Ontological Semantics for the Use of UML in Conceptual Modeling

Xueming Li[1] and Jeffrey Parsons[2]

[1]Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, Canada A1B 3X5

xueming@cs.mun.ca

[2]Faculty of Business Administration
Memorial University of Newfoundland
St. John's, NL, Canada A1B 3X5

jeffreyp@mun.ca

Abstract

Despite its origins in software modeling, there has been growing interest in using the Unified Modeling Language (UML) for conceptual modeling of application domains. However, the UML has many constructs that are purely software oriented. Consequently, the suitability of the UML for modeling "real world" phenomena has been questioned. This research aims to assign real-world semantics to a core set of UML constructs by proposing a set of principles for mapping these constructs to the formal ontology of Mario Bunge, which has been widely used in information systems modeling contexts. We conclude by outlining how the proposed principles can be evaluated in terms of their effectiveness in supporting conceptual modeling using UML.

Keywords: UML, Ontology, Conceptual Modeling.

1 Introduction

Before developing an information system, the business and organizational domain in which the information system is to be used must be examined and understood. The system analysis phase of information system development focuses on how to represent this domain. Such a description is termed a *conceptual model*. Developing a conceptual model that faithfully represents the domain it is intended to represent is critical for successful information system development. It is widely held that one important advantage of the Unified Modeling Language (UML) is that it can be used both for modeling software, and for modeling the problem domain that is supported by a system (i.e., conceptual modeling) (Fowler, 2004). However, since UML was developed in large part based on ideas from the implementation domain, such as object-oriented programming languages and databases, its suitability for modeling real world domains in the early development phases has been questioned. For example, Opdahl and Henderson-Sellers (2002) indicate that within requirements engineering (RE), OO constructs and their use appear less well-understood and well-defined and their value is controversial. Woodfield (1997) argues, using eight examples, that significant impedance mismatches exist between conceptual modeling concepts and the concepts of object-oriented programming. More generally, UML lacks theoretical foundations to support its use for conceptual modeling. While the semantics of its constructs (such as *object*, *class*, *attribute*, *link*, and *association*) are clear with respect to software design and coding, they are vague with respect to application domain modeling.

These situations must be taken seriously since many authors have argued that RE is critical for successful information system development (Jackson, 1995; Offen, 2002) – the cost of repairing requirements errors during maintenance may be two orders of magnitude greater than that of correcting them during RE (Davis, 1993).

We argue that, to develop a conceptual model that faithfully represents a real world domain, we must understand in advance what exists in the real world. In philosophy, ontology is devoted to the study of the nature and structure of reality. A number of researchers have argued that ontology is an appropriate foundation for identifying the fundamental constructs that need to be supported by a conceptual modeling language and the relationships among them (e.g. Wand, 1989; Wand and Weber, 1993; Parsons and Wand, 1997; Evermann and Wand, 2005a, 2005b; Opdahl and Henderson-Sellers, 2002). Our research addresses the question: *Can we define a core of UML that has formal ontological semantics and is suitable for conceptual modeling?*

2 Ontology and Ontological Evaluation in Conceptual Modeling

In philosophy, the purpose of ontology "is to study the most general features of reality" (Peirce, 1935). It aims to answer questions such as: what kinds of entities exist in the world? Ontologies have become very influential in the Information Systems discipline, with research focusing on topics related to system analysis and design, enterprise systems and web services. Such research generally follows one of two different lines: ontologies as technologies of information systems (covering the ontology-driven information systems that use domain, task and application ontologies) and the ontological models of information systems (ontologies as abstract models supporting the core of the information system discipline and contributing to

the improvement of conceptual modeling techniques). Our research follows the second line.

As noted above, various researchers have argued that ontology is an appropriate theoretical foundation for identifying the fundamental constructs and the relationships among them that need to be supported by a conceptual modeling language. Most work has been based on Wand and Weber's adaptation of Mario Bunge's ontology (Bunge, 1977, 1979) to the information systems field. The idea of ontological evaluation of modeling languages developed by Wand and Weber (1993) is to find a mapping from ontological concepts into language constructs and vice versa. The former mapping shows how ontological concepts can be represented by the language and is therefore called *representation* mapping. The latter shows how language elements are interpreted ontologically and is called *interpretation* mapping.

Based on these mappings, Wand and Weber identify four ontological discrepancies that may undermine the *ontological completeness* and *ontological clarity* of constructs in a conceptual modeling language: c*onstruct deficit*; c*onstruct overload*; *construct redundancy*; and *construct excess*. The mappings between ontological concepts and conceptual modeling language constructs can be used to transfer assumptions from ontology to the modeling language. If there are ontological rules or constraints that relate two or more concepts, then by virtue of the mapping, these same rules or constraints must also apply to the modeling language. Thus, the ontological mappings can lead to modeling rules and guidelines on how to use the conceptual modeling language to model real world domains (Evermann and Wand, 2001).

Wand (1989) and Parsons and Wand (1997) apply the main concepts of Bunge's ontology to examine principles of object-oriented concepts. They distinguish representation related constructs from implementation related ones. Parsons and Wand derive guidelines for using object ideas in systems analysis. They argue that "the key to applying object concepts in systems analysis successfully is viewing objects as representation, rather than implementation, constructs…applying them with an implementation focus during analysis may have undesirable consequences" (1997, p. 105).

More recently, Opdahl and Henderson-Sellers (2002) analyze and evaluate UML as language for representing concrete problem domains using Wand and Weber's adaptation of Bunge's ontology. Using UML as an example, Evermann and Wand (2005a) propose a method based on Bunge's ontology to restrict the syntax of a modeling language such that impossible domain configurations cannot be modeled.

Most of the work cited above based on Bunge's ontology evaluates aspects of a modeling language by mapping ontological constructs into those of the modeling language and vice versa in a somewhat inconsistent manner. Also, the equally important mapping of relationships between ontological and modeling language constructs is often ignored. In this paper, we use Bunge's ontology to establish a more precise and consistent semantic framework of an ontologically-grounded UML core for conceptual modeling, which can be used to resolve a number of confusions in the UML literature.

3 Core Concepts of Bunge's Ontology

Following is a summary of key elements of Bunge's ontology.

3.1 Thing and Construct

In Bunge's ontology, there are two kinds of objects: *concrete things* or simply *things*, and *conceptual things* or *constructs*. The world is viewed as composed of things. Constructs (e.g. mathematical concepts such as numbers, sets, and functions) are only creations of the human mind which take part in our representations of the real world.

3.2 Property and Attribute

All things possess properties. Properties do not exist independent of things; in other words, every property is possessed by some thing or other. Some properties, such as Height and Age of a person (which is a concrete thing), are inherent properties of things, called *intrinsic properties*. Other properties, such as 'being enrolled in a university,' are properties of pairs (in general, n-tuples) of things, called *mutual properties*. A mutual property of a thing is a property that has meaning only in terms of some other thing or things in the world.

Humans conceive of things in terms of models of things (called *functional schemata*). Such models are conceptual things (thus constructs). Attributes are characteristics assigned to models of things according to human perceptions. We may use different models for the same thing, and therefore assign different sets of attributes to the same thing. Thus humans conceive of properties of things in terms of the attributes of their conceptual models, and properties are known to humans only as attributes. In a given model of a thing, usually not every property of the thing will be represented as an attribute. Therefore, every functional schema only reflects partial aspects of a thing. Likewise, an attribute in a given model may or may not reflect a substantial property. For example, the Height of a person (which is a model of a concrete thing) is an attribute that reflects a substantial property of the concrete thing. The Name of a person (which is a model of a concrete thing) does not represent any specific substantial property of the concrete thing. It is an attribute that stands for the individual as a whole.

3.3 Kind

An arbitrary collection of things need not share a given set of properties. When they do, however, and no thing outside the collection has the properties of interest, the collection is called a kind.

3.4 Law, Law Statement, and Natural Kind

A *law* is any condition or restriction on properties of a thing. A *law statement* is any restriction on the possible values of attributes of models of things and any relation among two or more attributes. The relation between laws and law statements is similar to that of properties and

attributes discussed above: laws restrict and interrelate properties, whereas law statements restrict and interrelate attributes representing these properties (law statements represent laws). Laws are also properties.

A kind of things is determined by a set of properties. A natural kind, however, is determined by the laws the things obey. Indeed, the deepest method of grouping things is by the laws they obey.

3.5 State, Conceivable State Space, and Lawful State Space

We do not handle directly concrete things but their models. The attributes of a model (or functional schema, see below) of things represent properties of the things. They are also called *state variables* or *state functions* because their values contribute to characterizing or identifying the *states* the things of interest can be in.

An ontological hypothesis is that, every thing is – at a given time – in some state or other. There are two kinds of law statements: *state law statements*, which specify lawful states that a thing could actually stay in, and *transformation law statements*, which specify lawful transformations of a thing from a lawful state to another lawful state. While state law statements reflect the static characteristics of a thing, transformation law statements reflect its dynamic ones. More precisely, a state law statement can be viewed as a mapping from the possible states of things in a given functional schema into the set {lawful, unlawful}; a transformation law statement can be viewed as a mapping from a set of tuples reflecting lawful states of the things in a given functional schema into {lawful, unlawful}.

3.6 Functional Schema

In Bunge's ontology, models of things are called functional schemata. Any natural kind of things can be modeled by some functional schema. A thing may have multiple functional schemata, reflecting different views of the same thing.

3.7 History and Coupling

Every thing is changeable. Changes of states manifest a history of a thing. The notion of history allows us to determine when two things are interacting with each other thus bonded or coupled to each other. Intuitively, if two things interact, then at least one of them will not be traversing the same states as it would if the other did not exist.

For example, consider a husband and a wife who are married to each other. Their histories are not independent, i.e., the state history of the husband is not the same as that he would traverse if he were single (and vice versa). As a result, they are coupled.

Couplings are also mutual properties. Two things are coupled if and only if some changes in one are accompanied or preceded or followed by some changes in the other. We call couplings *binding mutual properties* and the mutual properties discussed earlier *nonbinding mutual properties*. Note that, *only a general nonbinding mutual property can be represented by an attribute function in a functional schema.*

The interaction between two or more things may give rise to a number of *semantically related* nonbinding mutual properties. For example, the interaction between an employee and an employer may incur a set of nonbinding mutual properties such as Salary, StartDate, and OfficePhone etc. Therefore a binding mutual property between two or more things always implies the existence of some semantically related nonbinding mutual properties shared between these things. The reverse however is not always true, i.e. two or more things may share a nonbinding mutual property without being coupled. Examples are all spatiotemporal relations like "Thing A is five kilometers away from thing B". Here, A and B share a nonbinding mutual property 'five kilometers away,' but thing A does not act on or is acted upon by thing B, thus they are not coupled. Moreover, the effects of an interaction between two things may be lasting. Thus if two things interact for a while and then cease to do so, they are still coupled.

4 Ontological Foundation of UML for Conceptual Modeling

In this section, we assign to UML ontological semantics for conceptual modeling by mapping ontological constructs discussed above to UML model elements and vice versa, leading to a set of principles *for the use of UML for conceptual modeling*. It is important to emphasize that we suggest these principles be followed in the early information systems analysis phases when a conceptual model is developed to faithfully represent the domain it is intended to represent. We contend that implementation-oriented considerations should be added only in later system design phases.

4.1 UML Object

In UML, "An object represents a particular instance of a class. It has identity and attribute values." (OMG 2003, p. 3-64). Since real world things are represented in UML conceptual models as objects, we propose that things are modeled in UML as objects, and objects model things. In software design, not every object corresponds to a thing. For example, as argued by Evermann and Wand (2005b), Job is a set of mutual properties (Salary, StartDate etc.) shared by an employee and an employer.

Principle 1: In a UML conceptual model, every object models a thing. The converse is also true.

4.2 UML Attribute

In UML, "An attribute is a named slot within a classifier that describes a range of values that instances of the classifier may hold." (OMG 2003, p. 2-24) Since in Bunge's ontology, general (intrinsic and nonbinding mutual) properties are represented by attribute functions, we propose that attribute function is modeled in UML as attribute and a UML attribute models an attribute function.

Principle 2: In a UML conceptual model, every attribute of a class/type models an attribute function representing a

general (intrinsic or nonbinding mutual) property. The converse is also true.

Note that, in Principle 2, if attributes of a class/type model attribute functions that represent general intrinsic properties, then that class/type is an "ordinary" class/type, instances of which are objects which models things in the real world; otherwise, if the attributes model attribute functions which represent general nonbinding mutual properties shared by things, then that class/type is an association class, instances of which are links connecting objects modeling those things.

4.3 UML Class/Type

In UML, "A class is a description of a set of objects that share the same attributes, operations, methods, relationships, and semantics." (OMG 2003, p. 2-26) Since a functional schema models things of a natural kind using a set of attribute functions and law statements, we propose that functional schema is modeled as class/type.

Principle 3: In a UML conceptual model, every functional schema in the domain is modeled as a class/type. However, not every class/type models a functional schema.

For example, Figure 1 is a UML conceptual model adapted from Rumbaugh et al. (1998, p. 159). The classes Company and Person model two functional schemata whose natural kinds are sets of companies and persons respectively. However, association class Job does not model any functional schema because its instances are links, not objects. Instead, the attributes of Job (Salary and StartDate) model two attribute functions representing two general nonbinding mutual properties shared between employees and employers. Salary and StartDate are in fact the only attribute functions owned by functional schemata Employee and Employer which are represented in Figure 1 as named places.

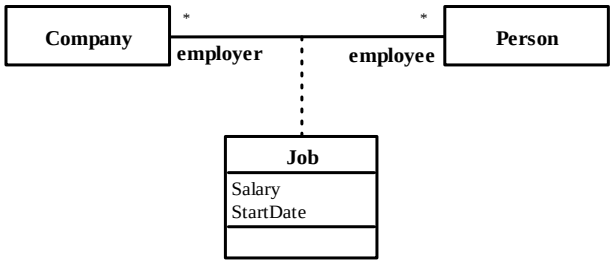

Fig. 1: Attributes of an association class

4.4 UML Link and Association

In UML, a classifier is a classification of instances describing a set of instances that have structural and behavioral features in common. There are several kinds of classifiers, including class, interface and data type, etc. Here, we only consider classes.

In UML, "A link is a connection between instances. Each link is an instance of an association, i.e. a link connects instances of (specializations of) the associated classifiers." (OMG 2003, p. 2-110) "An association defines a semantic relationship between classifiers. An association declares a connection (link) between instances of the associated classifiers (e.g., classes)." (OMG 2003, p. 2-64) Since individual nonbinding mutual property connects things which are modeled as objects in UML, we propose the following principle:

Principle 4: In a UML class or object diagram, every link between two or more objects models a tuple of values representing semantically related individual nonbinding mutual properties shared between things modeled by these objects. The converse is also true.

As a result, instances of an association (i.e. links) in a UML class diagram are more than "a tuple (list) of object references" (OMG 2003, p. 3-84). Moreover, from Principle 2, each attribute function representing a general nonbinding mutual property is modeled as an attribute of a UML association (or association class), thus we propose:

Principle 5: In a UML class diagram, every association between two or more classes/types models a tuple of attribute functions representing semantically related general nonbinding mutual properties shared by things that are modeled as instances of these classes/types. The converse is also true. Every association must have at least one attribute.

In a UML class diagram, attributes of an association can be illustrated using an association class attached to this association. Thus in UML class diagrams, association and association class are semantically equivalent.

For example, in Figure 1, association (or association class) Job models a pair of attribute functions Salary and StartDate, each of which represents a semantically related general nonbinding mutual property. Therefore, given a specific pair of employee and employer, the instance (link) of association Job is a pair of values that model individual nonbinding mutual properties of Salary and StartDate shared by the pair of employee and employer.

So far, the mutual properties we considered in a real world domain are nonbinding mutual properties as discussed in section 3.2. We suggest that, in a UML class diagram, every attribute of an association/association class models an attribute function representing a general nonbinding mutual property. Every association/association class in a UML class diagram must have at least one attribute shared by two or more classes/types because, otherwise, this association/association class is not needed at all. On the other hand, links between two or more objects in a UML collaboration model only binding mutual properties holding between things modeled by these objects. Therefore, links and their corresponding associations in UML class diagrams and collaborations are of fundamentally different ontological nature in that, while links and associations in UML class diagrams reflect static characteristics of a real world domain, those in UML collaborations reflect dynamic ones. Consequently, we may call associations and their links in UML class and object diagrams *nonbinding associations* and *nonbinding links*, and associations and their links in UML collaborations *binding associations* and *binding links*.

In UML, "A collaboration describes how an operation or a classifier, like a use case, is realized by a set of classifiers

and associations used in a specific way." (OMG 2003, p. 2-117) "A collaboration defines an ensemble of participants that are needed for a given set of purposes. Reasoning about the behavior of an ensemble of instances can therefore be done in the context of the collaboration as well as in the context of the instances." (OMG 2003, p. 2-124) From the UML specification, it seems that each link in a collaboration is supposed to be an instance of an association in the corresponding class diagram. However, this is a consequence of UML's implementation oriented origin from object-oriented programming in which links in class diagrams are merely communication passages for sending messages to linked objects.

In fact, this view has given rise to a considerable amount of confusion in the UML literature when considering the relationship between class diagrams and collaborations. For example, Stevens (2002), in order to remedy the so-called *Baseless Link Problem*, namely a link in a UML collaboration may not have a corresponding association in class diagram, proposes to classify associations in UML into *static* associations and *dynamic* associations. An association is static if one of the class definitions includes an attribute that contains a reference to an object of the other class. An association is dynamic if instances of the classes may exchange a message. Although Stevens' classification of associations into static and dynamic associations is somewhat similar to our classification of associations into nonbinding and binding associations, instead of focusing on the correspondence between real world domain and UML conceptual model, her proposal focuses on the correspondence between UML conceptual model and program code, therefore is not suitable for conceptual modeling.

Based on the above discussion, we have:

Principle 6: In a UML collaboration, every (binding) link between objects models a binding mutual property shared between things modeled by these objects.

As a result, in a UML collaboration, a (binding) link is completely determined by the objects it links. Note that, usually, not every coupling or binding mutual property shared between two or more things in the domain is modeled as a (binding) link in a UML collaboration. For example, consider a husband and a wife and the husband's employer. It is reasonable that the state history of the wife is not the same as that she would traverse if her husband's employer has not existed. As a result, the wife and the employer are coupled. However, in a UML collaboration, this (indirect) coupling is usually ignored and not modeled as a (binding) link.

Note that the interaction between two or more things (thus their coupling) will most likely give rise to a number of nonbinding mutual properties. Consequently, in a UML collaboration, whenever a binding link (representing a coupling or binding mutual property) exists between two or more objects, there will be a corresponding nonbinding link (representing a tuple of individual nonbinding mutual properties) between these objects in the corresponding class or object diagram. On the other hand, although rare, not every nonbinding link in a UML class or object diagram corresponds to a binding link in the corresponding collaboration. Examples are all spatiotemporal relations like "Thing A is five kilometers away from thing B". Here, thing A does not act on or is acted upon by thing B, thus they are not coupled.

4.5 UML Association Class

In UML, "An association class is an association that is also a class. It not only connects a set of classifiers but also defines a set of features that belong to the relationship itself and not any of the classifiers" (OMG 2003, p. 2-21).

From Principles 2 and 5, we know that an attribute of an association/association class models actually an attribute function representing a general nonbinding mutual property. Furthermore, an association class does not model any functional schema because its instances are links, not objects that model things in the real world domains. Therefore, as suggested by Evermann and Wand (2005b), an association class cannot have operations or methods. Instead, operations that change attribute values of an association/association class must be placed in participating *role* classes of the association. This idea has important implications for conceptual modeling (especially role modeling) and system design and implementation, but space precludes discussion of these issues here.

For example, in Figure 2 (adapted from Evermann and Wand, 2005b, p. 152), operations RaiseSalary and Terminate must be placed in either Employee or Employer, both are role types participating in the association Job. Note that, attributes of Employee and Employer (Salary and StartDate) model only Bunge-attribute functions representing Bunge-general nonbinding mutual properties shared by employees and employers, thus all of these attributes are placed in association/association class Job. As a result, the attribute compartments of Employee and Employer are empty. In system design phases however, we may treat Job as a "real" class and put implementation-oriented operations such as getSalary/setSalary and getStartDate/setStartDate into it. We may even put an operation Terminate into class Job as long as the semantic difference between Employer.Terminate (which means "lay off employee") and Employee.Terminate (which means "quit job") is not important for the design model.

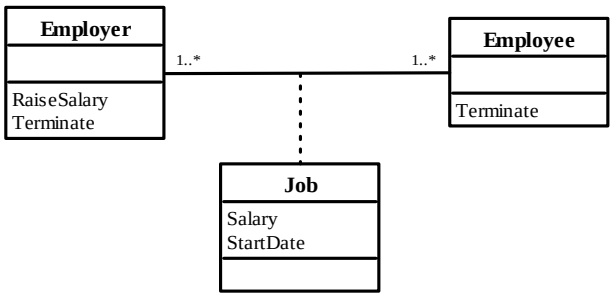

Fig. 2: Operations and methods of an association class (adapted from Evermann and Wand, 2005b)

Based on the above discussion, we have the following principle:

Principle 7: In a UML class diagram, an association class cannot have operations or methods, cannot be a composite class, and cannot be associated with other class(es).

4.6 UML State

In UML, "A state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event. … Conceptually, an object remains in a state for an interval of time." (OMG 2003, p. 3-137) Therefore, in UML, there is no precise definition even for such fundamental notion as state. We propose that a UML state models a lawful state and a lawful state is modeled as a UML state.

Principle 8: In a UML conceptual model, every state of an object models a lawful state of a thing modeled by the object. The converse is also true.

5 Conclusion

In this paper, we assign ontological semantics based on Bunge's ontology to a core set of UML constructs, namely UML object, attribute, class/type, association, link, association class, and state. The choice of these UML constructs is driven by Bunge's ontology. We also analyze consequences for conceptual modeling using UML based on this semantic mapping. For example, we have focused on UML association and link and indicate that links and their corresponding associations in UML class diagrams are of fundamentally different ontological nature from those in UML collaborations. As a result, the so-called *Baseless Link Problem* disappears naturally. Our analysis suggests that Bunge's ontology is an appropriate foundation for identifying the fundamental constructs and the relationships among them that need to be supported by a conceptual modeling language.

We argue that the proposed principles should be followed when using UML for conceptual modeling purpose. If they are, the ontological semantics of resulting models will be precise, allowing for unambiguous interpretation of a conceptual model expressed in UML. Thus, we argue that the models will more faithfully represent real-world phenomena. In this paper, we focus on discussing static aspects of UML. In the future, dynamic aspects of UML will be investigated. Research is also needed to determine whether following these rules leads to models that are easier to understand. Such questions are amenable to experimental and field-based empirical research methods.

6 References

Bunge, M. (1977): Treatise on Basic Philosophy (Volume 3), Ontology I: *The Furniture of the World*, Boston: Reidel.

Bunge, M. (1979): Treatise on Basic Philosophy (Volume 4), Ontology II: *A World of Systems*, Boston: Reidel.

Davis, A.M. (1993): *Software Requirements: Objects, Functions, States*, Prentice-Hall, Upper Saddle River, NJ.

Evermann, J. and Wand, Y. (2001): *Towards Ontologically-Based Semantics for UML Constructs*, in: H.S. Kunii, S. Jajodia, A. Solvberg , (Eds). Conceptual Modeling – ER 2001, Lecture Notes in Computer Science #2224, Springer,2001, Yokohama, Japan, November 27-30, pp. 341-354.

Evermann, J., Wand, Y. (2005a): *Toward Formalizing Domain Modeling Semantics in Language Syntax*, IEEE Transactions on Software Engineering, 31(1), pp. 21 - 37.

Evermann, J.; Wand, Y. (2005b): *Ontology based object-oriented domain modelling: fundamental concepts*, Requirements Eng. 10 (2): 146-160.

Fowler, M. (2004): *UML distilled: applying the standard object modeling language*, 3rd edition, Addison-Wesley.

Jackson, M. (1995), Software Requirements & Specifications – A lexicon of practice, principles and prejudices, ACM Press/Addison-Wesley, Workingham, England.

Object Management Group (2003): UML Specification 1.5.

Offen, R. (2002): *Domain understanding is the key to successful system development*, Requirements Eng., vol. 7. pp. 172-175.

Opdahl, A., Henderson-Sellers, B. (2002): *Ontological Evaluation of the UML using the Bunge-Wand-Weber Model*, Software and Systems Modelling, vol. 1, 1, pp. 43-67.

Parsons, J., Wand, Y. (1997): *Using Objects for Systems Analysis*, Communications of the ACM, (40)12, pp. 104-110.

Peirce, C. S. (1935): *Scientific Metaphysics*, Vol. VI of the Collected Papers, Cambridge, Mass.: Harvard University Press (original from 1892-1893).

Rumbaugh, J., Jacobson, I., Booch G. (1998): *The Unified Modeling Language Reference Manual*, Addison-Wesley Professional.

Stevens, P. (2002): *On the interpretation of binary associations in the Unified Modelling Language*, Software and System Modeling 1(1): 68-79.

Wand, Y. (1989), *A proposal for a formal model of objects*, In Won Kim and Frederick H. Lochovsky, editors, Object-Oriented Concepts, Databases, and Applications, chapter 21, pages 537—539, Addison-Wesley, ACM Press Frontier Series.

Wand, Y., Weber, R. (1993), *On the Ontological Expressiveness of Information Systems Analysis and Design Grammars*, Journal of Information Systems, pp. 217-237.

Woodfield, S.N. (1997): *The Impedance Mismatch Between Conceptual Models and Implementation Environments*, in ER'97 Workshop 4 Proceedings.