AN ENVIRONMENT BASED ON PRE-CONCEPTUAL SCHEMAS FOR AUTOMATICALLY GENERATING SOURCE CODE UNDER THE MVC PATTERN

UN ENTORNO PARA LA GENERACIÓN AUTOMÁTICA DE CÓDIGO BAJO EL PATRÓN MVC A PARTIR DE ESQUEMAS PRECONCEPTUALES

CARLOS MARIO ZAPATA

Ph.D., Facultad de Minas, Universidad Nacional de Colombia, Sede Medellín, cmzapata@unal.edu.co

JOHN JAIRO CHAVERRA

M.Sc., Facultad de Minas, Universidad Nacional de Colombia, Sede Medellín, jjchaver@unal.edu.co

Received for review November 11 th, 2011, accepted June 6th, 2012, final version July, 17 th, 2012

ABSTRACT: Computer-aided software engineering (CASE) tools are intended to support several phases of the software development lifecycle. At the present, it is almost impossible to develop software applications without them. One of the most desired features of CASE tools is its automated source code generation. Nowadays, several CASE and metaCASE tools attempted to do such generation, but they share two main problems: they use, as a starting point, technical languages which are not suitable for stakeholder validation, and they generate an incomplete and non-patterned source code. This is why in this paper we propose a CASE tool for automatically generating a PHP source code under the model view controller (MVC) programming pattern. We use pre-conceptual schemas as a starting point. We aim to generate a complete source code and to ease stakeholder validation.

KEYWORDS: CASE, MetaCASE, GEF, MOFScript, EMF, MVC, PHP

RESUMEN: Las herramientas CASE (computer-aided software engineering) apoyan las diferentes fases del desarrollo de software, a tal punto que hoy en día es casi imposible pensar en un desarrollo de software sin el apoyo de una de ellas. Una de las características más deseadas de estas herramientas es la generación automática de código. Actualmente, existen herramientas CASE y metaCASE que realizan esta función, pero con dos problemas principales: parten de lenguajes técnicos que no permiten la validación de los interesados y se genera código incompleto y sin un patrón arquitectónico definido. Por ello, en este artículo se propone una herramienta CASE para generar automáticamente código ejecutable bajo el patrón de programación MVC (model view controller), en el lenguaje de programación PHP, a partir de los esquemas preconceptuales. Se busca hacer más completo el código resultante y permitir la validación de los interesados en el proceso.

PALABRAS CLAVE: CASE, MetaCASE, GEF, MOFScript, EMF, MVC, PHP

1. INTRODUCTION

CASE tools are increasingly used for almost every phase of the software development lifecycle. Automated code generation is one of the most desired features of CASE tools, but a vast majority of these tools are still far from having a fully-automated source code generation process. As a matter of fact, these tools do not generate a fully functional code, but a "template" (mainly containing classes, attributes, and method headers) to be completed by the programmer. Some of these tools are: Together™ [1], Rose™ [2], and Fujaba® [3].

Also, CASE tools are based on technical languages (for example UML diagrams), distant from stakeholder understanding. In the software development lifecycle, the stakeholder must interact with the analyst to validate the information during the first phases, and such interaction is difficult when we use technical artifacts in the communication process.

MetaCASE tools try to overcome the main constraints of CASE tools, because they have devices for creating modeling patterns and checking syntactical rules [1]. However, these tools are still based on technical

languages and the process of code generation must be explicitly defined inside the MetaCASE tool.

Chaverra [5] proposes a set of rules in order to automatically obtain the source code under the MVC programming pattern. Pre-conceptual schemas are used as pre-conditions of such rules. This proposal is still unincorporated inside well-known CASE tools. Therefore we develop, in this paper, an environment based on Eclipse® Modeling Framework (EMF), Graphical Modeling Framework (GMF), Object Constraint Language (OCL), and MOFScript in order to represent pre-conceptual schemas and then to translate them into the PHP source code under the MVC pattern.

The rest of this paper is organized as follows: in Section 2 we define the domain concepts belonging to the theoretical framework; in Section 3 we summarize some work on CASE and metaCASE tools for automatically generating a source code; Section 4 is devoted to the elements of the EMF-based implementation environment; in Section 5, the case study for illustrating the use of the proposed CASE tool is developed. Conclusions and future work are presented in Sections 6 and 7, respectively.

2. THEORETICAL FRAMEWORK

The proposal given in this paper requires some understanding of the theoretical foundations related to the modeling language (pre-conceptual schemas), the paradigm for creating the solution (meta-modeling), and the environment for assembling the solution. We describe such elements in this section.

2.1. Pre-conceptual Schemas

According to Zapata et al. [6], pre-conceptual schemas are diagrams for representing the terminology belonging to a certain domain. They are commonly used for obtaining several conceptual schemas (mostly UML diagrams). The main pre-conceptual schema elements are shown in Fig. 1. The description of such elements is the following [6,7]:

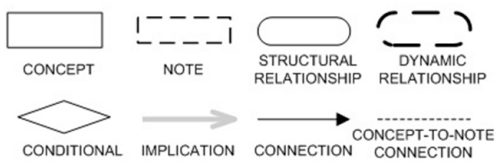

Figure 1. Main pre-conceptual schema elements [6]

- Concept: is either a noun or a noun phrase of the stakeholder discourse.
- Note: is used for listing the several values which can be assigned to concepts.
- Structural relationship: is understood to be a permanent relationship between concepts. Both the verbs "to be" and "to have" are allowed to be used inside this kind of relationship.
- Dynamic relationship: is associated with action verbs.
- Conditional: is understood to be the pre-condition a dynamic relationship must accomplish before its execution.
- Implication: establishes a cause-and-effect relationship between two dynamic relationships or between a conditional and a dynamic relationship.
- Connection: is used to link concepts to (structural or dynamic) relationships. Also, a special kind of connection is used for linking concepts and notes.

2.2. Meta-modeling

Meta models are useful for describing model components in a formal way [1]. In other words, with meta-models it is possible to formally define modeling languages. The Object Management Group (OMG) proposes a modeling architecture for grouping concepts by abstraction levels. So there are four levels in decreasing order of abstraction:

- The meta-meta-model level (M3). This is the highest level of abstraction. Meta object facility (MOF) is located at this level [1]. Concrete meta-models (e.g., UML models) can be defined by means of MOF.
- The meta-model level (M2). The unified modeling language (UML) is located at this level. Some concepts on the M2 level are: class, attribute, and association.
- The system model level (M1). Data models are elements at this level, for example entities like "Person" and "Animal", attributes like "name", and relationships between entities.
- The instance level (M0). The actual system is modeled at this level. Data examples are elements

at this level, e.g., “John Jairo” (name) and “Cr 72b No 90-10” (address).

2.3. The MVC pattern and the PHP language

Reenskaug [8] creates the Model-View-Controller pattern as a multiperspective-based solution to the organization of software applications. Since its inception in the design of software applications, such a pattern has been widely used by software designers. According to Reenskaug, the model represents the knowledge, the views are visual representations of the model, and the controllers are links between the users and the software application. The MVC pattern is useful for representing several types of applications; e.g., the work of Paredes et al. [9].

The hypertext preprocessor (PHP) was selected in this paper as the generated language for the source code due to several reasons: it has possibilities for connection to databases; it is scalable and portable; it is an open-source language; and it is useful for creating dynamic web applications.

2.4. Eclipse-based environment

The proposal in this paper is supported by the CASE tool Eclipse® and the following elements:

- The Eclipse® Modeling Framework (EMF) is a modeling structure for easing code generation processes. Also, EMF is used for building tools and software applications. EMF is based on a structured data model encoded into an XMI model specification [10].
- The Graphical Modeling Framework (GMF) is an Eclipse® plug-in for developing graphical editors, which can be jointly used with other plug-ins. Meta-model instances are expressed in GMF. As a graphical editor, GMF interoperates with a wide range of applications, such as single text processors, completely standardized or customized diagram (UML or any other modeling language) editors, graphical user interfaces (GUI), and so on.
- The Graphical Editing Framework (GEF) is the framework for creating views and graphical editors.
- The Object Constraint Language (OCL) is a language for describing UML model constraints. OCL expressions are used for checking the syntax of meta-model derived languages [11].
- MOFScript is an Eclipse® plug-in for defining rule sets, which, in turn, are used for transforming a structured model to text. Either EMF-supported plug-ins or customized models can be used as input models [10]. In the context of this paper, MOFScript will be used for defining rules related to the pre-conceptual-schema-to-PHP-code transformation.

3. PREVIOUS WORK

MetaCASE tools have been developed with the intention of speeding up and improving the software development process. Most of these tools are based on a graphical modeling language for defining meta-models; for example, class diagrams and entity-relationship diagrams. Some of the most common metaCASE tools are:

- The Domain Modeling Environment (DOME®) is based on the ProtoDOME graphical modeling language, a class-diagram-based language for defining meta-model elements [12].
- A Tool for Multi-Formalism Modeling and Meta-Modeling (AToM3®) is used for describing any conceptual schema. Meta-models are created and edited employing an entity-relationship-based graphical formalism [1].
- The Generic Modeling Environment (GME) uses UML as a modeling pattern. As suggested by the UML superstructure specification, OCL constraint management is included on this metaCASE tool [11].

CASE tools like Together™ [1], Rose™ [2], and Fujaba® [3] are suitable for generating the basic structure of the source code (commonly, they only generate classes and attributes). Some of them also try to complement the source code with object behavioral features. For example, Together™ uses the sequence diagram and Fujaba® uses story diagrams (mixing activity and communication diagrams). In Table 1, we include an analysis of previous research.

Table 1. Analysis of previous research

Tool	Language	Code Completeness	MVC pattern
DoME	Technical	Depends on the way in which they are programmed	Depends on the way in which they are programmed
AToM3			
GME			
Together	Graphical, technical	Only structure	Model
Rose		Only structure	Model
Fujaba		Structure and some behavior	Model and some controller

As a summary, the previous projects commonly employ technical modeling languages which are beyond stakeholder comprehension. As a consequence, stakeholders can barely validate the information in the software development lifecycle. Also, these projects only generate a portion of the source code. By doing so, the solutions provided are highly programmer-dependent. The above reasons lead us to propose a CASE tool for incorporating some code generation rules defined by Chaverra [5] oriented towards the PHP programming language under the MVC pattern, using pre-conceptual schemas as a starting point.

4. A PROPOSAL FOR A CODE-GENERATION ENVIRONMENT

Chaverra [5] proposes a set of heuristic rules for transforming a pre-conceptual schema-based domain description into source code under the MVC pattern. Pre-conceptual schemas use a graphical notation with similarities with the natural language. Also, they are useful for expressing several elements of a stakeholder's discourse. The proximity with the natural language structure leads the stakeholder to constantly validate such schemas with no need for additional training. In this paper, we propose the environment of a CASE tool for applying the transformation rules defined by Chaverra [5].

4.1. Meta-model definition

A meta-model, in the EMF environment, is specified by class diagrams. By using such notation, in Fig. 2 we define the pre-conceptual schema meta-model. Every meta-model class can be assigned to a graphical representation.

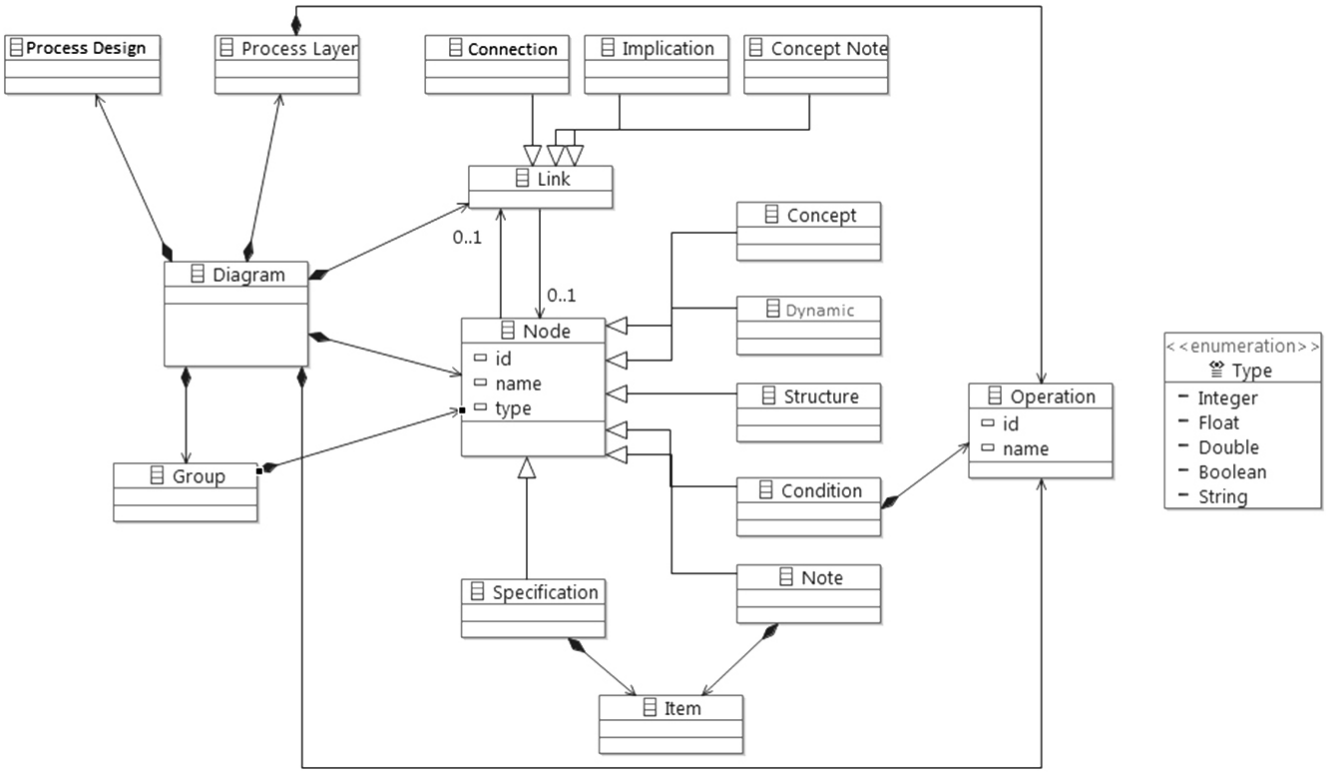

Figure 2. Pre-conceptual schema meta-model

Inside structural relationships, only the verbs "to be" and "to have" are allowed. For dynamic relationships, any other kind of verb is allowed. *Node* and *link* are the central classes of the meta-model, because they gather the elements for drawing the pre-conceptual schema. The *type* enumeration is defined in the meta-model, but is intended to be used in the transformation to languages other than PHP (Java, for example.)

4.2. Definition of the graphical elements of pre-conceptual schemas

By default, GMF represents every element of the meta-model as a rectangle. However, the user is allowed to change this representation by using several figures, like ellipses, rounded-corner rectangles, and customized figures. In Fig. 3, the image for the dynamic relationship is defined as a rounded rectangle with a name.

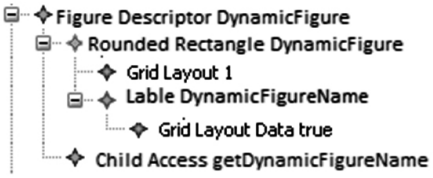

Figure 3. Definition of the graphic elements from pre-conceptual schema

4.3. OCL-based pre-conceptual schema validation

We use OCL in the context of this paper for validating the syntax of the pre-conceptual schema. With OCL, it is possible to define connection constraints among the artifacts. Figure 4 shows an example of OCL inside EMF for defining the one-to-one connection between a concept and a note. The overall validation rules are defined in the same way.

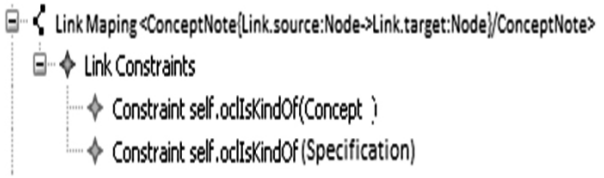

Figure 4. Definition of OCL rules on GEF

4.4. MOFScript implementation of the validation rules

MOFScript is used for automatically obtaining PHP source code from the pre-conceptual schema. The rules defined by Chaverra [5] are programmed inside MOFScript. Be advised that the MVC pattern emerges at this point because the rules are intended to generate the model, the views, and the controllers. In order to complete this task, we must design the pre-conceptual schema in GEF, and then MOFScript must follow the conceptual hierarchy of the XMI file belonging to the schema (as exemplified in Fig. 5, in the case of filling in the information of a concept with its name.) Once the hierarchy is recognized, we must apply the transformation rules. In Fig. 5, the attribute *id* is not present in the pre-conceptual schema because it is only used for identifying the concepts in the database (and, in such a case, it is a default attribute.)

5. CASE STUDY

A case study (based on the sale record of a supermarket) is proposed for the sake of exemplifying the heuristic rules defined by Chaverra [5].

```
texttransformation Preconceptual(in Model:"Preconceptual"){
        property concept:Hashtable = ""
        model.Diagram.main(){
                self.nodes->foreach(c:model.Concept){
                        concept.put(c.id, c.name)
                }
        }
}
```

Figure 5. MOFScript code for storing all the node information

"A supermarket wants to register daily sales. This task is performed by a cashier. The supermarket has products and, for each one, the number, the sale price, the real price, the description, and the amount are registered. When the client selects a product, the cashier registers the sale, which includes a code, a total, and a detail (i.e., detailed information about each product), expressed in terms of the quantity, the subtotal, and the product. Both the client and the cashier have an i.d., a name, a phone number, and an address."

Figure 6 contains the obtained pre-conceptual schema, as a representation of the domain in this case study.

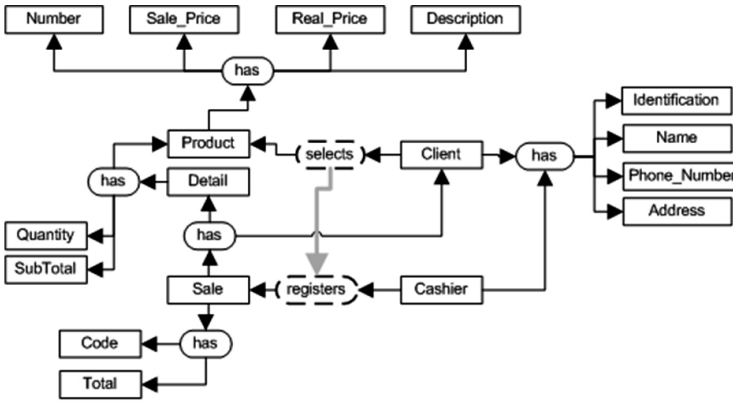

Figure 6. Pre-conceptual schema obtained for the case study

The following subsections detail the results—in the PHP programming language—for applying the rules of Chaverra [5] to the pre-conceptual schema of Fig. 6. We summarize the results for the model, the view, and the controller (the aforementioned MVC pattern). Due to space restrictions, we only show the results of the "sale" concept. For the sake of clarity, we must remark that the interface belonging to an object includes all the attributes of the object. In other words, there is no possibility for a specific stakeholder to fill in some part of the information while the rest is filled by another stakeholder. The current project has no additional rules for covering such differences among stakeholders.

The automation level of the process releases the analyst responsibilities in applying the heuristic rules. The resulting code is completely functional and it preserves traceability in the names associated to every concept, a highly desirable feature in an automatically-generated code. Be advised that the heuristic rules act as "templates" in which we take the elements belonging to the pre-conceptual schema and we map them onto the source code by filling in the blanks of the heuristic rules. By doing so, we are guaranteeing the usage of the concepts and relationships from the pre-conceptual schema in generating the PHP source code.

5.1. Model

When we register a sale (see Fig. 7) according to the pre-conceptual schema, we must fill in the information of the code, the total, and the client i.d. for the sale. If we want to fill the information of the details directly, we must put another dynamic relationship in the pre-conceptual schema. The SQL sentences belonging to this model are also generated by using the rules of Chaverra [5].

```php
<?php
class SaleModel extends ModelBase{

    public function register($code, $total, $clientId){
        $sql = $this->db->prepare(
            "INSERT INTO sales ('code','total','client_id')
            VALUES ($code, $total, $clientId)");

        $sql->execute();
    }
}
?>
```

Figure 7. Model for the case study

5.2. View

In the case of the view (see Fig. 8,) the information about the detail is provided in the graphical user interface. "Rice" and "salt" are product description values which have arisen from the information we previously put in the database. Hence, if we have more values for product description in the database, they will be displayed in this interface.

```html
<html>
<head>
    <title>Register Sale</title>
</head>

<body>
<FORM  action='register' method='post'>
    Code  :<INPUT type='text' name='code'><br/>
    Total :<INPUT type='text' name='total' ><br/>
    <fieldset>
        <legend>Detail</legend>
        Product:
        <select>
            <option>Rice</option>
            <option>Salt</option>
        </select>
        Quantity :<INPUT type='text' name='quantity'><br/>

        Sub Total <INPUT type='text' name='sub_total'><br/>

        <INPUT type='button' name='add' value='add'><br/>
        <table>
            <tr>
                <th>Name</th>
                <th>Quantity</th>
                <td>SubTotal</td>
            </tr>
        <table>
    </fieldset>
    <INPUT type='submit' name='register' value='Register'><br/>
</FORM>
</body>

</html>
```

Figure 8. View for the case study

5.3. Controller

The controller for registering sales (see Fig. 9) has some information missing from the pre-conceptual schema: the client i.d. and the product i.d. are internal numbers in the database for identifying the records belonging to clients and products, respectively.

```
<?php
class SalesController extends ControllerBase{

    public function register(){
        $code     = $data['Sale']['code'];
        $total    = $data['Sale']['total'];
        $clientId = $data['Sale']['client_id'];

        $sale = New SaleModel();
        $sale->register($code, $total, $clientId);

        $detail = new DetailModel();
        foreach($data['Detail'] as $key => $value){
            $quanity   = $value['Detail']['quantity'];
            $subTotal  = $value['Detail']['sub_total'];
            $productId = $value['Detail']['product_id'];
            $saleId    = $sale->id;

            $detail->register($quantity, $subTotal, $productId, $saleId);
        }
    }
}
?>
```

Figure 9. Controller for the case study

6. CONCLUSIONS

In this paper, we proposed a CASE tool based on the EMF environment and using GML, OCL, and MOFScript. For this environment, the domain representation is accomplished by using pre-conceptual schemas. This environment is also capable of generating a PHP source code under the MVC pattern by applying the rules defined by Chaverra [5]. The main contributions of this proposal are:

- An automated process—with no intervention of both analysts and programmers—is defined, improving code quality and avoiding human errors in applying the rules.
- Traceability is maintained along the entire software development process.
- A fully-functional prototype is quickly obtained. Early validation of code is possible for stakeholders without any technical knowledge on programming languages.
- Communication among stakeholders and analysts is improved.

7. FUTURE WORK

Some of the issues to be covered by future projects are:

- The aforementioned specialization of user interfaces by roles.
- The inclusion of data types inside the pre-conceptual schema as a way to prepare the conversion to other languages like Java and C#.
- The automated generation of highly-complex artifacts of documentation. The detailed use case specification is one possible example.

8. ACKNOWLEDGMENTS

This work is partially funded by the *Vicerrectoría de Investigación de la Universidad Nacional de Colombia,* through the research project *Transformación semiautomática de los esquemas conceptuales, generados en unc-diagramador, en prototipos funcionales*.

REFERENCES

[1] Borland Software Corporation. Borland Together Architect. Available: http://www.borland.com/us/products/together/index.html [cited september 2012].

[2] IBM Corporation. Rational Rose Architect™. Available: http://www.306.ibm.com/software/awdtools/architect/swarchitect/index.html [cited september 2012].

[3] Geiger, L. and Zündorf, A., TOOL modeling with fujaba. Electronic Notes in Theoretical Computer Science, (148), pp. 173–186, 2006.

[4] De Lara, J. and Vangheluwe, H., AToM3: A tool for multi-formalism and meta-modeling. European Joint Conference on Theory and Practice of Software (ETAPS), Fundamental Approaches to Software Engineering (FASE). Grenoble, France, pp. 174–188, April 2002.

[5] Chaverra, J., Generación automática de prototipos funcionales a partir de esquemas preconceptuales [M.Sc. Thesis]. Medellín, Colombia: Universidad Nacional de Colombia, 2011.

[6] Zapata, C. M., Gelbukh, A. and Arango, F., Pre-conceptual Schemas: A Conceptual-Graph-Like Knowledge

Representation for Requirements Elicitation. Lecture Notes in Computer Sciences, 4293, pp. 17–27, 2006.

[7] Zapata, C. M., Gelbukh, A. and Arango, F., UN–Lencep: Obtención Automática de Diagramas UML a partir de un Lenguaje Controlado. Arturo Hernández, José L. Zechinelli (Eds.) Avances en la Ciencia de la Computación, pp. 254–259, 2006.

[8] Reenskaug, T., Models-Views-Controllers. Technical Note. Palo Alto, CA: USA. Xerox PARC, December 1979.

[9] Paredes, M., Villamizar, J., Bautista, L. and Rodríguez, D., The structure of the computational signal algebra and its application in digital image processing. Dyna, 78(166), pp. 118–132, 2011.

[10] Moore, B., Dean, D., Gerber, A., Wangenknecht, G. and Vanderheyden, P., Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM International Technical Support Organization Redbooks, New York, 2004.

[11] Object Management Group (OMG). Unified modeling language specification. Version 2.1.1. Available: http://www.omg.org/uml/ [cited september 2012].

[12] DOME Users Guide. Available: http://www.htc.honeywell.com/dome/support.html#documentation. [Cited june 2010].