Deontic Database Constraints
From UML to SQL

Pedro Nogueira Ramos
ADETTI /ISCTE-IUL, Computer Science Department, Lisbon, Portugal

Keywords: UML, SQL, Case, Relational Model, Deontic Constraints.

Abstract: Deontic constraints (obligations, forbiddances, violations) can be easily and explicitly represented in UML Class Diagrams. They deal with formal representation of requirements, which ideally should always be fulfilled, but can be violated, in atypical situations. In this paper we adopt and extend previous work on deontic constraints. Our contribution is the development of a tool that fully generates code based on UML Class Diagrams representation of those constraints. We overcome some limitations of previous work, and consequently we only adopt standard UML notations. Our tool (a Sybase PowerDesigner plugin) generates OCL constraints, a standard relational model, and SQL code to hold the deontic requirements. SQL is coded in views and triggers. Since we adopt a commercial tool, these enrichments will benefit any non-professional users.

1 INTRODUCTION

The paper addresses the automatic generation of database constraints. Our approach is based on the formal representation of constraints, using OCL (Object Constraint Language) and UML (Unified Modeling Language) Class Diagrams. The advantages of CASE (Computer-Aided Software Engineering) Tools are well known, and a wide range of commercial tools are largely adopted by companies. Code generation usually ensures a well-documented, accurate and efficient code. Software maintenance costs can be highly reduced as a consequence of CASE tools adoption. Our main contribution is the development of a tool that fully generates code based on UML (Boock et. al., 1998) constraints represented in a diagrammatic notation.
Within the wide scope on software constraints, we have focused our work in database constraints, more specifically, the paper deals with a relational model enrichment that allows a flexible notion of the mandatory property in foreign key fields. The paper considers the distinction between the so called Soft and Hard constrains (Elliman, 1995). This distinction is explicitly represented in the graphical notation.

In the database domain, the boolean mandatory attribute is adequate for requirements that must hold unavoidably, but is not adequate to deal with requirements that ideally should always be fulfilled, but can be violated in atypical situations. If those violable requirements are explicitly represented it is possible to maintain both the requirement and its violation and, consequently, recur to monitoring procedures for violation warnings. Without that explicit representation, the designer must always choose a non-mandatory value if there is a chance, even if a small one, that in an atypical situation the field will not be filled in.

Consider the following real example - taken from (Ramos, 2008) - concerning the overtime hours control in an organization where sometimes the employees work for several days outside the organization (at the client's organization facilities).
An organizational rule states that all overtime hours must be authorized by the employee hierarchic superior. Using a workflow system the employee fills in a requirement form that, if authorized by his superior, will be stored in a table record in which the ID of the superior must be filled in. However, sometimes the need for overtime hours is only detected when the employee is working outside the organization. A problem may arise if the employee is outside during a change of month (the problem is related with technical issues regarding the total

Nogueira Ramos P..
Deontic Database Constraints - From UML to SQL.
DOI: 10.5220/0004415801020109
In *Proceedings of the 15th International Conference on Enterprise Information Systems* (ICEIS-2013), pages 102-109
ISBN: 978-989-8565-60-0

amount of monthly overtime hours). When the employee is working outside with his superior the usual procedure consists of the employee sending a fax or an email (and then someone fills in the requirement form for them) and their superior making a phone call or sending a fax authorizing the overtime hours. Due to organizational security procedures no one can electronically authorize overtime hours without a proper password. When the superior is outside and cannot access the system, the system cannot accept overtime hours. The solution adopted by the organization was to remove the requirement, which stated that all records of the overtime hours' authorization table should always have the superior's ID.

The important aspect of the previous example is that, because of one atypical situation (which nevertheless happens several times) an important requirement was abandoned (the overtime authorization control is now made manually, where previously it was validated by the database). That happened because the requirement was inflexible.

The paper is organized as follows: in section 2 we present some relevant related work; in section 3 we analyze the related work presented in (Ramos, 2008), and in section 3 we introduce its limitations, our counterproposal and the plugin we developed for the automatic generation of OCL, Views and Triggers. Some concluding remarks are presented in the last section

2 RELATED WORK

There are already some tools that generate code based on OCL constraints (Loecher and Ocke, 2010), (Briand, 2005). These tools haven't yet reached a final stage, some limitations were found during their testing. Also, they do not support graphical notation. Graphical notation is critical when we want to support the daily work of database designers in organizations. OCL(Warmer and Kleppe, 2003) hasn't yet become a common language, even in the database community. Its use requires a deeper knowledge than UML Class Diagrams.

We follow the work presented in (Ramos, 2008) where the author, based on deontic notions, already proposes a flexible notion of the mandatory property. However, the author does not support his approach on a computational implementation. Furthermore, we have detected that the solution presented in (Ramos, 2008) has some serious limitations, which can be avoided. Moreover, in (Ramos, 2008) the role of OCL seems useless since the author does not present any guidelines regarding the automatic generation of OCL expressions based on the UML Class Diagram. In this paper we present an integrated solution that generates OCL expressions and SQL scripts based only on the Class Diagram. Unlike the approach proposed in (Ramos, 2008), we don't need to extend the relational model with odd tables. In our approach the relational model is generated based only on standard rules. Additional knowledge is stored in triggers and views. In the next section this work is analyzed with more detail.

Borgida, in his work with exceptions in information systems (originally in (Borgida, 1985) with further extensions, e.g., (Borgida et. al., 1999)) considers that exceptional situations arise when some constraints are violated, and that exceptions are considered as violations. In his proposal, the occurrence of a violation is signaled by the creation of an object in a class called ANY_VIOLATION. The author proposes an exception handling mechanism to specify failure actions. Again, this author also recurs to an odd class (more precisely, he uses two classes: one for the violations as such and another for the violation constraints). Borgida proposes a much more general mechanism to deal with exceptions handling in object oriented programming languages. Our approach, apart from being only oriented on one particular constraint (not considered in Borgidas work), is focused on the database generation. Borgida, contrary to us, explicitly rejects triggers approach, because he wants to maintain the control in a middleware software level. What we call Contrary-To-Duties constraints isn't addressed in Borgidas work.

The distinction between violable requirements and mandatory ones is similar to the so called Soft and Hard constraints. The Semantics of Business Vocabulary and Business Rules (SBVR), an adopted standard of the Object Management Group (OMG) for a formal declarative description of business rules, considers the deontic modal operators obligation and necessity (www.omg.org/). Those operators also intend to deal with the distinction on hard and soft constraints. The idea is to use them in computer systems in the context of the OMG's Model Driven Architecture (MDA).

3 DEONTIC CONSTRAINTS

In (Ramos, 2008) the author addresses a specific subject: the mandatory property in relational models tables attributes. The proposed approach is inspired

in deontic logic, namely in the notions of Deontic Obligation, Deontic Prohibition and Deontic Necessity (Wieringa and Meyer, J., 1991). The main idea is to maintain in the system both the requirements (ideal situations) and their violations, and, additionally, the emerging consequences of those violations. In deontic terminology, we are talking about Obligations (requirements), their Violations and Contrary-To-Duties (new emerging Obligations or Prohibitions).

In the paper the author presents the following example to motivate the need for a deontic approach. "All students must have a zip code address". That requirement exists due to the fact that the university regularly needs to send correspondence to the student. However, is that a requirement that intends to capture an ideal situation or a requirement that the database should always fulfill? What will be the procedure if one student tries to register in the school and has forgotten his zip code? If we want to conditionally accept his registration (telling him that he must supply the zip code as soon as possible) then the requirement is about an ideal situation that sometimes doesn't happen (neither in the real world nor in the database). Blocking the registration could be a wrong choice because, apart from the fact that all data already inserted in the application form would be lost, the school would convey an unpleasant image of unnecessary bureaucracy. The zip code will be indispensable in the future, but during a short period of time the zip code is dispensable.

In (Ramos, 2008) the author distinguishes between two kinds of requirements:

- requirements that ideally should always be fulfilled, but can be violated in atypical situations and;
- requirements that must hold unavoidably.

Furthermore, the author, inspired by the deontic logic approach, considers the so-called Contrary-To-Duties scenarios (Carmo and Jones, 1996), i.e., new constraints that emerge as a consequence of the violations of the Violable Requirements. Those sub-ideal states represent situations where deontic obligations are violated and consequently new obligations or prohibitions arise to deal with that undesired but tolerated situation. In the paper the author considers different emerging scenarios, but since that distinction is not very clear, in this paper we would rather adopt a different classification (very similar to the OMG standard for the semantics of Business Vocabulary and Business Rules). In the presence of an unfulfilled obligation three different kinds of constraints may arise:

- New obligations;
- Forbiddance constraints (explicit representation of undesired situations);
- Necessary constraints (the same as hard constraints: requirements that must hold unavoidably).

Notice that those three situations can be expressed as soft and hard constraints, but from a user perspective, we agree that the deontic semantics of Obligation, Forbiddance and Necessity (also adopted by OMG) are more intuitive.

In Figure 1 we present an example taken from the author's paper that illustrates the proposed graphical notation.

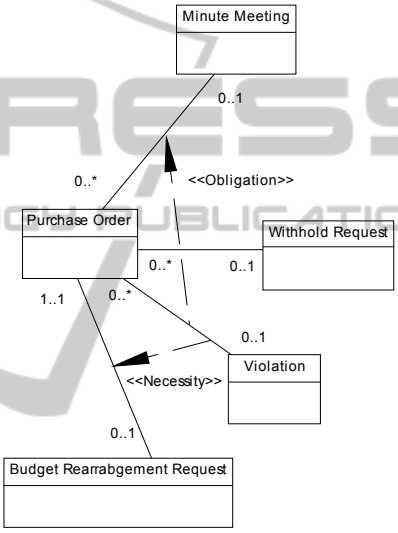

Figure 1: Graphical notation for deontic constraints.

The example refers to an application that supports the budget control of a building company. The example will also be used in the next section. *Every building project has its own budget. The budget is disaggregated into several items. When the project leader adjudicates a new work to a supplier, he fills in a Purchase Order (PO) (to be delivered to the supplier) and also fills in a Withhold Request (WR) (to ensure that the money will be available when the payment takes place). The Withhold Request (WR) is only allowed by the application if there is enough money in the corresponding budget item (a PO for a specific item). In order to control the budget the following rule is implemented in the application:* ***every PO must be associated to a WR****. Consequently, POs are only allowed if there is enough money available in the budget item. However there are situations where adjudications must take place (the PO must be created) even if there is no budget (for example, unpredictable*

works). In such situations the standard procedure is to request a budget rearrangement (for example, to exchange values between items). That request takes some time (even days) to be analyzed and sometimes the urgency of the work forces the project leader to violate the control rule (for example, an imminent land falling that requires a sustentation wall). In order to allow the violation of the rule, the application must accept that sometimes a PO is not associated to a WR. Given that flexible interpretation, what the rule really states is that ***ideally every PO must be associated to a WR****. In order to ensure a rigorous budget, when the rule is violated (a PO is not associated to a WR), apart from the Budget Rearrangement Request (BRR), the project leader must organize a work meeting to elaborate a formal minute that justifies the decision to adjudicate the new work (in this kind of meeting a third party entity – surveillance company - is always present). Notice that 'new obligation' (to have a minute signed by the three entities: the project leader, the supplier and the third party company) also represents an ideal situation. Sometimes (rare situations) the adjudication must occur before the meeting takes place.*

The obligation to have an association between the PO and the WR is represented with a special class: Violation. The disjunction (XOR) means that if the PO does not have a WR then a violation will be associated with the PO. In the paper the author adopts the approach first presented in (Tan and Torre, 1994). Deontic rules are represented with violation constants: the previous expression *Ideally(Φ)* (or *Obligation(Φ)*) is represented as $\neg V_i \rightarrow \Phi$, in which V_i represents a violation constant. The author considers a finite set of violation constants, each of them associated to one deontic rule. The semantics is intuitive: $\neg V_i \rightarrow \Phi$ means that if rule i isn't violated then Φ is a fact (in other words, rule i states that there is an obligation to ensure Φ). The author considers that the Class Violation corresponds to the finite violation constants ΔV ($\forall_i V_i \in \Delta V$).

In the example there are two contrary-to-duties situations. When a PO is not associated to a WR two new situations arise that can be expressed with two new rules:

- If a PO is not associated with a WR then a BRR is necessary;
- If a PO is not associated with a WR then the PO must be associated with a Minute Meeting (MM).

The author proposes the use of the UML Dependency Relation to capture the contrary-to-duties requirements. The stereotypes must be read as follows: "the Budget Rearrangement Necessity and the Minute Meeting Obligation depend on the Purchase Order violation". The notion of Forbiddance is also considered (not illustrated in the example) and also represented with a dependency relation.

The author presents an OCL representation for the deontic requirements (based on the Violation Class) but he does not link the OCL expressions to SQL code. The relational model implementation is based on the built-in table called Violation (with seven columns), and the SQL code depends on that table. We do not present any further details, because our approach does not take that odd table into account. We do not need to create any tables apart from the ones that are generated by standard transposition rules. We also generated complete SQL code that covers all requirements. Our approach is presented in the next section.

4 DEONTIC CONSTRAINTS: FROM UML TO SQL

We follow the deontic approach described in the last section because we believe that it is intuitive and also because OMG has adopted it has well. We consider the three contrary-to-duties scenarios that may arise when obligations are violated:

- New Obligations;
- Forbiddance;
- Necessity.

We represent all requirements only with stereotyped dependency relations: <<Obligation>>, <<Necessity>>, <<Forbiddance>>. For each stereotype we generate the OCL expression, and afterwards, the SQL code (triggers and views).

In Figure 2 we present the previous example with a new requirement and minor changes. We tested the diagram in the plugin. We adopt the Sybase PowerDesigner tool (www.sybase.com) because we think it is one of the best UML-To-Relational tools and it allows the development of plugins without any restriction. Since we adopt a commercial tool, any non-professional user can use our approach in their projects.

We create a new association between Collaborator and Zip Code to illustrate the Forbiddance requirement. If the system does not know the collaborator zip code, he cannot place

orders. We choose a "many to many" association between the Collaborator and the Zip Code just to enhance the potentialities of the plugin. It will be more realistic to consider that one collaborator is only associated with a zip code, but this situation (more than one zip code) is more complex in what regards the obligation and the consequent forbiddance. As we will see we only use standard Class Diagram and we do not make use of the Violation Class. The corresponding OCL constraints are the following:

Table 1: OCL constraints for deontic constraints.

Stereotype	OCL Expression	
Obligation	Context: Colaborator ZC-> notEmpty()	Context: source class. Requirement: there must be an object in the target class.
Obligation	Context: Purchase Order WR->notEmpty()	
CTD Obligation	Context: Purchase Order WR -> Empty() implies MM -> notEmpty();	Context: source class. of the primary obligation. Requirement: there must be an object in the target class.
CTD Necessity	Context: Purchase Order WR -> Empty() implies no_role_ctd -> notEmpty();	Context: source class. of the primary obligation. Requirement: there must be an object in the target class.
CTD Forbiddance	Context: Colaborator ZC->Empty() implies PO -> isEmpty();	Context: source class. of the primary obligation. Requirement: there should not be an object in the target class

Since we want to keep the deontic constraints in the database, triggers are the best choice to implement them. Using Before and After action triggers it is possible to ensure that all constraints are fulfilled. The procedure for monitoring the obligations fulfillment can be easily achieved using relational views. For each constraint (OCL invariant) we can generate a SQL view, which retrieves the records that don't satisfy it. That's, for example, what the Dresden Toolkit does, one of the few OCL-SQL generators (Loecher and Ocke, 2010), (dresden-ocl.sourceforge.net/). In what regards the SQL generated code, our plugin automatically generates the code explained in Table 2. Notice that in the current version we haven´t yet implemented the after/before update and delete triggers, however, the process is very similar to the after/before insert triggers:

Table 2: SQL code for deontic constraints.

Stereotype	SQL Code
Obligation	Generates a view that retrieves violations (null foreign keys or missing records)
CTD Obligation	Generates a view that retrieves violations (null foreign keys or missing records).
CTD Necessity	Generates a trigger that cancels an insert operation (an obligation not fulfilled) if it finds a null foreign key or doesn't find a joined record (a mandatory association not instantiated).
CTD Forbiddance	Generates a trigger that cancels an insert operation if it finds a null foreign key or doesn't find a joined record (an obligation not fulfilled).

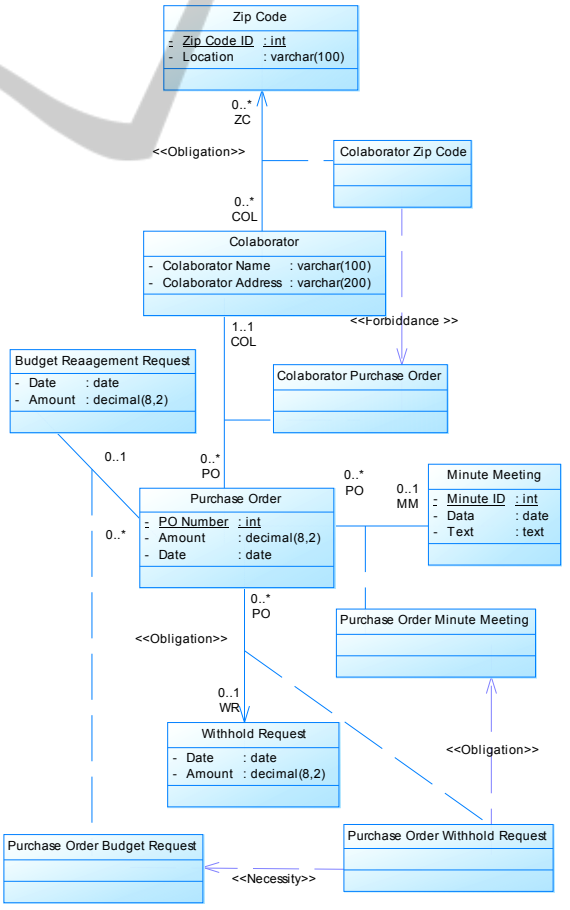

Figure 2: Deontic constraints, Order Example.

In Figure 3 we present the plugin steps to generate the desired code.

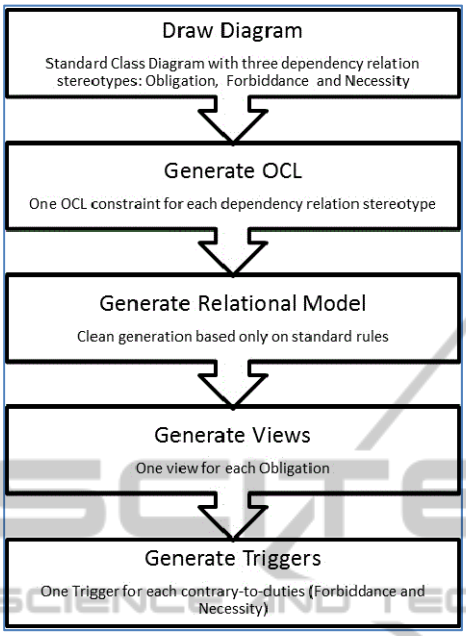

Figure 3: Steps to generate the SQL Code.

OCL constraints are generated based only on the Class Diagram. The association role names are used to name the constraints, but they aren't mandatory, as it can be noticed in the *CTD_Nec_Purchase_Order_PO_Withhold_Request_no_role_ctd* constraint (we haven't named the *Purchase_Order_ Budget_Request* roles, that's the reason for the *no_role* expression).

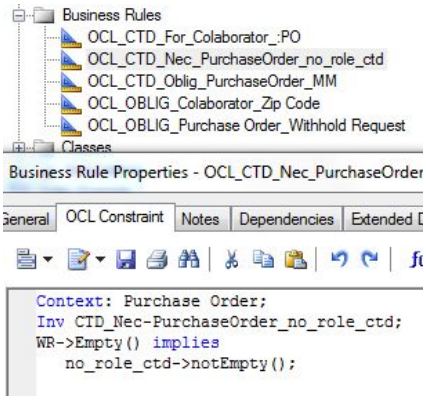

Figure 4: An example of an OCL constraint.

Primary obligations' names start with OCL_OBLIG_, and contrary-to-duties constraints start with OCL_CTD_. Contrary-to-duties can be new obligations (OCL_CTD_Oblig_), prohibitions (OCL_CTF_For_) or hard constraints (OCL_CTF_NEC_). The remainder of the name for the primary obligations is the conjunction of both classes names. For the contrary-to-duties constraints the remainder is the conjunction with the context class name and the role associated to the opposite class. An OCL code example can be checked in Figure 4 (the remainder are presented in Table 1).

The relational model depicted in Figure 5 results essentially from the application of standard transformation rules (Scott Ambler, 2013). We haven't followed exactly PowerDesigner rules because it has some limitations regarding one to one relations, foreign keys integrity constraints, primary keys, etc. The adaptations we made do not in any aspect relate to our extension. The adaptations only concern relational model optimizations and do not affect the plugin code covered by this paper.

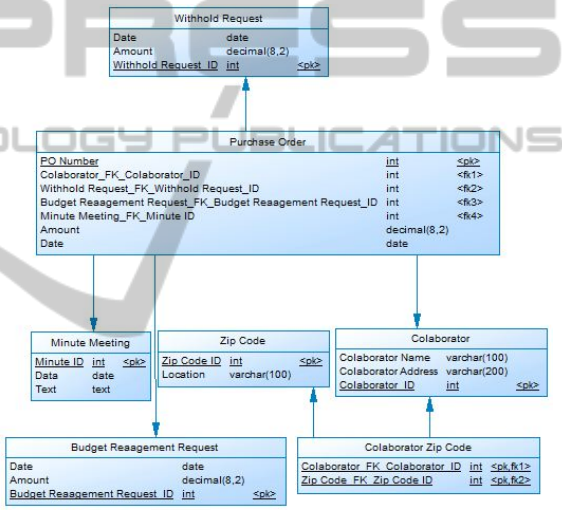

Figure 5: Relational Model, Order Example.

Each Obligation generates one view as presented in Figure 6, Figure 7 and Figure 8. Our criterion was not to generate the most efficient SELECT commands. Our goal was to find a generic algorithm, which copes with any obligation, that's the reason why we always recur to the EXISTS operator.

Considering the second example regarding the obligation, which stands that all collaborators must have a zip code. The view retrieves collaborators that do not exist in the colaborator_Zip_Code table, the table that holds the collaborators zip codes.

One trigger is generated for each Forbiddance and Necessity dependency relation. The corresponding code is presented in Figure 9 and Figure 10. Again, our criterion was not to generate the most efficient SQL code. We are aware that sometimes it is possible to generate a more simple

code, and in future versions we will address that subject. The current version generates an effective code.

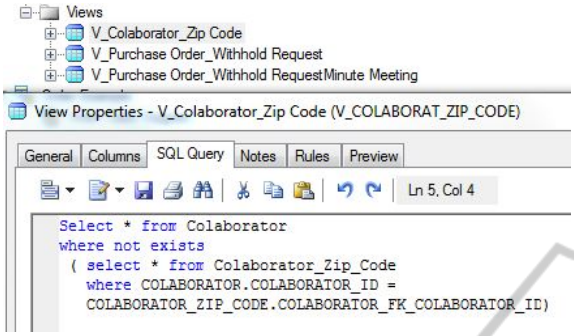

Figure 6: View for monitoring the fulfilment of the primary obligation of having a zip code.

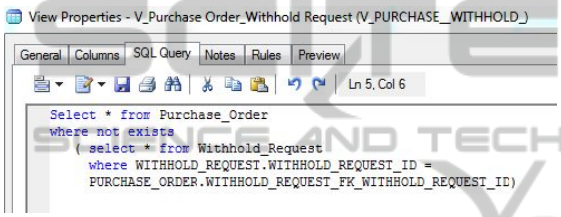

Figure 7: View for monitoring the fulfilment of the primary obligation of having a withhold request.

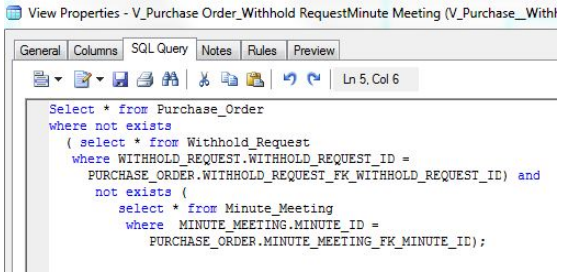

Figure 8: View for monitoring the fulfilment of the sub-ideal obligation of having a minute meeting.

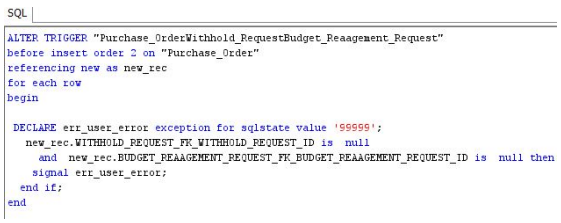

Figure 9: Trigger for cancelling and insertion that violates a forbiddance requirement.

In Figure 11 we can see an example of one trigger avoiding one insertion that will violate a Forbiddance requirement. When generating the Relational Model we can choose the database engine that will hold the database. In our example we choose Sybase Adaptive Server. We tried to use only standard SQL code in order to cope with the most known database engines.

Figure 10: Trigger for cancelling and insertion that violates a necessity requirement.

Figure 11: Cancelling and insertion that violates a forbiddance requirement.

5 FINAL REMARKS AND FUTURE WORK

In this paper we have extended existing work in order to provide us with a tool that fully supports an automatic generation of the extended relational (deontic) model. With the tool, a database designer can explicitly represent in standard UML class diagram deontic constrains and automatically obtain the final SQL code.

The reason for focusing our attention on the relational model is the fact that Object Databases Management Systems aren't yet an efficient solution for demanding applications. The object-oriented paradigm is becoming the main background for system modeling, but unfortunately the object-oriented databases don't progress at the same speed, making relational databases the standard for data storing. The reason for choosing UML is the fact that it has become a standard language for design and conception of systems.

Our tool needs to be tested with complex

scenarios (diagrams) in order to correct any bugs that there may be. For that reason we delivered the tool to information system course students (master degree).

Furthermore, in the near future it will be necessary to cover more complex situations like relations with more than two arguments and composite obligations. For example, how to represent that: all courses should be associated to a degree and all courses should be associated to a coordinator; but if obligation to be assigned to a coordinator disappears if the course isn't assigned to a degree (the two obligations must only be processed as a whole and not separately).

REFERENCES

Boock, G., Rumbaugh, J., Jacobson, I., 1998. The Unified Modeling Language Reference Manual. In *Addison-Wesley object technology series.*

Warmer, J., Kleppe, A., 2003. The Object Constraint Language: Getting Your Models Ready for MDA (2nd Edition). Ed *The Addison-Wesley Object Technology Series* .

Ramos, P. 2008.. Contrary-to-duties Constraints: From UML to Relational Model. In *ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems,*Toulouse

Wieringa, R. J., Meyer, J., 1991. Applications of Deontic Logics in Computer Science: a concise overview In J. Meyer and R. J. Wieringa, , editors, *Procs. First Int. Workshop on Deontic Logic in Computer Science*.

Elliman, D. G., Burke, E., Weare, R. F., 1995. The automation of the timetabling process in higher education. *Journal Educational Technol Systems*, 257-266.

Scott Ambler. Mapping Objects to Relational Databases: O/R Mapping In Detail, http://www.agiledata.org/essays/mappingObjects.html (visited 10/01/2013)

Carmo, J., Jones, A., 1996. A New Approach to Contrary-To-Duty Obligations. In, *Defeseasible Deontic Logic*, I, Donald Dute (ed.), Synthese Library

Tan, Y., T. L., 1994. Representing Deontic Reasoning in a Diagnostic Framework, in ILCP'94 Workshop on Legal Applications of Logic Programming, Genova, Italy.

Loecher, S., Ocke, S., 2010. A metamodel-based OCL-compiler for UML and MOF.In Elsevier, editor, Proceedings of the 6th International Conference on the Unified Modelling. San Francisco.

Borgida, A., 1985. Language features for flexible handling of exceptions, in *ACM Transactions on Database Systems* (TODS).

Borgida, A**.,** Murata, T**.**, 1999**.** Tolerating Exceptions in Workflows: A unified framework for Data and Process. in *Proc. International Joint Conference on Work Activities Coordination and Collaboration* (WACC), USA.

Briand, L. C., Dzidek, W., Labiche, Y., 2005. Using aspect-oriented programming to instrument OCL contracts in *Java. Proc. of IEEE International Conference on Software Maintenance* (ICSM), pp. 687-690, Budapest, Hungary

dresden-ocl.sourceforge.net/ (Dresden OCL Toolkit.) (visited in11/01/2010)

www.omg.org/ spec/SBVR/1.0/, Semantics Of Business Vocabulary And Business Rules (SBVR), Version (visited 10/01/2013)

www.sybase.com/products/modelingdevelopment/powerd esigner Sybase PowerDesigner. (visited 10/01/2013)