A Domain Specific Language to Generate Web Applications

Juan José Cadavid[1], David Esteban Lopez[1], Jesús Andrés Hincapié[1], Juan Bernardo Quintero[2]

[1]Archetypus Inc. {jjcadavid, jahincapie, delopez}@archetypus.net , [2] EAFIT University jquinte1@eafit.edu.co

Abstract. Nowadays building a web application is still a complex process that requires a big effort to get several tasks done. This article presents a domain specific language aimed to simplify web application development by using it within a MDSD generation process, based on the construction and transformation of high level models from a domain structure viewpoint. this leads software development to focus on understanding the problem and doing a better business analysis, leaving the construction of the solution to a code generation process, thus reducing time and cost.

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 domain specific language (DSL) aimed to simplify Web application development by transforming high level models from a domain structure viewpoint into code. Such DSL is framed within a whole generation process that also includes a transformation strategy and the definition and application of Model to Model (M2M) and Model to Text (M2T) transformations that allow generating the code of the Web application.

The starting point of this generation process is a UML Domain Model, a visual representation of conceptual classes or real-world objects in a domain of interest [2]. This model is also called conceptual model, domain object model or analysis object model [3]. 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.

The remaining part of this paper is as follows: section 2 describes the role of DSLs within MDSD. Section 3 explains the semantic of the DSL showing the meaning of each element in a Web application. Section 4 presents the **W**eb **A**pplication **M**eta**M**odel (**WAMM**), explaining its elements and its function in the transformation process. Section 5 presents the WebApp profile which plays the role of the concrete syntax of the DSL. Section 6 describes the process of generating a Web application when using the DSL depicted in this paper for that purpose. Section 7 presents conclusions and further work.

2 DSL role in Model Driven Software Development

A Domain Specific Language (DSL) is a language designed to represent or solve problems in a particular well-defined domain; that is, instead of a general purpose language, it is one that captures precisely the semantics of a given domain. This fact, as seen in the field of software development, presents DSLs as tools that help to close the semantic gap between a problem and an application.

DSLs are not just tools in a text form for modeling elements of a specific domain; they can also be represented in a graphical form. This kind of representation makes easier to understand the concepts of the concerning domain and the relationships between them, providing a more intuitive and natural way to model these concepts, in order to include them in the development process of a software application.

Völter and Stahl define as the three core concepts of a DSL: Semantics, Meta Model and a Concrete Syntax; in the following sections these concepts will be reviewed to explain how they were applied when the building a DSL to generate web applications.

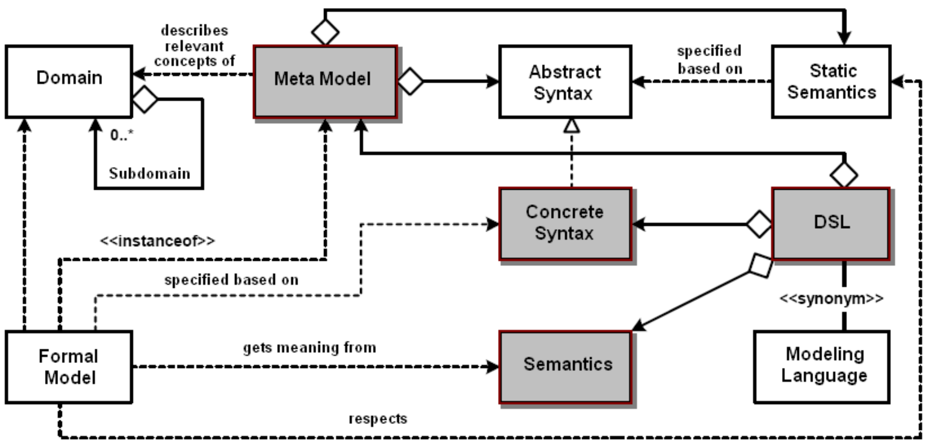


Fig. 1. A DSL as an aggregation of Semantics, Meta Model and Concrete Syntax in the MDSD context, adapted from [1]

3 Semantics for Web Applications

The semantics component of a DSL refers to meaning of web application elements that allows providing a well fit Human-Computer Interaction [4] to generated applications. The following table show the main concepts required to generate web applications from UML structure models.

Table 1. Description of the Semantics for Web Applications

Concept	Explanation
Web Form	Along Web List, it's one of the most basic interaction components in web applications. Typically used to view and manipulate the data associated to a specific instance of an object. The web developer picks which specific attributes of a domain object are shown, and which are editable.
Web List	Along Web Form, it's one of the most basic interaction components in web applications. Typically used to visualize a list of several instances of a domain object. It provides actions to create additional instances, modify the existing ones, delete them, etc. according to the web developer's choice.
Master Detail	Commonly found in applications whose domain model contains aggregation relationships ("one to many") between two business objects in the domain model e.g. Invoice and Item. The web application shows them in a way that portrays the master-detail dependency relationship – the master contains the detail, and when the former changes, so does the latter. The master can be presented in either a web form or a web list, whereas the detail is commonly presented in a web list manner.
Lookup	Some web applications contain forms or lists in which one or more fields take as a value an instance of an associated domain object. The UI facilitates this by providing a dynamically generated list ("look up") where the user can see the available choices and pick one.

Defined Selection	Some web applications contain forms or lists in which one or more fields take its value from a static list of predefined values. The web developer defines in the domain model such collection of values, and associates it to the domain object which will use it.
Primary Key	For each of the business objects in the domain model, the web developer defines a key property to identify each instance of the objects that will comprise the web application. Unlike all of the above concepts, it doesn't render to a visual, UI component, but rather it is used to generate the persistence layer of the application.

With these artifacts it is posible to generate fully functional web applications, ready for deployment; such web applications are, at its most basic form, transactional i.e. they are data-driven, using databases technology, upon which they perform DML (Data Manipulation Language) operations, more commonly known as CRUDEL operations (Create, Retrieve, Update, Delete, Exists and List).

This set of concepts comprise the semantics component of the proposed DSL for web applications; they compose the theorical terms for both the abstract and concrete syntax, as it will be seen in the following two sections.

4 Meta Model for Web Application (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 is the abstract syntax of the exposed DSL, and describes all the global components needed to generate a complete web application in an object oriented programming language. An instance of WAMM can be seen 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 in. 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 of the domain objects and the relations between them and the information required for generating the database scripts, such as relational constraints and 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 and the interaction with the user. The last version of WAMM can be seen in Fig 2.

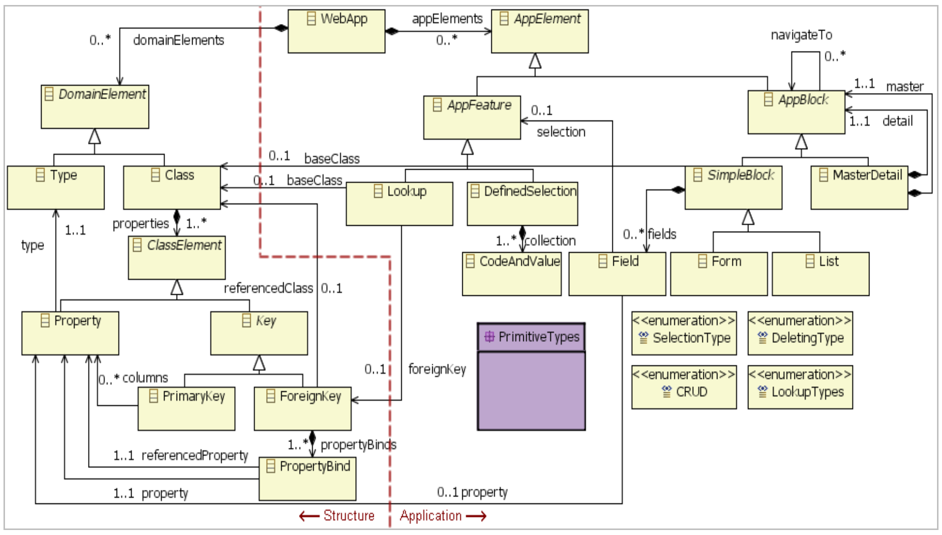


Fig. 2. Diagram of the Web Application Metamodel. Here all the elements of the metamodel are shown.

5 Concrete Syntax for Web Applications: WebApp UML Profile

Defining a concrete syntax is required to portray the elements of a language in order to make it usable and understandable to individuals [8]. In practice, MDA recommends the use of UML profiles to define a concrete syntax for a DSL.

A UML Profile is a mechanism to extend the semantics of models built with UML in a bounded area of knowledge or interest. In other words, a UML Profile can be seen as 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.

5.1 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 2 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.

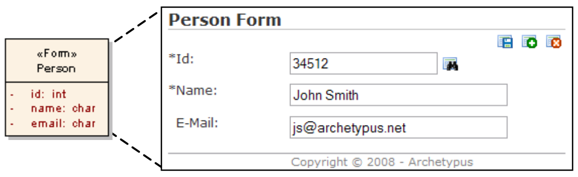
Table 2. Description of the operations for the Form stereotype tagged value CRUD

	Created controls	Control effects	Considerations
C	Insert Action a "+" icon or "new" button Save Action: a diskette icon	Insert: to put in "*Insertion Mode*" cleaning the form Save: to save the values in a new record.	If the primary key is auto increment, then those properties cannot be inserted.
R	Search Action: 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 "*Updating Mode*".
U	Save Action: a diskette icon	To save the changes made on a record.	If there are no pending changes, then this control is unavailable
D	Delete Action: a "x" icon	To delete the current record.	A confirmation message is shown before delete 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.



*Id:	34512
*Name:	John Smith
E-Mail:	js@archetypus.net

Copyright © 2008 - Archetypus

Fig. 3. Simple form generated form with the <<form>> stereotype of Webapp profile.

5.2 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 3 shows the effect of every letter in the CRUD tag.

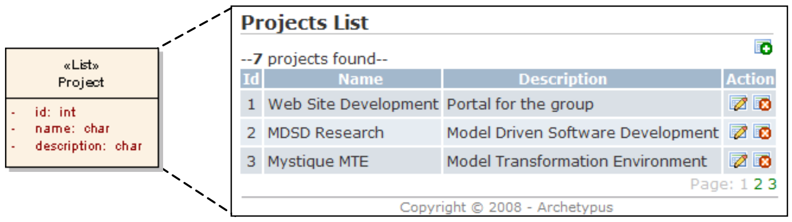
Table 3. Description of the operations for the List stereotype tagged value CRUD

	Forms generated	Controls in reading form	Cardinality of controls	Considerations
C	Creation Form	Insert Action: 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	Update Action: a pencil icon	One for each record.	
D	No form generated	Delete Action: 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.



Projects List

--7 projects found--

Id	Name	Description	Action
1	Web Site Development	Portal for the group	
2	MDSD Research	Model Driven Software Development	
3	Mystique MTE	Model Transformation Environment	

Page: 1 2 3

Copyright © 2008 - Archetypus

Fig. 4. List form generated with the <<list>> stereotype of Webapp profile.

5.3 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: [<u>CRUD</u> | 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: [<u>CRUD</u> | 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.

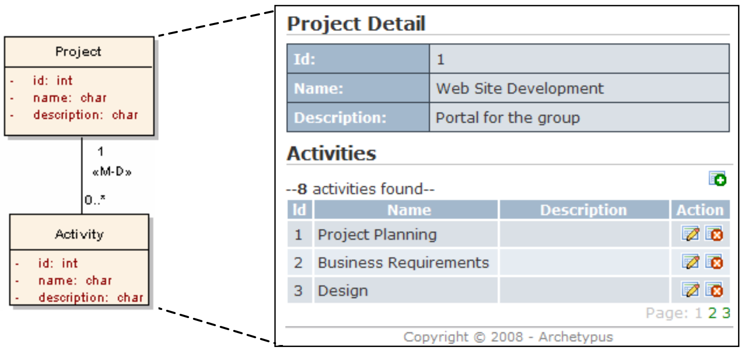
Effect:

- 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 <u>Master CRUD</u> tags depend on the value of <u>Master layout</u>. All effects of <u>Master CRUD</u> 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 <u>Detail CRUD</u> tag applies to Detail in the same way that CRUD tag applies to the “List” stereotype.
- If the value of <u>Master specific properties</u> is not set, then all master properties are displayed.
- If the value of <u>Detail specific properties</u> is not set, then all detail properties are displayed except the foreign key.
- If <u>Master layout</u> is “Form”, then the <u>Master records by page tagged value</u> has no effect.
- If <u>Master records by page value</u> is 0 or not set, then master records are not paged.
- If <u>Detail records by page value</u> 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 <u>Navigation order</u>, there must be a node in the navigation flow corresponding to the upper node in the navigation flow.
- If <u>Master CRUD</u> has not “D”, then <u>Master deleting behavior</u> 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.



Project Detail

Id:	1
Name:	Web Site Development
Description:	Portal for the group

Activities

--8 activities found--

Id	Name	Description	Action
1	Project Planning		
2	Business Requirements		
3	Design		

Page: 1 2 3

Copyright © 2008 - Archetypus

Fig. 5. Master-detail form generated with the <<M-D>>stereotype of WebApp profile.

5.4 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:

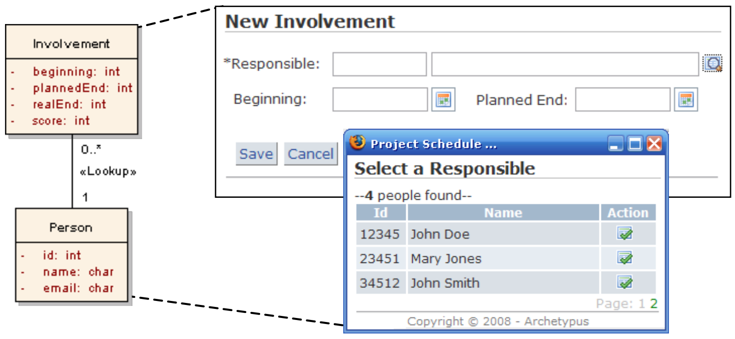
- Lookup type: [Pop-up window, Select list] it defines if the list of values is shown in an input select or in a new window with a list of records to select one record.
- Specific properties: comma-separated string listing the class properties shown in the records list.

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.



New Involvement

*Responsible:

Beginning: Planned End:

Select a Responsible

--4 people found--

Id	Name	Action
12345	John Doe	✓
23451	Mary Jones	✓
34512	John Smith	✓

Page: 1 2

Copyright © 2008 - Archetypus

Fig. 6. Pop-up window generated with the <<lookup>> stereotype of Webapp profile.

5.5 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:

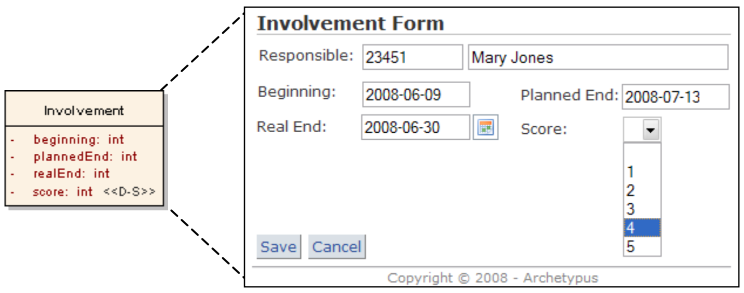
- Selection type: [Check box, Radio group, Select list] defines the presentation of pre-defined values from which the field is selected.
- Code and value collection: enumeration name with the list of values.

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 as an enumeration in the same package or namespace of the class that owns the property marked with this stereotype.



Involvement Form

Responsible: 23451 Mary Jones

Beginning: 2008-06-09 Planned End: 2008-07-13

Real End: 2008-06-30 Score:

1
2
3
4
5

Copyright © 2008 - Archetypus

Fig. 7. Defined selection generated with the <<D-S>> stereotype of WebApp profile.

5.6 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 * 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 <u>Auto increment</u> 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 <u>Primary key</u> stereotype.
- <u>Auto increment</u> can only be set for a single primary key; for composite ones it must be false or not set.

5.7 WebApp Profile Representation

All the concepts and ideas described above are represented in the profile shown in Fig 8. This diagram shows the WebApp profile stereotypes and their relations with the UML metaclasses; it gives the basis for marking the classes of any domain model.

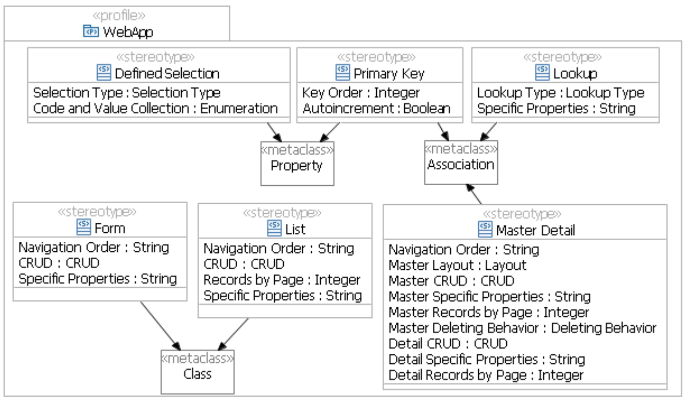
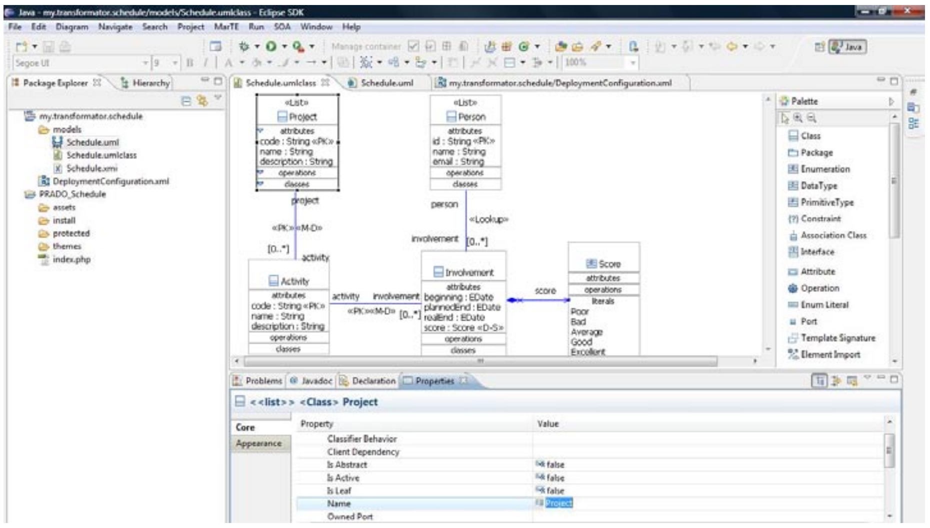

«profile»
WebApp
«stereotype»
Defined Selection
Selection Type : Selection Type
Code and Value Collection : Enumeration
«stereotype»
Primary Key
Key Order : Integer
Autoincrement : Boolean
«stereotype»
Lookup
Lookup Type : Lookup Type
Specific Properties : String
«metaclass»
Property
«metaclass»
Association
«stereotype»
Form
Navigation Order : String
CRUD : CRUD
Specific Properties : String
«stereotype»
List
Navigation Order : String
CRUD : CRUD
Records by Page : Integer
Specific Properties : String
«stereotype»
Master Detail
Navigation Order : String
Master Layout : Layout
Master CRUD : CRUD
Master Specific Properties : String
Master Records by Page : Integer
Master Deleting Behavior : Deleting Behavior
Detail CRUD : CRUD
Detail Specific Properties : String
Detail Records by Page : Integer
«metaclass»
Class

Fig. 8. WebApp profile representation

6 Generating Web Applications

The ultimate goal of the presented DSL is to automate the web application development process, which involves the assembly of tools for code generation depending on the target platform. Such code generation, in this case, would imply a two-phased process: a Model-to-Model Transformation from the user domain model, which yields an intermediate model; and a Model-to-Text transformation, from the intermediate model to the executable code.

Using the power of the Eclipse platform and plug-ins like EMF (Eclipse Modeling Framework) [9], and UML2 Tools [10] to load and manipulate UML models, an environment to use the DSL described in previous sections was set up. A screenshot of such environment can be seen in Fig 9.



Property	Value
Classifier Behavior	
Client Dependency	
Is Abstract	false
Is Active	false
Is Leaf	false
Name	Project
Owned Port	

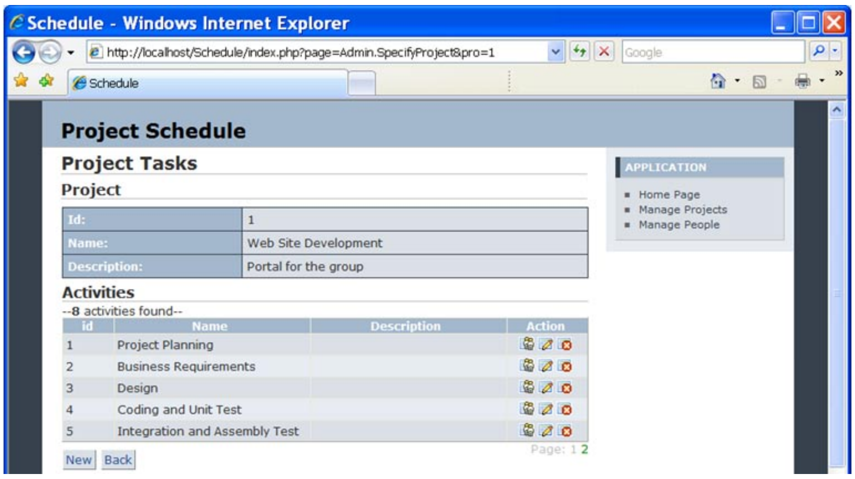
Fig. 9. Screenshot of the model transformation environment. In the right side of the windows a marked UML domain model of a *Project PRADO_Schedule* example is opened.

To apply the transformation process it is necessary to execute certain activities in the transformation environment. 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 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 the transformation. This action consists in applying a M2M transformation to the marked domain model so a WAMM instance can be obtained. This task is done by using ATL (Atlas Transformation Language) [11]. After the M2M transformation is applied, the final step is to perform a M2T transformation implemented using JET (Java Emitter Template)[12] in order to convert a WAMM instance to source code depending on a target platform.

Fig 10 shows a screenshot of the Web application generated from the marked UML domain model of the *Project Schedule*. The platform selected for deploying this example was PHP with *Prado Framework*.


Schedule - Windows Internet Explorer
http://localhost/Schedule/index.php?page=Admin.SpecifyProject&pro=1
Google
Schedule
APPLICATION
Home Page
Manage Projects
Manage People
New
Back

Project Schedule

Project Tasks

Project

Id:	1
Name:	Web Site Development
Description:	Portal for the group

Activities

--8 activities found--

id	Name	Description	Action
1	Project Planning		
2	Business Requirements		
3	Design		
4	Coding and Unit Test		
5	Integration and Assembly Test		

Page: 1 2

Fig. 10. Screenshot of the generated Web application deployed in a server.

8 Conclusions and Further Work

Building software is a task that might incur in 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 MDSD approaches. However proposals such as those experienced with a tool such as the proposed DSL in this work, 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.

The development automation that the presented DSL provides goes beyond the basic code generation facilities provided by CASE tools, since it is now possible to generate data definition language scripts; furthermore the application code is generated according to a target platform and a target architecture, it transforms the domain model to an intermediate artifact that precedes the code generation, and it generates the required files to deploy or expose in a server the generated Web application.

The source of the web application generation process inherent to the proposed DSL 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 [13], using a specific purpose language of this kind, like ATL [11] or the latest m2m component of the Eclipse community called Operational QVT.

Referencias

1. Völter, M. y Stahl, T. Model-Driven Software Development (Technology, Engineering, Management) ISBN: 978-0-470-02570-3, 444 p, 2006.
2. Fowler, M. Analysis Patterns: Reusable Object Models. Reading, MA.: Addison-Wesley, 1996.
3. 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.
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: Metamodeling 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. Greenfield, J., Short, K.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley. (2004)
9. Wolfe, A. Eclipse: A platform becomes an Open-Source Woodstock. ACM Queue. Vol 1, No 8. ACM, 2003.
10. Eclipse Foundation. Model Development Tools (MDT): UML2 Tools. http://www.eclipse.org/modeling/mdt/
11. Eclipse Foundation. M2M. 2008. ATL: Atlas Transformation Language. http://www.eclipse.org/m2m/atl/
12. Eclipse Foundation. M2T. 2008. JET: Java Emitter Templates. http://www.eclipse.org/modeling/m2t/?project=jet#jet
13. Object Management Group. Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. OMG, 2007.