. BPMN 2.0 metamodel

The BPMN 2.0 specification presents a MOF(Ecore)-based metamodel which defines all entities with their attributes and relationships. This facilitates the exchange, interoperability and execution of models. This article uses a metamodel according to the BPMN 2.0 specification prepared for the Business Process Management Systems (BPMS) Bonita Open Solution [21].

B. UML 2 metamodel

For UML 2 metamodel, an implementation MOF (Ecore) for Eclipse is used based on the OMG specification [8]. The UML 2 metamodel can be obtained from package org.eclipse.uml2.uml_3.2.100.v201108110105/model integrated into the SDK of Model Development Tools (MDT) Project [22].

C. Java EE Profiles

The official metamodel and UML Profile of the OMG is for EJB specification version 1. The EJB technology radically changes from version 1 to 3. Then it is necessary to use a
profile according to specification 3.1 due to the advantages it offers. As part of this article, a profile for EJB 3.1-Platform Web Services (JAX-WS) is developed based on information provided by Java EE 6 Tutorial [23]. Figure 1 shows part of the EJB 3.1-Web Services (JAX-WS) profile.

![Fig. 1. Ejb 3.1-Web Services Profile](image)

For the JPA profile, we base on the profile proposed by genesz.org [24]. Figure 2 shows part of the JPA profile.

![Fig. 2. JPA Profile](image)

II. TRANSFORMATIONS

Model transformations using QVT-Relations standard were made in the context of the MOF metamodel architecture. Figure 3 shows the transformation proposed. The use of MOF standard and the metamodel concept is essential in MDA. MOF describes an architecture based on four levels of abstraction called M0, M1, M2 and M3.

![Fig. 3. Transformation proposed in the context of the OMG architecture supporting MOF.](image)

The BPMN 2.0 and UML 2 metamodels are represented as instances of the MOF metamodel. The UML 2-Profile EJB 3.1-JPA 2 metamodel refers to the fact that the UML metamodel has been extended with a profile-Ejb3.1 JPA 2 to have own concepts of the implementation platform that will be used.

The first transformation (BPMN 2.0 to UML 2) is form a computer independent model (CIM) into a platform independent model (PIM). It is defined in M2 level having BPMN 2.0 as MOF source metamodel and UML 2 as MOF target metamodel.

The second transformation (UML 2 into UML 2-Profile EJB 3.1- JPA 2) is from a platform independent model (PIM) to a platform specific model (PSM). It is defined in M2 level having UML 2 as MOF source metamodel and UML 2- Profile EJB 3.1-JPA 2 as the MOF target metamodel.

To make the transformations the tool Medini QVT [25] is used, which is an integrated tool in the Eclipse development environment that allows the execution of QVT transformations expressed with Relations language textual syntax.

Taking the metamodel Jee (EJB 3.1-JPA2) and the instance of this model generated from the transformation explained above as inputs, the tool MOFScript [26] is applied to generate the application code. The tool MOFScript is an implementation of a model transformation language to MOFScript text.

II. DEFINITION OF TRANSFORMATIONS WITH QVT

The following subsections describe the QVT transformations and MOFScript of some of the relationships developed.

A. Transformation from BPMN 2 to UML 2

These first three relationships transform a BPMN model with the TDefinitions element (containing all the elements of BPMN) into a UML2 model consisting of packages Business and Entities. In the Business package classes are created that will then become SessionBeans. In the Entities package classes
are created that will then become business entities (Entity). This is represented in Figure 4.

```
transformation bpmn2uml(bpmn2:model, uml2 :uml) {
  top relation ModeloBpmnToPackageUml {
    pn : String;
  checkonly domain bpmn2 b :model::TDefinitions {
    name = pn;
  }
  enforce domain uml2 u :uml::Model {
    packagedElement = pack :
      uml::PackageableElement {};
    name = pn;
  }
  where {
    ModeloBpmnToPackageBusinessUml(b,pack);
    ModeloBpmnToPackageEntitiesUml(b,pack1);
  }
  }
relation ModeloBpmnToPackageBusinessUml {
  pn : String;
  checkonly domain bpmn2 b :model::TDefinitions {
    name = pn;
  }
  enforce domain uml2 p : uml::Package{
    name = 'Business' ;
  where {
    ProcessToClass(p,b);
    LaneToClassS(p,b);
  }
  }
relation ModeloBpmnToPackageEntitiesUml {
  pn : String;
  checkonly domain bpmn2 b :model::TDefinitions {
    name = pn;
  }
  enforce domain uml2 pack:uml::Package{
    name = 'Entities';
  where {
    ParticipantToClass(b,pack);
    LaneToClassE(b,pack);
    DataStoreToClass(b,pack);
  }
  }]
```

Fig.4. Main Transformation from BPMN 2.0 into UML 2

Once we have the basic structure of the target model we need to use a criterion for creating classes that will then represent the SessionBeans in the second transformation. One way is to obtain for each case of system use, one SessionBean. For this we use the concept Step that consists of defining a use case for each task sequence without interruption of an actor [19].

The clause “where” of ModeloBpmnToPackageBusinessUml relationship contains ProcessToClass and LaneToClass relations. These produce a SessionBean Class per process if there are no Lane elements in it. In the case that the process contains a TLaneSet element composed of one or more TLane then the LaneToClassS relationship generates a Class for each TLane object.

The ServiceTasks become a method in the class defined from a Lane.

**B. Transformation from UML2 to UML2 with EJB3.1 -JPA2 profile.**

From the model obtained in the previous item, the developer can add associations, attributes and operations from a tool that allows working on UML2. Then the second transformation can be applied on this model that acts as the source and results in a UML 2 model with stereotypes dependent on Java EE platform.

This transformation takes classes of the Business package and transforms them into UML classes with stereotype "<<Stateless>>". For each attribute, it generates the set and get methods. It also generates remote interfaces with their respective methods and stereotypes for each class. Figure 5 shows the main ClassToSession relationship. Figure 6 describes the AnnotatedClasses and ClassToInterfaceRemote relationships, which are called in the clause “where” of the previous relationship.

```
relation ClassToSession {
  className : String;
  st:ejb3::Session;
  checkonly domain source src : uml::Package {
    packageElement = srcCls:Class {
      name = className 
    };
  }
  enforce domain target dst : uml::Package {
    packageElement = dstCls : Class {
      name = className+SessionBean,
    }
    ownedOperation = setCls : Operation {
      name = 'set' + name ,
    }
    ownedOperation = getCls : Operation {
      name = 'get' + name ,
    }
    interfaceRealization = int :
      uml::Class::InterfaceRealization {
      
    };
  }
  where {
    [src.name = 'Business';]
  }
relation AnnotatedClasses {
  className : String;
  checkonly domain source src : uml::Class {
    baseClass = src : uml::Class {
    }
  }
  enforce domain target dst :ejb3::Session {
    [baseClass = src : uml::Class {
    }
  }
  }
relation ClassToInterfaceRemote{
  st:ejb3::Remote;
  checkonly domain source src : uml::Class {
  }
  enforce domain target dst : uml::Package {
    packageElement = int : uml::PackageableElement::Interface {
      name = 'Interface Remote'+src.name 
    };
  }
  where {
    [AnnotatedInterface (int, st)];
  }
```

Fig.5. Transformation from Class into Session Bean

Fig.6. Transformation of AnnotatedClasses and ClassToInterfaceRemote

Entities package classes are transformed into classes with the notation "<< Entity >>". The associations between classes are added the stereotype "<<Association>>" and using the attribute AssociationType, they relate to the cardinality one-to-one, one-to-many, many-to-one, or many-to-many as appropriate.
C. Transformations MOFScript

From model to model transformation of the previous section, the source code of the application is generated from MOFScript tool.

II. CASE STUDY

In this section we introduce a generic example in which transformations explained in section IV and V are applied. Figure 7 shows the BPMN 2.0 diagram from which the instance was generated in xmi format of the BPMN metamodel. This is shown in Figure 8 using a display of objects that constitute the xmi file.

Figure 9 shows the result of applying the first transformation. The output model is obtained, which is an instance of the UML2 metamodel.

From UML file resulting from the first transformation, the transformation from UML 2 to UML 2-Profile-EJB3.1 JPA 2 is performed. Figure 10 shows the transformation of Business Package classes into UML model with the addition of stereotypes corresponding to the EJB3.1 and JPA profile.

Finally, using the UML2 metamodel with EJB3.1-JPA2 profile and the corresponding generated xmi instance, the transformation MOFScript is executed. With the Java EE source code files are obtained which are then packaged using the ant tool to generate the ear file, these files are feasible to be displayed by a Java EE application server.

The complete transformations, metamodels, profiles and example can be accessed from the website: https://sites.google.com/site/mdapractico/
II. CONCLUSIONS AND FUTURE WORKS

Today organizations are constantly changing so there is a need to have fully or partially automated Business Processes. For this reason, it is essential to maintain the articulation between business processes and software systems. The transformations in the context of MDA are a valid alternative to meet this goal.

This article introduces a proposal so that, through transformations of an executable BPMN 2.0 model, a code is obtained in a Java EE server display. The proposal shows the rules that allow, from the information contained in an executable BPMN 2.0 diagram, obtaining a logical architecture of Enterprise Java EE application. In addition to the transformations, the study provides a profile for EJB 3.1-Web Services (JAX-WS) Platform.

The proposal benefits the development of software that allows increasing the quality and reducing design errors. Based on standard UML2, the developer can take UML models of transformations and add the necessary details to comply with the requirements of object-oriented analysis.

When using an instance of the standard BPMN 2.0 metamodel as a source model, this model serves as input for a BPMS and for the MDA tool developed in this work, allowing the interoperability between them through BPMN 2.0 task types as ServiceTask.

This work contributes transformations that allow generating Java EE business components related to business processes, so it helps improve development productivity and reduce design errors.

Future work is aimed at adding new transformations related to the elements of BPMN 2.0 metamodel, such as Timer and business rules.

REFERENCES