Association for Information Systems
AIS Electronic Library (AISeL)

CONF-IRM 2010 Proceedings

International Conference on Information Resources Management (CONF-IRM)

5-2010

3P. A First Approach to a Domain Specific Language for Constructing Code Generators

Julián Vargas
National University of Colombia, jdvargasa@unal.edu.co

Helga Duarte
National University of Colombia, hduarte@unal.edu.co

3P. A First Approach to a Domain Specific Language for Constructing Code Generators

Julián Vargas[1]
National University of Colombia
jdvargasa@unal.edu.co

Helga Duarte[2]
National University of Colombia
hduarte@unal.edu.co

Abstract
The present work defines the components and architecture of a Domain-specific Language that takes advantage of patterns and common tasks performed by code generators based on relational databases. This DSL (Domain-specific Language) allows developers to easily and rapidly build and maintain custom code generators that meet their particular requirements instead of building them from scratch using general purpose programming languages, which is more expensive in terms of time. Current work includes the definition of the Code Generation Domain constrained to the transformation of relational models into applications source code, high level architecture and features models. The implementation of the DSL is a future work.

Keywords
Code Generators, Code Templates, Domain-specific Language.

1. Introduction

Code generation for applications based on relational databases has been treated extensively by a large number of tools that vary in complexity and sizes, from simple parsers as SAX [53] to layer and class generators such as Angie [3]. The term also refers to the process undertaken by the compilers that take as input the program source code of program and translate it into operation codes of the processor [1]. In the present work Code Generation is defined as *to produce source code files for an application, based on information taken from a relational model and customizable code templates.*

A code generator is a software application that facilitates the construction of applications that make use relational databases. The generator retrieves from the database, information about entities and relationships, and merging this information with the custom code templates, produces a set of files with the source code on which a software application is made of.
The code templates give the generator the capability to meet new requirements of the developers because they allow changing the way the final code is generated.

[1] Student in *Master in Systems and Computers Engineering*, Faculty of Engineering, National University of Colombia. Bogotá, Colombia
[2] Associate Professor, Faculty of Engineering, National University of Colombia. Bogotá, Colombia

1.1 Advantages of using code generator in the software development process

According to [1] there are several key benefits offered by code generators to the developers:

- Quality: Code generation from templates creates a consistent code base instantly, and when the templates are changed and the generator is run, the bug fixes or coding improvements are applied consistently throughout the code base.
- Consistency: The code that is built by a code generator is consistent in the design for the APIs and the use of variable naming. This results in a no-surprises interface that is easy to understand and use.
- Design decisions that stand out: High-level business rules are lost in the minutiae of implementation code. Code generators use abstract definition files to specify the design of the code to be generated. These files are much shorter and more specific than the resulting code.

1.2 Disadvantages of using code generators

When a code generator is needed in a software development process, the developers have to face these disadvantages:

- Oriented to one platform: Generate code for a given platform, i.e. only web apps, only windows apps, only databases, only documentation [12]-[39], [44].
- Integrated to one development environment: Integrated to one IDE (Integrated Development Environment) improving the efficiency of the users of that IDE but it's not possible to integrate them to other IDEs [7], [41].
- Oriented to one technology: are the ones capable of generation code for a given technology, for example .Net but are not able to generate code for other technologies such as Java or PHP [9]-[22], [25]-[28], [32], [34], [35], [43], [44].
- High cost: some of these tools are quite expensive [3], [17], [19], [25], [30], [32], [35], [40].

On the other hand, if the decision to construct a custom code generator is made, one has to take into account that the building costs could be high in terms of time. Additionally the developer should perform maintenance tasks over the long term and adapt the product to new technologies and methodologies that come out in time [1]. This makes the custom code generator part of the problem rather than the solution.

The rest of the document is organized in three parts as follows: in the next section the problem that we are working out is described. Then, a brief description of Domain Specific Languages that is our approach for building the solution, after this, a description of Expert Coder which is a project related with building code generators. Then we proceed to define the Code Generation Domain starting with the more abstracts elements and descending to the more detailed and technical descriptors. The last three sections present conclusions, future work and references, respectively.

2. Problem description

Code generators available on the market are heterogeneous and possess qualities that are not capable of being integrated into a single product due to their differences in platform, technology and licensing.

Code generators exist to reduce the time it takes to develop an application, but it takes time to build a custom generator and not having a tool to reduce the time needed to build, the code generator becomes part of the problem rather than the solution.

According to the above, a Domain Specific Language is necessary to facilitate the construction of custom code generators in order to have a tool that can be used in any context of software development involving the use of relational databases.

Domain-specific languages (DSL) offer a more complete solution to software engineering problems than general purpose programming languages can offer since they provide a notation tailored towards an application domain and are based on the relevant concepts and features of that domain. Such languages provide a natural vocabulary for concepts that are fundamental to the problem domain. A Domain-specific language (DSL) allows one to develop software for a particular application domain quickly and effectively, yielding programs that are easy to understand, reason, and maintain. The benefits of DSL derive from two basic principles of language design: abstraction and restriction. The choice of appropriate abstractions aids the phases of requirements, design, coding, and maintenance by providing high-level entities and relationships that the domain closely. Restriction of language expressiveness allows for greater automated analysis and hence supports verification, modification, and maintenance [55], [56], [57].

Domain-specific Languages such as ColdFusion Markup Language [47], MediaWiki templates [48] and PowerShell [49], for instance, allow development of quick solutions applied to each of their contexts without the need to build everything from scratch, increasing user productivity.

The DSL are designed to accurately describe a specific domain, such a task, a platform or a process. Instead of generic abstractions, they use concepts taken directly from the domain [54].

3. Definition of the code generation domain

ExpertCoder is a toolkit for the .NET platform that supports the creation of code generators based on expert systems. It's not a generator of code generators, but rather a set of libraries useful to write generators.

The purpose is to build a toolset that provides the code generator writer with the clarity that results from using templates for code generation and the flexibility provided by the power of the .NET platform, along with its huge class library.

Besides, as it is based on the principles of expert systems, the resulting generators are easily extensible, modular, and their structure is more declarative than imperative.

The idea is to create an expert system, to write a set of rules and then to specify the distinct precedence between them. These rules are evaluated by an execution engine, which determines, based on the precedence and every rule's activation state, which rule must be executed.

The execution engine provides an environment, which provides three sources of information:

- Parameters: these are stored in configuration files.

- Input model: the model to be converted.
- Inferred knowledge: the expert system is able to modify its active memory. By using this you can implement an indirect interaction mechanism for rules.

A typical generator is composed by two kinds of rules: navigation rules and production rules. Navigation rules are activated in presence of a given element type at the input, and proceeds to "navigate" that element's relationships, changing the current element from the input model. Production rules, are active in presence of an input element (and possibly certain kinds of elements at the output,) apply a developer-written algorithm in order to generate nodes at the output, using the information currently available at the input and in the active memory [58].

3.1 Goals of the code generation domain

The purpose of code generation is to reduce the time required to develop an application, as well as to reduce the repetitive tasks of writing code that can be automated by a code generator.

The scope of this work requires two main elements for defining the domain, one is the code generation process and the other is the code generator itself.

The code generation process involves the relational model taken from the database and the code templates defined by the user.

The code generator is the tool that performs the code generation process. This tool requires interaction from the user and is compounded by user interfaces and parameter options.

3.2 Concepts/Entities/Requirements

The 1st to the 44th references correspond to the bibliographical revision in the dominion of the code generators. Here some key patterns have been found in the way they treat the information retrieved from the database and in the way they produce the final code. Also, common functionalities such as a screen for prompting the connection string to the database and a synchronization option to reload the relational model in the case that it was changed, among others that will be described in more detail further in this document.

3.2.1 Iterations

The code generation process is performed mainly as a nested iteration since the relational model can be represented in a hierarchy [59], a code generator iterates over the properties of a given column, the columns of a given table, the tables of a given database, and the databases of a given set. In these iterations, the custom code templates are applied to the appropriate object to produce the final code.

3.2.2 Rules

Another key element of code generation process is a set of rules that are applied when a given condition is met. These conditions allow the code generator to apply templates or run special routines to cases where the standard model should not be applied.

3.2.3 Domain vocabulary

This section is dedicated to define the vocabulary that is going to be used for the definition of the rest of the DSL.

Relational model*:* As the name suggests, it is the raw relational model as is persisted in the database; this excludes any customization made by the user.
Code template: A text file with the description of how the final code should be generated.
Entity: The representation of an object inside the model, this can be a table inside the database or an artificial object defined by the user.
Level: A set of objects present in a given position of the hierarchy.
Code generator: A piece of software that by itself is able to transform pieces from the model into final source code or an output that can be consumed by other code generators.
User**:** In this scope, a user is a software developer or programmer that uses code generators in his/her daily work developing software applications.

3.3 High-Level requirements specification

To achieve the goal of allowing users to create custom code generators, the language should meet these requirements:

- Create user interface for prompting information that will be used in the code generation process.
- Create code templates that will be applied to the model in the code generation process.
- Allow the user to synchronize the actual model with the model persisted in the database in the case that new objects are created or some are deleted.
- Allow the user to manipulate the model to add, delete, update or list objects.
- Allow the user to use code generators as building blocks to construct a higher-level code generator.

3.4 Domain specific software architecture

Figure 1 shows the architecture proposed for the DSL. The following section explains each one of the parts that compose it.

3.4.1 Database reader/writer and raw relational model

The base information needed by a code generator resides in a relational database. This information, more precisely is the database catalog, can be represented using a hierarchy of objects, and a set of functions can be used to retrieve information from such structure (Figure 2) [59] as well as some additional functions to manipulate the database, if the code generator requires it to create new objects such as stored procedures, functions, views, etc.

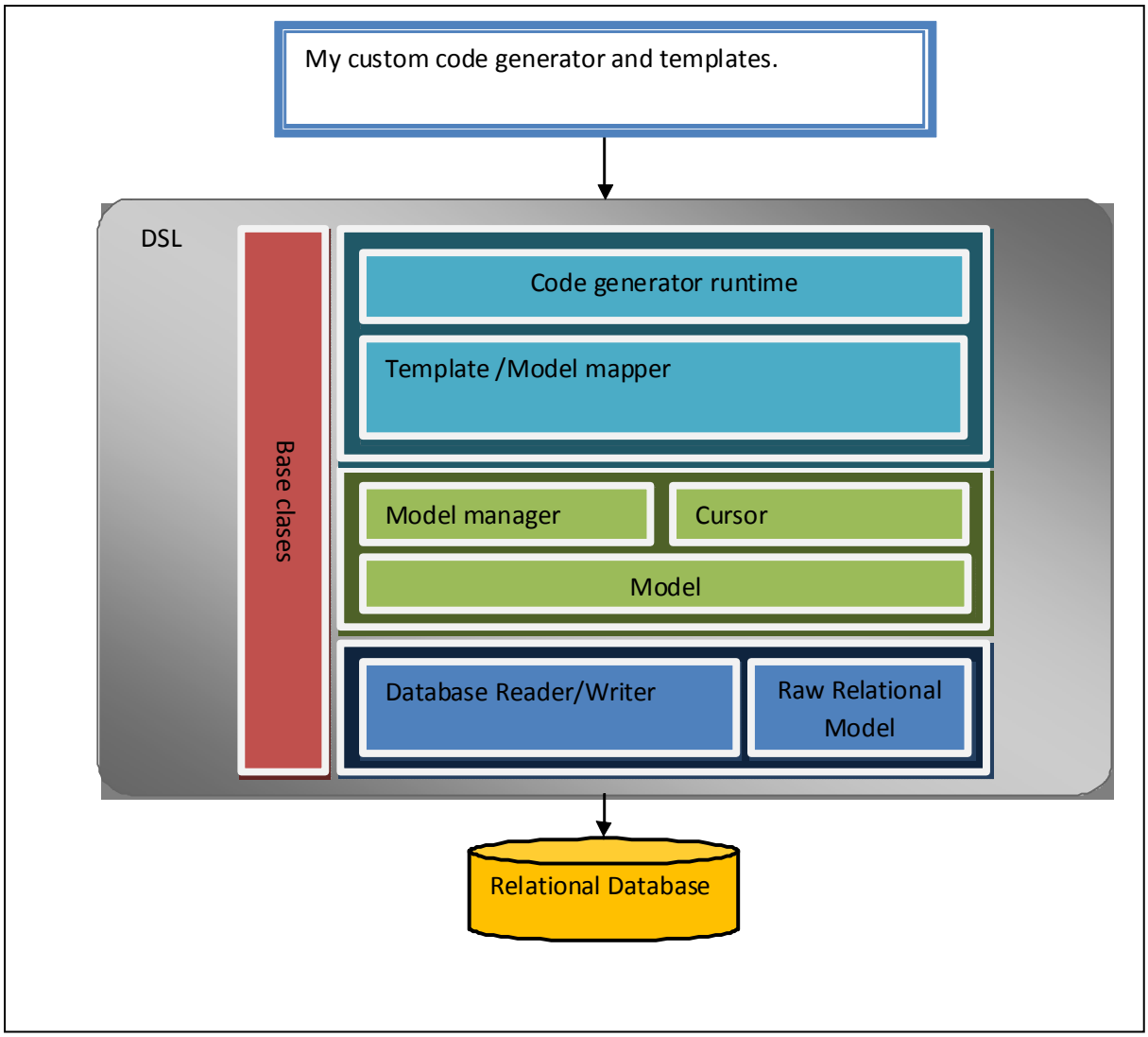


Figure 1: Architecture of the DSL. The language contains the building blocks that make up the code generators and code templates

3.4.2 Model

The model is an extension of the raw relational model; it is aimed at creating artificial objects that actually do not really exist in the database. This allows the user to create custom properties, flags, tags and any additional object in order to apply rules to accomplish particular code generation requirements or scenarios.

3.4.3 Model manager

Is a set of functions that allow adding, editing, deleting and listing, among others, information inside the model.

3.4.4 Cursor

The cursor is responsible for the iterations performed over the model in order to apply the custom code templates. This process starts over the higher level in the hierarchy and nests iterations over the children until the last level. This process also can be run directly over a

given set of objects, for instance, over all the columns without iterating first over databases and tables.

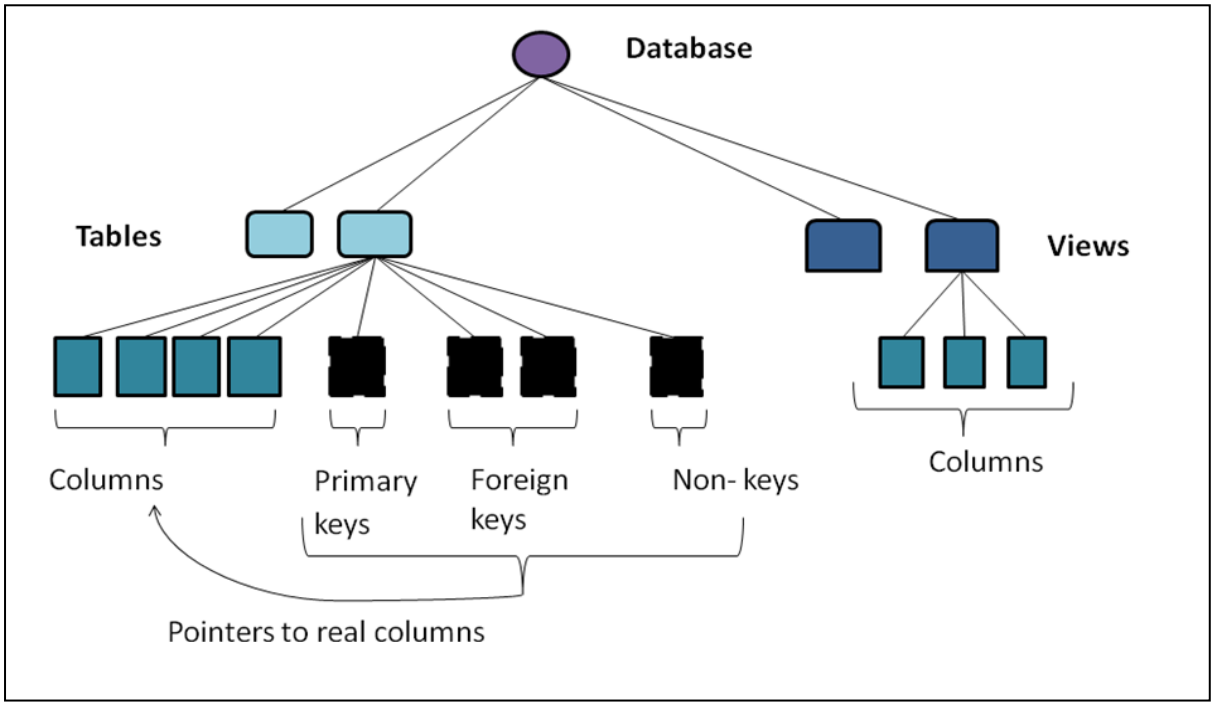


Figure 2: Representation of the relational model as a hierarchy taken from [59]

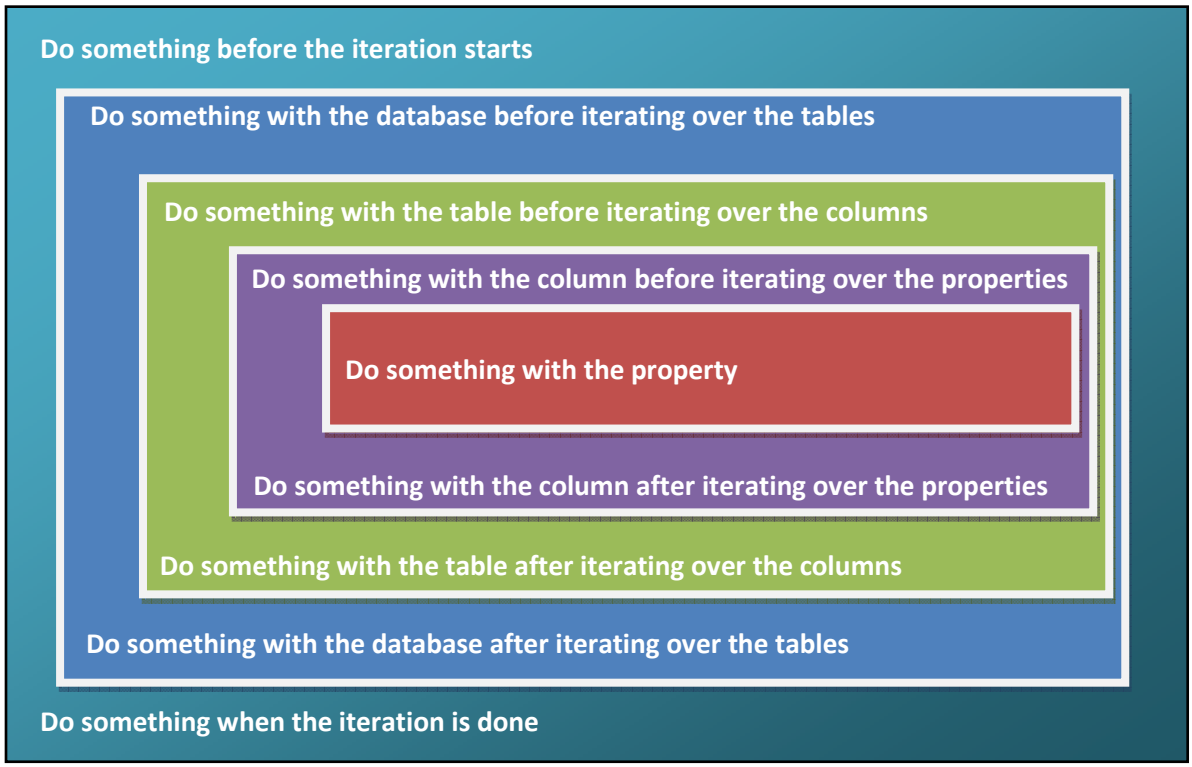


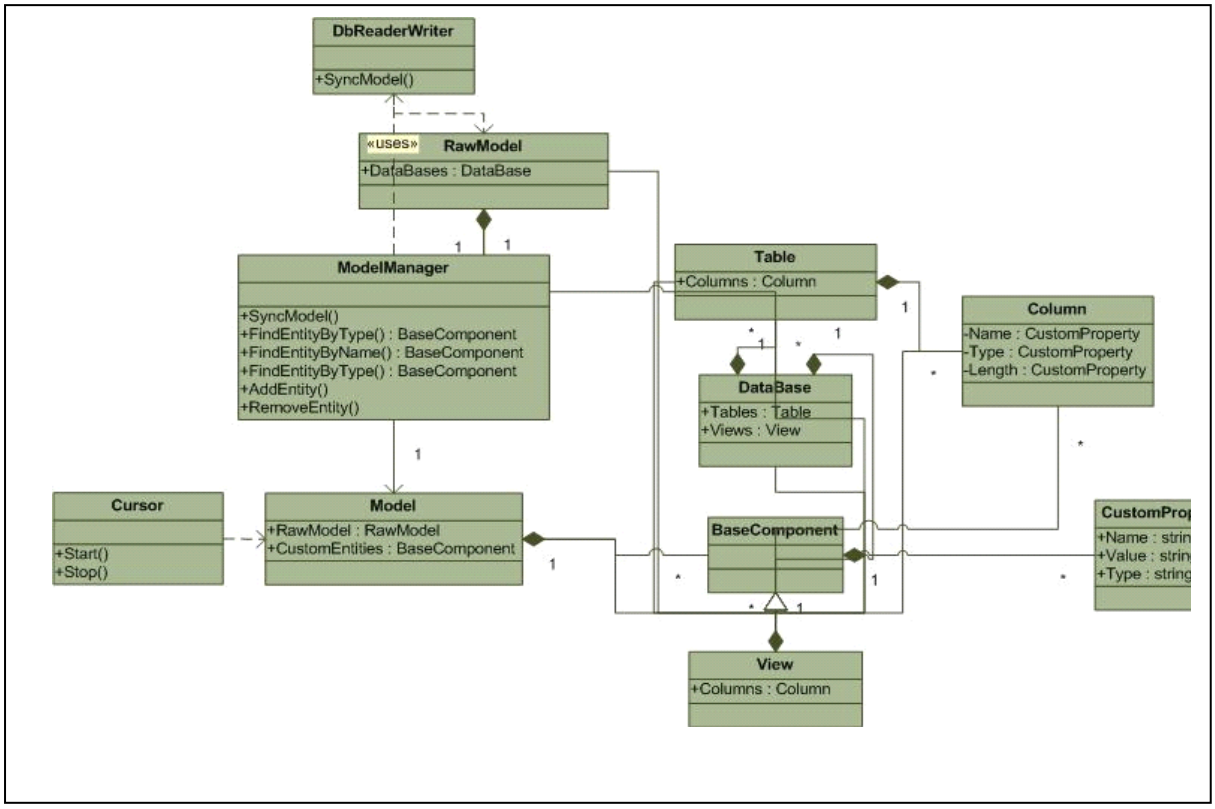
Figure 3: Iteration process performed by code generators over the model

3.4.5 Template/Model mapper

Since the code templates are text files that define what to do with the information retrieved from the catalog, the Template/Model mapper is responsible for finding inside the Model, the appropriate information required for the templates.

3.4.6 Base classes

Here we present the main classes that are part of the core library of this DSL. These classes



are defined taking into account the need of allowing customization to the model.

Figure 4: Static class diagram of the core components over which the DSL is built

The Figure 4 does not show the whole model due to space limitations but shows the interaction between the main components and the methods that run on the code generation process.

- *CustomProperty* allows the creation of properties in runtime and attaches them to an object. This class allows the user to add custom data to the objects stored on the model.
- *BaseComponent* is the base class from which the rest of the model related classes inherit. The internal composition of this class is a set of *Custom properties*.
- *View, Column, Table and DataBase* are the representation of the raw relational model, they all inherit from *BaseComponent* and have static properties that are internally represented as *DynamicProperty*.

3.4.7 Code templates definition

One of the most important parts of any of the code generators reviewed, is the set of code templates. Code templates are text files that mix pieces of the final code that shall be produced by the generator with the properties inside de model. These templates should be as clear as possible in order to facilitate code maintenance and rapid customization, and should also avoid unnecessary content. Following these guidelines, we have developed a declarative language for code templates based on the syntax of CSS (Cascading Style Sheets) [60]. The basic structure of a code template is:

```
@Selector{ expression }

#SelectorId{ expression }

.Class{ expression }
```

- ***@Selector***: Specifies an element inside the model, this could be a Table, Column, Database or any user-defined element.
- ***#SelectorId***: Specifies the *Id* of any element inside de model, when this type of template is used, the *expression* is applied only to that element identified with *#SelectorId.*
- ***.Class****:* The *class* is a grouping value shared for various elements inside the model.
- ***Expression***: A mixture of plain text, properties of an element and nested templates. The following example will be useful to clarify an *expression*.

```
@Table{
        Public Class $id{
                @Column{ Private $Type $id; }
        } }
```

This template is applied to every element of type *Table* inside the model, the inner expression defines that the text "*Public Class* " should be written followed by the value of the property *$id* of the current *Table*, then, for each *Column* element that is directly related to the current *Table*, the text "*Private*" is written followed by the values of the properties *$Type* and *$id* of the current *Column.*

4. Proposed solution

Based on the models described in section 3 and trying to address the problems that developers face when building code generators with general purpose programming languages, we are developing a cross-platform DSL aimed at simplifying the process of building custom code generators, reducing development time and improving the quality since the language already includes tested components and functionalities commonly used in code generation process.

Extending the architecture shown in Figure 1, our final solution will offer to the developers the possibility of building custom code generators that run over different platforms and can be extended or integrated with others to reuse functionalities and meet particular needs.

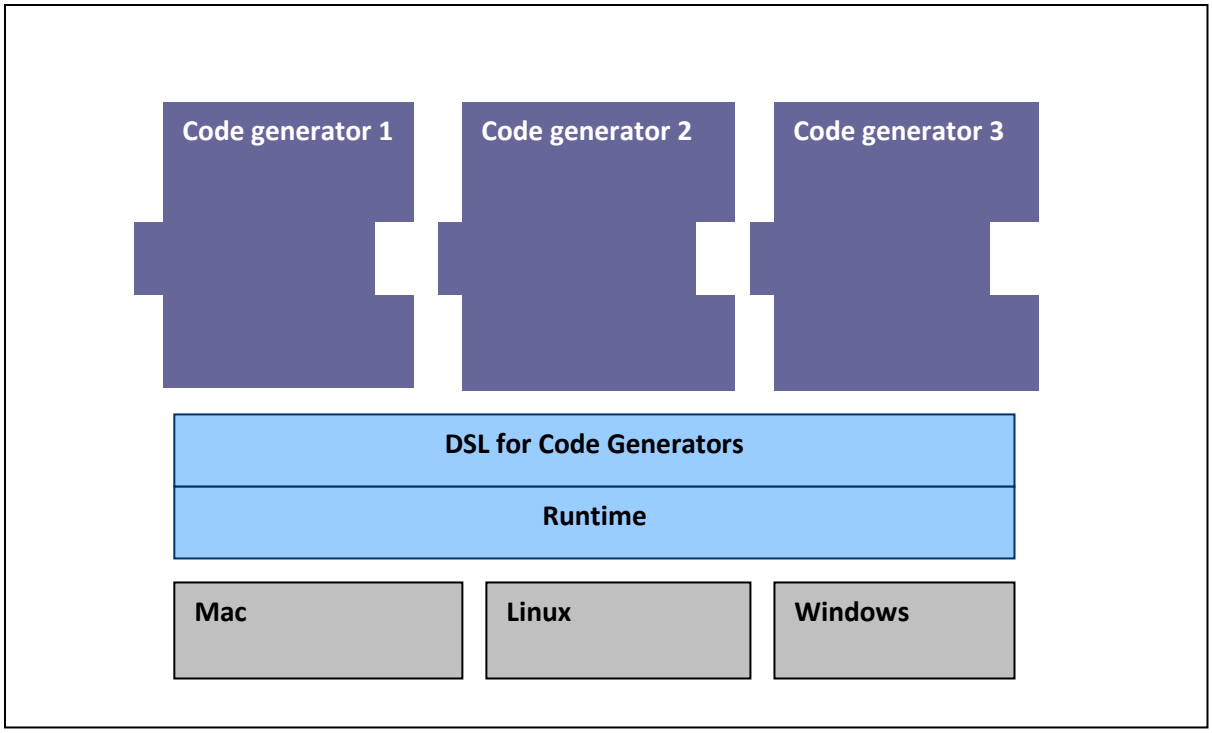


Figure 5: Cross platform architecture of the DSL.

Figure 5 shows how code generators can be integrated and even run on different platforms. This is because they are built with a standardized tool that works with different operating systems.

Benefits of using this solution are wide as developers are able to create and maintain their code generators easier than before since less code is needed to implement the same functionalities. Software vendors can spend more resources improving their code generators instead of building them from scratch. In the case that one client does not want to use a code generator available on the market, she is able to create a custom tool that meets particular needs in a code generation scope taking advantage of functionalities and components available in this solution. In all cases, efficiency improvement and time reduction are big gains where a code generator is an important part of any software development project.

With the aim of giving a clearer overview of what we are currently doing, here we present a conceptual example of how a custom code generator would look like implemented with our solution proposed. These examples are part of the test cases that we will use to measure the accuracy of the DSL creating a code generator and running it over different platforms and with different database engines.

Conceptual example

In this section we present some high level conceptual examples of how this DSL should be used and how it should behave.

Let us suppose that we have a database called StudentsDB. There are two tables as follows.

- Student

    - StudentId, Int
    - Name, varchar(100)
    - CountryId, Int
- Country
    - CountryId, Int
    - Name, varchar(100)

Both tables are related by CountryId column.

When these tables are loaded into the *Raw Model* they are represented as specified in [59]. Then the *Raw Model* is extended in the *Model*. Now we define the code templates as follows:

```
@Table{
        Public Class $id{
                @Column{
                    Private $Type $id;
                    Public $Type get$id (){ return $id;}
                        Public void set$id ($Type p$id){ $id = p$id;}
                }
        }
}
```

When the *Cursor* passes through the *Model*, the output would be:

```
Public Class Student{
    Private int StudentId;
    Public int getStudentId(){ return StudentId;}
    Public void setStudentId (int pStudentId){ StudentId = pStudentId;}

    Private string Name;
    Public string getName(){ return Name;}
    Public void setName (string pName){ Name = pName;}

    Private int CountryId;
    Public int getCountryId (){ return CountryId;}
    Public void setCountryId (int p CountryId){ CountryId = pCountryId;}
}

Public Class Country{
    Private int CountryId;
    Public int getCountryId (){ return CountryId;}
    Public void setCountryId (int p CountryId){ CountryId = pCountryId;}

    Private string Name;
    Public string getName(){ return Name;}
    Public void setName (string pName){ Name = pName;}
}
```

Notice that in the *Raw Model* the column "*name"* in both tables is of type *varchar* and not of type *string*. The user who builds the code generator according to his particular requirements should provide this type mapping.
Now let us suppose that we need to change the way primary keys are generated. We want to change the getter and setter to GetID and SetID respectively, the rest of the columns stay the same.

```
@Table{
    Public Class $id{
        .PrimaryKey{
            Private $Type $id;
            Public $Type GetID (){ return $id;}
                Public void SetID ($Type p$id){ $id = p$id;}
        }
        @Column{
            Private $Type $id;
            Public $Type get$id (){ return $id;}
                Public void set$id ($Type p$id){ $id = p$id;}
        }
    }
}
```

This template uses two selectors, one for any element of class "PrimaryKey" which is provided by the *Raw Model* representation and one for every *Column* element inside the *Model*.

5. Conclusions

This work gives an overview of the Code Generation Domain constrained to applications based on relational databases. With this approach one can implement the models and components described above in different platforms and technologies to allow developers to easy build and maintain code generator that meet their particular code generation requirements. This language includes most common patterns found in code generators and the architecture allows extending its components in order to enrich the base platform.

The solution proposed in this work solves the problems derived from building custom code generators with general purpose programming languages.

6. Future work

The next phases of this work are the implementation and performance tests. Python and Ruby are the candidates as implementation tool since they are easy to learn by programmers, allow rapid program writing due to their meaningful and flexible structure and also provide an easy manner to introspect the code which is a plus in this domain. Performance tests after the implementation phase will be needed to evaluate the accuracy of the components defined in this work.

In addition, bearing in mind that design-science research has become more popular and accepted as a research method in Information System, we want to frame this work inside the

premises of the design science approach (Hevner et al. (2004) [61]). For instance, in agreement with Järvinen P. (2005) [62], our work is framed in a case of study oriented at solving a specific problem, where the technological rules (of Domain Specific Language) are being developed and tested in close collaboration with the people in the interest area (designers, developers, programmers).

Acknowledgements

The authors acknowledge the anonymous reporters who, by their comments, contributed to the improvement of the text.

References

[1] Herrington, Jack (2003) *Code Generation in Action*. Greenwich, USA: Manning Publications Co.

[2] Krzysztof, Czarnecki (1998) *Generative Programming, Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*. Department of Computer Science and Automation. Technical University of Ilmenau.

[3] Delta Software Technology Schmallenberg, Germany, ANGIE. Available from: http://www.d-s-t-g.com/angie [Accessed 12 Oct 2007]

[4] NCodeGen. Available from: http://ncodegen.sf.net [Accessed 12 Oct 2007]

[5] MDef. Available from: http://mdef.sourceforge.net [Accessed 12 Oct 2007]

[6] Eva-lsx Generator Available from: http://www.elxala.de [Accessed 12 Oct 2007]

[7] Smith Eric, CodeSmith. Available from: http://www.ericjsmith.net/codesmith/ [Accessed 12 Oct 2007]

[8] NVelocity. Available from: http://nvelocity.sourceforge.net/ [Accessed 12 Oct 2007]

[9] Velocity. Available from: http://jakarta.apache.org/velocity/ [Accessed 12 Oct 2007]

[10] X-Code .NET. Available from: http://www.arithex.com/xcc.html [Accessed 12 Oct 2007]

[11] e-GEN. Available from: http://www.gentastic.com/ [Accessed 12 Oct 2007]

[12] nAlliance Corporation P.O. Box 7051 Romeoville, IL 60446, nGeneration. Available from: http://www.nDevelopment.net [Accessed 12 Oct 2007]

[13] MyGeneration. Available from: http://www.MyGenerationSoftware.com [Accessed 12 Oct 2007]

[14] Lattice Business Software Intl, Inc., Lattice.SPGen. Available from: http://www.latticesoft.com [Accessed 12 Oct 2007]

[15] Propel. Available from: http://propel.phpdb.org [Accessed 12 Oct 2007]

[16] EazyCode. Available from: http://www.eazycode.com [Accessed 14 Oct 2007]

[17] Solutions Design, LLBLGen Pro. Available from: http://www.llblgen.com [Accessed 14 Oct 2007]

[18] Oxygen Code Generator. Available from: http://www.techinceptions.com/codegenerator.html [Accessed 14 Oct 2007]

[19] Deklarit. Available from: http://www.deklarit.com/ [Accessed 14 Oct 2007]

[20] MetaStorage. Available from: http://www.meta-language.net/metastorage.html [Accessed 14 Oct 2007]

[21] Iron Speed Designer. Available from: http://www.ironspeed.com/download [Accessed 14 Oct 2007]
[22] VBeXpress.NET. Available from: http://www.vbexpress.com/index.asp [Accessed 14 Oct 2007]
[23] XCoder. Available from: http://sourceforge.net/projects/xcoder/ [Accessed 14 Oct 2007]
[24] Finalist IT Group Finalist IT Group b.v. P.O. Box 1354 3000 BJ Rotterdam The Netherlands, JAG. Available from: http://jag.sourceforge.net [Accessed 14 Oct 2007]
[25] RFG Software Ltd, CG Pro. Available from: http://www.rfgsoftware.com/RFG.aspx?ID=12 [Accessed 18 Oct 2007]
[26] JCodeBox. Available from: http://www.jcodebox.com/dwn_eval_frm.asp [Accessed 18 Oct 2007]
[27] Clarion/PHP templates. Available from: http://www.softvelocity.com/php/php.htm [Accessed 18 Oct 2007]
[28] realMethods Framework. Available from: http://www.realmethods.com [Accessed 18 Oct 2007]
[29] BrightSword Designer Professional Edition. Available from: http://www.brightsword.com [Accessed 18 Oct 2007]
[30] Razor Source, Inc. P.O Box 34594 Indianapolis, IN 46234, Source Cutter 2.0. Available from: http://www.RazorSource.com [Accessed 18 Oct 2007]
[31] Automated Architecture, Inc Reston VA 20190, Blue Ink. Available from: http://www.blueink.biz [Accessed 18 Oct 2007]
[32] Leeonsoft Company, Dreamsource. Available from: http://www.leeonsoft.com [Accessed 18 Oct 2007]
[33] EPFL - LTI, WebLang. Available from: http://ltiwww.epfl.ch/WebLang [Accessed 18 Oct 2007]
[34] TurnObjects Louisville, KY, TurnObjects. Available from: http://www.turnobjects.com/Overview/Default.aspx [Accessed 18 Oct 2007]
[35] Vgo Software Manchester, CT, USA, Rev. Available from: http://www.vgosoftware.com/products/rev/index.php [Accessed 18 Oct 2007]
[36] CodeCharge. Available from: http://www.codecharge.com/ [Accessed 18 Oct 2007]
[37] Fenix ASP.NET generator. Available from: http://net.radventure.nl/Fenix/Index.aspx [Accessed 18 Oct 2007]
[38] Genexus. Available from: http://www.genexus.com/main/hgenexus.aspx [Accessed 18 Oct 2007]
[39] CompileX. Available from: http://www.compilex.com/ [Accessed 18 Oct 2007]
[40] GENNIT Code Generation, GENNIT. Available from: http://gennit.com [Accessed 19 Oct 2007]
[41] Nolics.net. Available from: http://www.nolics.net/download_nolicsnet2005.aspx#latest [Accessed 19 Oct 2007]
[42] THINKUI SOFTWARE INC, ThinkUI SQL Client. Available from: http://www.thinkui.com [Accessed 19 Oct 2007]
[43] Pattern By Example. Available from: http://pbe.d-s-t-g.com/ [Accessed 19 Oct 2007]

[44] phpCodeGenie 3.0. Available from: http://phpcodegenie.sourceforge.net/ [Accessed 19 Oct 2007]

[45] Using Domain Specific Languages. Available from: http://martinfowler.com/dslwip/UsingDsls.html [Accessed 19 Oct 2007]

[46] Linguo Yu (2006) "Prototyping, Domain Specific Language, and Testing". *Indiana University South Bend, South Bend, IN 46615 USA.*

[47] About ColdFusion MX, Adobe. Available from: http://livedocs.adobe.com/coldfusion/6.1/htmldocs/introb6.htm#wp1116971 [Accessed 19 Oct 2007]

[48] MedisWiki templates, MedisWiki.org. Available from: http://www.mediawiki.org/wiki/MediaWiki [Accessed 19 Oct 2007]

[49] Windows PowerShell, Microsoft Corp. Available from: http://www.microsoft.com/windowsserver2003/technologies/management/powershell/default.mspx [Accessed 19 Oct 2007]

[50] Python Programming Language. Available from: http://www.python.org/ [Accessed 19 Oct 2007]

[51] Ruby Documentation. Available from: http://www.ruby-lang.org/en/documentation/ [Accessed 19 Oct 2007]

[52] Haskell. Available from: http://haskell.org/haskellwiki/Haskell [Accessed 19 Oct 2007]

[53] About SAX. Available from: http://www.saxproject.org/ [Accessed 19 Oct 2007]

[54] Jack Greenfield (2006) "Bare Naked Languages or What Not to Model". *The Architecture Journal, 9*, pp. 2-9.

[55] David Atkins et al (2001) "Mawl: a Domain-specific Language for Form-based Services". *Software Production Research Department. Bell Laboratories, Lucent Technologies. Room 2A-314, 263 Shuman Blvd., Naperville, IL 60566.*

[56] Arie van Deursen and Paul Klint (1998) "Domain-Specific Language Design Requires Feature Descriptions". *CWI. ACM Computing Classification System: D.2.2, D.2.9, D.2.11, D.2.13. P.O. Box 94079, 1090 GB Amsterdam, The Netherlands.*

[57] Paul Hudak (2008) "Modular Domain Specific Languages and Tools". *Department of Computer Science, Yale University. New Haven, CT 06520.*

[58] Expert Coder. Available en http://expertcoder.sourceforge.net/ [Accessed 19 Oct 2007]

[59] Julian Vargas and Helga Duarte (2009) "An Approach to a Query Functional Language for Relational Models Represented in Hierarchic Structures". *Tendencias en Ingeniería de Software e Inteligencia Artificial, 3, Escuela de Sistemas de la Universidad Nacional de Colombia, Medellín, Colombia*. pp 37-42.

[60] CSS Syntax, w3schools. Available from: http://www.w3schools.com/Css/css_syntax.asp [Accessed 1 Nov 2007]

[61] Hevner A.R., March, S.T., Park, J., and Ram, S. (2004). Design science in information systems research. MIS Quarterly, 28 (1), 75-105

[62] Järvinen P. (2005) *Action research as an approach in design science. D*epartment of Domputer Sciences. University of Tampere, Finland. Presented in THE EURAM (European Academy of Management) Conference, Munich, May 4-7, 2005, in track 28: Design, Collaboration and Relevance in Management Research.