STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

A PROPOSED DSL FOR DATA INTENSIVE APPLICATION DEVELOPMENT

PAUL HORAŢIU STAN

Abstract. Model Driven Architecture (MDA) defines three layers of abstraction for a domain: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). Nowadays in software industry the translation from PIM to PSM is made by developers that implement the model diagrams. This paper presents a new Domain Specific Language (DSL) for developing data intensive applications. The proposed DSL contains a grammar for specifying the PIM and a transformation engine to .NET PSM. The benefits of the proposed DSL resides in the possibility to write transformations to many PSM not only to .NET web applications. Finally a comparison with WebML and WebDSL is presented.

1. Introduction

In software engineering, a DSL represents a custom programming language dedicated to a particular problem domain. It contains a minimal set of statements understood by the people who use domain specific concepts.[11].

This paper presents a DSL for specifying data intensive web applications. Using the proposed DSL, a PIM of an application can be defined, then using transformations to given PSM the source code can be automatically generated.

The terms PIM and PSM are most frequently used in the context of the MDA [1] approach. This MDA approach corresponds to the OMG vision of Model Driven Engineering. The main idea is that it should be possible to use a Model Transformation Language (MTL) to transform a PIM into a PSM.

The paper is structured as follows: Section 2 presents the problem that the current research intend to solve, divided into two subsections that describe (1) current DSL for data intensive web applications and (2) the proposed solution;

Received by the editors: September 26 2011.

2010 *Mathematics Subject Classification.* 68N15, 68N20.

1998 *CR Categories and Descriptors.* D.2.11 [**Software**]: SOFTWARE ENGINEERING – *Software Architectures* D.2.13 [**Software**]: SOFTWARE ENGINEERING – *Reusable Software*; D.3.4 [**Software**]: PROGRAMMING LANGUAGES – *Processors*;

Key words and phrases. Domain Specific Languages, Model Transformation, Code generators.

Section 3 shows the technical details of the proposed DSL for data intensive applications together with a comparison with WebML and WebDSL two of the most actual DSLs for web modeling, and finally, Section 4 summarizes the research results.

2. The Problem

In order to improve the quality and the development time of the final software products, the software engineering industry should use abstraction more and more. By retaining knowledge about lower level operations in higher-level abstractions, developers can work with higher-level concepts and save the effort of composing the lower-level operations [11, 10].

The conventional abstraction technique that involves methods and classes are no longer sufficient for creating new abstraction layers [15, 1]. Libraries and frameworks are good at encapsulating functionality, but it is often awkward for developers to reach that functionality, using, in many cases the application programmers interface (API) [11].

The common parts of the domain are implemented by code generation templates, while the custom variables are configurated by the application developer using configuration interfaces. These configuration interfaces can take the form of a wizard for simple domains, or complex languages for larger domains [11].

The scope of our research is web applications with a rich data model. That is, applications with a database for data storage and a user interface providing several views on the data in the database, including CRUD operations. An additional assumption is that the data model cannot be changed at run-time.

2.1. Current DSLs for web applications.

There are many development environments and programming languages that can be used to design and implement data intensive applications, but these are general purpose languages, in other words, are not specialized only on a given domain.

This section presents two of the actual DSLs for web application development: Web Modeling Language (WebML) and Web Domain Specific Language (WebDSL).

Web Modeling Language (WebML) can be used to define web sites under distinct dimensions: (1) structural model which expresses the data content of the site, in terms of the entities and relationships, (2) composition model contains the pages that compose it, (3) navigation model represents the topology of links between pages, (4) presentation model defines the layout and graphic requirements for page rendering, (5) personalization model contains the customization features for one-to-one content delivery. All the concepts of WebML have two representations: graphic notation and a textual XML

syntax. WebML specifications are independent of programming languages or development platforms. WebML is a model-driven approach to web site development [6, 14]. WebML enables developers to define the core features of a web application at a higher level avoiding architectural details. All proposed concepts are associated with intuitive graphical symbols which can be easily supported by CASE tools and useful for the non-technical members of the application development team [16, 14].

WebML defines basic units such as: Data unit, Index unit, Entry unit, Create unit, Delete unit, etc that have graphical representation and a default implementation; for instance the Create unit enables the creation of a new entity instance [16, 2]. These basic units are translated to a PSM in order to become an application. One of the commercial tools that implement WebML specifications is WebRatio.

WebDSL [8, 3, 9, 4] is another DSL for developing web applications with a data model. The main features of WebDSL are: (1) Domain modeling, (2) Presentation, (3) Page-flow, (4) Access control, (5) Data validation, (6) Workflow, (7) Styling, (8) Email. WebDSL applications are translated to Java web applications, and the code generator is implemented using Stratego/XT and SDF [11, 10].

WebDSL has the following types of statements: (1) function definition, (2) variable declaration, (3) assignment, (4) if, (5) return, (6) for loop, (7) while and (8) switch [4].

The definition of a page has three parts: (1) the name of the page, (2) parameters definition, and (3) a presentation of the data encapsulated in the parameters. WebDSL provides basic markup operators such as: (1) section, (2) header, and (3) list for defining the structural model of a page. Data from the object parameters are displayed in the page by data access operations such as `output`. Collections of data can be iterated using the `for` construct. It is possible to display the content of an object based on a condition. Using custom templates the developer can define reusable parts of code. Finally, WebDSL supports the separation of the concepts, user-defined concepts can be grouped in modules [4].

2.2. **The proposed DSL for web applications.** This section presents the current DSLs for data intensive web applications and the big picture of the proposed DSL for developing data intensive applications.

2.2.1. *Problem Motivation.* DSLs for web applications intend to solve the same problem, all of them are focus on web applications domains and intend to provide a common language for these domains. A new DSL should contain fewer statements than a general-programming language like Java or .NET and should work with more abstract things and should be able to (1) automatically

transform a PIM into a general-programming language and to (2) simulate the PIM before the transformation process.

At this moment, WebML and WebDSL have transformation engines only to Java. The main motivations of the proposed DSL are: the PIM should be easily translated to different programming languages, the syntax should contains fewer "words" than WebDSL, the language should be easily extended with platform dependent routines.

2.2.2. *Conceptual view of the proposed solution.* Nowadays many applications process data, these data are shown to the user via windows, web pages, different type of mobile forms etc. In the proposed solution these things are modeled using the *Page* concept. A page shows/reads data to/from the user interface and receives user's actions. For displaying and reading data few statements are needed, and for catching user actions the event concept is introduced into the proposed solution. So, it is a good thing for each defined event to have an event handler, the body of the event handler contains a set of statements that control the application flow. In the proposed solution these statements are structured into four categories: (1) User Interface Statements, (2) Domain Statements, (3) Persistence Statements and (4) Control Flow Statements. They will be detailed later into this paper. The statements work with domain objects: (1) Entities, (2) Value Objects and (3) Specifications. Domain objects are used to define the static model of the application, while statements define its dynamic model.

3. Technical details

This section describes the proposed solution's details. First subsection shows the architecture of the proposed DSL, the second describes how the solution works in a real context and the last one makes a comparison with WebML and WebDSL during the development process of a web site that manages books, authors and members.

3.1. **The proposed DSL's architecture.** Figure 1 shows the metamodel of the proposed DSL for developing data intensive applications. The solution involves (1) a simple language with few concepts to ensure the syntax will be easily understandable by developers and (2) a transformation engine to different PSMs. At this moment only the transformation to .NET web sites is available, but the PIM allows transformations to Java web sites, .NET desktop applications, Java desktop applications, Java and .NET mobile applications, etc.

The metamodel classes are divided into three main categories:

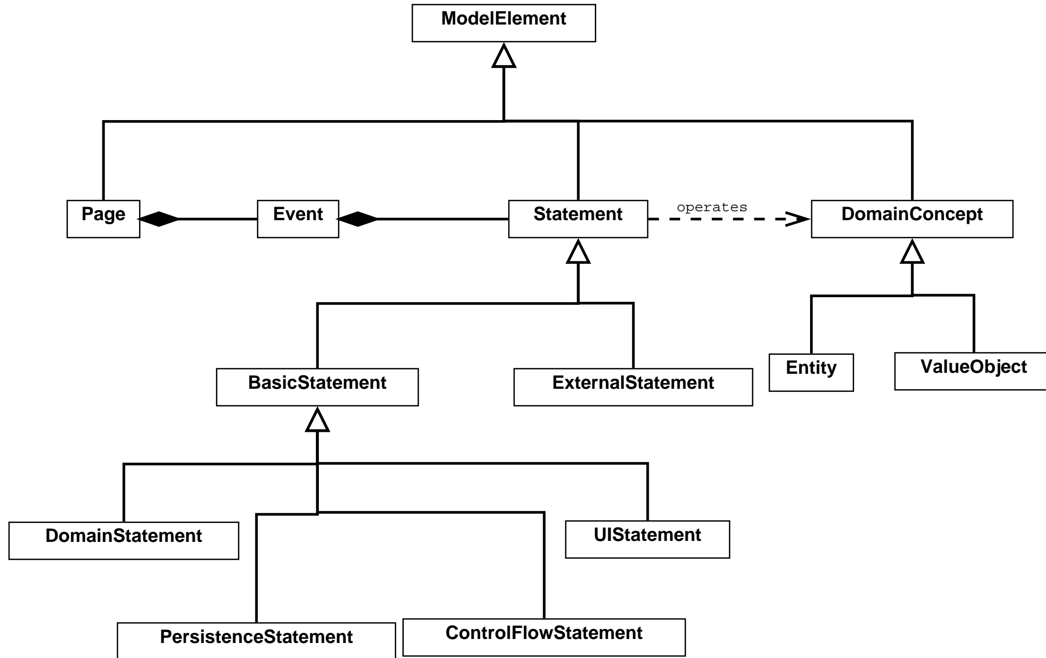

FIGURE 1. Proposed DSL's metamodel

- *Pages* and *Events*, used in order to manage the interaction with the human user, display data and read user actions;
- *Domain Concepts*, used for defining the business model of the system, that includes entities and relations between them;
- *Statements*, used for specifying the behavior of the system. There are *basic statements* and *external statements* the latter representing custom operations defined by the development team. An external statement is a link between a new platform-independent statement and a platform-dependent statement, acting as a wrapper.

Each page has a list of events with at least one event called *main event*, raised when the page is loaded. An event other than the *main event* represents a user action. Each event has an *event handler* with the same name. The *event handler* has a list of statements for describing the system response when a given event appears.

In order to define the static model of an application a developer can use the following domain concepts:

- *Entities*, can be stored into the persistent storage, can have properties of primitive types and can be associated with another entities in one

of the following ways: one to one, one to many, many to one and many to many. The Domain Driven Design (DDD) patterns have been used for designing the *Entity* metamodel class [12].

- *Value Objects*, which cannot be stored into the persistent storage, can have only properties of primitive types and cannot be associated with another domain concept. The name *Value Object* has been adapted from [12].
- *Specification*, validation rules for *Entities* or *Value Objects*, containing a list of properties like an entity and a logical expression that will be evaluated for a given domain object. Available operators for composing the logical expression are: *and, or, not, not=, <=, >=, <, >, ==, -, +, *, /*.

Listing 1 presents a specification example. The name of the specification is *FilterMembersByName* and it is applicable to *Member* entities. The specification has a single property of type *String* called *myMemberName*. The logical expression checks if the first name or the last name of a given member contains the *myMemberName* property as a substring. The logical expression can be composed with logical operators and with *specification functions* which are either (1) Basic functions, defined into the proposed DSL or (2) External functions being like external statements links between new platform-independent functions and platform-specific functions implemented by the development team.

Listing 1. A specification example

```
1 Specification FilterMembersByName for Member{
2   myMemberName:String
3   expression: (contains(Member.FName,myMemberName) or contains
        (Member.LName,myMemberName))
4 }
```

For defining user interfaces and system behavior, the developer should create pages and enter control statements in the bodies of event handlers for controlling the processing flow.

The basic statements are divided into four main categories:

- *Domain Statements*, which operate over domain concepts such as *Entities*, *Value Objects* and *Specifications*.
- *User Interface Statements*, are used to display data to the user or to read data from the pages.
- *Persistence Statements*, useful when storing and loading entities to, respectively from, the repository.

- *Control flow statements*, a subset of similar statements from a general programming language.

3.2. **The transformation process.** The Transformation engine has two inputs: (1) the model specified in the proposed DSL and (2) platform-specific templates, in our case .NET templates. These templates contain code snippets for .NET. The result of the process is the generated source code of the application which is the PSM.

3.3. **How it works.** This section describes how the entire solution works, presents simple code examples written in the proposed DSL and the transformation process from the DSL code to the Microsoft C#.NET code. Listing 2 presents the essential elements of the proposed language in the Xtext grammar specification language.

Listing 2. DevDSL's grammar

```
1 /*Level 1*/
2 Model:{Model}
3    'config' '{'Config+=Config*'}'
4    elements+=AbstractElement*;
5
6 /*Level 2*/
7 AbstractElement: DomainObject | Specification | Page;
8 DomainObject: Entity | ValueObject;
9
10 /*Level 3*/
11 Entity: 'Entity ' name=ID '{'
12   (attributes+=Attribute)*
13 '}';
14
15 ValueObject:'ValueObject' name=ID'{'
16   (attributes+=ValueObjectAttribute)+
17 '}';
18
19 Specification: 'Specification 'name=ID'
20      for '(domainObject=[DomainObject] | 'Object')'{'
21   (attributes+=Attribute)*
22     'expression'':'expression=LogicalExpression
23 '}';
24
25 Page: 'Page ' name=ID '{'
26   FirstEvent=Event
27   NextEvents+=Event*
28 '}';
```

```
29
30 /*Level 4*/
31 Attribute: name=ID ':' type=(RefType|StdType);
32 ValueObjectAttribute:name=ID':'type=StdType;
33
34 Config: name=QualifiedName '=' value=STRING;
35
36 Statement:
37 //Domain Statements
38 CreateObject|EraseObject|SetObjectProperty|GetObjectProperty|
       AddElementToList|RemoveElementFromList|Satisfy|Filter|
39 //UI Statements
40 DisplayObject|DisplayList|DisplayButton|ReadObjectFromUI|
       Redirect|
41 //DB Statements
42 SaveEntityToRepository|LoadList|DeleteEntityFromRepository|
43 //ControlFlow Statements
44 IfStatement|WhileStatement|ForEachStatement |
45 //External statements
46 ExternalStatement;
47 ...
```

In order to create a PIM of an application, a developer should write a text file named `[filename].app` which should contain the following sections:

- the *config* section
- the *domain objects* section
- the *pages* section.

Each application should start with the *config* section, which can contain database connection properties for instance hibernate configurations.

Listing 3. Config Section

```
1  config {
2    proxyfactory.factory_class=
3      "NHibernate.ByteCode.Castle.ProxyFactoryFactory,
4      NHibernate.ByteCode.Castle"
5    connection.provider=
6      "NHibernate.Connection.DriverConnectionProvider"
7    dialect=
8      "NHibernate.Dialect.MsSql2005Dialect"
9    connection.driver_class=
10     "NHibernate.Driver.SqlClientDriver"
11   connection.connection_string=
12     "Server=servername;Initial Catalog=database;
```

```
13     User Id=user;Password=*****"
14   hbm2ddl.auto="update"
15 }
```

The *domain objects* section contains definitions for *Entities*, *Value Objects* and *Specifications*, the static model of the application. *Entities* and *Value Objects* are similar to classes in a general object oriented programming language.

Listing 4. Domain Objects Section

```
 1 Entity Book{                          Entity Author{
 2   Title: String                         FName:String
 3   ISBN : String                         LName:String
 4   Price: Double                         BDate:String
 5   Pages: Integer                        Royalties:Royalty*
 6   Royalties:Royalty*                  }
 7   LibraryBooks:LibraryBook*
 8 }
 9 Entity LibraryBook {                  Entity Royalty{
10   RealBook:Book                         RBook:Book
11   StockId:String                        RAuthor:Author
12   Value:Double                          RAmount:Double
13   Lends:Lend*                         }
14 }
15 Entity Member {                       Entity Lend {
16   FName:String                          LMember:Member
17   LName:String                          LLibraryBook:LibraryBook
18   BDate:String                          LendDate:String
19   Sex:String                            ReturnDate:String
20   Address:String                        Days:Integer
21   StartDate:String                      Fee:Double
22   Dues:Double                         }
23   Lends:Lend*
24 }
```

The listing 4 defines the same model as the figure 2 which is an WebML data model diagram.

The last section a developer should write is the *Pages* section, which contains the dynamic model of the application. The listing 5 presents a demo page specification using the proposed DSL.

Listing 5. Page Demo

```
1 Page CurrentBook{
2   Page_Load{
```

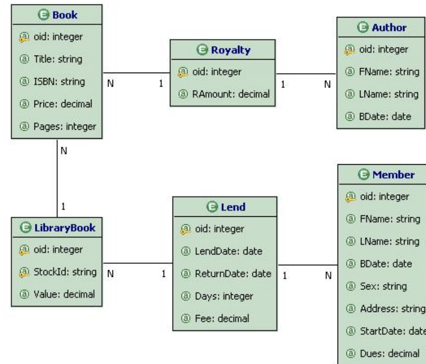

FIGURE 2. The Library Model

```
3      if(Satisfy("cBook",ObjectNotNull()))
4      {
5        DisplayObject("cBook",editable=true,
6          [("Library Books", btnLibraryBooks)
7           ("Royalties",btnRoyalties)])
8        DisplayButton("Back",btnBack)
9        DisplayButton("Save",btnSave)
10     }
11     else
12     {
13       Redirect(MainPage)
14     }
15   }
16   btnLibraryBooks{
17     ReadObjectFromUI("cBook")
18     Redirect(ManageLibraryBooks)
19   }
20   btnRoyalties{
21     ReadObjectFromUI("cBook")
22     Redirect(ManageRoyalties)
23   }
24   btnBack{
25     EraseObject("cBook")
26     Redirect(ManageBooks)
27   }
28   btnSave{
29     ReadObjectFromUI("cBook")
30     SaveEntityToRepository("cBook")
31     EraseObject("cBook")
```

```
32     Redirect(ManageBooks)
33   }
34 }
```

3.4. **The Library web site.** This section presents fragments from a demo web site called *Library* that manages books, authors and members, and has been developed using the proposed DSL. Also, parts of it have been developed using WebML and WebDSL in order to be able to make a comparison between our DSL and the others.

The conceptual model of the web site has been presented in the figure 2 for WebML and in the listing 6 for WebDSL. The conceptual model designed using our DSL is presented into the listing 4.

Listing 6. Library Model in WebDSL

```
1  entity Book{
2    Title:: String
3    ISBN :: String
4    Price:: Double
5    Pages:: Int
6    Royalties -> Set<Royalty>
7    LibraryBooks -> Set<LibraryBook>
8  }
9  entity LibraryBook {
10   RealBook -> Book
11   StockId::String
12   Value::Double
13   Lends -> Set<Lend>
14 }
15 entity Member {
16   FName::String
17   LName::String
18   BDate::String
19   Sex::String
20   Address::String
21   StartDate::DateTime
22   Dues::Double
23   Lends -> Set<Lend>
24 }
```

```
entity Author{
  FName::String
  LName::String
  BDate::String
  Royalties -> Set<Royalty>
}

entity Royalty{
  RBook -> Book
  RAuthor -> Author
  RAmount::Double
}
entity Lend {
  LMember::Member
  LLibraryBook->LibraryBook
  LendDate::DateTime
  ReturnDate::DateTime
  Days::Int
  Fee::Double
}
```

WebDSL offers the possibility to specify the kind of a relationship: (1) reference −> and (2) composite <>. The difference between reference and composite property kinds is that composite indicates that the referred entity is

part of the one referring to it. Deletion of the composite entity is also deleting the referred entities.

WebML uses graphic diagrams for defining the static model of an application. Unfortunately this graphic representation can confuse the developers because it is not similar to UML. For instance the relationship *one to many* in WebML is like the *many to one* in UML.

The proposed DSL does not offer the possibility to specify the association between two entities as reference or composite like WebDSL, but this can be added in a future version. In contrast with WebML, the proposed solution does not produce confusions regarding associations between entities. Entities are defined as classes in a GPL. Attributes of an entity are typed inside it's body, if an attribute is a collection of elements then after the attribute type there is an `*`.

The behavioral model of the library web site can be expressed in WebDSL using action code. Functions can be defined globally and as methods in entities. At this moment WebDSL model of an application can be translated only to Java PSM.

The behavioral model of the library web site can be expressed in WebML using units and links between them. The disadvantage of the WebML is that it is very hard to maintain a model with many lines and boxes; for complex sites, the model becomes a spaghetti picture. Current version of WebML supports only transformations to Java. On the other hand, if new units need to be defined, the developer should write the Java code which implements the unit functionality.

The dynamic model specified with the proposed DSL is easily understandable and manageable. At this moment the eclipse plug-in works with the textual models but, as a feature improvement, a visual plug-in can be developed. Unlike WebML, the model for complex systems, expressed using the proposed textual DSL, does not become an unmanageable model. Unlike WebDSL the proposed DSL does not allow developers to define methods inside the body of an entity. This fact ensures that the static and dynamic models are not mixed.

An advantage of our DSL compared with WebML and WebDSL is that it has a simple syntax with few "words". The conceps presented in the proposed DSL are structured in two main categories: concepts used to define static model: *Entities*, *Value Objects* and *Specifications*, and conceps used to define the behaviour of the system: *Pages*, *Events* and *Statements*. Due to this simple syntax, it is easy for a developer to understand the language and to quiqly ramp-up in an open project.

In contrast with both WebML and WebDSL, the PIMs described using the proposed DSL can be easily translated to different PSMs such as .NET, Java,

PHP, etc. It is in progress an implementation of a transformation engine to PHP PSM of the models specified using the proposed DSL.

This is a short comparison of the proposed DSL with WebML and WebDSL, the main target of this paper is to introduce the basic concepts of the new DSL, a more complex comparison can be the subject of a future paper.

4. Conclusions and future work

The main contributions of this paper are: (1) a new DSL for data intensive applications, (2) a textual representation of the proposed language and (3) a transformation engine from the proposed PIM to .NET web application PSM.

The novelty of the proposed solution resides in: (1) proposing a set of basic statements that can be easily translated to PSMs, (2) offering the possibility to add and call external statements, (3) introducing the concept of specification for defining validation rules for entities and value objects and (4) offering the possibility to translate the PIM to different PSMs such as web, desktop and mobile applications, all of these written in different programming languages.

The limitations of the proposed solution are: (1) it does not offer the inheritance relationship between entities, (2) it is not possible to specify if an association is container or reference, (3) a developer is not able to compose statements into parent statements for modularizing the system.

All these will be subject of future developments.

5. Acknowledgment

The author wish to thank for the financial support provided from programs co-financed by the SECTORAL OPERATIONAL PROGRAMME HUMAN RESOURCES DEVELOPMENT, Contract **POSDRU 6/1.5/S/3** "Doctoral studies: through science towards society".

References

[1] OMG. Model Driven Architecture. http://www.omg.org/mda/specs.htm. Last accessed on December 10, 2011.

[2] Willian Massami Watanabe, David Fernandes Neto, Thiago Jabur Bittar, and Renata P. M. Fortes. Wcag conformance approach based on model-driven development and webml. In *Proceedings of the 28th ACM International Conference on Design of Communication*, SIGDOC '10, pages 167–174, New York, NY, USA, 2010. ACM.

[3] Danny Groenewegen, Zef Hemel, and Eelco Visser. Separation of concerns and linguistic integration in webdsl. *IEEE Softw.*, 27:31–37, September 2010.

[4] WebDSL. A domain-specific language for developing dynamic web applications with a rich data model. http://webdsl.org. Last accessed on December 12, 2010.

[5] Debasish Ghosh, *DSLs in Action*, Manning Publications, 2010.

[6] Stefano Ceri, Marco Brambilla, and Piero Fraternali. Conceptual modeling: Foundations and applications. chapter The History of WebML Lessons Learned from 10 Years of Model-Driven Development of Web Applications, pages 273–292. Springer-Verlag, Berlin, Heidelberg, 2009.
[7] Guotao Zhuang and Junwei Du. Mda-based modeling and implementation of e-commerce web applications in webml. In *Proceedings of the 2009 Second International Workshop on Computer Science and Engineering - Volume 02*, IWCSE '09, pages 507–510, Washington, DC, USA, 2009, IEEE Computer Society.
[8] Eelco Visser Danny M. Groenewegen. Weaving web applications with webdsl: (demonstration). In *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 797–798, 2009.
[9] Danny M. Groenewegen, Zef Hemel, Lennart C.L. Kats, and Eelco Visser. Webdsl: a domain-specific language for dynamic web applications. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, OOPSLA Companion '08, pages 779–780, New York, NY, USA, 2008. ACM.
[10] Z. Hemel, L.C.L Kats, E. Visser *Code Generation by Model Transformation. A Case Study in Transformation Modularity*, Delft University of Technology Software Engineering Research Group 2008.
[11] Eelco Visser *WebDSL: A Case Study in Domain-Specific Language Engineering*, Delft University of Technology Software Engineering Research Group 2008.
[12] Eric Evans, *Domain-Driven Design Quickly*, C4Media Inc 2006.
[13] Nathalie Moreno, Piero Fraternalli, and Antonio Vallecillo. A uml 2.0 profile for webml modeling. In *Workshop proceedings of the sixth international conference on Web engineering*, ICWE '06, New York, NY, USA, 2006, ACM.
[14] Andrea Schauerhuber, Manuel Wimmer, and Elisabeth Kapsammer. Bridging existing web modeling languages to model-driven engineering: a metamodel for webml. In *Workshop proceedings of the sixth international conference on Web engineering*, ICWE '06, New York, NY, USA, 2006, ACM.
[15] K. Czarnecki. *Overview of generative software development. In J.-P. Bantre et al., editors, Unconventional Programming Paradigms (UPP 2004)*, volume 3566 of Lecture Notes in Computer Science, pages 313328, Mont Saint-Michel, France, 2005.
[16] Aldo Bongio Stefano Ceri, Piero Fraternali. *Web Modeling Language (WebML): a modeling language for designing Web sites*, Politecnico di Milano, 2000.

Babeş Bolyai University, Department of Computer Science, 1 M. Kogălniceanu St., 400084 Cluj-Napoca

E-mail address: horatiu@cs.ubbcluj.ro