Journal of Theoretical and Applied Information Technology
10th September 2015. Vol.79. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MAPPING OF A PLATFORM SPECIFIC MODEL TO A PARTICULAR PLATFORM USING AN EAV DESIGNED PLATFORM MODEL

[1]AHMED MOHAMMED ELSAWI, [2]SHAMSUL SAHIBUDDIN

[1]PhD Student, Department of Computing, Universiti Teknologi Malaysia (UTM), Malaysia

[2]Prof., Advanced Informatics School (AIS), Universiti Teknologi Malaysia (UTM), Malaysia

E-mail: [1]elsawi@gmail.com, [2]shamsul@utm.my

ABSTRACT

The Model Driven Architecture (MDA) aimed to produce applications that support multiple platforms using models instead of the conventional coding with less cost and time. The MDA development process separated to Platform Independent Model (PIM) and Platform Specific Model (PSM). Both, PIM and PSM are standing in different level of abstraction. The PIM focused on the business rules definition with no concern about the platform environment. It is the PSM role to emphasis on the implementation environment of the targeted platform. Although, the PSM models holding some technical implementation details about the targeted platform, but it's still considered too abstract to be executed in a run-time platform environment. The degree of abstraction can be controlled by the model mapping, given the MDA standard Platform Model (PM). The PM can be described as a platform's system manual that provides model mapping with a specific technical details required by the targeted run-time environment. In this work we explicitly employed a Platform Model (PM) designed by the Entity-Attribute-Value (EAV) concept to support the model mapping from PSM to Java platform. A case study provided as proof of concept to generate executable Java code that supports desktop and mobile platform.

Keywords: *MDA, PSM, Platform Model, Mapping to Java Platform, EAV*

1. INTRODUCTION

Applications The Model Driven Architecture (MDA) is focusing on Architectural Modeling where the concern of the business requirement is separated from the implementation details and platform requirements. The development approach divided into platform independent model (PIM) and platform specific models (PSM). Both models are working in different level of abstractions [1]. UML/MOF are a common OMG standard tools that used in model driven development to design the PIM and PSM models and metamodels [2, 3]. Model mapping or mapping is one of the major activities in model driven software. Given the Platform Model (PM), it serve in transforming high level models (PIM) to low level (PSM) models or vice versa. Also, the mapping can be within the same level of abstraction (PIM-to-PIM) or (PSM-to-PSM). These scenarios of mapping can be classified as Model-to-Model (M2M) mapping or Model-to-Text (M2T). Both are under the MDA umbrella and supported by a good number of tools that furnished to address each scenario and type of mapping [4].

Model mapping faced by several challenges. Specifically, when it comes to generating concrete platform's code out of Models.

The abstraction of the data structures specified in the MOF models is one of the model mapping challenges. Where, concrete syntax derived from MOF/UML instances model is a common syntax. There is no reason to expect that the concrete syntax of the method calls for a particular platform conform to any of the MOF related concrete syntaxes. Platforms runtime environment require a very specific concrete syntax to execute [5, 6].

On the other hand, The UML does have a wide-ranging of behavioral models [7]. These behavioral models permit the specification of a complete range of behaviors. These specifications are normally static. The class diagram can be one of these models that commonly used to describe the model specifications. But on the other hand, the semantics of the behaviors are not included in the models as it is not included in the static model specifications

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[8]. For example, one of the actions included in a behavioral model might be print(). The behavioral model can include pre and post conditions for the validity and outcome of the print() action, but what the action actually does is not specified in the model. What print() actually does depends on the hardware and software platforms which implement the operation represented by the action. In MDA terms, the print() action is specified in the PIM, while its semantics are specified by the platforms generalized in a PSM which represent the dynamic part. Both of the models are independent from the platform but collaboratively upon mapping should map to a single executable object in the targeted platform.

From other prospective, the adoption of separation of concern and models along with model mapping, gives the MDA a great advantage on the automation of software development and hypothetically supports the deployment of software on different platforms [9]. Yet, the implicit employment of the PM during the mapping limited the model mapping scope to address a single presumed platform [10]. The thing that create an uncertainty of the possibility of using this model mapping for other platforms different than the one for which it was designed. Considering the diversity and volatility of the computer platforms, the possibility of having a model mapping that can be used by different platforms will be compromised. Even if we consider a portable platform like Java, it is not guaranteed that a model mapping designed for J2SE can be reused to address J2ME mobile platform.

This work is addressing the above challenges by focusing on the mapping from PSM models to Java platforms. We propose an explicit use of a PM designed by the Entity-Attribute-Value (EAV) concept to complement the model mapping and generate a concrete Java code out of a given PSM metamodel. We also illustrated how to address the platform diversity and volatility by an explicit employment of EAV-PM.

The next part is section 2 where we introduce the related works, and initiatives that focused on code generation out of PSM models and the explicit use of the PM by model mapping. This is beside the work that address the issues of platform diversity in the MDA context. Section 3 shows the capability of EAV in modelling a computer platform and how we utilize it to design an explicit PM to support the model mapping from PSM models to Java platform. A case study presented in Section 4 to practically demonstrate the possibility of generating an executable java code out of PSM model supported by our novel explicit EAV-PM. The discussion and results shown in Section 5. While the conclusion and future work provided in Section 6.

2. RELATED WORK

The work in [11], is closely related to ours. Where the authors suggested an explicit use of an ontology based platform model to address the platform dependencies of the model mappings and their validity to specific platforms. Using the ontology to represent the platform model comes with limitations. In general, the automation of handling big ontology is impossible due to the number of classes an instances. This is beside the fact that the time consumed in a manual construction of ontologies is growing more complex upon the diversity and platform's data volume rapid increase. Consequently, this is reflected in the number of ontologies required to cope with this technology volatility. Although some ambitious ontology automation [12, 13] works on reducing the time and complexity, however, it is not yet mature enough to be adopted. Putting into account that the current ontology automation methods are failing to merge different ontologies to produce new mature one [14, 15]. Another ontology platform representation limitation, is the lack of flexible validators that capable to validate all different types of platform ontologies. Especially when it comes to complex inheritance relationship. On the other hand the OWL language that adopted to create the platform ontologies is also drawing some limitation on ontology creation. Since, the OWL doesn't stand on a backend database during the ontology creation. This is beside the fact that their explicit ontology base PM model serve the model mapping from PIM to PSM. Meaning, more effort required to tackle the mapping from PSM model to platform concrete code.

Nevertheless, we are following some of their steps as we also explicitly employed the platform model. However, our EAV-PM serve the mapping from PSM to concrete executable code. Instead of using the ontology we used the EAV to reasoning for platforms and to design the PM to support the model mapping with the necessary technical details required by the targeted platforms.

In SPL terminology the platform model here can be described as a core assets that can be utilized by model mappings to address different platforms.

In the next section we highlight EAV capabilities in knowledge representation and specifically in

Journal of Theoretical and Applied Information Technology
10th September 2015. Vol.79. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

representing computer platforms models and metamodels along with the design of the explicit Platform Model.

3. COMPUTER PLATFORM REPRESENTATION USING EAV

Our work in [16] illustrated EAV capabilities in representation of models and metamodels. Its knowledge representation capability and open structure makes it suitable for modeling and representing computer platform. The open structure gives EAV a flexibility to cope with the platform diversity and volatility. Our work in [17] demonstrated the possibility of merging models with different level of abstraction in a single EAV repository. This capability opens the door for more technology merger and integration flexibility. On the other hand it gives valuable support to MDA objective of speed up and cost cuts the software development process. With consideration to satisfy the technology diversity factor. In this section we adopted the technique provided by [18, 19] to describe a computer platform. They provide an ontology based platform vocabulary that utilized by [11] and so we do. However we presented in EAV model. Figure 7 shows a partial view of their platform description.

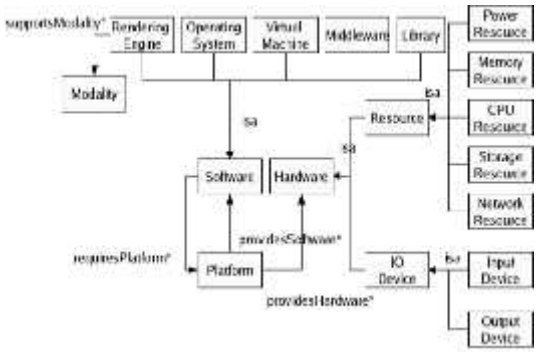

Figure 7, A Partial View Of A Computer Platform Description Presented By[11]

Each platform component in the above figure, can be represented as an Entity with its associated attributes and values. Attributes can be described as a separate Entity with its own attributes and values. For example the virtual machine component presented as Platform.software.virtualmachine. The "isa" is a subsumption relationship that indicate that the group of Virtual Machines subsumes the group of software. The Entity associated attributes and values, list out the information and futures provided by this Entity. This is along with its correspondent dependencies and constrains that govern its interactivities with other platform components. The table in appendix A table A.1 illustrate a partial EAV representation for figure 7. As we described in [16] a database view can be used to distinct a certain area of interest in the targeted platform or to set a particular contains or dependency. For sake of space saving and simplicity we narrow down the amount of attributes, so we can present the whole concept above.

The Virtual Machine in appendix A table A.1 is prefixed by Software to indicate it refers to the Software, where the '.' employed as a namespace delimiter for easy navigation and information retrieval. Similar to our EAV representation in section 3.1, the platform formation rules and Constrains, will be checked by structural queries as well. For example, to satisfy the fact that each software platform is in a cyclical relationship with software and hardware platform.

```
SELECT * FROM  EAV_PM AS B
WHERE
B.ENTITY IN (SELECT SUBSTR( A.VALUE_ , -8,
8 ) FROM  EAV_PM AS A
                  WHERE
                  B.ENTITY    =    SUBSTR(
A.VALUE_ , -8, 8 )
             )
```

It is important to mentioned that the the database is not intruded here as a technology since the attribute value pairs concept can be handled and formulated by other methods like XML/XMI. So the SQL is here to help in formalization of the EAV concept. Beside the fact that the database structures can easily be transformed to XML format.

From the above we can see that the highlighted output results in the below XML format illustrated that for each software platform, there are multiple hardware and software platform involved. Having the constrains among other EAV PM results in XML format, is opening the door for more integration and automation possibilities with different tools in different branch streams concerned with model and software development as well as code generation.

```
<database name="test">
<!--  Table eav_pm  -->
<table name="eav_pm">
<column name="ENTITY">Platform</column>
<column
name="ATTRIBUTE">Relationship.type</column>
```

Journal of Theoretical and Applied Information Technology
10th September 2015. Vol.79. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
<column
name="VALUE_">Multiplicity.provideHardware.
Platform.hardware</column>
</table>
<table name="eav_pm">
<column name="ENTITY">Platform</column>
<column
name="ATTRIBUTE">Relationship.type</column>
<column
name="VALUE_">Multiplicity.provideSoftware.
Platform.software</column>
</table>
</database>
```

The high abstract level platform representation in figure 7, can be extended to detailed lower levels of abstraction. For example the Java Runtime Environment (JRE) in figure 8 come with Java Virtual Machine.

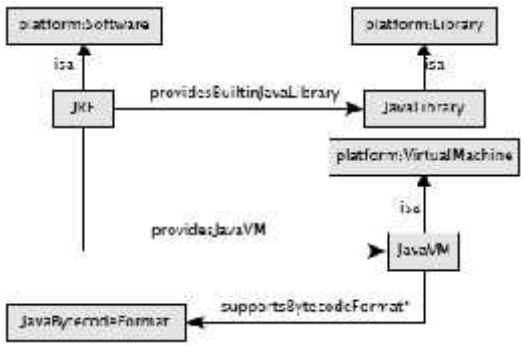

Figure 8 An Ontology Fragment Describe A Java Runtime[11]

Each version of the Java virtual machine has its own byte code format. In general Java has different releases, each of which addressing specific technology. The major difference between these releases is mainly in the build in libraries. J2me libraries addressing mobile platform technologies, while the J2EE releases addressing different technologies through its build in APIs. In appendix A table A.2 is apart from table A.1. The "Platform.software.virtualmachine.JRE" is a new entity associated with its attributes values pair. For each JRE there is a dedicated virtual machine associated with build in libraries. These libraries depending on the Java specifications for the particular JRE. For example the libraries of the J2me PP 1.1 are different than the one work for J2EE. Accordingly, appendix A table A.2 shows a partial description of the EAV representation of Figure 8. It is important to mention that this representation is stored in the same EAV platform repository in appendix A table A.1 that we separated for the sake of space saving.

The below structural queries can be saved in a database view to group the specifications of the JRE along with its constrains and API and/or other technical details of the targeted platform.

SELECT value_ FROM EAV_PM WHERE ENTITY = 'Platform.software.JRE'

The above code is a straight "SELECT" statement to retrieve the pair value associated to "Platform.software.JRE" Entity. By enquiring each of the highlighted entities we will get all the APIs beside the constrain we presented in the above EAV_PM example. Where the "Multiplicity.provideSoftware.Platform.software" and "Multiplicity.provideSoftware.Platform.hardware" entities are retrieved as a dependency constrain required by "Platform.software.JRE". Similarly, the "Platform.software.Library.JavaLibrary" and "Platform.software.VirtualMachine.JavaVM" can be enquired to drill down more information and/API associated to them.

In the next part we provide a briefing about an Eclipse Modeling Framework.

4. ECLIPSE MODELING FRAMEWORK (EMF)

The Eclipse Modeling Framework (EMF) is conforming to the OMG Meta Object Facility (MOF) standard for metamodel definition. The EMF metamodelling language called Ecore. Figure 9 presents a fragment of the Ecore Metamodel.

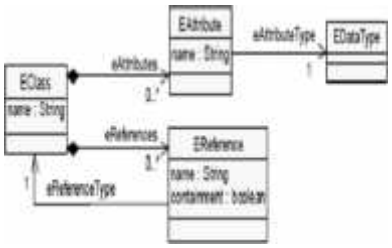

Figure 9: A Fragment From Ecore Metamodel

The capital E in the Ecore used to mark the Ecore classes from UML classes. Similarly, the Ereference and Eattribute. While the Eclass represent the classes the Ereference represent the association between classes. The Eattribute name

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the properties of each class and the attribute type address by the Etype.

Beside the Ecore, the EMF has another metamodel called Genmodel. While the Ecore handle the information about the defined classes. The Genmodel, provide the necessary information required for code generation.

Beside its support to the M2T mapping standards [2], the EMF have been chosen for its modelling capabilities where a domain model is visibly established. This is addition to the notification feature that the EMF provided upon model modification. Also, the EMF managing changes in models upon object creation through its notification feature. Consequently, applications will be immune from discrete classes' employment. On the other hand, the targeted code (Java code) can be generated when desired from the source model.

In the next part a case study introduced to demonstrate the mapping from PSM model to concrete Java code utilizing the explicit EAV-PM. The EMF version adopted to implement this case study is Eclipse JUNO Service Release 21.

5. THE WEBSITE FACTORY CASE STUDY

The MDA principles has been adopted in this case study to build a website factory by mapping a PSM model for a website to Java. In other word we are building a product line to produce Java based websites. The eclipse JUNO is equipped with a visual editor that allow us to create Ecore diagrams. Our Ecore model in this case study called "webpage.ecore". The full project of this case study can be downloaded from (https://www.dropbox.com/l/RtjY1X7AwYbc6FjLhAs24a?). The UML class diagram shown in Figure 10, define 4 classes (Web, Webpage, Category and Article).

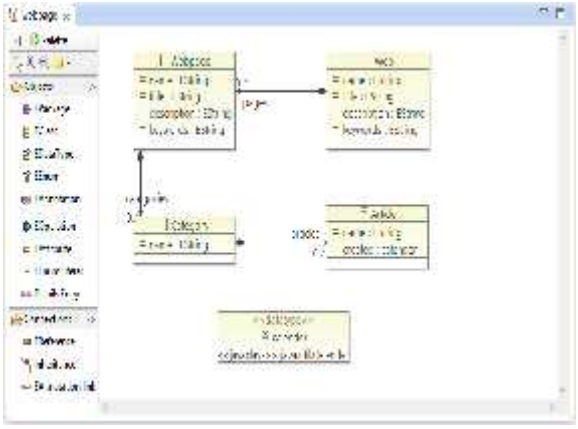

Figure 10 A fragment from a website class diagram

Unlike the other attributes, we assigned the attribute "created" in the "Article" class to a user defined "calendar" type defined in an Edatatype named "calendar" with type "java.util.Calendar". The saved diagram stored in "webpage.ecore" in the format shown in Figure 11. This feature enable for including a precompiled core asset as user defined data type.

As we mentioned in the previous section the EMF supported beside the Ecore metamodel, the "genemodel" metamodel. The genmodel should contain all the information concerned with the code generation.

*Figure 11 Part Of The "*Webpage.Ecore*" Model's File Format*

Through the EMF Generator Model, we generate a new version based on the "webpage.ecore" model called "webpage.genmodel". This step is done by the EMF generator Model. Figure 12 shows the "webage.genmodel" file format.

[1] http://www.eclipse.org/downloads/packages/release/juno/sr2

Journal of Theoretical and Applied Information Technology
10th September 2015. Vol.79. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

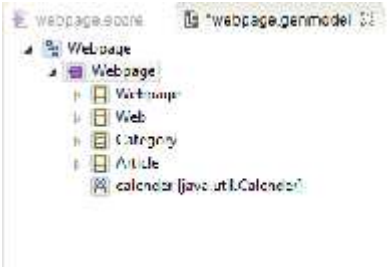

Figure 12 Sample from the "webpage.genmodel" file format

Both model files, the ".ecore" and ".genmodel" are required to generate the java code. By write click on the webpage node in the webpage.genmodel file (see figure 13) we get the following three files:The "spl.mda.emf.webpage.model.webpage" which representing the interfaced and the factory to great the Java classes. While the second is the "spl.mda.emf.webpage.model.webpage.impl" which holding the concrete code of the webpage model. The last is file is the "spl.mda.emf.webpage.model.webpage.util" which act as the adaptor factory.

Figure 13 Show How To Generate The Website Concrete Java Code.

In the next step we are creating an editor to our generated website model. Similar to the step shown in figure 13, but this time will choose "Generate edit code" and then "Generate editor code" respectively. This is resulting to a ".editor" plug-in that can be run in a new eclipse instance as eclipse application. In the new runtime eclipse instance, we have the leverage of using the "Example EMF Model Creation Wizards" where we choose our "Webpage Model". Then we can create an object for each of our model classes as shown in figure 14.

Figure 14 Example Of Creating An Object From The Webpage And Category Classes.

In the above example, we straight forward generated an executable website Java code form a website UML class diagram model. There is no employment the platform model here since our targeted platform in a normal personal computer. But if we want to extend this website to be browsed in a mobile platform. The code need to be adjusted to suite for mobile browsing.

To handle this issue we utilize our EAV platform model to update the model mapping with the necessary information to adjust our website to be brows in a mobile environment. Since the EAV platform is already updated with different platform's information up to the concrete library code and instead of start the mapping from scratch, we retrieve the required information from the EAV platform model as described in section 3.2. The required code lies under the library part. So we need to retrieve the mobile browsing requirement as per below code, by retrieving the content of the mobile library API from EAV PM.

```
SELECT VALUE_ FROM (
SELECT * FROM EAV_PM A
      WHERE A.Entity =
'Platform.software.Library.JavaLib.Mobile.M
obileBrowes'
         AND
      A.ATTRIBUTE =
'LIB.Type.Smartphone.General'
        AND
      A.VALUE_ =
'API.Platform.software.Library.JavaLib.Mobi
le.MobileBrowes') AS B
WHERE
 B.ATTRIBUTE = 'LIB.API.Content'
```

The below code is a partial API content for mobile browsing, resulting of the above query.

```
public class SmartPhoneBrowsingInfo
{
   //Stores some info about the browser and
device.
   private String BrowerAgentInfo;

   //Stores info about what content formats
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
the browser can display.
   private String httpAccept;

   //    strings that list out smart phone
device's capabilities.

 public static final String deviceType =
"SmartphoneBrand";

   //The constructor. Initializes several
default variables.
   public SmartPhoneBrowsingInfo(String
BrowerAgentInfo, String httpAccept) {
        if (BrowerAgentInfo != null) {
            this.BrowerAgentInfo =
BrowerAgentInfo.toLowerCase();
        }
        if (httpAccept != null) {
            this.httpAccept =
httpAccept.toLowerCase();
        }
    }

   //*************************
   //Returns the contents of the broweser
Agent value, in lower case.
   public String getBrowerAgentInfo()
   {
       return BrowerAgentInfo;
   }

   //*************************
   // Detects if the current device is a
SmartphoneBrand.
   public boolean detectSmartphoneBrand ()
   {
        if (BrowerAgentInfo.indexOf(device
SmartphoneBrand) != -1 &&
!detectsmartphone()) {
            return true;
        }
        return false;
   }}
```

The ".Impl" files need to be updated by the above code. This code will be imported as an API retrieved from our "EAV_PM" platform model. The following code illustrate how this update will take place in the ".Impl" files starting with the "spl.mda.emf.webpage.model.webpage.impl".
Below is a sample code of the ".Impl" file, after updating the mobile browsing information (see the highlighted code).

```
package
spl.mda.emf.webpage.model.webpage.impl;


import
eav_pm.lib.javalib.mobile.mobilebrowse.Smar
tPhoneBrowsingInfo

import java.util.Calender;
import org.eclipse.emf.ecore.EAttribute;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EDataType;
import org.eclipse.emf.ecore.EPackage;
import org.eclipse.emf.ecore.EReference;

import
org.eclipse.emf.ecore.impl.EPackageImpl;

import
spl.mda.emf.webpage.model.webpage.Article;
import
spl.mda.emf.webpage.model.webpage.Category;
import
spl.mda.emf.webpage.model.webpage.Web;
import
spl.mda.emf.webpage.model.webpage.Webpage;
import
spl.mda.emf.webpage.model.webpage.WebpageFa
ctory;
import
spl.mda.emf.webpage.model.webpage.WebpagePa
ckage;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model
<b>Package</b>.
 * <!-- end-user-doc -->
 * @generated
 */
public class WebpagePackageImpl extends
EPackageImpl implements WebpagePackage {

          private EClass webpageEClass =
null;


          private EClass webEClass = null;


          private EClass categoryEClass =
null;


          private EClass articleEClass =
null;

          /**
           * <!-- begin-user-doc -->
           * <!-- end-user-doc -->
           * @generated
           */
          private EDataType
calenderEDataType = null;

          /**
           * Creates an instance of the
model <b>Package</b>, registered with
           * {@link
org.eclipse.emf.ecore.EPackage.Registry
EPackage.Registry} by the package
           * package URI value.
           * <p>Note: the correct way to
create the package is via the static
           * factory method {@link #init
init()}
, which also performs
           * initialization of the package,
or returns the registered package,
           * if one already exists.
           * <!-- begin-user-doc -->
           * <!-- end-user-doc -->
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
          * @see
org.eclipse.emf.ecore.EPackage.Registry
          * @see
spl.mda.emf.webpage.model.webpage.WebpagePa
ckage#eNS_URI
          * @see #init()
          * @generated
          */
         private WebpagePackageImpl() {
                  super(eNS_URI,
WebpageFactory.eINSTANCE);
         }

         /**
          * <!-- begin-user-doc -->
          * <!-- end-user-doc -->
          * @generated       */
// Initialize classes and features; add
operations and parameters

         initEClass(webpageEClass,
Webpage.class, "Webpage", !IS_ABSTRACT,
!IS_INTERFACE,
IS_GENERATED_INSTANCE_CLASS);

         initEAttribute(getWebpage_Name(),
ecorePackage.getEString(), "name", null, 0,
1, Webpage.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEAttribute(getWebpage_Title()
, ecorePackage.getEString(), "title", null,
0, 1, Webpage.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEAttribute(getWebpage_Descrip
tion(), ecorePackage.getEString(),
"description", null, 0, 1, Webpage.class,
!IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEAttribute(getWebpage_Keyword
s(), ecorePackage.getEString(), "keywords",
null, 0, 1, Webpage.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEReference(getWebpage_Categor
ies(), this.getCategory(), null,
"categories", null, 0, -1, Webpage.class,
!IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
IS_COMPOSITE, !IS_RESOLVE_PROXIES,
!IS_UNSETTABLE, IS_UNIQUE, !IS_DERIVED,
IS_ORDERED);

                  initEClass(webEClass,
Web.class, "Web", !IS_ABSTRACT,
!IS_INTERFACE,
IS_GENERATED_INSTANCE_CLASS);

         initEAttribute(getWeb_Name(),
ecorePackage.getEString(), "name", null, 0,
1, Web.class, !IS_TRANSIENT, !IS_VOLATILE,
IS_CHANGEABLE, !IS_UNSETTABLE, !IS_ID,
IS_UNIQUE, !IS_DERIVED, IS_ORDERED);

         initEAttribute(getWeb_Title(),
ecorePackage.getEString(), "title", null,
0, 1, Web.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEAttribute(getWeb_Description
(), ecorePackage.getEString(),
"description", null, 0, 1, Web.class,
!IS_TRANSIENT, !IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEAttribute(getWeb_Keywords(),
ecorePackage.getEString(), "keywords",
null, 0, 1, Web.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEReference(getWeb_Pages(),
this.getWebpage(), null, "pages", null, 0,
-1, Web.class, !IS_TRANSIENT, !IS_VOLATILE,
IS_CHANGEABLE, IS_COMPOSITE,
!IS_RESOLVE_PROXIES, !IS_UNSETTABLE,
IS_UNIQUE, !IS_DERIVED, IS_ORDERED);


         initEClass(categoryEClass,
Category.class, "Category", !IS_ABSTRACT,
!IS_INTERFACE,
IS_GENERATED_INSTANCE_CLASS);

         initEAttribute(getCategory_Name()
, ecorePackage.getEString(), "name", null,
0, 1, Category.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEReference(getCategory_Articl
es(), this.getArticle(), null, "articles",
null, 0, -1, Category.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE, IS_COMPOSITE,
!IS_RESOLVE_PROXIES, !IS_UNSETTABLE,
IS_UNIQUE, !IS_DERIVED, IS_ORDERED);


         initEClass(articleEClass,
Article.class, "Article", !IS_ABSTRACT,
!IS_INTERFACE,
IS_GENERATED_INSTANCE_CLASS);

         initEAttribute(getArticle_Name(),
ecorePackage.getEString(), "name", null, 0,
1, Article.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);

         initEAttribute(getArticle_Created
(), this.getcalender(), "created", null, 0,
1, Article.class, !IS_TRANSIENT,
!IS_VOLATILE, IS_CHANGEABLE,
!IS_UNSETTABLE, !IS_ID, IS_UNIQUE,
!IS_DERIVED, IS_ORDERED);
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
                    // Initialize data types

          initEDataType(calenderEDataType,
Calender.class, "calender",
IS_SERIALIZABLE,
!IS_GENERATED_INSTANCE_CLASS);

                    // Create resource
                    createResource(eNS_URI);
          }

} //WebpagePackageImpl
```

Now the ".genmodel" need to be reload after the above updates (see figure 15). After we reload the new updated model, the webpage can be regenerated. But this time the website is suite to be browse in a mobile environment.

Figure 15 shows how to reload the ".genmodel"

It worth to mention that the changes was only limited to the ".genmodel", while the ".ecore" model remain with no changes. This operation can be fully automated process with minimal manual interference.

6. DISCUSSION AND RESULTS

In this work we explicitly used a platform model designed by EAV concept to support the mapping from PSM model to Java.

EAV act as a means of knowledge representation that we utilizes to design an EAV platform model. The advantage of EAV representation over the ontology platform representation is that the open structure of EAV allows for representing multiple platforms, with different level of details up to the code level. This is beside the flexibility and the dynamicity of EAV structure in defining and representing any upcoming new platforms or modifying the current one.

On the other hand, the possibility of presenting an EAV platform model in XML format is giving an integration room with different systems, tools and software engineering approaches. For example EAV platform can be used as a core asset repository in any software product line. The platform information and APIs can be stored and retrieved from it. Consequently, Both MDA and software product line approaches can share this platform model as a centric area of collaboration, integration and information exchange. In reverse the core assets in the software product line can be stored in the EAV Platform model, adopting the same semantic presented in this work. This will positively reflect on the number of platforms that the model mapping can support.

From other prospective the possibility of having the EAV-PM in XML can allow for an automating update to its repository by the platform vendor's website feeds (RSS). This way the platform model will be up to date with new platform releases or versions. Consequently, the software productivity will rapidly increase the support to a great number of platforms with minimal changes and cost. While the coding effort can be shifted in more architectural and design work to improve the software quality.

On top of the above, the explicit employment of EAV-PM provide a great support to the model evolution: with proper mechanisms and tools like the EMF in place, there is a flexibility to ensure that models will work, open in editors, produce the code etc. with the newer metamodel too (e.g. updates automatically the models to the new metamodel).

There are also other advantages like faster metamodel/language development, easier management, possibility to couple various generators based on the metamodel together, etc.

The limitation of the EAV-PM is inherited from EAV representation drawbacks. Where a considerable up-front programming is needed to do many tasks that a conventional architecture would do automatically. Moreover, such programming needs to be done only once, and availability of generic EAV tools could remove this limitation. Also, for bulk retrieval EAV design is considered less efficient than a conventional structure. Consequently, performing complex attribute-centric queries, which are based on the values of attributes, and returning a set of objects is both significantly less efficient as well as technically more difficult.

7. CONCLUSION AND FUTURE WORK

Computer, platforms are combining a set of features and component that allow to formally control different functions and contexts (Hardware/software activities) that normally limited to a particular stream of technology. Having a capability of representing multiple platforms for different technologies in different levels of abstraction, in a single EAV-PM repository, is dramatically increasing the quality and the productivity of the software development process.

The EAV-PM introduced in this work as a novel approach that explicitly employed to support the mapping from PSM models to two different Java environment. The EMF used for modeling, mapping and code generation. The Website factory

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

case study presented to show the MDA capability in producing end to end software product. The website designed for desktop browsing. However, we put the EAV-PM in action to update the model mapping by retrieving the mobile browsing information from EAV platform model. The model reloaded and regenerated in a mobile browse friendly version with minimal interference and modification.

In the near future the focus will be on the automatic update of the EAV-PM. Where the Domain Specific Language will be explored as a solution to produce a user friendly interface that allow easy update to EAV-PM model. Also, we intend to create a plug-in to automatically populate EAV-PM repository from the vendor's website feeds.

More case studies will be implemented to target different platforms other than java for more testing and to generalization.

REFRENCES:

[1] Frankel, D.S., *MODEL DRIVEN ARCHITECTURE APPLYING MDA*. 2003: Wiley. com.

[2] OMG Document: formal/01-11-02, *OMG: Meta Object Facility (MOF) v1.3.1*, in *OMG Document: formal*2001.

[3] 3. formal/01-09-67, O.D., *OMG: Unified Modeling Language v1.4. OMG Document: formal/01-09-67, Sept. 2001.*, 2001.

[4] Mellor, S.J., *MDA distilled: principles of model-driven architecture*. 2004: Addison-Wesley Professional.

[5] Henderson-Sellers, B. and C. Gonzalez-Perez, *Multi-Level Meta-Modelling to Underpin the Abstract and Concrete Syntax for Domain-Specific Modelling Languages*, in *Domain Engineering*. 2013, Springer. p. 291-316.

[6] Ráth, I., A. Ökrös, and D. Varró, *Synchronization of abstract and concrete syntax in domain-specific modeling languages.* Software & Systems Modeling, 2010. **9**(4): p. 453-471.

[7] Son, H.S., W.Y. Kim, and R.Y.C. Kim, *Concretization of the Structural and Behavioral Models based on model Transformation Paradigm for Heterogeneous Mobile Software.* development, 2013. **10**: p. 15.

[8] Hoisl, B., et al., *A Catalog of Reusable Design Decisions for Developing UML-and MOF-based Domain-Specific Modeling Languages.* 2012.

[9] Völter, M., et al., *Model-driven software development: technology, engineering, management*. 2013: John Wiley & Sons.

[10] Soares, I.W., et al., *Modeling of embedded software on MDA platform models.* Journal of Computer Science & Technology, 2012. **12**.

[11] Wagelaar, D. and R. Van Der Straeten, *Platform ontologies for the model-driven architecture.* European Journal of Information Systems, 2007. **16**(4): p. 362-373.

[12] Noy, N.F. and M.A. Musen. *Algorithm and tool for automated ontology merging and alignment*. in *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00). Available as SMI technical report SMI-2000-0831*. 2000.

[13] Raunich, S. and E. Rahm. *ATOM: Automatic target-driven ontology merging*. in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. 2011. IEEE.

[14] Antonkulaga, *Limitations of Semantic Web; Ontology.* [Online] http://denigma.de/data/entry/limitations-of-semantic-web-ontology, 2013.

[15] Flahive, A., D. Taniar, and W. Rahayu, *Ontology as a service (OaaS): a case for sub-ontology merging on the cloud.* The Journal of Supercomputing, 2013. **65**(1): p. 185-216.

[16] ELSAWI, A.M., S. SAHIBULDIN, and A. ABDELHADI, *INTRODUCING THE OPEN SOURCE METAMODEL CONCEPT.* Journal of Theoretical & Applied Information Technology, 2013. **57**(3).

[17] ELSAWI, A.M., S. SAHIBUDDIN, and A. ABDELHADI, *PROPOSE AN INTEGRATION BETWEEN UML STATIC AND DYNAMIC MODELS USING ENTITY-ATTRIBUTEVALUE UNDER THE MDA CONTEXT.* Journal of Theoretical & Applied Information Technology, 2014. **68**(1).

[18] Lédeczi, Á., et al., *Composing domain-specific design environments.* Computer, 2001. **34**(11): p. 44-51.

[19] Tolvanen, J.-P. and M. Rossi. *MetaEdit+: defining and using domain-specific modeling languages and code generators*. in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. 2003. ACM.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

.

APPENDICES

APPENDIX A:

State
Name
WishTravel
Completed
HoldRes
ReadyTravel
WBoardCard

Event
Name
reservation
reschedual
reqCheckIn
checkIn
complete
urgeFly

Transition		
Source	*Target*	*Triggeredby*
WishTravel	HoldRes	reservation
HoldRes	ReadyTravel	reqCheckIn
ReadyTravel	HoldRes	reschedual
ReadyTravel	WBoardCard	checkIn
WBoardCard	Completed	complete
Completed	WishTravel	urgeFly
WishTravel	HoldRes	reservation

Appendix A: Airline Passenger State Model Of Figure 1 Represented As A Conventional Database Population

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

APPENDIX B:

ENTITY	ATTRIBUTE	VALUE_
EVENT	NAME	checkIn
EVENT	NAME	Complete
EVENT	NAME	reqCheckIn
EVENT	NAME	Reschedule
EVENT	NAME	Reservation
EVENT	NAME	urgeFly
STATE	NAME	Completed
STATE	NAME	HoldRes
STATE	NAME	ReadyTravel
STATE	NAME	WBoardCard
STATE	NAME	WishTravel
TRANSITION	SOURCE	Completed
TRANSITION	SOURCE	HoldRes
TRANSITION	SOURCE	ReadyTravel
TRANSITION	SOURCE	ReadyTravel
TRANSITION	SOURCE	WBoardCard
TRANSITION	SOURCE	WishTravel
TRANSITION	TARGET	Completed

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TRANSITION	TARGET	HoldRes
TRANSITION	TARGET	HoldRes
TRANSITION	TARGET	ReadyTravel
TRANSITION	TARGET	WBoardCard
TRANSITION	TARGET	WishTravel
TRANSITION	TRIGGEREDBY	checkIn
TRANSITION	TRIGGEREDBY	complete
TRANSITION	TRIGGEREDBY	reqCheckIn
TRANSITION	TRIGGEREDBY	reschedule
TRANSITION	TRIGGEREDBY	reservation
TRANSITION	TRIGGEREDBY	urgeFly

Appendix B: Airline Passenger State Model Of Figure 1 Represented In EAV Database Population

APPENDIX C:

ENTITY	ATTRIBUTE	VALUE_
Metamodel	ID	1
Metamodel	Name	State Machine
Metamodel.Element	ID	1.1.1.1
Metamodel.Element	Name	NamedElement
Metamodel.Element.NamedElement	DataType	String
Metamodel.Element.NamedElement	Attribute	Name
Metamodel.Element.NamedElement.EVENT	NAME	checkIn
Metamodel.Element.NamedElement.EVENT	NAME	complete
Metamodel.Element.NamedElement.EVENT	NAME	reqCheckIn
Metamodel.Element.NamedElement.EVENT	NAME	reservation
Metamodel.Element.NamedElement.EVENT	NAME	urgeFly
Metamodel.Element.NamedElement.EVENT	NAME	Completed
Metamodel.Element.NamedElement.EVENT	NAME	HoldRes
Metamodel.Element.NamedElement.EVENT	NAME	ReadyTravel
Metamodel.Element.NamedElement.EVENT	NAME	WBoardCard
Metamodel.Element.NamedElement.EVENT	NAME	WishTravel
Metamodel.Element.NamedElement.NAME	NAME	Completed
Metamodel.Element.NamedElement.NAME	NAME	HoldRes

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Metamodel.Element.NamedElement.NAME	NAME	ReadyTravel
Metamodel.Element.NamedElement.NAME	NAME	WBoardCard
Metamodel.Element.TRANSITION	SOURCE	Completed
Metamodel.Element.TRANSITION	SOURCE	HoldRes
Metamodel.Element.TRANSITION	SOURCE	ReadyTravel
Metamodel.Element.TRANSITION	SOURCE	WBoardCard
Metamodel.Element.TRANSITION	SOURCE	WishTravel
Metamodel.Element.TRANSITION	TARGET	Completed
Metamodel.Element.TRANSITION	TARGET	HoldRes
Metamodel.Element.TRANSITION	TARGET	HoldRes
Metamodel.Element.TRANSITION	TARGET	ReadyTravel
Metamodel.Element.TRANSITION	TARGET	WBoardCard
Metamodel.Element.TRANSITION	TARGET	WishTravel
Metamodel.Element.TRANSITION	TRIGGEREDBY	checkIn
Metamodel.Element.TRANSITION	TRIGGEREDBY	complete
Metamodel.Element.TRANSITION	TRIGGEREDBY	reqCheckIn
Metamodel.Element.TRANSITION	TRIGGEREDBY	reschedule
Metamodel.Element.TRANSITION	TRIGGEREDBY	reservation
Metamodel.Element.TRANSITION	TRIGGEREDBY	urgeFly

Appendix C: Fragment Of EAV Representation To Models In Figure 1 And Figure 2