Enforcing Ontological Rules in UML-Based Conceptual Modeling: Principles and Implementation

Shan Lu[1] and Jeffrey Parsons[2]

[1]Department of Computer Science, [2]Faculty of Business Administration
Memorial University of Newfoundland
St. John's, NL A1B 3X5 Canada
[1]lu@cs.mun.ca, [2]jeffreyp@mun.ca

Abstract. UML is used for at least two purposes: OO software design, and conceptual modeling. However, UML's origins in software engineering may limit its appropriateness for conceptual modeling. Evermann and Wand [5,6,7] have developed a set of formal ontological rules that constrain the construction of UML diagrams to reflect underlying ontological assumptions about the real world. This paper examines issues in implementing that functionality in a UML CASE tool. The main contribution of our research is to distinguish four categories of rules for implementation purposes, reflecting the relative importance of different rules and the degree of flexibility available in enforcing them. We further propose four implementation strategies that correspond to these rule categories, and identify some rules that cannot be implemented without changing the UML specification. We have implemented the rules in an open-source UML CASE tool, providing a proof-of-concept demonstration of the feasibility and usefulness of the approach.

1 Introduction

The Unified Modeling Language (UML) has emerged as an important graphical modeling tool for designing complex software. Today, the UML is very popular, and has de facto become a standard for software engineering.

The origins and initial focus of the UML lie in the area of Object-Oriented (OO) software design. More recently, the UML has also emerged as a language for *conceptual modeling* – representing aspects of a real-world domain for which a system is required. In this context, the UML is a language for communicating between users and developers in understanding and eliciting requirements, and also for documenting the outcome of analysis.

The UML is generally viewed as well-suited for software design. However, extending UML for use in conceptual modeling has raised questions about its suitability for that purpose. More generally, the applicability of object-oriented modeling in the early development phases is controversial [10]. In addition, the UML is very complex; Version 1.5 has nine types of diagrams and an extensive specification encompassing 736 pages [9]. In short, "although UML 1.x has enjoyed widespread acceptance, its shortcomings include: excessive size and gratuitous complexity" [7].

In this setting, Evermann and Wand [3,4,5] provide ontology-based guidance for the use of the UML in conceptual modeling. Their work is grounded in the view that, since conceptual modeling involves representing aspects of the real world, *ontology*, the branch of philosophy dealing with the nature and structure of the real world, is an appropriate foundation to guide UML-based conceptual modeling. Accordingly, they

propose a set of ontological rules that place constraints on the construction of UML diagrams to ensure they properly represent underlying ontological assumptions. If followed, these constraints effectively impose a "method" for using the UML for conceptual modeling that reduces the degrees of freedom available to an analyst in creating a model.

However, Evermann and Wand propose a series of 75 rules and corollaries, some of which are quite complex. Enforcing them manually would be very difficult. Simply remembering and applying the rules would be a challenge in practice. Moreover, like most prior work on ontological analysis in conceptual modeling, they do not consider practical issues in implementing these rules.

To examine how ontological rules for using UML in conceptual modeling can be enforced effectively, this paper explores the extent to which and how Evermann and Wand's rules might be implemented in a UML CASE tool. We explore methods to check UML diagrams as they are being constructed, detect when violations of one or more rules occur, and take appropriate action depending on the violation. We implement these methods in a CASE tool, following the seven research guidelines proposed in the "design science" approach of Hevner et al. (see [6] for details of the approach).

Our research addresses the following two research questions:

Are Evermann and Wand's ontological rules practical? How can we implement these rules into CASE tools to provide effective support for UML-based conceptual modeling?

The remainder of this paper is structured as follows. First, we briefly summarize the parts of the UML relevant to conceptual modeling and introduce Evermann and Wand's ontological rules in Section 2. Section 3 explains the approach taken in analyzing and implementing the ontological rules. In Section 4, we offer some conclusions and present areas for future research.

2 Ontological Rules for Conceptual Modeling with UML

2.1 Elements of UML Relevant to Conceptual Modeling

We assume readers have basic knowledge of the UML and do not explain UML concepts here. Readers can refer to the UML 1.5 specification [9] for details.

Since our research is based on Evermann and Wand's ontological rules, we (and they) focus on diagrams related to those rules for conceptual modeling purposes, rather than consider all diagrams in the language. Our research covers the three types of UML diagrams (based on the UML 1.5 specification) indicated in Table 1.

2.2 Ontological Foundation

In addition to analyzing and examining the suitability of UML for domain modeling, Evermann and Wand have proposed constraints (rules) to make UML better suited for that purpose [3,4,5]. They point out that conceptual modeling involves representing aspects of the real world. When people use UML for conceptual modeling, the goal is to describe and help people to understand the real world. Therefore, we need a theory to describe the real world, against which to map UML constructs. Ontology is the

branch of philosophy dealing with the nature and structure of the real world. Ontology defines constructs that describe what the world consists of and how the world works.

Table 1. UML diagrams and components covered

Diagrams	Describe	Diagrams	Components included
Static structure	Static structure of things	Class diagram	Class, Attribute, Operation, Association, Binary association, Association class, N-ary association, Composition, Link, Generalization.
Change	Change within things	Statechart diagram, Activity diagram	State, Composite states, Events, Simple transitions, Transitions to and from Concurrent states, Transitions to and from Composite states, Submachine states, Synch states, Action state, Subactivity state, Call states, Swimlanes.
Interaction	Interaction between things	Sequence diagram, Collaboration diagram	Interactions, Messages, Stimulus

There are several ontologies. Evermann and Wand's research chooses Mario Bunge's ontology [1,2] because:

1. It is well formalized in terms of set theory;
2. It has been successfully adapted to information systems modeling and shown to provide a good benchmark for the evaluation of modeling languages and methods;
3. It has been used to suggest an ontological meaning to object concepts; and
4. It has been empirically shown to lead to useful outcomes [4, p3].

Bunge's ontology introduces basic concepts, including: Thing, Property, Change, Law, State, and Interaction, to describe the world. The world consists of things that possess properties. Properties are either mutual or intrinsic. Every thing has states defined by the specific value of its properties. States change as the property value changes. All these changes follow rules, which are called laws. Every thing can change. Change is either qualitative, in which a thing acquires or loses properties, or quantitative, in which property values of a thing changes. Two things are said to interact when they act on each other.

2.3 Evermann and Wand's Ontological Rules

Following Bunge's ontology, Evermann and Wand propose three groups of ontological rules: Static rules, Change rules, and Interaction rules, following an ontological commitment in which "our world consists of a static structure of things with their properties, changes in things and interactions of things" [5, p37]. In this section, we present examples of each kind of rule. The complete list of rules and corollaries is provided in [5]. We use the original numbering scheme below for ease of reference.

2.3.1 Static Rules

Rule 1: Only substantial entities in the world are modeled as objects.

In Bunge's ontology, "thing" is the basic element in the world and it refers to "substantial entity" [2, p110]. Substantial entities are things that physically exist in the world. They can be seen, heard, or felt by humans. For example, a 'book' can be seen; 'sound' can be heard, and 'wind' can be felt. However, 'job', and 'order' are not substantial entities since we cannot see, hear or feel them.

2.3.2 Change Rules

Rule 21: A UML-state represents a specific assignment of values to the attributes and attributes of association classes of the objects for which the state is defined.

According to ontology, a thing can be represented by state functions, whose values are determined by properties of the thing [2, p126]. That is, states of things associate with properties of things. The different property value sets correspond to different states. In Evermann and Wand's research, UML-states and UML-state transitions are mapped with ontology's states and ontology's state transitions, respectively. Also, "there exist no states which are independent of attributes because properties express all the characteristics of a thing" [7, p71]. In addition, properties consist of intrinsic properties and mutual properties, which are represented by attributes of ordinary and association classes respectively. Thus, we get this rule. For example, a 'Person' object has an attribute of 'location.' Different values of 'location' specify different states of the 'Person.' A value of 'office' implies state of 'work'; a value of 'bus' implies state of 'go home'; and a value of 'home' implies state of 'rest.' (Note that 'work,' 'go home,' and 'home' are high-level composite states that can be decomposed into sub-states.)

2.3.3 Interaction Rules

Rule 33: For every class of objects between which message-passing is declared, there exists an association class.

Interactions are expressed by message-passing, wherein a "message defines a particular communication between instances that is specified in an interaction" [12, p2-119]. That is, when message-passing exists, there is an interaction. According to Bunge [2], "two different things X and Y interact iff each acts upon the other" (p259). That means both objects participating an interaction undergo changes. Since "change may be quantitative, in which case the values of one or more properties is changed" [5, p33], property values of both objects should change. Also because every property must lawfully relate to other properties [1, p77], the changes of both objects must obey laws. That is, an interaction must lawfully change properties of both participating objects. However, a law is any restriction on the possible values of properties of a thing [1, p129]. A law only constrains property values of a single object [5, p32]. Therefore, in order for an interaction to change property values while satisfying laws in both objects, it must change the mutual properties of participating objects. Mutual properties can be mapped to association classes (Rule 3). Thus, for every class of

objects between which message-passing is declared, there exists an association class. For example, a 'Company' interacts with an 'Employee' by promotion. This promotion interaction changes the value of mutual property 'salary' of both 'Company' and 'Employee.'

The above examples illustrate the three kinds of rules proposed by Evermann and Wand. There are in total 36 rules and 39 corollaries (see [5]for a complete list).

3 Analyzing and Implementing Ontological Rules

To implement the ontological rules, we chose ArgoUML from several existing UML CASE tools. Our choice was motivated by several factors.

First, ArgoUML supports the function of critiquing designs. Critiques are activated when the program launches. When users draw UML diagrams, the system can detect mistakes and advise users. However, ArgoUML's native critiquing is based on UML syntax. Our research extends the critiquing function to enforce Evermann and Wand's ontological rules during diagram construction.

Second, ArgoUML is open source software. Thus, unlike a proprietary product, it is relatively easy to add our desired functionality to the tool. A disadvantage of using ArgoUML in this project has been the fact that, as an ongoing project, it does not fully realize the UML specification. As a result, the implementation of some rules had to be adapted to account for this.

3.1 Categorizing Rules by Priority

Examination of the rules and corollaries shows clearly that there are important differences in how they can be implemented practically within a UML CASE tool. For example, some rules are critical in the sense that failing to detect and correct violations *as they occur* during the construction of a diagram can lead to problems that are difficult or impossible to rectify later. In other cases, rule violations cannot reasonably be detected without human guidance and interpretation, and implementation support is primarily a matter of flagging possible violations. After analyzing all rules and corollaries, we categorize each under one of four general implementation approaches.

The first category is for critical (CRITICAL) rules. CRITICAL rules specify that something must be or must not be in a particular way (e.g., Rule 1). For rules of this type, the violation warning must be shown immediately to prevent further problems, and the cause of the violation must be addressed before proceeding. That is, as soon as the rule is violated, the system should alert the user. In the CRITICAL approach, the implementation displays a dialogue informing users of the violation and resets the related space so that the user can reenter correct information. For example, in Rule 1, when a non-substantial noun is detected (by a method such as that described later, the system will display a violation warning as well as clear the string of the non-substantial noun in the diagram and waiting for the new input string. Thus, the user cannot continue until he/she supplies a substantial noun as a class name. Figure 1 shows a system response to violation of a CRITICAL rule under the current implementation.

Not all violations are easy to correct. In some cases, it is not reasonable to delete the violating diagram element as in the CRITICAL approach. For example, "*Sets of*

mutual properties must be represented as attributes of association classes" (Rule 3). We cannot simply delete the 'mutual attributes.' Since this kind of violation is difficult to correct, a user may need time to fix it or may not want to fix it immediately.

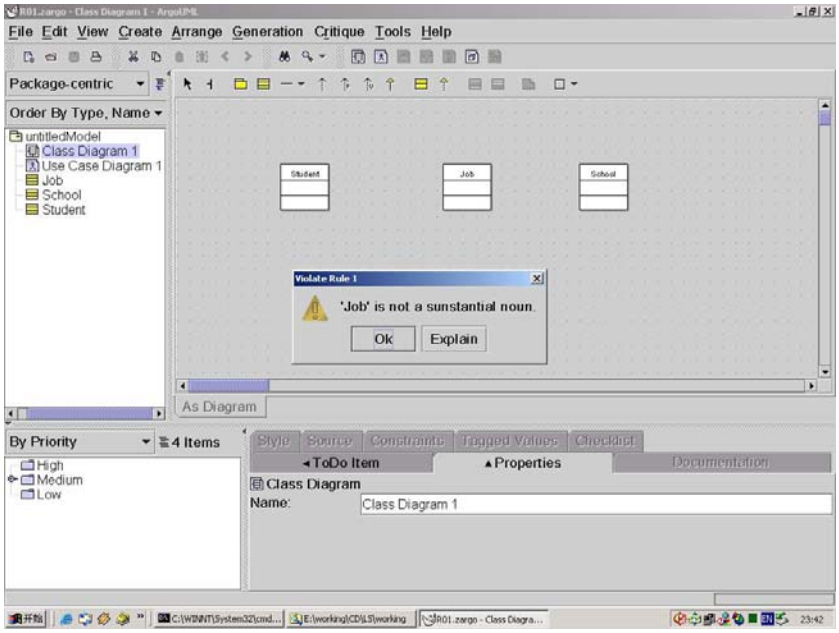

Figure 1.CRITICAL Rule Violation Warning

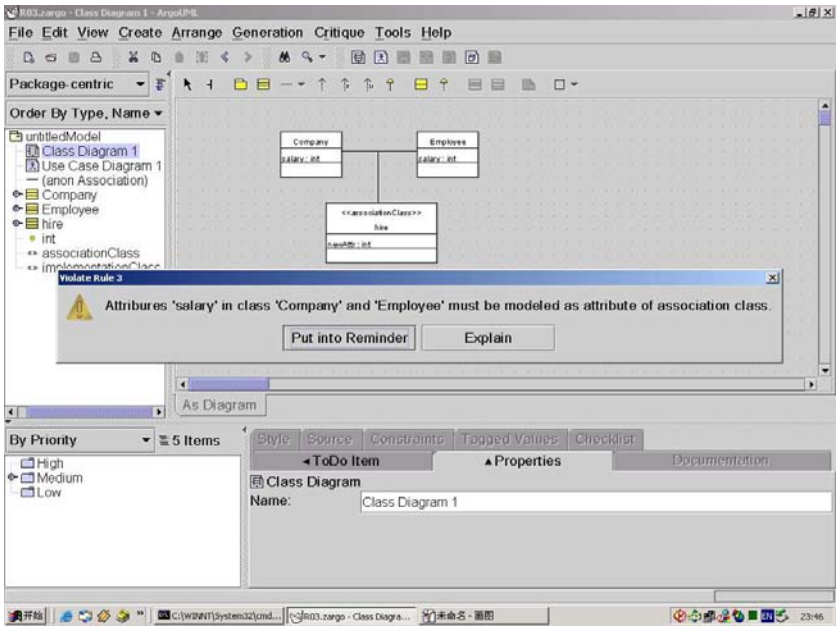

Figure 2. HIGH Priority Rule Violation Warning

In such cases, even though the rules indicate conditions that must be true in an ontologically sound representation, a CRITICAL implementation will result in a system that inhibits the modeler from working effectively on developing a diagram. To prevent this, we adopt a high priority (HIGH) approach: that is, once a rule violation occurs, the system displays a dialogue to warn the user. However, it does not clear anything in diagrams. Instead, it keeps the violating input and gives the user the option to correct it later (or even not to change it at all). To prevent the user from forgetting about the violation and to avoid continually displaying dialogues indicating the same violation, this violation is put into a HIGH reminder list. That means the popup dialogue will be shown only once for each violation instance of a rule. The system continues examining the diagram in real time. This is done in the background. If the user does not change the input, the violation message will remain in the reminder list, but the violation will not interfere with the task of diagram construction. If the user checks the list, he/she can view all outstanding HIGH violations.

To illustrate, Rule 3 is a HIGH priority rule. The system should inform users as soon as the violation appears. Thus, we design this examination to display a dialogue when it finds the same attribute in two associated classes, indicating that mutual properties may be modeled improperly. However, this violation cannot be corrected immediately. The user needs to create an association and put this attribute in that association class (and since this may be complex, may wish to defer the task). During that period, the system should not trigger further warnings of this violation. To accommodate this, the rule is stored in a high priority reminder list (see Figure 2).

The approach in this rule (as well as in some others) is imperfect. It may indicate a violation when in fact there is none. Consider the example "customer buys book from bookstore." Both customer and bookstore may have the intrinsic attribute "name", which are synonyms that refer to the different properties customer name and bookstore name. However, this situation still satisfies the violation of this rule in our approach. To accommodate situations such as this, our approach also allows designers to dismiss reminders of violations that are erroneously detected.

The third category is called medium priority (MED), corresponding to situations in which something must be or must not be included in a diagram. For example, "*Every UML-aggregate must possess at least one attribute which is not an attribute of its parts or participate in an association*" (Rule 7). For this kind of rule, a system cannot easily determine whether the user has forgotten to do something, or simply not done it yet. However, the system examines diagrams in real time. Thus, in situations involving rules of the kind "A must have B", as soon as A is created, the system will find there is no B in A before a user can input B. In this case, the system should not display a dialogue saying, A must have B. Instead, these kinds of violations are added directly to a medium priority reminder list. The system does not display any dialogue at all. Using this MED approach, we can keep track of the violation without interrupting and annoying users. In Rule 7, immediately after the user creates an aggregate class, the system will find that there is no additional attribute in the aggregate class than in its parts. The system detects a violation. However, the fact is that the user has no time to input an attribute yet. To avoid annoying users, the MED approach simply adds this violation directly to the reminder list. Moreover, if the user models an addi-

tional attribute later (thereby correcting the 'violation'), the warning will disappear. All this takes place without the user's awareness.

We call the fourth approach low priority (LOW) reminders. These do not entail violation warnings, but consist of reminders. Such situations are based on rules involving newly proposed structures (by Evermann and Wand) that do not exist in current UML; for example, "*A UML-state represents a specific assignment of values to the attributes and attribute of association classes of the objects for which the state is defined*" (Rule 21). There is no way for the program to detect violations of this kind of rule. In this case, the program will display reminders to users and ask users whether they want to keep this reminder. If the user indicates, the system will put the reminder into the LOW reminder list. In Rule 21, because there is no mechanism connecting states and attribute values in current UML, we use the LOW approach. The program gets all states and state machines they belong to. Then it gets the model element owning the state machine and reminds users to follow this rule.

3.2 Selective Rule Implementation

We did not implement all rules and corollaries proposed by Evermann and Wand. Specifically, our analysis of how certain rules might be implemented identified four general situations in which rules or corollaries did not need to be implemented: those *covered* or logically implied when others are followed; those that propose a kind of *solution* to violations of others; those that appear to be *inconsistent* with Bunge's ontology, and those that are impossible to violate and thus are *redundant*. We explain each case with examples.

Covered

> *Rule 29: An operation is not directly specified by state machines. Instead, the methods that implement an operation are specified by state machines.*
>
> versus
>
> *Corollary 25:A state chart either expresses the external behaviour of an object (SC0), a method, a signal reception or is a composite state contained in another state machine.*

In Corollary 25, we know that state machines represent behavioural features of model elements including objects, methods and signal receptions. The implementation approach of Corollary 25 will shows a violation warning if a state machine specifies a model element other than an object, a method or a signal reception. That is, if an operation is specified by state machine, the program that implements Corollary 25 will detect the violation. Thus, Rule 29 is covered by Corollary 25.

Solution

> *Corollary 3: If an association class of an n-ary association is intended to represent substantial things, the association should instead be modelled as one with arity (n+1).*
>
> *versus*

Corollary 2: An association class cannot represent substantial entities or composites of substantial entities.

Corollary 2 specifies that an association class cannot represent substantial things. Corollary 3 actually offers a solution to a violation of Corollary 2. If we enforce Corollary 2, Corollary 3 is not necessary.

Inconsistent

Corollary 37: Asynchronous communication of objects with expected response implies the existence of at least one state transition caused by the object acted upon, signifying the return interaction after the state transition signifying the original communication.

"An asynchronous invocation is the transmission of a request from a requestor to a target object in which the requestor continues execution immediately, without waiting for a reply" [9, p2-312]. The motivation of this corollary is that even though the acting object does not wait for a reply the acted object may still return a reply, which is considered "expected response" [5, p127]. However, UML specifies that "it is permissible to asynchronously invoke a request to a procedure that eventually issues a reply; the reply message is simply discarded" [9, p2-312]. One may argue that if the reply message actually sends some useful information, it should not just be discarded? However, "if the invocation is repliable, a subsequent reply by the invoked execution is transmitted to the requesting object as an asynchronous request" [9, p2-322]. That is, the response actually invokes another new communication. This is because "the target object might later communicate to the requestor, but any such communication must be explicitly programmed and it is not part of the asynchronous invocation action itself" [9, p2-312]. As we can see, there is no "expected response" for asynchronous communication. The motivation of this corollary is inconsistent with the UML specification.

Redundant

Corollary 18: All states in an activity diagram must be states of the same object.

In UML, "an activity graph is a special case of a state machine" [9, p2-172]. In addition, UML specifies that a state machine is a behavior that specifies the sequences of states of an object [9, g14]. That is, an activity diagram only represents one object. Consequently, all states in an activity diagram must be states of the same object. Thus, it is impossible to violate this corollary. Thus, it is redundant.

3.3 Supporting Features of the Implementation

To support the analysis and implementation of the ontological rules, we have also used several other approaches: inquiry for required information, use of a dictionary, and alternatives to proposed changes of UML. The following examples illustrate these approaches.

Inquiry for required information

Our implementation uses the approach of querying users for required information. When our program needs more information to determine whether a rule has been followed or violated, it queries users. Consider Rule 5. It is difficult to find a way for a program to tell whether attributes in an association class come from the same interaction. Once the program finds an association class, it gets all attributes from that association class and asks users whether these attributes are from one interaction (and provides an example to illustrate the concept). If the user indicates that they are not from the same interaction, the program gives a violation warning.

Use of a dictionary

Consider again Rule 1. We cap map "substantial entity" into human language, or into a domain ontology of terms. "Entities" can be mapped to nouns. To determine whether a noun is substantial or not, we used the following naive approach. First, classify all nouns in a dictionary into two groups: Substantial and Non-substantial. Then, create a file to store those nouns. When checking UML diagrams, the program gets the "object" and searches for a match in the file. If an object is "Non-substantial", the program shows the violation. If an object cannot be found in our database, the program asks the user whether that object is substantial or not. The program will add the new substantial or non-substantial noun to its database.

This approach clearly has limitations. If the user uses an abbreviation for an object name or a class name, the system cannot find it in the database. However, we propose an approach to mitigate this problem. A heuristic search algorithm can be adopted. If the user's input is similar to a word in our database, the system can ask the user whether the input is the same as the database's word.

Alternative implementation for change of UML

We do not change or delete any aspect of current UML, but provide alternative implementations in those cases. This is because Evermann and Wand's ontological rules currently have not been adopted by UML. Consider Corollary 21. Currently, there is no mechanism in UML to specify whether a state is stable or not. We use an alternative solution for this corollary. To determine if a state is stable or unstable, our program first gets all transitions with this state as the source state in SC0. Then each of these transitions is checked to see whether it has a trigger event. If any transition has no trigger, it can happen spontaneously. Thus, this state is unstable and the program shows a violation warning.

Finally, we have evaluated Evermann and Wand's ontological rules and, as a result, have revised, deleted, and created rules and corollaries. For example, the following is a rule proposed by our research.

Rule-New04: Neither stimuli, nor message should be modeled.

"Stimuli are not things in the world. They are abstract concepts that serve as descriptions, illustrations, abstractions or representations of interaction" [5, p118]. In addition, "a message is a specification of stimulus" [9, p3-111]. Thus, both stimuli and messages have no equivalent in ontology [5, p120].

4 Conclusions and Future Research

4.1 Feasibility of Implementing Ontological Rules

We have developed a general (i.e., not CASE tool specific) implementation approach for most of the ontological rules proposed by Evermann and Wand. We have done so for 35 rules and 26 corollaries. Of these, 3 rules and 11 corollaries are realized by enabling, disabling or modifying mechanisms (features) in the UML specification; 32 rules and 15 corollaries are realized by a critiquing mechanism that operates in real time as diagrams are being constructed. We have implemented algorithms for these (complete details are given in [8]). We found that the remaining 1 rule and 13 corollaries did not need to be implemented. Of these, 1 rule and 8 corollaries are covered by others (i.e., they can be deduced from other rules and corollaries.), 2 corollaries are actually the solution of violations of others, 2 corollaries are inconsistent with Bunge's ontology, and 1 corollary is always true and thus redundant.

Of the rules and corollaries that were implemented, our analysis shows that these can be divided into four categories, depending on their importance and the ease with which they can be detected. This is the key contribution of this research. The requirement to implement an artifact that supports the rules led to the categorization of levels of support in Table 2. Such a classification cannot be deduced from the theoretical analysis underlying the rules proposed by Evermann and Wand.

4.2 Proof-of-concept implementation

Evermann and Wand proposed a set of rules to ensure the ontological soundness of UML diagrams constructed for conceptual modeling purposes. However, they did not consider the usability of the rules in the sense of being able to implement them in a working UML CASE tool. We have been able to evaluate whether and how these rules can be implemented. In doing so, we shed light on differences among the rules in terms of the strategies needed to implement them effectively. Thus, we have empirically demonstrated their value. In addition, by considering implementation issues, we have identified some redundancies and inconsistencies among the rules which otherwise would be difficult to detect.

Table 2. Categories of feasible (to implement) rules

Rule Type	Implementation Approach	Explanation
Must Be; Simple to correct	CRITICAL	User is prevented from proceeding in the construction of a diagram until the violation is corrected.
Must Be; Difficult to correct	HIGH	User is warned once of the violation, and the violation is added to a reminder list with high priority.
Must Have	MED	There is no popup indicating the violation. Instead, the potential violation is added to a reminder list with medium priority.
Relates to non-existing UML component	LOW	User is queried about the issue and asked whether he/she wants to keep this reminder. Based on the answer, either a low priority reminder is added to the reminder list, or the reminder is discarded

4.3 Future research

The UML is recognized to lack theoretical foundations. Evermann and Wand propose rules that can be used to ensure that UML diagrams constructed for conceptual modeling purposes adhere to certain ontological principles. We have shown that most of these rules can be supported within a UML case tool. Thus, Evermann and Wand's and our work sets the stage for empirical (experimental or field-based) research to test the effectiveness of providing this kind of support. A critical question is whether modelers following ontological rules (either manually or supported by a tool such as ours) produce models that are in some sense "better" than those who construct models without ontological support. Related questions involve developing an understanding of ease of use and satisfaction associated with using a tool that supports the enforcement of the rules.

References

1. Bunge, Mario Augusto. (1977). Ontology I: The Furniture of the World: Volume 3 Treatise On Basic Philosophy. Dordrecht, Holland: D. Reidel Publishing Company.
2. Bunge, Mario Augusto. (1979). Ontology II: A World of Systems: Volume 4 Treatise On Basic Philosophy. Dordrecht, Holland: D. Reidel Publishing Company.
3. Evermann, Joerg and Yair, Wand. (2001a). An Ontological Examination of Object Interaction. Proceedings of the Workshop on Information Technologies WITS 2001, New Orleans, LA.
4. Evermann, Joerg and Yair Wand. (2001b). Towards Ontologically based Semantics for UML Constructs. Proceedings of the 20th International Conference on Conceptual Modeling ER'2001, Yokohama, Japan. Berlin: Springer Verlag.
5. Evermann, Joerg. (2003). Using Design Languages For Conceptual Modeling: The UML CASE. Unpublished doctoral dissertation, The University of British Columbia.
6. Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sudha Ram (2004). Design Science in Information Systems Research. MIS Quarterly, 28(1), 75-105.
7. Kobryn, Cris. (2002). UML2 for System Engineering. INCOSE 2002 Symposium.
8. Lu, Shan. (2004). Enforcing Ontological Rules in Conceptual Modeling Using UML: Principles and Implementation. Unpublished M.Sc thesis, Memorial University of Newfoundland.
9. Object Management Group (OMG). (2003). OMG Unified Modeling Language Specification. Version 1.5. Retrieve at June, 2003 from http://www.uml.org
10. Opdahl, A. L. and B. Henderson-Sellers. (2002). Ontological Evaluation of the UML Using the Bunge–Wand–Weber Model. Software Systems Modeling (2002) 1: P.43–67 / Digital Object Identifier (DOI) 10.1007/s10270-002-0003-9.