World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:11, No:2, 2017

Definition of a Computing Independent Model and Rules for Transformation Focused on the Model-View-Controller Architecture

Vanessa Matias Leite, Jandira Guenka Palma, Flávio Henrique de Oliveira

***Abstract*—**This paper presents a model-oriented development approach to software development in the Model-View-Controller (MVC) architectural standard. This approach aims to expose a process of extractions of information from the models, in which through rules and syntax defined in this work, assists in the design of the initial model and its future conversions. The proposed paper presents a syntax based on the natural language, according to the rules agreed in the classic grammar of the Portuguese language, added to the rules of conversions generating models that follow the norms of the Object Management Group (OMG) and the Meta-Object Facility MOF.

***Keywords*—**Model driven architecture, model-view-controller, bnf syntax, model, transformation, UML.

I. Introduction

SOFTWARE development is an area that still presents several challenges. From the unfolding of the problem to the code, there are several steps, and it is a labor intensive one. The challenges are to streamline, facilitate and improve the unfolding of the various stages of development. Another challenge is that every time technology changes or evolves, a lot of work still needs to be redone [1]. However, some software development methodologies minimize some of these difficulties, such as Model Driven Development (MDD). MDD is an approach that uses models as primary development artifacts, starting from a higher level of abstraction [2]. MDD provides an increase in productivity, greater automation, reduced vulnerability to technological changes and other benefits [3], [4]. An implementation of MDD is the Model Driven Architecture (MDA) maintained by the Object Management Group (OMG).

According to the OMG, the MDA is a set of standards used in a joint way, being the Unified Model Language (UML) and Meta-Object Facility (MOF) standards, both also defined by the OMG [5]. MDA makes use of three types of model abstractions. These levels of abstractions have the purpose of describing a software system from a particular perspective and specified by the OMG, being represented by the following acronyms CIM (Computing Independent Model), PIM (Platform Independent Model) and PSM (Platform Specific Model). The CIM has a view of the independent computing system, not showing details of the system structure [5]. Already PIM focuses on the operation of the system; however it does not care about details of the implementation in a specific platform [5]. Finally, the PSM is a view of the system that has characteristics, elements and information of the technology that will be employed [5].

From the abstractions of models defined above, the MDA establishes the transformations between the models of the different visions from mapping responsible for correlating the elements present in a source model with the elements of a target model [5], being that these transformations commonly are automated. However, it is possible to highlight the difficulty found to perform the transformations of CIM to PIM automatically, and the transformations between these models occur in a manual way [6]. Consequently, the CIM model ends up being excluded, starting directly from the PIM model [6].

In addition to software development methodologies, there are standards of architectures that help both software development and maintenance. Model-View-Controller (MVC) is one of these standards, which is based on the separation of the application into three layers, being the Model, View and Controller [7].

The purpose of this work is to present a BNF syntax, which was elaborated for the conception of the use case of the CIM model of the MDA, and this syntax is also used for the elaboration of information extraction rules for UML models. It should be noted that in addition to the syntax and mapping rules, this article is directed to the MVC architectural model.

This work is organized as follows: Section II discusses the MDA concepts and related works. In section III, the BNF grammar and the rules for the development of the proposed CIM model are presented. Section IV presents the rules for extracting information from the models. In section V, the application of the proposal is developed. Section VI presents the analysis of results, and finally in section VII, the conclusions of this paper.

II. Theoretical Foundations

In this section, Model Driven Architecture (MDA) is presented, describing its characteristics, specifications and steps. Work related to this article will also be presented and discussed.

V. M. Leite is a master's student at the State University of Londrina in Brazil (e-mail: vanessa.matiasleite@gmail.com).

J. G. Palma, is Prof. Dr. in State University of Londrina in Brazil (e-mail: jandirapalma@gmail.com).

F. H. de Oliveira is student in the State University of Londrina in Brazil (e-mail: fho1996@gmail.com).

A. Model Driven Architecture

MDA was defined by OMG in 2001 and is a specification to support Model Driven Development (MDD), its main objective being to extract value from models and modeling processes, thus enabling a way to deal with high complexity and interdependencies existing in software systems [5].

The main feature of the MDA focuses on transformations between models. Automating the transformation of high-level executable information systems models provides a number of advantages, citing cost reduction, time, production and system maintenance risks, by operating on system refinement for the problem-specific domain [5]. However, not all models are capable of automatic transformations; a model must be sufficiently complete and accurate for information about the developing system to be well specified by the models.

The Meta Object Facility (MOF) is a specification created by the OMG, in which it defines an abstract language and a framework to create a behavior pattern and the implementation of repositories for the metamodels and metadata, respectively [8].

Unified Modeling Language (UML) is a visual language used to model software based on the model-driven paradigm. In 1997, UML was adopted by the OMG as standard modeling language [9]. In this paper, three diagrams contained in UML will be used: the use case diagram, class diagram and the sequence diagram. For the CIM, the use case diagram will be developed, and for PIM, the diagrams will be the class and the sequence diagrams.

In MDA, the development of models and the transformations between them are one of the primordial points; three categories of models are established:

- Computation Independent Model (CIM), also called a business model [10], is one that presents a vision of the system, thereby helping to understand what the system is expected to execute. This type of model is independent of knowledge in computing having a high level of abstraction [11].
- Platform Independent Model (PIM) describes a system without any knowledge of the final platform of execution. PIM possess a higher level of abstraction than the PSM, however less than the CIM, and represents a formal specification of the structure and functioning of the system, such as it hides the features of a particular platform that enables the reuse in different platforms [11], [12].
- Platform Specific Model (PSM) is created from the PIM. In PSM, specific platform details are considered, being the basis for the transformation of the model to code [11], [12].

The transformations in MDA deal with the production of different models or artifacts, starting from a model based on a transformation pattern. It should be noted that there is a difficulty in performing the automatic transformation of CIM to PIM, and that this transformation is performed manually. This is due to the lack of specific artifacts and/or rules that define which elements should be used in the CIM, and consequently, there is no standardization for the mapping of CIM-PIM, this can be observed in several works [13]-[16].

B. Related Work

Article [17] presents an MDA approach that proposes a development process based on models and transformations that allows the implementation of Graphical User Interface (GU) for Rich Internet Applications with JavaFX platform respecting the MVC pattern. The article implements from the PIM model with transformation rules for PSM and uses the QVT (Query View Transformation) transformation mechanism to get to the code.

Articles [18] and [19] also present an MDA approach to the automatic generation of web application code, and propose the use of UML diagrams and the MVC architectural standard, using a framework, Spring MVC. However, these articles do not present a separation in CIM, PIM or PSM.

III. Proposal for the Elaboration of the CIM Model

This paper proposes formalism for the CIM and the rules for mapping the CIM to PIM models for MVC architecture.

In the CIM model proposed, it was considered a well-structured language with a syntax defined in Backus-Naur Form (BNF). The mapping performed was oriented to the MVC architecture. For clarity:

- The Backus-Naur Form (BNF) is a notation used to express the grammar of a language in the form of production rules. The BNF grammars are composed of terminals and non-terminals, and non-terminals can be expanded to one or more terminals or non-terminals [20].
- The MVC structure consists of three types of objects. The model is the application of the object, it is where they implement the logic of the application (domain of the problem) [21]. The Controller interprets the user inputs, informing the model and/or the view to change as appropriate. The View must ensure that its appearance reflects the state of the model. By approaching the isolation of the functional units, it facilitates the understanding and the modifications, and therefore, there is no need to understand all other units [7].

The rules of development and conversion are determined from the syntax created and defined by the BNF grammar.

For the elaboration of the CIM, the use case diagram and the screen interface will be elaborated. For each sentence of the use case description, the BNF grammar rules defined in this work will be used, so the use case sentences must be within the established syntax. The screen interface will be the means of communication with the user to validate the specification of the requirements, just as it will correspond to the view in the MVC architecture. The distribution of the activities described by each use case will be executed by a controller with the same name as the case of use.

The extraction of information, which comes from the formulated rules based on the BNF, generate the models of the PIM. In the PIM phase two diagrams are produced, the class diagram and the sequence diagram. The two diagrams are

developed from the derivation of the use case diagram and the screen interface, which derives from the defined rules.

For the editing of the diagrams presented in this article, the Enterprise Architect (EA) tool was used. This tool supports the MDA and is recognized by the OMG.

A. BNF Syntax

In this work, the BNF will be used with the objective of creating a formal notation to describe a syntax for creating the CIM use case description. It is also used to extract information and rules for mapping between diagrams.

The defined BNF syntax is based on the natural language based on syntax of the Portuguese language, being composed of nine non-terminal elements. Fig. 1 shows the syntax defined without the terminal symbols, in which it establishes a standardization for writing the use case description, thus allowing a transformation from CIM to PIM.

<sentence> := <subject> <predicate>

<predicate> := <verb> <object1> | <verb> <object> <complement> | <verb> <object> <complement1> <complement2>

Fig. 1 BNF Syntax

It is worth noting that the non-terminals <subject>, <object>, <object1>, <complement>, <complement1> and <complement2> of this syntax must be composed of a single noun, and the non-terminal <verb> for a single verb, with all these non-listed terminals can be accompanied by articles and/or prepositions. Therefore, for the writing of the description of the use case presented in this article, the sentences defined by the BNF was used, and some more rules will be exposed later in this paper. Each use case description will generate your terminals according to the specifications and generated sentences.

Some rules of writing in addition to the syntax itself are imposed, for example, non-terminals <object1>, <complement>, and <complement2> must be key words to the important requirements of the use case, and these requirements usually answer certain questions like: Who? Where? What? This specification is important because these elements have been transformed into classes, for example, so this restriction.

English : **System checks room occupation**
Syntax

BNF : **System checks occupation of room**
Syntax

Fig. 2 Example of writing by BNF syntax

In this paper some sentences do not adjust well with the English language, since the BNF was based on the Portuguese language and the way of structuring sentences is different. To illustrate this differentiation between languages, Fig. 2 shows what would be correct in the syntax of the English language and how the phrases will follow the defined BNF.

B. Proposed CIM Model

For the CIM phase, the use case diagram is defined, and this diagram will be used to document the requirements. The use case diagram was chosen because it is suitable for the CIM phase because it is high level and is independent of computation. To elaborate the description of the use case, the rules of the defined syntax of the BNF grammar were adopted, so all the sentences elaborated must follow what was specified by the BNF grammar.

As an example of the elaboration of the use case description, a representation of the elements proposed in this paper is presented in Fig. 3. What is emphasized in red are fields that need to be filled in when there are alternate specifications or exceptions to the normal specification path. It is noteworthy that these alternatives and exceptions must have an origin, title and an end, but must also have the sentence(s) that follow the defined BNF.

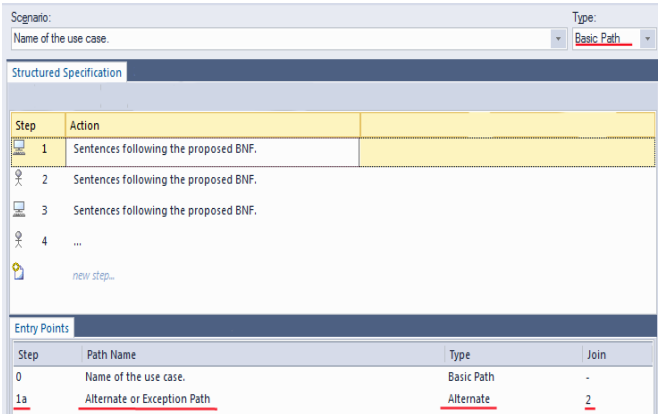

Fig. 3 Example of use case description

Some rules, in addition to the BNF syntax, are established for the development of the use case description, these rules are:

- The verb contained in each sentence of the form must be in a specific verbal conjugation, the verb tense used will be Simple Present.
- A specific verb will be used to indicate the input of data by the actor, that verb will be the INSERT.
- The verb INSERT will be used to indicate the input of data by the actor.
- The non-terminals <object1>, <complement> and <complement2> must contain keywords of the requirements.

The screen interface is used to support the development of CIM. With the screen interface, in addition to being able to have a clearer view of the system, it is possible to convert important information, such as data and relevant specification iterations. Another point, on which the interface assists in this article, is with the architecture of the development MVC; the screen interface makes the role of the View. The screen interface should be in accordance with the use case, so when the diagrams are referring to the same functionalities, these should have the same appointment.

IV. Proposal of Rules for the Extraction of Information for the Generation of Class Diagrams and Sequence Diagrams

For the conversion of the description of the use case and the screen interface into UML models, rules were created from the BNF grammar, as shown in Fig. 1. In this section, the rules will be defined for conversion into two diagrams, the class diagram and the sequence diagram.

A. Extraction of Information from the Use Case Diagram and the Screen Interface to the Class Diagram

In the MVC architecture, every use case that has at least one system interface will have a class with the stereotype interface. The Controller class must also be created because it will receive the methods of the View class and will distribute the Activities of the Use Case. The classes referring to the Model will be added as it happens to the transformation.

An established rule, which must be taken into account since the development of the use case itself, is that any sentence describing the use case should be possible for conversion. Therefore, from this rule, we must analyze all the sentences of the description of the use case, decomposing them and analyzing them according to the rules given below:

1. The sentence with the syntax <subject> <verb> <object1> has the following rules:

- <verb> followed by <object1> is transformed in the name method;
- <object1> is transformed into class.

In this rule, the <verb> and the <object1> join and form the name of the method, the class is already formed by <object1>. To define which class the method should be inserted, <subject> must be parsed. If the <subject> is an actor that interacts with the system, the method must be inserted in the View and Controller class. However, if the <subject> of the sentence is a system, then <verb> <object1> of the sentence is a method of the <object1> class.

2. The sentence with the syntax <subject> <verb> <object><complement> has the following rules:

- <verb> followed by <object> is transformed in the name method;
- <complement> is transformed into class.

In this rule the <verb> and the <object> join and form the name of the method, since the class is formed by <complement>. To define which class the method should be inserted, <subject> must be parsed. If the <subject> is an actor that interacts with the system, the method must be inserted in the View and Controller class. However, if the <subject> of the sentence is the system, the method must be part of the class represented by the <complement> contained in the sentence.

3. The sentence with the syntax <subject> <verb><object><complement1> <complement2> has the following rules:

- <verb> followed by <object> is transformed in the name method;
- <complement2> is transformed into class.

In this rule, the <verb> and the <object> join and form the name of the method, since the class is formed by <complement2>. To define which class the method should be inserted, <subject> must be parsed. If the <subject> is an actor that interacts with the system, the method must be inserted in the View and Controller class. However, if the <subject> of the sentence is the system, the method must be part of the class represented by the <complement2> contained in the sentence.

Other rules that are valid for all sentences of the use case description are:

- The verb INSERT is a reserved word that means the input of data by the user of the system, so when the sentence contains this verb it should not be converted to the class diagram.
- After the sentence with the verb Insert, the next sentences may use the data entered as its parameters in the class diagram.
- Another aspect to be considered is the question of the prepositions, articles and pronouns that compose the sentences, for the conversion these should be disregarded.
- When a use case description generates more than one class in the course of its sentences, this will imply that there is a relationship between these generated classes. This relationship can be of the type association, aggregation or composition.

After the rules for the use case, information can be extracted from the screen interface, as well as establishing a class with the stereotype interface in the class diagram, the interface also allows the extraction of data that are represented on the screen. Therefore, when there are fields and representations that represent data of the possible classes already created, this data must be added as attributes and must be private.

B. Extraction of Information from the Use Case Diagram and the Screen Interface to the Sequence Diagram

In the MVC architecture, every use case that has at least one screen interface will be a Lifeline View instance, there will also be an instance of the lifeline Controller. These instances have the same name as the use case under analysis.

The actor system will be the lifeline controller itself and the other actors that interact with the interface are also actors in the sequence diagram.

In the same way that all the sentences from the use case for the class diagram were analyzed, the same will be done for the sequence diagram. Therefore, the conversion rules are given below:

1. The sentence with the syntax <subject> <verb> <object1> has the following rules:

- <verb> followed by <object1> is transformed in the message;
- <object1> is transformed into lifeline;
- <subject> sends the message;
- <object1> receives the message.

In this rule, the <verb> and the <object1> join together and form the name of the sent message. The <object> transforms in a lifeline. The <subject> of the sentence is that it sends the message, it is the <subject> also who defines where the message will be received, if the <subject> is an actor that interacts with the system the message must pass through the

View and then reach the Controller. However if the <subject> of the sentence is the system, the <object1>, which has transform lifeline, it is who receives the sent message.

2. The sentence with the syntax <subject> <verb> <object><complement> has the following rules:

- <verb> followed by <object> is transformed in the message;
- <complement> is transformed into lifeline;
- <subject> sends the message;
- <complement> receives the message.

In this rule, the <verb> and the <object> join together and form the name of the sent message. The <complement> transforms in a lifeline. The <subject> of the sentence is that it sends the message, it is the <subject> also who defines where the message will be received, if the <subject> is an actor that interacts with the system the message must pass through the View and then reach the Controller. However, if the <subject> of the sentence is the system, the <complement>, which has transform lifeline, it is who receives the sent message.

3. The sentence with the syntax <subject> <verb><object><complement1> <complement2> has the following rules:

- <verb> followed by <object> is transformed in the message;
- <complement2> is transformed into lifeline;
- <subject> sends the message;
- <complement2> receives the message.

In this rule, the <verb> and the <object> join together and form the name of the sent message. The <complement2> transforms in a lifeline. The <subject> of the sentence is that it sends the message, it is the <subject> also who defines where the message will be received, if the <subject> is an actor that interacts with the system the message must pass through the View and then reach the Controller. However, if the <subject> of the sentence is the system, the <complement2>, which has transform lifeline, it is who receives the sent message.

Other rules that are valid for all sentences are:

- The verb INSERT is a reserved word that means the input of data by the system user, so when the sentence contains this verb it should not be converted to the sequence diagram.
- After the sentence with the verb Insert, the next sentences may use the data entered as its parameters in the sequence diagram.
- Execution of specifications (ExecutionSpecifications) is defined according to the sequential order of the existing sentences in the use case description.
- The alternative and exception flows of the use case description generate a combined fragment in the sequence diagram.

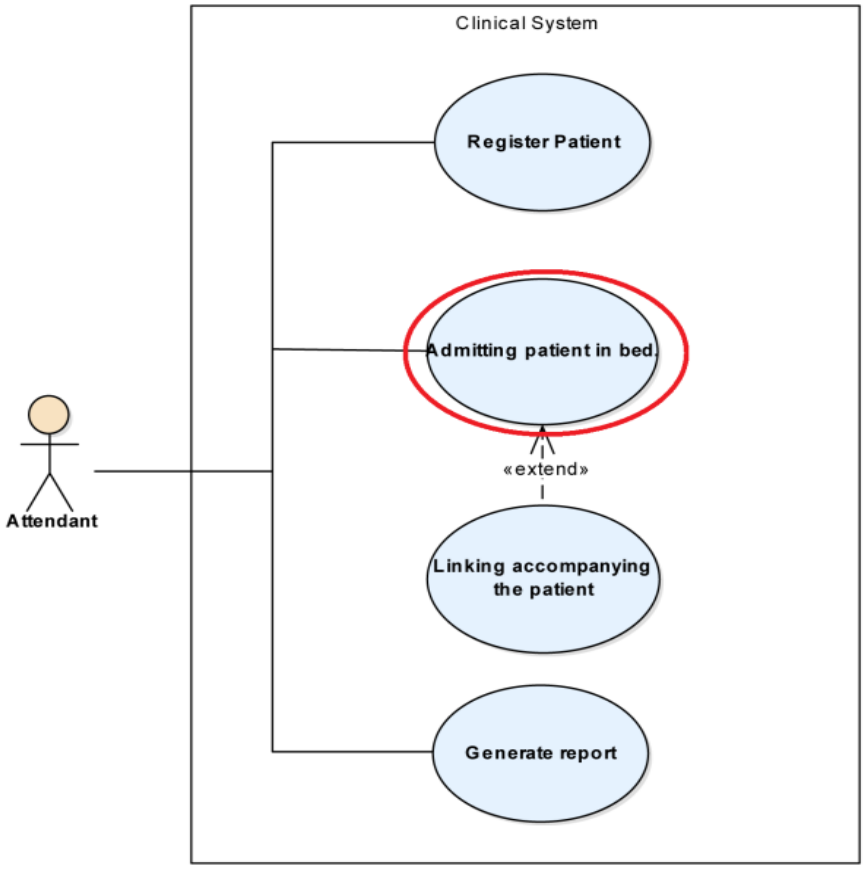

Fig. 4 Use case of a clinic

V. Application of the Proposals

The problem that is studied in this paper, for which the diagrams were drawn up, portrays the systems of a medical clinic. One of their primary needs is related to patient admissions, and the need for a system that interactively and visually addresses the issue of beds and occupied wards. It should be noted that this problem is a real case study, and the requirements were provided by a local software developer, so that it would be possible to apply this study in a more concrete scope. The use case for all medical clinic requirements is represented in Fig. 4; however, for this article only the description of the featured use case will be presented.

The use case that will be described is related to a patient's admission to a bed on a ward; this is depicted in Fig. 5. The description of the use case present in Fig. 5 follows the BNF grammar defined in Fig. 1, so all sentences present in the main and alternative scenarios are within the sentences defined by the BNF grammar of this paper.

The screen interface corresponding to the use case described in Fig. 5 is shown in Fig. 6. It is noted that the functionalities that are equivalent between the diagram and the interface have the same denominations. After the CIM models already built, information can be extracted for the other UML models through the established rules. Using the information extraction rules for the class diagram, Fig. 7 presents an example of how to apply the rules in the sentences of the use case description. The sequence diagram will also be developed in the same way. Fig. 8 shows an example of how to apply the rules in the sentences of the use case description.

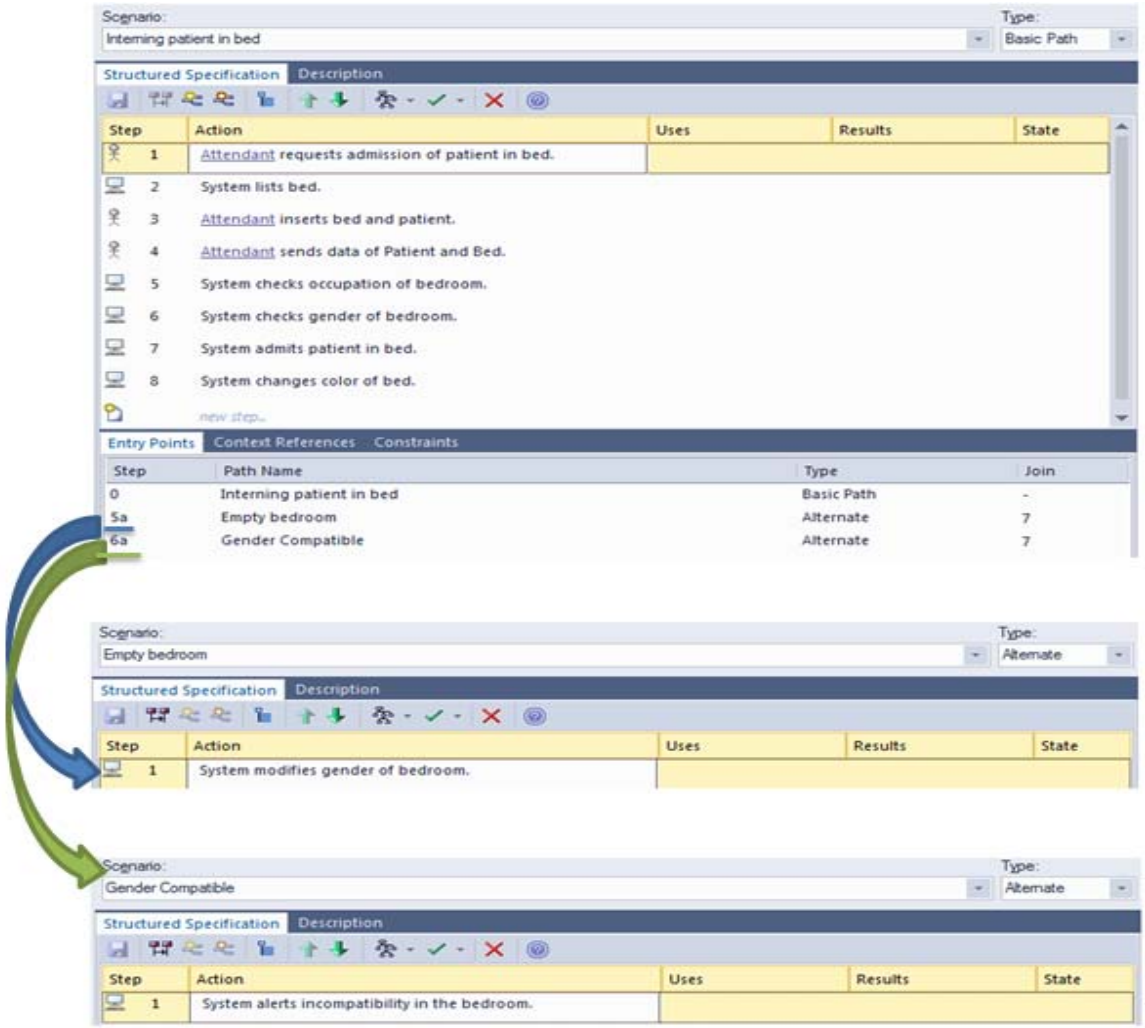

Fig. 5 Description of use case of a clinic

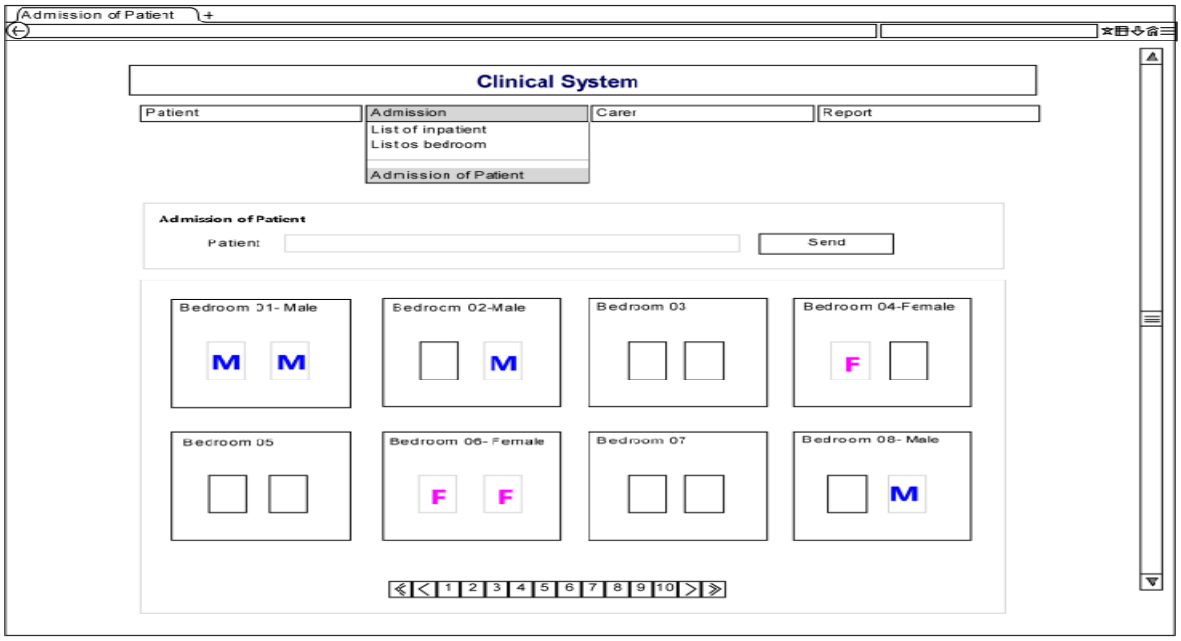

Fig. 6 Screen Interface

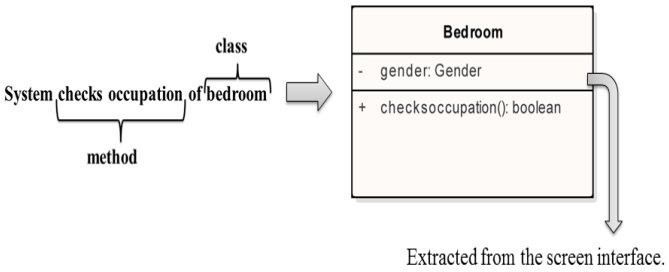

Fig. 7 Example of applying the rules to the class diagram

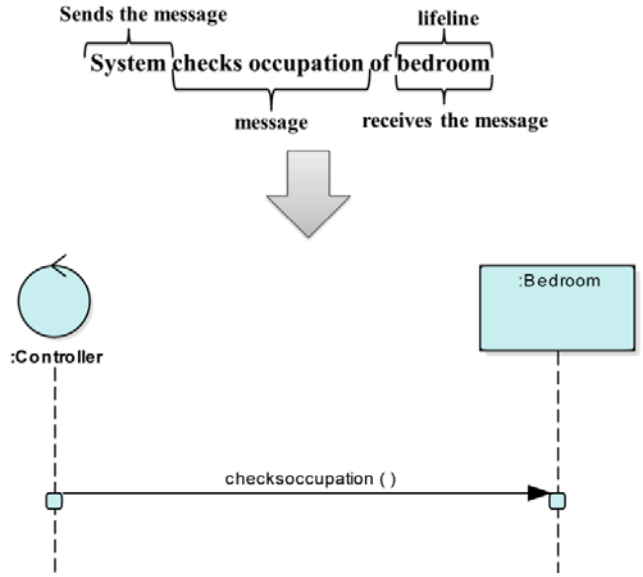

Fig. 8 Example of applying the rules to the sequence diagram

Figs. 7 and 8 illustrate how to apply the rules presented in this paper, and in order to obtain complete diagrams, it is necessary to apply the appropriate rules for all sentences of the use case description.

VI. Results Analysis

For the construction of the CIM models there was a standardization, with rules for creating the use case description and the screen interface. Thus, standardized models are created that facilitate the extraction of information.

The rules for extracting information for the generation of class diagrams and sequence diagrams were also presented in this paper, and as it was presented and exemplified, rules are able to extract in a simple way several pieces of information for the proposed diagrams.

Another relevant point of this article is that the BNF grammar defined is simple because it is based on natural language, so the developers of the models do not need to learn a new language. The rules based on the BNF are also clear and easy to understand and apply. Therefore, it can be concluded that the diagrams that were generated, both those of the CIM models and those that were produced from the rules, are susceptible to automated transformations within the proposed MDD, and the MDA proposed by the OMG.

VII. Conclusion

The aim of this paper is to propose the development of the MDA approach in view of the CIM model, as well as to create a standard for the development of the same. The rules of both construction and conversion are based on the defined BNF grammar developed, which was constructed from the natural language, and which facilitates the understanding and use of the same.

The transformation of CIM to PIM is generally little discussed in the literature, since the OMG itself does not

establish a standard for the CIM. Given what has been defined using the BNF syntax for the elaboration of the use case diagram and screen interface, it allows a writing pattern for the CIM, with which, this standard enables the conversion from the established rules of the CIM to the PIM to the MVC architectural standard.

The mapping performed in this article of the CIM models for the PIM models was performed manually; however, with the BNF grammar defined and the rules established it is possible to create a tool that performs this transformation automatically.

This work does not present the transformations of the PIM for the PSM and the PSM for the code, however, it should be noted that these transformations are supported by several tools recognized by the OMG [13].

REFERENCES

[1] Kleppe, Anneke G., Jos B. Warmer, and Wim Bast. MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional, 2003.
[2] Parviainen, P. et al. Model-driven development processes and practices. VTT Technical Research Centre of Finland, 2009.
[3] Souza, R. X. de O.; Oliveira, A. A. de; Nascimento, R. P. do. Modeler: Abordagem baseada em modelos aplicada ao processo de elicitação de requisitos.
[4] Torchiano, M. et al. Relevance, benefis, and problems of software modelling and model driven techniques—a survey in the italian industry. Journal of Systems and Software, Elsevier, v. 86, n. 8, p. 2110–2126, 2013.
[5] OMG. MDA Guide version 2.0. 2014
[6] Shirado, W. Hissamu, J. Guenka Palma, and V. Matias Leite. "Model Driven Architecture for Development Test Automation Tools." Proceedings of the International Conference on Software Engineering Research and Practice (SERP). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016.
[7] Isikdag, Umit, and Jason Underwood. "Two design patterns for facilitating Building Information Model-based synchronous collaboration." Automation in Construction 19.5 (2010): 544-553.
[8] OMG. Meta object facility (mof). 2002.
[9] Guedes, Gilleanes TA. UML: uma abordagem prática. Novatec Editora, 2008.
[10] Magri, J. A. Arquitetura dirigida a modelos (mda): Utilizando modelos no desenvolvimento de sistemas. Augusto Guzzo Revista Acadêmica, n. 8, p. 29–43,2008.
[11] OMG. MDA Guide version 1.0. 2003
[12] Mellor, S. J. MDA distilled: principles of model-driven architecture. (S.l.): Addison-Wesley Professional, 2004.
[13] Calic, T.; Dascalu, S.; Egbert, D. Tools for MDA Software Development: Evaluation Criteria and Set of Desirable Features. Las Vegas: Conference on Information Technology: New Generations, 2008.
[14] Gailliard, G. et al. Transaction Level Modelling of SCA Compliant Software Defined Radio Waveforms and Platforms PIM/PSM. Design, Automation & Test in Europe Conference & Exhibition. Nice: (s.n.). 2007
[15] Guttman, M.; Parodi, J. Real Life MDA: Solving businessproblems with Model Driven Architecture. San Francisco: Elsevier, 2007.
[16] Alves, E. L. G.; Machado, P. D. L.; Ramalho, F. Automatic generation of built in contract test drivers. Software and Systems Modeling Journal., New York, v. 13, 2014.
[17] Roubi, Sarra, Mohammed Erramdani, and Samir Mbarki. "Modeling and generating graphical user interface for MVC Rich Internet Application using a model driven approach." 2016 International Conference on Information Technology for Organizations Development (IT4OD). IEEE, 2016.
[18] Kateros, Dimitrios A., et al. "A methodology for model-driven web application composition." Services Computing, 2008. SCC'08. IEEE International Conference on. Vol. 2. IEEE, 2008.
[19] Kapitsaki, Georgia M., et al. "Model-driven development of composite web applications." Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services. ACM, 2008.
[20] Ryan, Conor, J. J. Collins, and Michael O. Neill. "Grammatical evolution: Evolving programs for an arbitrary language." European Conference on Genetic Programming. Springer Berlin Heidelberg, 1998.
[21] Dennis, Alan, B. H. Wixom, and David Tegarden. "Systems Analysis with UML Version 2.0." (2005).