

Generating Web Applications from UML Domain Models: A Model-Driven Approach Using Profiles and MetaModels

Juan José Cadavid¹, David Lopez², Jesús Andrés Hincapié², Carlos Andrés Ospina³, Juan Bernardo Quintero¹

¹ EAFIT University, ² Archetypus Inc., ³ ABCFlex Ltda, Medellín, Colombia
{jcadav10, jquinte1}@eafit.edu.co, {jahincapie, delopez}@archetypus.net,
caospina@abcflex.net

Abstract. The rise and potential of the World Wide Web as a way to ease life have made the development of web applications one of the most widely known topics on software engineering, which is the reason why we find a large number of tools and platforms to support web development. However, building a web application is still a complex process that requires a big effort to get several tasks done. This article presents a generative framework aimed to simplify web application development, based on the construction and transformation of high level models from a domain structure viewpoint, using a UML profile plus a Domain Specific Language built for that purpose.

Keywords: Metamodels, UML Profiles, Web applications, MDA, MDSD, Model Engineering, Model Transformations, Domain Specific Languages, Web Development, Web Development Tools.

1 Introduction

Despite several tools and platforms that exist nowadays to support Web development, building a Web application is still a complex process that requires a big effort to get several tasks done. Over past years a form of engineering called model-driven engineering, by which all or at least central parts of software application are generated from models, has arise leading to the construction of software tools that helps building applications with a model-driven development approach.

Model-driven software development (MDSD) is a software development paradigm with roots in software product line engineering, which is the discipline of designing and building families of applications for a specific purpose or market segment [1]. MDSD unlike software product line engineering emphasizes on a highly agile software development process. One of the highest priorities in MDSD is to produce working software that can be validated by end users and stakeholders as early as possible. MDSD also includes several topics such as domain specific languages, model to model transformations, template languages, code generation among others that contribute to the goal of making models the way of building software and not just documentation.

The work described here presents a generative framework aimed to simplify Web application development, based on the construction and transformation of high level models from a domain structure viewpoint, using a UML profile plus a Domain Specific Language built for that purpose. Such framework consists of an adaptation of a transformation strategy, a profile that permits marking a high level domain model, a metamodel that describes the most relevant element of Web applications, and the definition and application of transformations that allow generating the code of the Web application.

This work is part of a project called **Model Transformation Environment (MTE)** that pretends to increase the quality and productivity of software development, focusing on the construction and transformation of models, so some activities that are done manually, can be automated. It also pretends to get the developer to be more focused on the business rather than technology, thus leading to a faster software development.

The remaining part of this paper is as follow: section 2 describes the selected transformation strategy. Section 3 shows the Web Application Profile (**WebApp Profile**) and describes how it is used to mark models. Section 4 presents the **Web Application MetaModel (WAMM)**, explaining its elements and its function in the transformation process. Section 5 explains how to transform a domain model marked with the profile to a WAMM instance. Section 6 describes the process of converting a WAMM instance to actual code of a Web application. Section 7 shows an example of the transformation process. Section 8 presents some conclusion and further work.

2 Selecting a Transformation Approach

There are two approaches of model transformation in MDA: the elaborationist [2] and the translationist[3]; in the former a developer can build the PIM, the PSM, and the code; in the latter the developer can build the PIM, which is translated directly into code. The main difference of these approaches is the level where a developer can change the functionality of the system.

The selected strategy for the generation of web applications is the elaborationist approach where the roles of the PIM, the PSM and the code are the following:

- **PIM:** The Platform Independence Model is a UML domain model, which can be marked with the stereotypes of a profile to indicate how the web application is going to be.
- **PSM:** The Platform Specific Model corresponds to a UML class model with the necessary elements to develop a web application with appropriated human computer interaction directives.
- **Code:** This item refers to the source code of web applications. In this case the target platforms are based on PHP or JEE.

This transformation approach includes two phases: the first is a PIM to PSM (M2M) transformation and the second is a PSM to Code (M2T) transformation. In order to complete this phases is important to use two artifacts: a UML profile and a metamodel to provide elements that are essential in the human computer interactions of web applications.

3 Web Application Profile (WebApp Profile)

A UML Profile is a mechanism to extend the semantic of models built with UML in a bounded area of knowledge or interest. In other words, a UML Profile is like a special type of metamodel, whose classes specialize UML metaclasses.

WebApp Profile is a UML profile for web applications that offers a mechanism to mark the UML domain model in order to provide a good Human Computer Interaction [4] to generated applications. The following sections show the main concepts required to generate web applications from UML structure models.

3.1 Form Stereotype (<<Form>>)

This stereotype applies to classes and is used when there is the need of manipulating in a web form the information of a single record based on the marked class.

Tagged Values:

- **CRUD:** [CRUD | CRU | CRD | RUD | CR | RU | RD | R] defines combinations for data manipulation actions in the form (Create, Read, Update and Delete).
- **Navigation order:** string that specifies the navigation flow of the form in the application.
- **Specific properties:** string listing the class properties shown for every record in the reading form. Each field is derived from one property.

Effect:

- With this stereotype a web form is created with a field for each property to manipulate a single record. Table 1 shows the effect in the web form of every letter in the CRUD tag.
- Navigation order is a sequence of dot-separated integers. An integer with no dots specifies a root in the navigation flow. Several integers separated by dots specify a lower node in the navigation flow, derived from the upper node identified by the same Navigation order without the last integer. For example: the 4.3.1 is a node immediately lower than the 4.3 node.

Table 1. Description of the operations for the Form stereotype tagged value CRUD

	Created controls	Control effects	Considerations
C	<u>Insert Action</u> : a “+” icon or “new” button <u>Save Action</u> : a diskette icon	<u>Insert</u> : to put in “ <i>Insertion Mode</i> ” cleaning the form <u>Save</u> : to save the values in a new record.	If the primary key is auto increment, then those properties cannot be inserted.
R	<u>Search Action</u> : a binocular icon	This icon is put next to the primary key field, to find a record that matches the value inserted in that field.	If the letter U is present in the CRUD tag, then when a record is retrieved, the form is set to “ <i>Updating Mode</i> ”.
U	<u>Save Action</u> : a diskette icon	To save the changes made on a record.	If there are no pending changes, then this control is unavailable
D	<u>Delete Action</u> : a “x” icon	To delete the current record.	A confirmation message is shown before deleting the record.

- If the value of Specific properties is not set, then all properties are displayed.

Constraints:

- In Navigation order, there must be a node in the navigation flow corresponding to the upper node in the navigation flow. For example: if Navigation order is 1.2.1.1 there must be a stereotyped element marked with 1.2.1 in the Navigation order tag.
- Elements listed at Specific properties must exist as properties of the class marked with this stereotype.



Fig. 1. Simple form generated form with the app profile.

3.2 List Stereotype (<<List>>)

This stereotype applies to classes and is used when there is the need of manipulating in a web form the information of multiple records based on the marked class.

Tagged Values:

- CRUD: [CRUD | CRU | CRD | RUD | CR | RU | RD | R] defines combinations for data manipulation actions in the record list (Create, Read, Update and Delete).
- Navigation order: string that specifies the navigation flow of the list in the application.
- Specific properties: string listing the class properties shown in the record list in the Reading Form.
- Records by page: number of records shown in every page of the list.

Effect:

- With this stereotype a Reading Form is created, displaying a list of records, as well as other two web forms for creating and updating a record. Table 2 shows the effect of every letter in the CRUD tag.

Table 2. Description of the operations for the List stereotype tagged value CRUD

	Forms generated	Controls in reading form	Cardinality of controls	Considerations
C	Creation Form	<u>Insert Action</u> : a “+” icon or “new” button	One for each form.	If the primary key is auto increment, then those properties are no shown.
R	Reading Form	A pager with navigation controls	One for each form.	These controls are used to navigate through pages.
U	Updating Form	<u>Update Action</u> : a pencil icon	One for each record.	
D	No form generated	<u>Delete Action</u> : a “x” icon	One for each record.	A confirmation message is shown before deleting the record.

- The Creation and Updating Forms have two controls: Save and Cancel; the first one to save the changes, and the second one to return to the Reading Form.
- If the value of Specific properties is not set, then all properties are displayed.
- The records can be sorted by clicking on the column header in the Reading Form.
- If Records by page is 0 or not set, then records are neither paged nor filtered.

Constraints:

- In Navigation order, there must be a node in the navigation flow corresponding to the upper node in the navigation flow.
- Elements listed at Specific properties must exist as properties of the class marked with this stereotype.
- Records by page must be greater than or equal to 0.
- If a class is marked with the stereotype Form, then it cannot be marked with the stereotype List.



Fig. 2. List form generated with the app profile.

3.3 Master Detail Stereotype (<<M-D>>)

This stereotype applies to associations and is used to create a web page with a master detail form to manipulate information involved in the two associated classes. When a navigation action change the current record in the master block, then the record or records in the detail are changed too.

Tagged Values:

- Navigation order: string that specifies the navigation flow of the master detail in the application.
- Master layout: [Form, List] define the presentation of the master block.
- Master CRUD: [CRUD | CRU | CRD | RUD | CR | RU | RD | R] defines combinations for data manipulation actions in the master block.
- Master specific properties: string listing the class properties shown in the master block form.
- Master records by page: number of records shown in the master block.
- Master deleting behavior: [Cascade, Isolated] define the action to carry on the detail block when a record is deleted in the master block. “Cascade” delete records and “Isolated” avoid this action until all record in the detail are deleted.
- Detail CRUD: [CRUD | CRU | CRD | RUD | CR | RU | RD | R] defines combinations for data manipulation actions in the detail block.

- Detail specific properties: string listing the class properties shown in the detail block form.
- Detail records by page: number of records shown in the detail block.

Effect:

- The master block corresponds to the class with multiplicity 0 or 1 in the association, and the detail block corresponds to class with multiplicity 0..* or 1..* in the association.
- The effects of the Master CRUD tags depend on the value of Master layout. All effects of Master CRUD tag applies to the Master block in the same way that CRUD tag applies to “Form” and “List” stereotypes.
- For the Detail block a “List” layout is assumed. All effects of Detail CRUD tag applies to Detail in the same way that CRUD tag applies to the “List” stereotype.
- If the value of Master specific properties is not set, then all master properties are displayed.
- If the value of Detail specific properties is not set, then all detail properties are displayed except the foreign key.
- If Master layout is “Form”, then the Master records by page tagged value has no effect.
- If Master records by page value is 0 or not set, then master records are not paged.
- If Detail records by page value is 0 or not set, then detail records are not paged.

Constraints:

- This stereotype is only accepted in associations between classes with multiplicity 0 or 1 and multiplicity 0..* or 1..*.
- In Navigation order, there must be a node in the navigation flow corresponding to the upper node in the navigation flow.
- If Master CRUD has not “D”, then Master deleting behavior must be not set.
- The elements listed at Master specific properties must exist as properties of the class that represents the master block in the relationship.
- The elements listed at Detail specific properties must exist as properties of the class that represents the detail block in the relationship.
- Master records by page must be greater than or equal to 0.
- Detail records by page must be greater than or equal to 0.



Fig. 3. Master-detail form generated with the app profile.

3.4 Lookup Stereotype (<<Lookup>>)

This stereotype applies to associations and is used to get a field from a list of dynamic values associated to another class.

Tagged Values:

- Lookup type: [Pop-up window, Select list] it defines if the list of values is shown in an input select or in a new window with a list of records to select one record.
- Specific properties: comma-separated string listing the class properties shown in the records list.

Effect:

- The field that will be selected corresponds to a property from the class with multiplicity 0 or 1 in the association. If Lookup type is “Pop-up window” a magnifier icon is put next to this property to open a new window with the list.
- The list of values corresponds to records associated to the class with multiplicity 0..* or 1..* in the association.
- If the value of Specific properties is not set, then all properties are displayed

Constraints:

- This stereotype is only accepted in association between classes with multiplicity 0 or 1 and multiplicity 0..* or 1..*.
- Elements listed at Specific properties must exist as properties of the class with multiplicity 0..* or 1..* in the association.

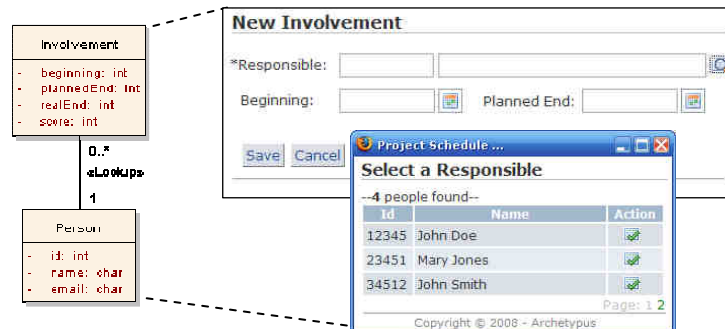


Fig. 4. Pop-up window generated with the app profile.

3.5 Defined Selection Stereotype (<<D-S>>)

This stereotype applies to properties and is used to get a field from a list of static values, which has its source in a set of pre-defined values.

Tagged Values:

- Selection type: [Check box, Radio group, Select list] defines the presentation of pre-defined values from which the field is selected.
- Code and value collection: enumeration name with the list of values.

Effect:

- A static list of values is created to select the field value depending on the Selection type.

- There are two types of enumerations: one with just values, for example payment type: credit and cash; and other with code and value, for example gender: f, female and m, male.

Constraints:

- If Selection type is a check box, then the size of the code and value collection must be two.
- Code and value collection must exist like an enumeration in the same package or namespace of the class that owns the property marked with this stereotype.



Fig. 5. Defined selection generated with the app profile.

3.6 Primary Key(<<PK>>)

This stereotype applies to properties or associations and is used to avoid repeated values in a field or in a field group, composing a unique identification.

Tagged Values:

- Auto increment: Boolean that defines if the key is auto increment or not.
- Key order: defines the field order in the index of the primary key.

Effect:

- A primary key constraint is created in the table based on the class corresponding to the property marked with this stereotype or with multiplicity 0..* or 1..* in the association.
- If an association is marked with this stereotype, then the columns derived from the association in the class at the end with 0..* or 1..* compose the primary key.
- If Auto increment is true, then a constraint or trigger is created to guarantee it, and this property is not displayed in a creation form.
- For a class several properties can be marked with this stereotype, including the property corresponding to the association, in this way a composite primary key is generated.

Constraints:

- If this stereotype is applied to an association, then it can only be applied to one between classes with multiplicity 0 or 1, and multiplicity 0..* or 1..*.
- An association can only be marked if the class at the end with multiplicity 0 or 1 has a Primary key stereotype.
- Auto increment can only be set for a single primary key; for composite ones it must be false or not set.

3.7 WebApp Profile Representation

All the concepts and ideas described above are represented in the profile diagram shown in Fig. 6. This diagram shows the WebApp profile stereotypes and their relations with the UML metaclasses; it gives the basis for marking the classes of any user domain model.

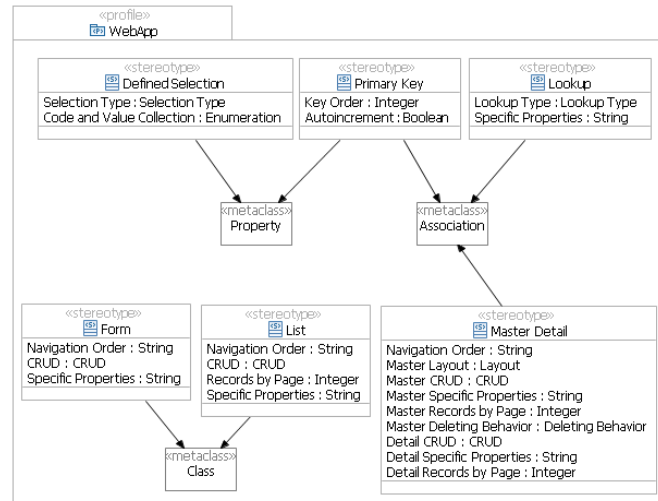


Fig. 6. WebApp profile representation

4 Web Application MetaModel (WAMM)

In order to support the transformation from the UML domain model marked with the profile elements and tags to the code, an intermediate step must occur. This intermediate step involves merging all the elements needed to generate the code in a single model. “A metamodel defines a consensual agreement on how elements of a system should be selected to produce a given model.”[5]. A metamodel is a simple ontology. “An ontology is an explicit specification of a shared conceptualization”[6]. A metamodel abstracts concepts from an ontology to compose models.

WAMM means Web Application Metamodel, and describes all the global concepts needed to generate a complete web application in an object oriented programming language. WAMM acts as an information holder, because it contains all the information required to generate the application code. It also describes how the elements are related in the application. The use of an intermediate metamodel is suggested as a best practice by Eclipse and JET project [7]. WAMM can be seen as a generic web application platform, which any web application could be defined with. It also can be seen as domain specific language of web applications.

The metamodel can be divided in two parts, a structure part and an application part. The Structure part contains the structure and description of the domain objects and has the information required for generating the database scripts, relational constraints,

models or value objects. The instances of those elements are the domain concepts. The Application part contains the relations between the domain objects and the web UI, this is how the information will be requested and presented to the user. The instances of those elements are related to the presentation. The metamodel can be seen in Fig. 7

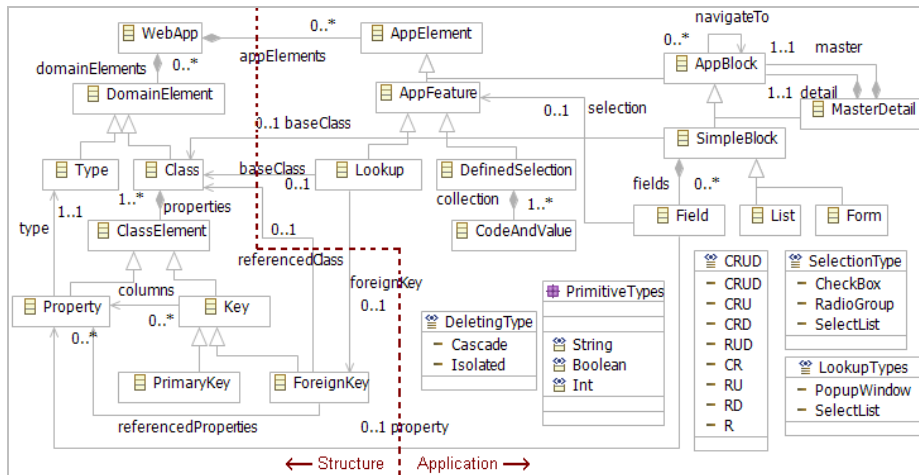


Fig. 7. Diagram of the Web Application Metamodel. Here all the elements of the metamodel are shown. The structure elements are at the left of the red line, and the applications elements at the right.

5. Transform Marked Domain Model to WAMM Instance

According to MDA OMG specification [8], one of the transformation approaches is “marking”. When generating Web applications from UML domain models, it is necessary to adapt this transformation approach.

Fig. 8 shows such adaptation.

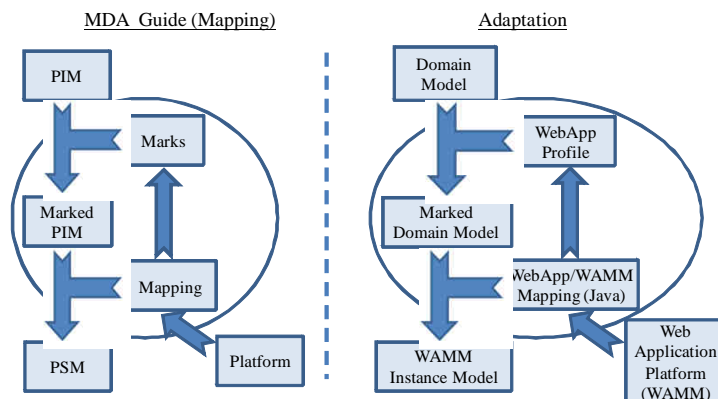


Fig. 8. Adaptation of M2M transformation

A kind of transformation from Model to Model (M2M) is the direct manipulation approach [9]. To transform the marked domain model to a WAMM instance, MTE applies this approach using Java to manipulate the model instance that can be loaded from a XMI file [10].

The start point of this transformation is a UML Domain Model, a visual representation of conceptual classes or real-world objects in a domain of interest [11]. This model is also called conceptual model, domain object model or analysis object model [12]. Although domain models are not models of software components, it is important to stand out that the detail level in the domain model defines the detail level of final web application.

6 Transform WAMM Instance to Code

According to MDA OMG specification [8], another transformation approach is “Metamodel Transformation”. In the second phase of generating Web applications from UML domain models, it is necessary the adaptation of this transformation approach. Fig. 9 shows the main concepts in the adaptation.

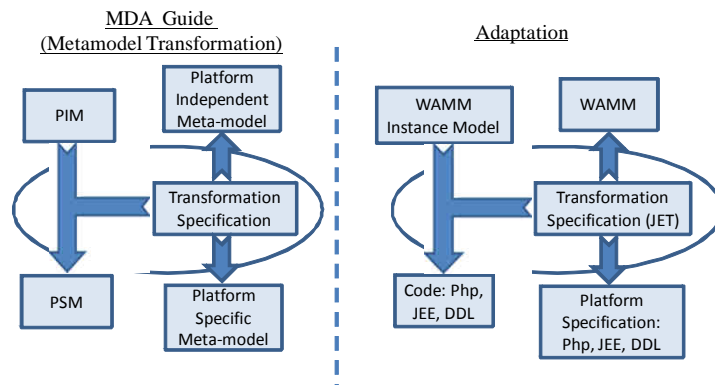


Fig. 9. Adaptation of M2T Transformation

A kind of transformation from Model to Text (M2T) is the Template-Based approach [9]. To transform a WAMM instance to code, MTE applies this approach using the Eclipse project JET [7], writing the transformation from WAMM to code in the JET template language. One main template to control the loading of the WAMM instance and subsequent templates to generate code from each WAMM instance entity.

The adaptation to MDA specification has the objective of converting the WAMM Instance in a web application working in a web browser.

Transforming a WAMM Instance to code might be seen as a task to generate source code similar to what a case tool does [13], the difference lies in the generative architecture of MTE, that is the elements and activities described in this paper that are required to transform a domain model in a web application.

The activity has as a precondition to define a target platform, after that it is necessary to create a software architecture [12] according to the web application in order to generate and select a web server to deploy. With the preconditions, the next task consists in defining the strategy or metalanguage to convert the WAMM Instance in source code of the selected target platform, following the guidelines proposed by software architecture and deployment in the web server selected to expose the application.

With the metalanguage selected, the “end point” consists in metaprogramming [14], specifically write in a template language the code necessary to create the source code of the target platform. The information necessary to build the web application according to the initial domain model is extracted from the WAMM Instance.

7 Applying the Transformation Process

MTE is built with EMF (Eclipse Modeling Framework) [15] using the power of a platform like Eclipse, additionally is integrated with UML2 Tools [16] to load and manipulate UML models. A screenshot of the MTE can be seen in Fig. 10.

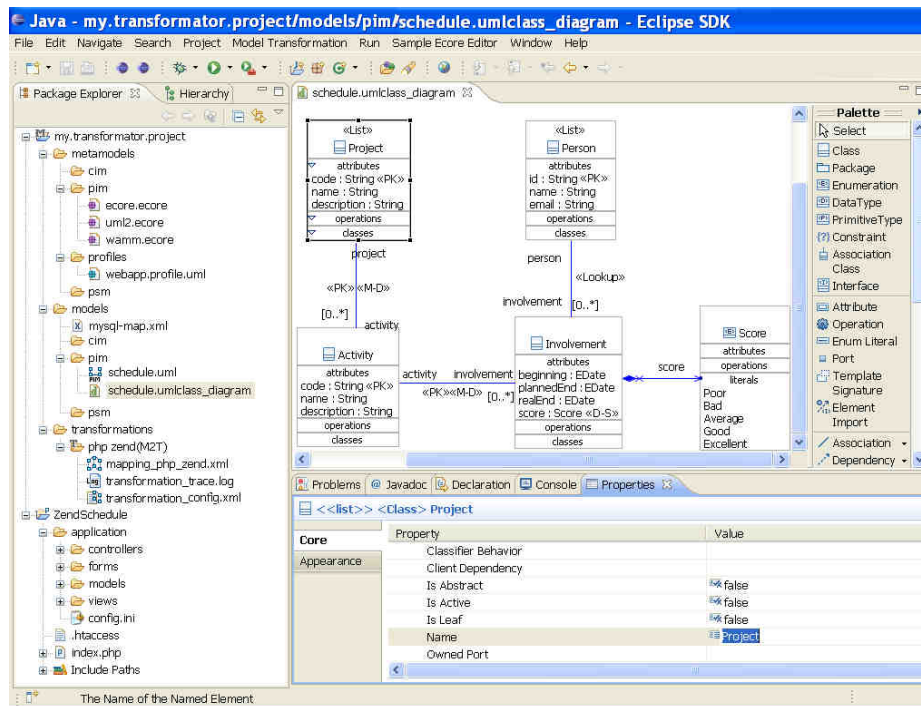


Fig. 10. Screenshot of the Model Transformation Environment (MTE). In the right side of the windows a marked UML domain model of a *Project Schedule* example is opened.

To apply the transformation process it is necessary to execute certain activities in the MTE tool. The first step is to build a UML domain model that represents the basic structure and relationships between identified business entities. The model can be created in MTE or imported from another tool using the XMI 2.1 standard.

The second step is based on the WebApp profile and consists in marking the domain model elements with the profile stereotypes according to the patterns of human-computer interaction each stereotype represents.

The third step is to execute a MTE action that consists in performing a M2M transformation to the marked domain model so a WAMM instance can be obtained. This task is performed by an internal class that implements the transformation logic.

The final step is to perform a M2T transformation in order to convert a WAMM instance to source code. For this purpose two platforms were defined to base the generated Web application on: PHP with Zend Framework [17] and Java Enterprise Edition [18].

Fig. 11 shows a screenshot of the Web application generated from the *Project Schedule* marked UML domain model. The platform selected for deploying this example was PHP with Zend Framework.

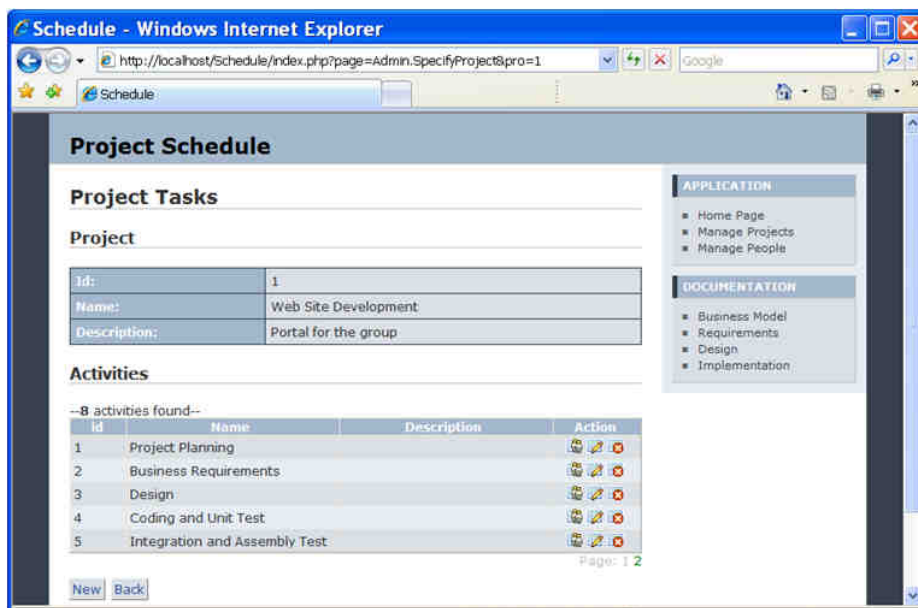


Fig. 11. Screenshot of the generated Web application deployed in a server with PHP and Zend Framework.

8 Conclusions and Further Work

Building software is a discipline that may involve a high level of complexity. Approaches such as those presented in MDSD tend to accelerate significantly the development process. The industrial and academic adoption of software development

paradigms based on the construction and transformation of models will be possible as tools adequately support such MDSO approaches. However proposals such as those experienced with a tool like MTE, allow demonstrating that the promises of these paradigms are now a reality: models can be transformed, programmed, and executed for the automatic generation of applications.

This work has described the elements used in the process of transforming a UML domain model in a deployable Web application. Such elements include an extension of UML by means of a Web application profile called WebApp; a specification of a domain specific language represented in a Metamodel called WAMM; the definition of a M2M transformation in order to transform a domain model, to which the WebApp profile was applied, into an instance of the WAMM Metamodel; and finally the definition of a M2T transformation, so executable code can be generated from an instance of WAMM.

Transformation capabilities of MTE tool are different from case tool source code generation since MTE generates data definition language scripts, the application code is generated according to a target platform and a target architecture, it transforms the domain model to artifact that are previous to code, and it generates the required files to deploy or expose in a server the generated Web application.

The source of the transformation process in MTE is a domain model, which represents one of most relevant structure diagrams in UML. In order to get a complete web application, other UML diagrams like use case or sequence are required. This kind of UML diagrams are used to represent behavior. An important future work is to complement UML domain models with behavior diagrams as sources of the transformation process, in this way the generated web applications can have a fully functionality.

One future plan is to formalize the model to model transformation process according to the OMG QVT (Query – View – Transformation) standard [19], using a specific purpose language of this kind, like ATL [20] or the latest m2m component of the Eclipse community called Operational QVT.

References

1. Völter, M. y Stahl, T. Model-Driven Software Development (Technology, Engineering, Management) ISBN: 978-0-470-02570-3, 444 p, 2006.
2. Kleppe, A. Warmer, J. and Bast, W. MDA Explained: The Model Driven Architecture - Practice and Promise. Addison–Wesley, 2003.
3. Mellor, S. J. and Balcer, M. J. Executable UML: A Foundation for Model Driven Architecture. Addison–Wesley, 2002.
4. Molina, P. “Especificación de interfaz de usuario: De los requisitos a la generación automática.” Universidad Politécnica de Valencia, 2003.
5. Bézivin J., Gérard S., Muller P-A. and Rioux L. "MDA components: Challenges and Opportunities", in: Metamodelling for MDA, York, England, 2003.
6. T. R. Gruber. A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220, 1993.
7. Eclipse Foundation. “JET Tutorial Part 2 (Write Code that Writes Code)”

- http://www.eclipse.org/articles/Article-JET2/jet_tutorial2.html
8. Object Management Group. Model Driven Architecture (MDA) Guide Version 1.0.1. OMG, 2003.
 9. Czarnecki, K y Helsen, S. 2006. Feature-based survey of model transformation approaches. s.l. : IBM Systems Journal. Vol 45, No. 3, 2006.
 10. Object Management Group. XML Metadata Interchange (XMI) Specification. OMG, 2003.
 11. Fowler, M. Analysis Patterns: Reusable Object Models. Reading, MA.: Addison-Wesley, 1996.
 12. Larman, C. Applying UML and Patterns: An introduction to Object analysis and design and the unified process. 2 ed. s.l. : Prentice Hall, 2005. 627 p.
 13. Software Engineering Institute. "Computer-Aided Software Engineering Environments". http://www.sei.cmu.edu/legacy/case/case_what.html
 14. Bartlett, J. The art of metaprogramming, Part 2: Metaprogramming using Scheme. The macro facility can generate code to simplify large projects. <http://www.ibm.com/developerworks/linux/library/l-metapro2.html>
 15. Wolfe, A. Eclipse: A platform becomes an Open-Source Woodstock. ACM Queue. Vol 1, No 8. ACM, 2003.
 16. Eclipse Foundation. Model Development Tools (MDT): UML2 Tools. <http://www.eclipse.org/modeling/mdt/>
 17. Zend Technologies. "Zend Framework Documentation". <http://framework.zend.com/>
 18. Sun Microsystems. "Java EE APIs & Docs". <http://java.sun.com/javaee/>
 19. Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. OMG, 2007.
 20. Eclipse Foundation. M2M. 2008. ATL: Atlas Transformation Language. <http://www.eclipse.org/m2m/atl/>