A Rule-Based System for Assessing Consistency Between UML Models

Carlos Mario Zapata[1], Guillermo González[1], and Alexander Gelbukh[2]

[1] Escuela de Sistemas, Universidad Nacional de Colombia,
Carrera 80 N° 65-23, Bloque M8. Medellín, Colombia
Tel.: 4255350
{cmzapata,ggonzal}@unalmed.edu.co
[2] Computing Research Center (CIC), National Polytechnic Institute (IPN),
Col. Zacatenco, 07738, DF, Mexico
www.Gelbukh.com

Abstract. The main goal of requirements specification is the transformation of a "rough draft" of stakeholder needs and expectations into a semi-formal specification, represented by several diagrams, commonly UML diagrams. These diagrams must be consistent with each other, but consistency among different UML diagrams is not defined by the UML specification, and the research about inter-model consistency is still immature. We propose, in this paper, a rule-based system to detect consistency problems among UML diagrams. In order to complete this task, we have defined a set of rules in OCL, and then we use a novel approach for implementing the system by means of Xquery and Xpath languages. The use of these languages helps the rule-based system to interact with traditional CASE tools.

1 Introduction

The initial specification of a software application is often informal and possibly vague, and it is usually a "rough draft" of the final specification [1]. This commonly incomplete and inconsistent "rough draft" must be translated to a correct requirement specification, and then presented to the stakeholders for their validation. One of the critical tasks in requirements engineering tries to assure the quality of the step-by-step specification, in order to find consistency, correctness, and completeness mistakes as soon as possible in the software lifecycle [2]. The Unified Modeling Language (UML) is often used to make such specification [3].

Consistency has been, particularly, one of the most important concerns of software development process, and there are lots of works about it. However, there are still problems to be solved:

- UML superstructure [3] and other works [4] only define intra-model consistency. The inter-model consistency is not formally specified [5] and it is not supported by the common CASE (Computer-Aided Software Engineering) tools.
- Consistency checking is carried out automatically among some of the models and the executable code [6]. Executable code is the final step in the software

A. Gelbukh and A.F. Kuri Morales (Eds.): MICAI 2007, LNAI 4827, pp. 215–224, 2007.

development lifecycle, and we need to perform consistency checking in the previous stages (definition, analysis, and design).
- Several works [7, 8, 9, and 10] establish transformation processes instead of consistency checking processes. We need to specify the way consistency checking is performed among diagrams, and transformation processes do not help in such task.
- Some works [4 and 10] are done in a semi-automated way; if the analyst must participate in the consistency checking process, this process will probably be human-error-prone.
- There is one approach to specify consistency checking in a semi-formal way [11]. Lack of formalism can cause ambiguity problems in the final specification of a software application.

The reviewed works use rules in order to define the consistency checking process. In essence, rules are the main elements of the rule-based systems, a contribution of the Artificial Intelligence (AI) to the solution of this kind of problems. Ligêza [12] defined a set of principles for designing rule-based systems, and one of the most important is functional capability, a principle linked to the functionality of the language in which the rule-based system is programmed. In reference to the problem of consistency checking, we need to guarantee that the models, commonly made in CASE tools, can be accessed by the rule-based system in order to check consistency among them. Because of this, we need to introduce XML (Extended Markup Language) capabilities in the rule-based system.

In this paper we propose a method for verifying consistency between UML diagrams by means of a novel approach to rule-based systems. The rules of the system are defined in OCL [3], a formal language for constraint definition, and then they are programmed in Xpath and Xquery, special languages for selecting and processing parts of XML code. For sake of exemplification, the rules are related to consistency between class and use case diagrams, two of the most important diagrams of UML.

The structure of this paper is as follows: in Section 2 we review specialized literature about consistency checking. Section 3 describes Xpath and Xquery and justifies the use of these languages in the rule-based system. Section 4 presents the rule-based system for consistency checking. Finally, in Section 5 the conclusions and future work are given.

2 Literature Overview on Consistency Checking in UML Diagrams

The main source of consistency rules for UML diagrams is the UML superstructure, emitted by the OMG [3]. This document includes some intra-model consistency rules in OCL (Object Constraint Language), a formal language for constraint specification; some of these rules are implemented in some of the conventional CASE tools. However, inter-model consistency rules are not defined. Some works have been developed [4–11] for dealing with the problem of consistency between different kinds of models.

Dan Chiorean [4] used OCL [3] for checking the intra-model consistency in UML diagrams, with the help of the CASE tool OCLE. This tool is compatible with XMI

[13], and it can process UML models generated by many of the available CASE tools (Together, Rational Rose, MagicDraw, Poseidon, ArgoUML, etc.). A software specification is integrated by many diagrams, and intra-model consistency checking is only a part of the job; in order to guarantee the quality of a complete specification, we need to check consistency between several UML diagrams.

Xlinkit [6] is an environment for consistency checking of heterogeneous distributed documents. This approach uses several languages like XML, Xpath, Xlink, and DOM [13], and is conformed by a first-order-logic-based language to express constraints among documents, a document management system, and an engine for checking the constraints against the documents. In the runtime, Xlinkit applies Xpath expressions in order to review all the documents of the collection, and then it constructs a list of nodes to be checked. The documents can be referred to UML class diagrams or to source code of the software application. However, such comparison does not make sense; the source code is available in the implementation stage of software lifecycle, and we need to discover potential mistakes in the specification in previous stages.

Four parallel works: Kösters, Pagel and Winter [7], Liu *et al.* [8], Shishkov *et al.* [9], and Buhr [10], use case diagram to derive the class diagram. Transformation between diagrams is a way to guarantee the consistency between diagrams, but only in the case that we can completely generate the second diagram from the first. This is not the common case of software specification, where every diagram contains both proper information and shared information. In other words, by means of the transformation process, we only can generate the shared information of the second diagram, and the independent information must be completed in a manual process.

Glinz [5] defines a manual method for assessing consistency between class and use case diagrams, and he uses as a starting point a textual specification of the use cases in a special format. In this method, the presence of the analyst is highly required in the consistency checking process among the two diagrams, and this situation makes it difficult for the partial or total automation of the process.

Sunetnanta and Finkelstein [11] present an approach for checking the inter-model consistency, and they based this approach on the UML diagram conversion into conceptual graphs, and on the definition of consistency rules referred to the same graphs. The conceptual graphs can not be considered as a formal approach for this kind of specification elaboration, but a semi-formal approach. While the approaches have still low formal level, there is a high probability of ambiguity problems in the consistency checking process.

3 Xpath, Xquery, and Rule-Based Systems

XML is an extremely versatile markup language, capable of labeling the information content of diverse data sources, including structured and semi-structured documents, relational databases, and object repositories. Throughout the last few years, the use of XML [14] has grown, and this language has become a standard language for communication purposes among applications. The reasons why the use of this language has increased are the strange mix of suitability and standardization that it can achieve. Also, XML has an important suite of standard languages surrounding it,

and this suite gives it the power to interact among different applications. Two of the most important languages of this suite are Xquery and Xpath [13].

A query language that uses the structure of XML can intelligently express queries across all these kinds of data, whether physically stored in XML, or viewed as XML via middleware. Query languages have been traditionally designed for specific kinds of data. Existing proposals for XML query languages are robust for particular types of data sources, but weak for other types. The Xquery specification has been designed to be broadly applicable across all types of XML data sources. Xquery is a functional language to acquire data in multiple document formats (including XML documents) and then to produce XML-based results [13]. Xquery extensively uses the so-called Xpath, an expression language to select parts of an XML document by means of a matching process [13]. Both Xquery and Xpath languages are used to retrieve information pieces, especially from XML documents; for this reason, these languages have been commonly used for processing information in the semantic web.

The growing use of XML-based languages has motivated the extension of some of their capabilities, and rule-based systems have been employed for this purpose, especially in the fields of query optimization [15, 16, and 17] and logic programming [18]. By contrast, Xquery and Xpath can be used to support the elaboration of rule-based systems, as suggested by Eguchi and Leff [19], who discussed the use of XML-based languages to create artificial intelligence applications in the legal environment. Following the same trend, Schaffert [20] proposed a special rule-based language called Xcerpt, which is suitable to create rule-based systems from web documents.

The UML superstructure [3] suggests XMI (XML Metadata Interchange) as a standard language to share information among UML-based applications. Most of the UML-based CASE tools are capable to export diagrams in XMI format, for the purpose of communication among them. In order to create a rule-based system to check consistency between UML diagrams, the previous approaches do not use XMI; only Xlinkit [6] uses a sort of XML-based environment, but the goal of this environment is the comparison of UML diagrams against source code. Due to the fact that Xcerpt [20] is suitable for the semantic web, it does not use XMI as a way to interchange information. By this reason, in the next section we propose the use of Xquery and Xpath for creating a rule-based system to check consistency between two of the most common UML diagrams: class and use case diagrams.

4 A Rule-Based System for Consistency Checking Between UML Models

The construction of a rule-based system to define the consistency rules between ML class and use case diagrams requires the definition of some principles:

- Every class is distinguished by its name, by a collection of properties, and by a collection of operations offered by the class.
- The use case model has actors, which represents the roles which different users can play, and use cases, which represents the actions performed by the actors in the future software application.

Due to the fact that the UML superstructure [3] only defined intra-model rules, a heuristic analysis of the experience of software analysts was performed to define a set of consistency rules between class and use case diagrams. In this section, we present and specify two of such rules; the rules are presented in natural language, OCL, and Xquery-Xpath source code.

Rule 1. The name of a use case must include a verb and a noun; the noun should correspond to the name of one class in the class diagram. In other words, for each use case U in the class diagram, there should be a class C belonging to the class diagram, so that *U.name* equals *C.name*. Figure 1 depicts the graphical representation of this rule.

The OCL expression that represents this rule is shown in Figure 1.

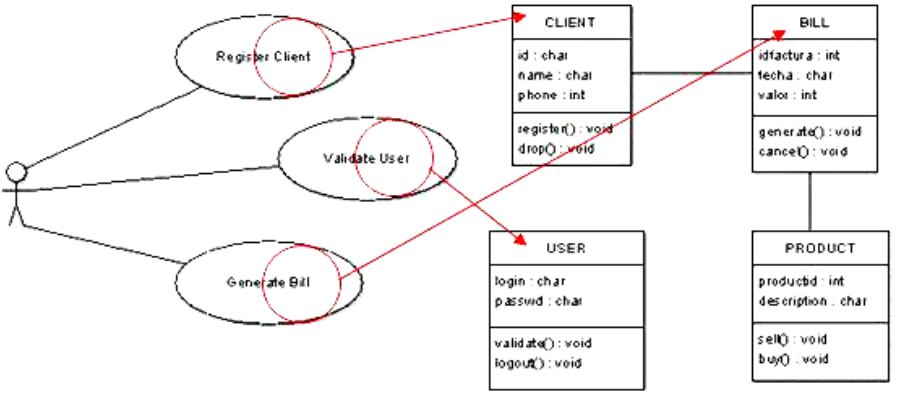

Fig. 1. Graphical expression of Rule 1

Classifier

self.UseCase->exists(us: Usecase, c: Class, x: Integer, y: Integer | y > x and us.name.toUpper.substring(x,y)=c.name.toUpper)

The Xquery-Xpath expression that represents this rule is:

```
<rule1>{
 for $i in 1 to count($class)
  for$j in 1 to count($usecase)
   return
        if (contains(upper-case($usecase[position()=$j]/@name), upper-
            case($class[position()=$i]/@name))) then
                ("<br/>The class <b> ", upper-
                case($clase[position()=$i]/@name), "</b> exists in the use case
                <b>", upper-case($casouso[position()=$j]/@name), "</b>")
```

```
        else
            ("<br/>The class <b> ", upper-case($clase[position()=$i]/@name),
            "</b> not exists in the use case <b>", upper-
            case($casouso[position()=$j]/@name), "</b>")
}</rule1>
```

Rule 2. The name of a use case must include a verb and a noun; the verb should correspond to an operation of a class in the class diagram that was identified in rule 1. In other words, for each use case U there should be a class C that contains an operation *Operationx* so that *U.name* contains *C.Operationx*. Figure 2 depicts the graphical representation of this rule.

The OCL expression that represents this rule is shown in Figure 2.

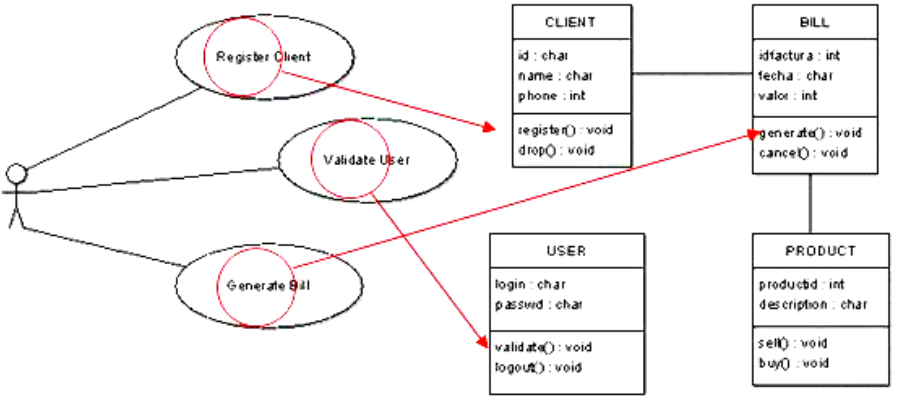

Fig. 2. Graphical expression of Rule 2

Classifier

self.UseCase->exists(us: Usecase, c: Class, x: Integer, y: Integer | y > x and us.name.toUpper.substring(x,y)=c.operation.toUpper)

The Xquery-Xpath expression that represents this rule is:

```
<rule2>{
 for $i in 1 to count($operation)
  for$j in 1 to count($usecase)
   return
        if (contains(upper-case($usecase[position()=$j]/@name), upper-
            case($operation[position()=$i]/@name))) then
            ("<br/>The operation <b>",upper-
            case($operation[position()=$i]/@name), "</b> exists in the use case
            <b>",upper-case($usecase[position()=$j]/@name), "</b>" )
```

```
        else
            ("<br/>The operation <b>",upper-
            case($operation[position()=$i]/@name), "</b> not exists in the use
            case <b>",upper-case($usecase[position()=$j]/@name), "</b>" )
}</rule2>
```

The complete set of rules was programmed in a rule-based system for checking consistency between class and use case diagrams. The inputs of the system are the two diagrams in XMI format [3]; ArgoUML® was the CASE tool selected to make these diagrams, and then to export them to XMI. The rule-based system was programmed in Java®, and uses Xquery and Xpath languages by means of an API (Application Program Interface) named *Saxon* for the validation of the rules. The rule-based system assesses the diagrams and creates a report to inform if the rules are followed (correct state), or are not (error state). Also, the rule-based system informs if there is an error of synonyms; to achieve this goal, the system uses a word list, which includes possible synonyms of every word. When the system detects two synonyms used in the same diagram, a warning message is presented.

If we apply the described process to the diagrams of Figure 1, after we introduce the XMI file resulting from ArgoUML®, in the rule-based system we achieve a XML file with the information of the recognized class diagram (see, for example, the class "BILL" in Figure 3; all of the classes will exhibit the same appearance), use case diagram (see Figure 4), and the results of consistency checking process (see Figure 5).

```
<class>
<name>BILL</name>
<attribute>ID</attribute>
<attribute>DATE</attribute>
<attribute>AMOUNT</attribute>
<operation>GENERATE</operation>
<operation>CANCEL</operation>
</class>
```

Fig. 3. XML file corresponding to the class "BILL"

```
<usecase>
<name>REGISTER CLIENT</name>
</usecase>
<usecase>
<name>VALIDATE USER</name>
</usecase>
<usecase>
<name>GENERATE BILL</name>
</usecase>
```

Fig. 4. XML file corresponding to the use case diagram

<consistency>
<existsclassinusecase>
The class **BILL** exists in the use case **GENERATE BILL**
The class **USER** exists in the use case **VALIDATE USER**
The class **CLIENT** exists in the use case **REGISTER CLIENT**
The class **PRODUCT** not exists in the use case **REGISTER CLIENT**
The class **PRODUCT** not exists in the use case **VALIDATE USER**
The class **PRODUCT** not exists in the use case **GENERATE BILL**
</existsclassinusecase>

<existsoperationinusecase>
The operation **GENERATE** exists in the use case **GENERATE BILL**
The operation **CANCEL** not exists in the use case **GENERATE BILL**
The operation **REGISTER** exists in the use case **REGISTER CLIENT**
The operation **DROP** not exists in the use case **REGISTER CLIENT**
The operation **VALIDATE** exists in the use case **VALIDATE USER**
The operation **LOGOUT** not exists in the use case **VALIDATE USER**
The operation **SELL** not exists in the use case **REGISTER CLIENT**
The operation **SELL** not exists in the use case **VALIDATE USER**
The operation **SELL** not exists in the use case **GENERATE BILL**
The operation **BUY** not exists in the use case **REGISTER CLIENT**
The operation **BUY** not exists in the use case **VALIDATE USER**
The operation **BUY** not exists in the use case **GENERATE BILL**
</existsoperationinusecase>
</consistency>

Fig. 5. XML file corresponding to the consistency checking process

5 Conclusions and Future Work

A novel approach to use Xquery and Xpath in the development of a rule-based system was presented in this paper. The main goal of the system is the assessment of consistency rules between UML class and use case diagrams.

This work makes contributions to Requirements Engineering and Artificial Intelligence. In the first case, the problem of definition of inter-model rules for consistency checking was dealt with by means of a formal specification of consistency rules between the use case and class diagrams in OCL. With the integration of the OCL in the rules definition, we assure that there is a formal way to check them, in order to avoid ambiguities and to guarantee well formed models. As a future work, a possible integration with the well-formedness rules of the UML specification can be defined. Related to Artificial Intelligence, this work has showed a novel way to incorporate XML-based languages in the development of rule-based systems. XML technology facilitates the access to several sources of information (for example, the semantic web and, in this particular situation, to the diagrams made by

means of CASE tools) and, in conjunction with the Artificial Intelligence theory, becomes a better way to develop rule-based systems.

Additional future work must be directed to extend the rule-based system to other UML diagrams (for example activity and sequence diagrams) and to other requirements engineering diagrams (for example goal diagram and process diagram). Also, we need to examine languages like Xcerpt, for assessing the suitability of these approaches to rule-based systems, with the possibility of accessing the diagrams made by means of many CASE tools.

Acknowledgements. The work was done with partial support of Mexican Government (SNI, CONACYT, COFAA-IPN) to the third author.

References

1. Jackson, M.: Software Requirements & Specifications: a lexicon of practice, principles and prejudices. Addison Wesley, Great Britain (1995)
2. Zowghi, D., Gervasi, V.: The Three Cs of requirements: consistency, completeness, and correctness. In: International Workshop on Requirements Engineering: Foundations for Software Quality, Essen, pp. 155–164. Essener Informatik Beitiage, Germany (2002)
3. OMG – Object Management Group. http://www.omg.org
4. Chiorean, D., Pasca, M., Carcu, A., Botiza, C., Moldovan, S.: Ensuring UML models consistency using the OCL Environment. In: Sixth International Conference on the Unified Modelling Language - the Language and its applications, San Francisco (2003)
5. Glinz, M.: A lightweight approach to consistency of Scenarios and Class Models. In: En: Fourth International Conference on Requirements Engineering, Illinois, USA, June 10-23 (2000)
6. Gryce, C., Finkelstein, A., Nentwich, C.: Lightweight Checking for UML Based Software Development. In: Workshop on Consistency Problems in UML-based Software Development. Dresden, Germany (2002)
7. Kösters, G., Pagel, B.-U., Winter, M.: Coupling Use Cases and Class Models. In: BCS FACS/EROS Workshop on Making Object-oriented Methods more Rigorous, London (1997)
8. Liu, D., Subramaniam, K., Far, B.H., Eberlein, A.: Automating transition from use-cases to class model. In: IEEE CCECE 2003. Canadian Conference on Electrical and Computer Engineering, vol. 2, pp. 831–834 (2003)
9. Shishkov, B., Xie, Z., Lui, K., Dietz, J.: Using norm analysis to derive use case from business processes. In: 5th Workshop on Organizations semiotics, Delft the Netherlands, June 14-15, 2002 (2002)
10. Buhr, R.J.A.: Use Case Maps as Architectural Entities for Complex Systems. IEEE Transactions on Software Engineering 24(12), 1131–1155 (1998)
11. Sunetnanta, T., Finkelstein, A.y.: Automated Consistency Checking for Multiperspective Software Specifications. In: Proceedings of the 26th Australasian computer science conference, vol. 16, pp. 291–300 (2003)
12. Ligêza, A.: Logical Foundations of Rule-Based Systems. Studies in Computational Intelligence (SCI) 11, 191–198 (2006)
13. W3C – World Wide Web Consortium. http://www.w3.org
14. XML – Extensible Markup Language. http://www.w3.org/XML

15. Travers, N., Dang, T.: An Extensible Rule Transformation Model for XQuery Optimization. In: ICEIS. International Conference on Enterprise Information Systems, INSTICC, Madeira (2007)
16. Pal, S., Istvan, C., Seeliger, O., Rys, M., Schaller, G., Yu, W., Tomic, D., Baras, A., Berg, B., Churin, D., Kogan, E.: XQuery implementation in a relational database system. In: Proceedings of the 31st international conference on Very large data bases, pp. 1175–1186. Trondheim, Norway (2005)
17. Che, D., Aberer, K., Özsu, M.: Query optimization in XML structured-document databases. The VLDB Journal 15(3), 263–289 (2006)
18. Almendros, J., Becerra, A., Enciso, F.: Magic Sets for the XPath Language. Journal of Universal Computer Science 12(11), 1651–1678 (2006)
19. Eguchi, G., Leff, L.: Rule-based XML: Rules about XML in XML To Support Litigation Regarding Contracts. Artificial Intelligence and Law 10, 283–294 (2002)
20. Schaffert, S., Xcerpt, A.: Rule-Based Query and Transformation Language for the Web. PhD thesis, University of Munich (2004)