Comparison of Two Model Driven Architecture Approaches for Automating Business Processes, Moskitt Framework and Bizagi Process Management Suite

Oskeol Gjoni, PHD Student

European University of Tirana, Tirana, Albania

Doi:10.5901/mjss.2015.v6n2p615

Abstract

The Model Driven Architecture (MDA) approach to develop information systems is a model-oriented approach since it focusses on the business logic (the "what") rather than on the specific implementation technology (the "how"). Models are the authentic protagonist of information systems development. They represent the information system developed while the implementation in a certain technology is obtained through model transformation. The business process that will be automated in both frameworks is the user access management flow that employees follow in order to gain access into systems. The same business process will be modeled in two different formalisms, UML and BPMN, as per the capabilities supported from each framework. Starting from the model the automation steps in order to have an information system which can be executed and which automates the respective process will be clarified. A comparison study in each step of the process automation for both frameworks is presented. The advantages and disadvantages of each of the approaches will be considered and also evidence which approach is more suitable in specific cases.

Keywords: *Model driven architecture, MOSKitt, Bizagi, UML, BPMN.*

1. Introduction

In order for modern companies to survive in a world always more competitive, more difficult to predict, more connected and challenging, their business environment changes and adapts dynamically. Regulatory compliance, mergers and acquisitions, joint ventures, outsourcing activities just to name a few, impose changes in the business model and therefore on the information system. At the same time the progress and introduction of new technologies impose changes in the business environment.

The development process of information systems is still highly focused on the codification (the "how") rather than modeling (the "what") which affects in a negative way the productivity and the quality of the final product. Most of the efforts are spend in absorbing the technological complexity of a particular implementation rather than concentrating on understanding the business process for which the information system is being build [10, 12, 31]. Programing is still the most important task in creating a software product. It is true that programming technologies have improved with the passing of years, but the problem is that if we consider the bad results programming-oriented technologies have achieved historically is logical to ask the following: is reasonable to search for better methods of creating information systems? It would be desirable to have a new approach in developing information systems that separates business logic from the implementation technology, which takes the software development process to a higher level of abstraction using expressions that are more close to the business logic.

The Object Management Group (OMG), as a response to the above concerns, has defined a new framework for developing information systems, the Model Driven Architecture (MDA) [22, 25]. As per OMG definition of MDA, "*MDA separates business and application logic from underlying platform technology... No longer tied to each other, the business and technical aspects of an application can each evolve at its own pace – business logic responding to business need, and technology taking advantage of new developments – as the business requires*" [25]. Models are at the heart of the MDA approach; the process of software development is driven by constructing models that represent the software under development, creating software means creating the model. The specific code that represents the implementation of the model in a certain underlying technology is obtained through model transformation [31].

MDA is a model-centric approach in developing information systems since if focusses on the business logic (the "what") rather than on the specific implementation technology (the "how") on a particular programming environment such as Java, .NET etc [11, 12, 31]. This separation enables also portability, interoperability and reusability [24]. MDA makes models the authentic protagonist of information systems development, not the source code. This separation decreases

the impact that technology evolution has on application development and also enables the possibility to profit from new implementation technologies. Once the model is created, it can be transformed automatically into code in different software platforms. Knowledge and intellectual property engaged in application development are moved from source code to models. Models are the most valuable assets since code is obtained from them through automatic transformations [32].

There are different MDA solutions that are based on the same OMG guidelines. Some of the most known implementations both proprietary and open source solutions are; Abstract Solutions (http://www.abstractsolutions.co.uk/), AndroMDA (http://www.andromda.org), Virtual Enterprise (http://www.intelliun.com), Oliva Nova [32] etc. Some of the biggest companies like Lockheed Martin, Credit Suisse, Austrian Railways etc, have obtained success stories while using the MDA approach [14]. In this study we will compare two known MDA implementations, MOSKitt framework (http://www.moskitt.org) and Bizagi business process management suite (http://www.bizagi.com).

2. MOSKitt and Bizagi

Modeling Software Kit (MOSKitt) is a free software framework built on the Eclipse platform developed by "*Conselleria de Infraestructuras, Territorio y Medio Ambiente*" in Valencia, Spain [20]. Our attention will be focussed in "*MOSKitt Code Generation*" module which supports the semi-automatic generation of OpenXava [27] applications through a chain of model transformations, from models (UML models, User Interface Models) to code. Through model transformation is provided automatic generation of the OpenXava code necessary to ensure the persistence of the entities in the database and the generation of the AJAX user interface. An operational enterprise web application with create, read, update, delete and export functionalities is automatically generated. OpenXava is an open source AJAX Java Framework used for Rapid Development of Enterprise Web Applications [27].

MOSKitt Code Generation has a development environment called OXPortal that deploys and executes OpenXava applications. It consists of a compressed package that contains the following components: Apache Tomcat container, Liferay Portal, HSQLDB database etc. [16, 19].

Bizagi business process management suite implements the MDA principles or the "modeling over programming" philosophy. It is comprised of three main tools: Bizagi Modeler, Bizagi Studio and Bizagi Engine that provide the capability to manage the complete lifecycle of business processes from modeling, execution and improvement [2].

- **Bizagi Modeler**: enables business experts to design, document and evolve their process models. The modeling notation that the Bizagi process modeler supports is BPMN (Business Process Modeling and Notation) [23].
- **Bizagi Studio**: provides to business experts everything they need to transform process models into real, running applications and workflows, from defining the data model, business logic and user interface to integrating IT assets and everything in between [2].
- **Bizagi Engine**: executes and controls the business processes automated by Bizagi Studio. It supports deployment in JEE or .Net and provides a set of useful performance KPIs for process improvement [2].

3. Modeling the User Access Management Business Process

The business process to be automated is the classical User Access Management (UAM) flow; the process that employees within a company need to follow in order to obtain or revoke access rights in specific applications (CRM, ERP etc).

The employee starts the process by raising an access request. The request follows a predefined approval path which consists of line manager and Corporate Security manager, which is a central entity within the company responsible for the security of the systems and which approves all permission requests. In case the request is approved by both line manager and Corporate Security manager, it will be implemented from the implementation team, otherwise the request is rejected. In all cases the employee is informed regarding the status of the request. After the request is implemented, the Corporate Security analyst is informed and the process ends. The employee is being granted with the access required in the respective system.

In the MDA approach to create an information system means to create the specific model that represents the domain under study. Models in MOSKitt are created using UML while Bizagi supports Business Process Modeling and Notation (BPMN), which is a standard for business process modeling that provides a graphical notation for specifying business processes based on a flowcharting technique [26, 34].

In Figure 1 is shown the UML model created using the MOSKitt UML2 Modeling Module, an UML class diagram, containing the classes with their attributes and relationships for all concepts necessary to cover functionalities required to deal with the UAM process. MOSKitt approach uses only UML class diagrams [17].

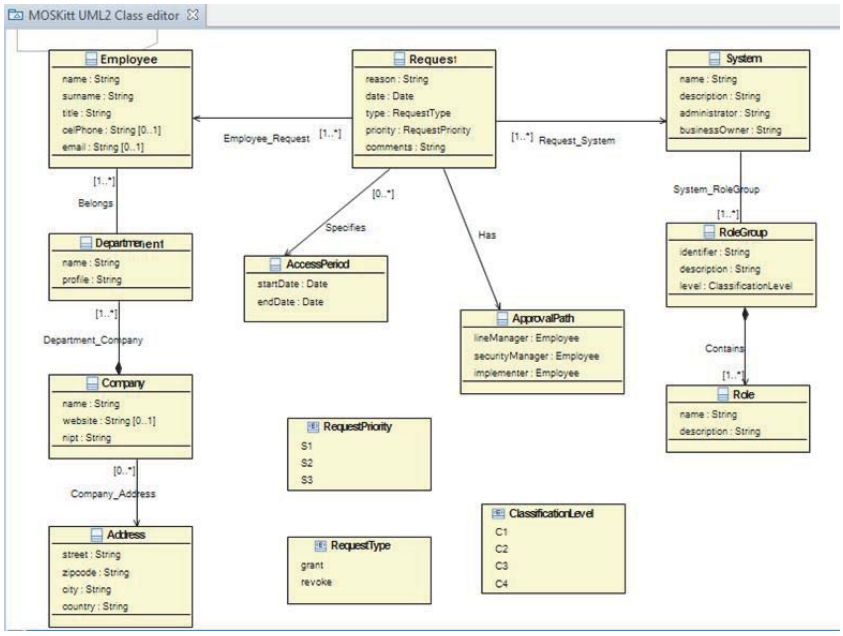

Figure 1: UML model of the User Access Management flow.

The BPMN model created in Bizagi is represented in Figure 2, containing all the stakeholders involved in the process with the respective tasks/activities that each should perform. The start and end event are also easily distinguished [23, 34].

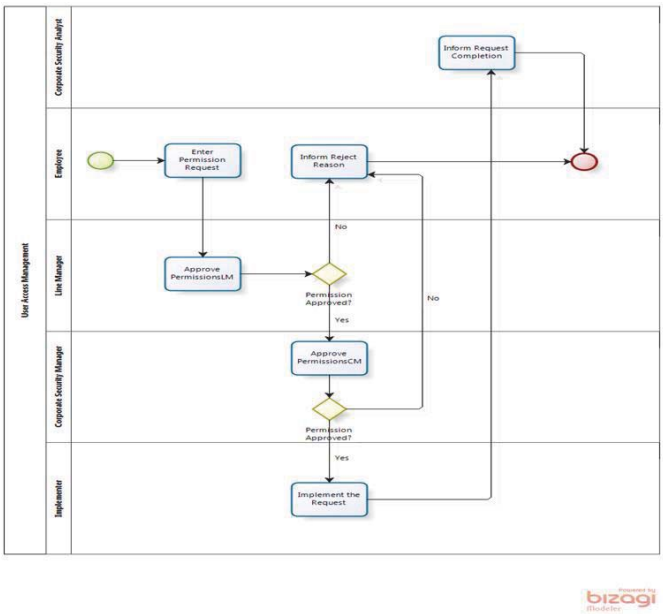

Figure 2: BPMN model of the User Access Management flow.

The unified modelling language (UML) takes an object-oriented approach to the modeling of applications, while BPMN takes a process-oriented approach to modelling of systems. In particular MOSKiit uses UML class diagrams that describe the static structure of the system by showing the classes, their attributes and relationships between them. On the other

hand Bizagi uses BPMN models that more easily represent the business flow using the flowcharting technique. The model is more readable and understandable. Where BPMN has a focus on business processes, the UML has a focus on software design and therefore the two are not competing notations but are different views on systems. The BPMN and the UML are compatible with each other [9].

4. MOSKitt and Bizagi Automation Steps

The model driven development adopted from MOSKitt is a chain of model transformations as presented in Figure 3 [18]. The UML model is the starting point.

Step 1: Generate the Java classes with the JPA annotations for the persistence of the UML entities. This is obtained from UML2JPA transformation from the UML Model.

Step 2 [Optional]: Add the OpenXava annotations to the Java classes to express the user interface information specified by means of the Sketcher model. First the Sketcher model is transformed in the User Interface Model (Sketcher2UIM) and the latter through UIM2OX transformation provides the final OpenXava code.

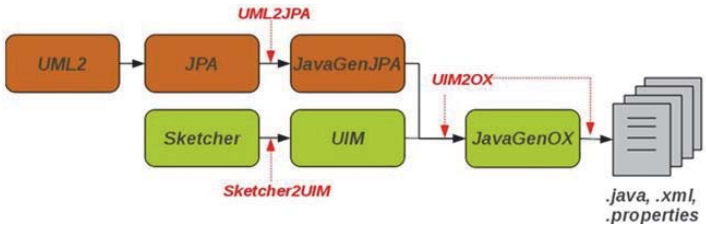

Figure 3: MOSKitt model driven development strategy – chain of model transformations.

The process automation steps followed in Bizagi Studio as shown in Figure 4, automate the BPMN model creating a running application without the need to write code. Bizagi adopts 100% the MDA guidelines [1].

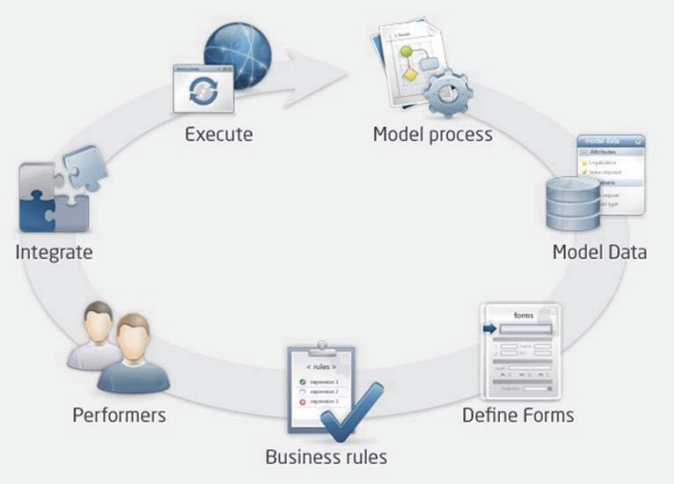

Figure 4: Bizagi Studio process automation wizard.

5. Data Modeling

The first step is to define the data that the process requires for its execution. Bizagi allows to structure business data in a graphical and logical way similar to an Entity-Relationship schema, resulting in an easy to understand data model. To provide an organized and coherent structure Bizagi provides different types for Entities and different types of Relationships that can be used in order to build the data model [5].

6. Forms – User Interface

The second step is to define the user interface for all the activities in the model that require human interaction. In the *User Access Management* process all the activities require human interaction since there are not present automatic activities. To define the user interface for an activity the ***Forms Designer*** is used which provides an intuitive and user friendly structure to drag and drop the data fields onto a form and arrange them accordingly without the need of programing [4].

7. Business Rules

Bizagi provides powerful business rules that can be implemented in various places throughout the process [6].The following points clarify the use of business rules.

- *Perform actions in Activities - calculate and validate*: It is very common to perform validations to control the values entered by end users in a field. Bizagi also allows making calculations to manage the data model and assign or delete values to any attribute.
- *Route Processes - Sequence flow:* Business rules are implemented as transition conditions in a process to route the sequence flow. The expressions used for this purpose always return a value of ***True*** or ***False*** (Boolean) in order to tell the process where to go next.
- *Managing the user interface:* it is very important to be able to control what fields are displayed or hidden to the end users, as well as control what fields are mandatory, editable or read-only.
- *Users allocation:* Business rules can be used to control the users that will be allocated to tasks according the business conditions [6].

8. Define performers

Work allocation is the next step of the process automation wizard where *Performers* are defined for each activity of the process. *Performers* are the users that have the qualities to be assigned to activities. Each activity created for end user interaction requires definition that will allow Bizagi to allocate the correct users within the organization. Bizagi automatically evaluates the allocation rules defined for each activity and selects one or more users that meet the given conditions from the user's list. Only these users will have access to work on the activity allocated to them [8].

9. Integrations and Execution

The next step in the automation wizard provides the capability to integrate the process with other external systems present within the enterprise [7]. Once all the process automation steps are completed an application that automates the business process is obtained that can be executed in the Bizagi Engine [3].

10. User Access Management Automation Comparison

Starting from the UML model in Figure 1 and following the UML2JPA automatic transformation an OpenXava application with *CRUD (Create, Read, Update, Delete)* and *Export (Excel, PDF)* functionalities is generated that can be executed in the developing environment. The transformation rules are:

- A UML class will be transformed into a Java class.
- A class attribute in the UML model will be transformed into a private attribute in the respective Java class.
- For each attribute in the UML class two public methods (a get and a set method) are generated in the corresponding Java class.
- For each association end, there is a private attribute of the same name in the opposite class of type equal to the type of the class in the case when the association is ***one to one***. In the event the association is of type ***one to many***, then the type of the private attribute is ***Set***. In the case the association is navigable (directed association) then the private attribute will be added only to the class origin of the association [28].

The generated code will contain the standard Java types and the annotations for the classes, their properties and operations [13, 17]. The output OpenXava application is responsible to automatically generate a default user interface and ensure the persistence in the database. In case the default user interface is not satisfactory there is the possibility to define specific user interface layout. MOSKitt supports this through Sketcher model diagrams, which are user interface

models, created by drag and dropping user interface widgets in the Sketcher editor [18]. In order to ensure consistence between the UML and Sketcher models every widget of the Sketcher model needs to be linked with the corresponding UML model property.

In Figure 5 is shown the Sketcher Diagram for the Request module, presenting the list view and part of the detailed view for the module. The "***Name***" widget that will contain the name of the employee performing the request is mapped with the "name" property of the UML Model, "***name***" property of the class **Employee**. To ensure this the attributes *Data Model Element* and *Data Model Path* are set accordingly [18].

Data Model Element: <Class> Employee :: <Property> name : String
Data Model Path: <Class> Request :: <Association> Employee_Request :: <Class> Employee :: <Property> name : String

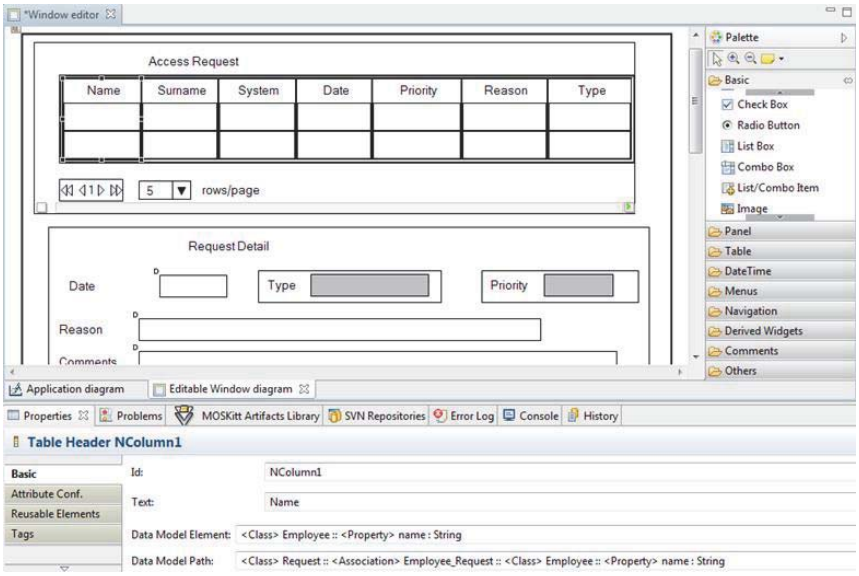

Figure 5: Sketcher diagram for the Request Module.

In Figure 6 is presented the generated web application in execution visualizing the list view of requests.

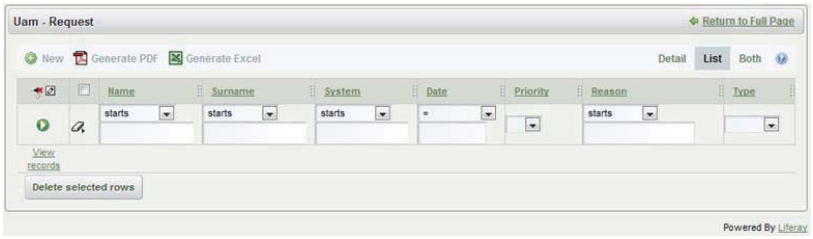

Figure 6: UAM application in execution, Request module.

In Bizagi, once the BPMN model in Figure 2 is completed the next step in the automation process is to create the data model. In Figure 7 is presented the data model created for the UAM flow containing all the data needed during the process execution.

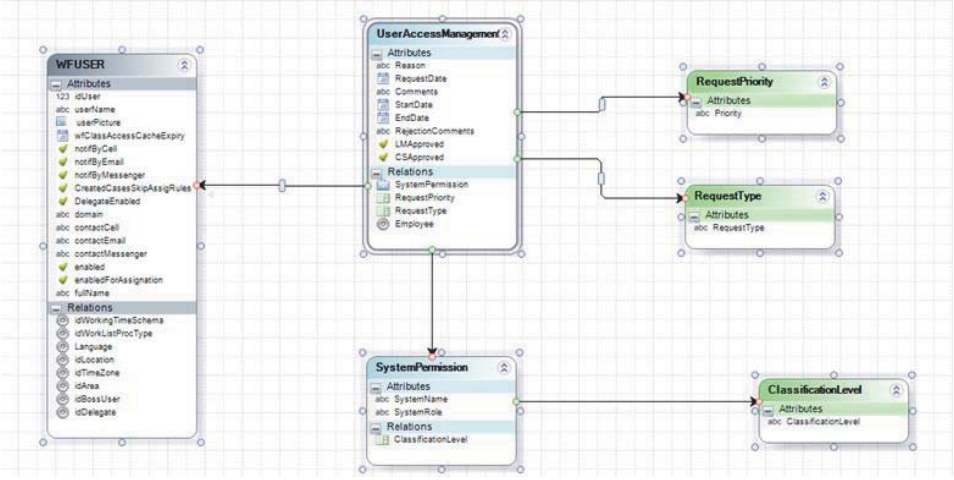

Figure 7: The data model for the User Access Management flow.

Bizagi Studio provides ready for use a set of entities which belong to the Bizagi's internal data model. These are called System entities and the ***WF USER*** entity presented in Figure 7 is one of them. System entities include information concerning the end users, areas, locations etc. These entities are created by default for each project and help in the creation of the data model for the specific process. All is needed is to include and link these entities in the data model created. They represent general entities which are present in any process and thus help to focus our attention in creating the entities which are particular for our process and utilizing System entities for general use cases (like end users in Figure 7). This is a big advantage compared to MOSKitt, where it is necessary to create in the UML class diagram the respective classes that will contain information regarding the end users (Employee) and the organization (Company).

In Figure 8 is presented the user interface created for the first activity of the *User Access Management* BPMN process, *Enter Permission Request*. The same logic is applied in order to create the user interface for the remaining activities.

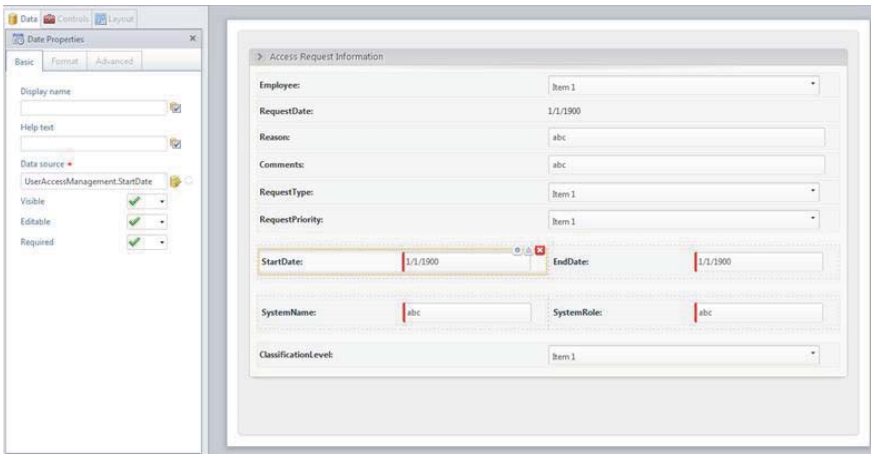

Figure 8: User interface for the Enter Permission Request activity.

Each item or element included in the design area is known as a *Control* in Bizagi. *Controls* can be added individually by means of drag and drop from the left panel. When a *Control* is added in the design area it does not have a reference in the data model to the specific attribute it represents. In order to link the *Control* with the corresponding attribute in the data model the *Data Source* property of the *Control* should refer the correct attribute in the data model. In Figure 8 is shown how the ***Data Source*** property of the ***StartDate*** control is related with the ***StartDate*** attribute in the data model (*UserAccessManagement.StartDate*) [4].

In Figure 9 is presented how business rules are used in order to route the user access management process. Let's consider the case when the Line Manager has approved the request. This is represented by the following expression (*UserAccessManagement.LMApproved is equal to true*). LMApproved is a boolean attribute present in the data model as shown in Figure 7 that will store the choice performed from the Line Manager. In a similar way are defined the business rules for the other routes of the process.

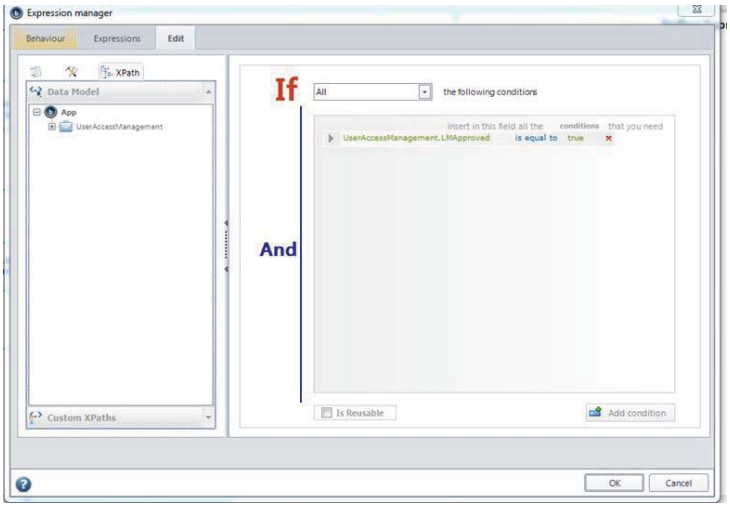

Figure 9: Business rules to route the *User Access Management* process

To allocate performers, it is necessary to have a user account created for everyone that intends to work with Bizagi. It is important that each user account must be set up correctly to ensure Bizagi selects appropriately. This powerful feature can be configured based on different criteria like positions, geographical location, skills and roles among others.

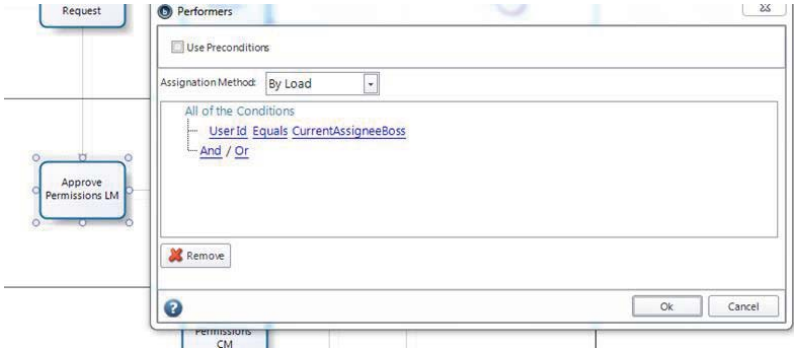

Figure 10: Define performers for the activity *Approve Permissions LM*

In Figure 10 is shown how is defined that the activity *Approve PermissionLM* is always performed from the line manager of the user that has raised the request. The expression used is ***UserId Equals CurrentAssigneeBoss*** that instructs Bizagi that the activity *Approve Permissions LM* should be performed from the line manager of the user that submitted the request in the first place. Similarly in Figure 11 is shown how the activity *Approve Permission CM* is allocated to the Corporate Security Manager through the application of positions present within the company [8].

Figure 11: Allocation of the Corporate Security manager to the activity *Approve Permissions CM*.

In the *User Access Management* process there are considered no integration with external systems. The process has completed all the steps of the process automation and an enterprise web application is generated automatically which is ready for execution in the Bizagi Engine. The user access management application in execution is presented in Figure 12.

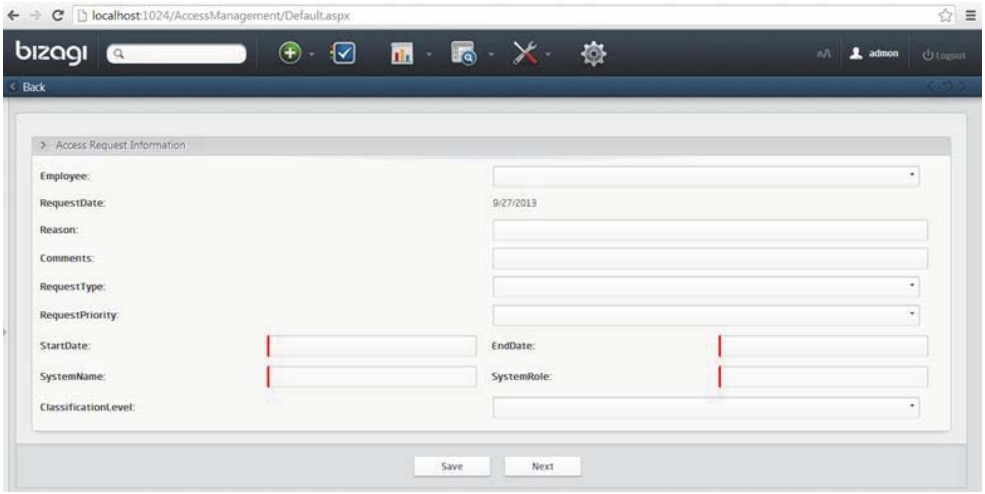

Figure 12: User Access Management application in execution

11. Comparison and Conclusions

11.1 Model Driven Development implementation

Starting from an UML Model (UML class diagram) through MOSKitt code generation it is possible to generate automatically a Java web application with *CRUD (Create, Read, Update, Delete)* and *Export* functionalities without the need to write any Java code. The web application generated does not contain information regarding the specific business logic of the domain under study. In order to complete the OpenXava application generated, incorporating into it business logic details, it's the responsibility of the application developer to write the necessary OpenXava code specifications. That is the reason why MOSKitt code generation is referred to as semi-automatic generator of OpenXava applications. It's a semi-automatic generation because the OpenXava code generated must be completed by the developers to include specific business logic for each specific application [15]. *There is not a full compliance with MDA principles*.

In Bizagi, starting from a BPMN model and following the process automation wizard is possible to generate a web application that automates the business process without the need to write any code. *Bizagi business process management suite fully supports the MDA principles.*

11.2 Security, User Management and Module Navigation

OpenXava framework does not include security, user management and module navigation. In order to add these features to an application though, there are several ways to do it [29]:

- *Integrate the application in a Java Portal such as Liferay, WebSphere Portal or JetSpeed*: OpenXava generates portlet applications that can be deployed in any standard Java Portal, so the application can be deployed in a Java portal and use the security, user management and navigation provided by the portal [30].
- *Use NaviOX:* NaviOX is an add-on for OpenXava that adds navigation, security and user management to applications easily [21].
- *Write the security and navigation code:* An OpenXava application is a regular Java web application, therefore security can be added in the same way as for any other Java web application.
- *Add spring security*: spring-security is an authentication and authorisation framework from spring framework [33]. No change in the OpenXava code is required.

Security, user management and module navigation are fully supported in Bizagi engine. The resulting application is executed without the need for further integrations or customizations.

12. Application Development

Bizagi is a suite for process automation and is not applicable for general purpose application development. MOSKitt is a tool for the development of web applications, suitable for business applications-oriented databases and not only process oriented applications.

13. Proprietary

Bizagi is a proprietary solution. Processes automated with Bizagi can be executed only within the Bizagi Engine and respective license fees need to be paid for enterprise deployments. MOSKitt is a free platform using the EPL (Eclipse Public Licence).

References

Bizagi BPM suite documentation, available http://help.bizagi.com/bpmsuite/en/.
Bizagi BPM suite overview, available http://help.bizagi.com/bpmsuite/en/index.html?overview_what_is_bizagi_bpm_suite.htm
Bizagi Engine, available at http://help.bizagi.com/bpmsuite/en/index.html?process_execution.htm
Bizagi Studio, creating the user interface, available http://help.bizagi.com/bpmsuite/en/index.html?creating_the_user_interface.htm.
Bizagi Studio data modelling, available http://help.bizagi.com/bpmsuite/en/index.html?modeling_data.htm.
Bizagi Studio defining business rules, available http://help.bizagi.com/bpmsuite/en/defining_business_rules.htm
Bizagi Studio integration, available http://help.bizagi.com/bpmsuite/en/index.html?integrating.htm
Bizagi Studio work allocation, available http://help.bizagi.com/bpmsuite/en/index.html?work_allocation.htm
Business Processing Modeling Notation information, available at http://www.omg.org/bpmn/Documents/FAQ.htm
Forester Consulting, Modernizing Software Development through Model-Driven Development. A commissioned study conducted by Forrester Consulting on behalf of Unisys, August 13, 2008.
Frankel, S. D.: Model Driven Architecture. Applying MDA to Enterprise Computing. Wiley Publishing, Inc. OMG Press 2003.
Guttman M, Parodi J: Real-Life MDA: Solving Business Problems with Model Driven Architecture, (The MK/OMG Press), 2007.
Java annotations, available http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html
MDA Success Stories, available http://www.omg.org/mda/products_success.htm
MOSKitt code generation, available http://www.moskitt.org/eng/openxava_que_es/
MOSKitt code generation installation guide, http://www.moskitt.org/eng/openxava_instalation00/
MOSKitt code generation, "Tutorial 1: Developing an OpenXava application from UML2 MOSKitt models", available http://www.moskitt.org/eng/openxava_manual0/
MOSKitt code generation, "Tutorial 2: Including User Interface design in OpenXava applications with MOSKitt", available http://www.moskitt.org/eng/openxava_manual0/
MOSKitt code generation, "Tutorial 3: How to launch an OpenXava application with OXPortal", available http://www.moskitt.org/eng/openxava_manual0/
MOSKitt home page, available http://www.moskitt.org
NaviOX, available http://www.openxava.org/naviox
Object Management Group (OMG), available http://www.omg.org

Object Management Group (OMG), Business Process Model and Notation, available http://www.omg.org/spec/BPMN/
OMG, "MDA Guide Version 1.0.1", available http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf 12th, June 2003.
OMG Model Driven Architecture, available http://www.omg.org/mda
OMG Unified Modeling Language Infrastructure, available http://www.omg.org/spec/UML/2.4.1/
OpenXava home page, available http://www.openxava.org/
OpenXava Reference Guide, available http://openxava.wikispaces.com/reference_en
OpenXava; Security, user management and module navigation, available http://openxava.wikispaces.com/security_en
Paniza J.: Learn OpenXava by example, 2011.
Papajorgji P., Pardalos, P.: Towards a Model-Centric Approach for Developing Enterprise Information Systems.
Pastor O., Molina, J.C.: Model-Driven Architecture in Practice A Software Production Environment Based on Conceptual Modeling, Springer 2007.
Spring security, available http://static.springsource.org/spring-security/site/
White S.A.: Introduction to BPMN, July 2014.