December 2004

An Ontological Evaluation of Use Case Grammar

Gretchen Irwin
Colorado State University

Daniel Turk
Colorado State University

An Ontological Evaluation of Use Case Grammar

Gretchen Irwin
Computer Information Systems Department
College of Business, Colorado State University
Gretchen.Irwin@ColoState.edu

Daniel Turk
Computer Information Systems Department
College of Business, Colorado State University
Dan.Turk@ColoState.edu

ABSTRACT

Use Case modeling is a popular technique for representing the functional requirements of an information system. The simple graphical notation of Use Case Diagrams, accompanied by well-structured narrative descriptions, makes Use Case models fairly easy to read and understand. This simplicity, however, belies the challenges associated with *creating* Use Case models. There is little, if any, theory underlying Use Cases, and little more than loose guidelines for creating a set of complete, consistent, and integrated set of Use Cases. We argue that Use Cases have strong potential value as a modeling and communication tool for different audiences and purposes, but there is a need for more rigor and consistency in the methods for creating and integrating Use Cases. Toward this end, we present a theoretical evaluation of Use Case modeling grammar. We use the findings from this evaluation to make recommendations for future research to understand and improve the Use Case modeling technique.

Keywords

Use Case, System Requirements, Unified Modeling Language (UML), Ontology.

INTRODUCTION

Use Case modeling is, for many organizations, the *de facto* technique for representing the functional requirements of a new information system (Dobing and Parsons, 2000: 29). Use Case Diagrams are part of the Unified Modeling Language (UML) and are based on a simple and rather intuitive grammar. The simple graphical notation of Use Case Diagrams, accompanied by textual Use Case Specifications, provides a conceptual model of the functional requirements for a new system in a readable and understandable manner.

The popularity, simplicity, and ease of reading Use Case models, however, belie the difficulty of *creating* them. A number of practitioners discuss the pitfalls and abuses of use cases observed on systems development projects (Cockburn, 2001, Fowler, 1998, Lilly, 2000, Rosenberg and Scott, 2001). They identify problems such as ambiguity about the system boundary in Use Case Diagrams (Lilly, 2000)); Use Case descriptions that include implementation-specific details or are otherwise "so detailed that the slightest change to the requirements will cause them to be rewritten" (Berard, 1998); and Use Cases that are so abstract they are practically useless (Berard, 1998, Korson, 1998). However, there is very little theoretical or empirical analysis of Use Case modeling that helps us understand the nature or extent of the problems, or the effectiveness of solutions to the problems. Two theoretical analyses of UML raise some concerns about Use Cases but do not address the issues most often cited in practice (Opdahl and Henderson-Sellers, 2002, Dobing and Parsons, 2000). A case study by Regnell and Davidson (1997) generally confirms the observations from the practitioner literature, but provides little theoretical explanation for the problems or the authors' proposed solutions.

This paper analyzes the constructs for Use Case Diagrams and Use Case Specifications from an ontological perspective. Ontology is "a theory that articulates those constructs needed to describe the structure and behavior of the world in general" (Wand and Weber, 2002: 365). Wand and Weber (1990, 1995) built on Bunge's (1979) ontology to create the Bunge-Wand-Weber ontology, hereafter referred to as the BWW model. The BWW ontology articulates the constructs that a conceptual-modeling grammar should have in order to be able to "create a 'faithful' representation" of real-world domains (Wand and Weber, 2002: 366). The BWW model has been successfully used to evaluate the "ontological expressiveness" of conceptual modeling grammars including the UML (Opdahl and Henderson-Sellers, 2002), NIAM (Weber and Zhang, 1991), the object model (Wand, 1996), and entity-relationship and data-flow diagrams (Wand and Weber, 1993).

Our analysis provides a theoretically-grounded evaluation of the effectiveness of use case models for representing one view of a real-world domain. The analysis also provides a basis for hypothesis development regarding the cognitive errors we expect authors and users of the models to encounter.

USE CASE MODELING GRAMMAR

Use Case Diagrams are used primarily to capture the functional requirements of an information system by focusing on *usage situations* for an information system. A Use Case Diagram shows "*who* does *what* with the system, for what *purpose"* (Malan and Dana, 2001). Figure 1 shows an example for an online order processing system that illustrates the grammatical constructs in Use Case Diagrams (OMG, 2003). The constructs are defined in Table 1.

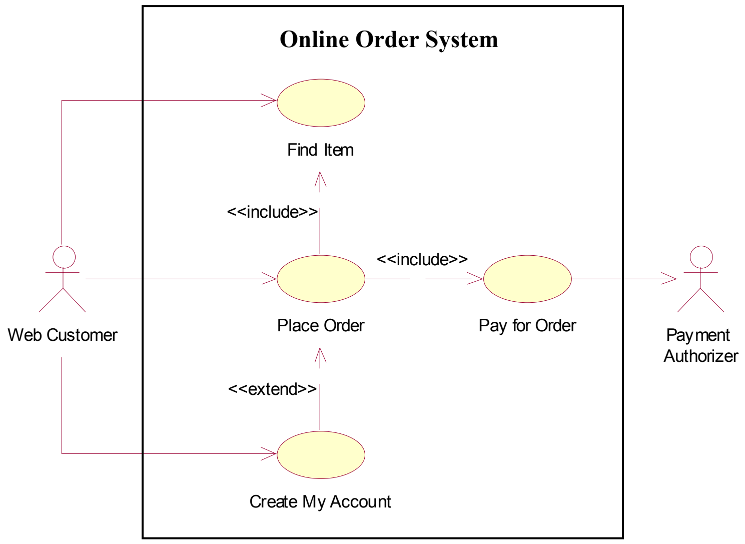


Figure 1. Sample Use Case Diagram.

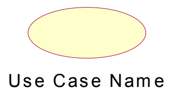
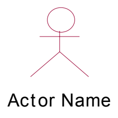
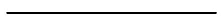
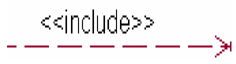
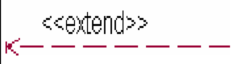
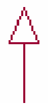
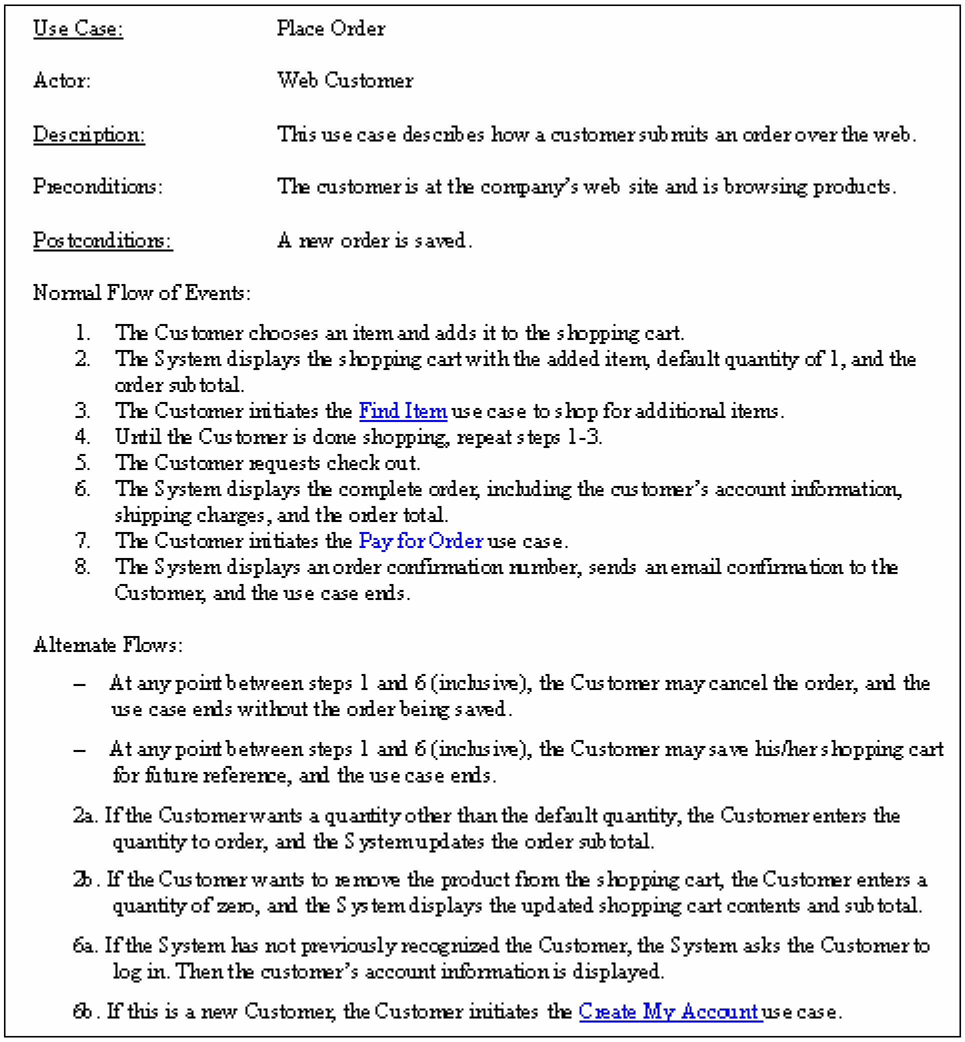
Construct	Representation	Definition(s)
Use Case		"A description of a set of sequence of actions, including variants, that a system performs that yields an observable result of value to a particular actor" (Jacobsen et al., 1999: 432).
Actor		A "coherent set of roles" that users can play when interacting with the system (3-97).
Association		A relationship that represents "the participation of an actor in a use case" (3-97).
Include		A relationship showing that an instance of one use case will also contain the behavior specified by another use case.
Extend		A relationship showing that an instance of one use case may be augmented, under some circumstances, by the behavior of another use case.
General-ization		"A generalization relationship from use case C to use case D indicates that C is a specialization of D"…"A generalization relationship from an actor A to an actor B indicates that an instance of A can communicate with the same kinds of use-case instances as an instance of B (3-98).

Table 1. Grammatical Constructs for Use Case Diagrams (as defined by OMG, 2003).

To provide a more detailed view of the system's functionality, Use Case Diagrams are often supplemented by Use Case Specifications. These specifications are not a part of the UML and there are many styles and formats (e.g., Cockburn 2001; Schneider and Winters 2001). However, most specifications include the constructs defined in Table 2 and illustrated in Figure 2. For the purposes of this paper, we consider these commonly-accepted constructs to be part of the Use Case Specification grammar.

Grammatical Construct	**Definition**
Normal Flow of Events	The set of sequential steps that describes the visible interaction between the actor and the system. (Cockburn 2001).
Alternate Flows	The exceptional, infrequent, or error-handling steps that may occur as alternate branches from the Normal Flow of Events.
Preconditions & Post-conditions	Preconditions state conditions that are assumed to be true before the use case begins. Post-conditions state conditions that are guaranteed to be true after the use case finishes successfully.

Table 2. Grammatical Constructs in Use Case Specifications.



Use Case: Place Order

Actor: Web Customer

Description: This use case describes how a customer submits an order over the web.

Preconditions: The customer is at the company's web site and is browsing products.

Postconditions: A new order is saved.

Normal Flow of Events:

1. The Customer chooses an item and adds it to the shopping cart.
2. The System displays the shopping cart with the added item, default quantity of 1, and the order subtotal.
3. The Customer initiates the Find Item use case to shop for additional items.
4. Until the Customer is done shopping, repeat steps 1-3.
5. The Customer requests check out.
6. The System displays the complete order, including the customer's account information, shipping charges, and the order total.
7. The Customer initiates the Pay for Order use case.
8. The System displays an order confirmation number, sends an email confirmation to the Customer, and the use case ends.

Alternate Flows:

– At any point between steps 1 and 6 (inclusive), the Customer may cancel the order, and the use case ends without the order being saved.

– At any point between steps 1 and 6 (inclusive), the Customer may save his/her shopping cart for future reference, and the use case ends.

2a. If the Customer wants a quantity other than the default quantity, the Customer enters the quantity to order, and the System updates the order subtotal.

2b. If the Customer wants to remove the product from the shopping cart, the Customer enters a quantity of zero, and the System displays the updated shopping cart contents and subtotal.

6a. If the System has not previously recognized the Customer, the System asks the Customer to log in. Then the customer's account information is displayed.

6b. If this is a new Customer, the Customer initiates the Create My Account use case.

Figure 2. Use Case Specification for the Place Online Order Use Case.

THE BWW ONTOLOGICAL FRAMEWORK

The BWW model specifies that the structure and behavior of a real-world domain can be represented by: (1) *things* that possess *properties*; (2) *events* that cause (the properties of) things to change according to certain *transformation laws*; and (3) *systems*, which are *composite things* made up of component things that interact with each other and with things in the *environment* of the system. The fundamental constructs of the BWW ontology are described in detail in Weber and Zhang (1991), Wand (1996), and Wand and Weber (1990).

Wand and Weber describe four "ontological discrepancies" that may arise if there is not a one-to-one mapping between constructs in the BWW ontology and constructs in the conceptual-modeling grammar (Wand, 1996). These discrepancies represent possible shortcomings in the conceptual modeling grammar, and are defined below.

1. *Construct overload* occurs when more than one ontological construct maps to a single grammatical construct.
2. *Construct redundancy* occurs when one ontological construct maps to more than one grammatical construct.
3. *Construct excess* occurs when a grammatical construct has no counterpart in the ontology.
4. *Construct deficiency* occurs where an ontological construct has no corresponding grammatical construct.

ONTOLOGICAL EVALUATION OF USE CASE GRAMMAR

We compared the grammatical constructs from Tables 1 and 2 to the BWW ontological constructs described in Wand and Weber (1990). This section discusses the main findings of this comparison, summarized below in Table 3.

The System Construct

A Use Case Diagram is a model of one thing, namely, a *BWW system thing*. The system in a Use Case Diagram may represent an organizational system, an information system, or a particular application within an information system. In UML version 1.4, the system construct is depicted by a rectangular box surrounding the use cases, to show that the use cases are within the system and the actors are external to the system. However, UML 1.4 states that the system construct is optional, and version 1.5 fails to mention the system construct at all in its discussion of Use Case Diagrams. Many published examples of Use Case Diagrams omit the system construct (e.g., OMG, 2003, Rosenberg and Scott, 2001), and in other cases it is shown but poorly defined (Lilly, 2000).

Consider the diagram in Figure 2 of a patient appointment system (Dennis, Wixom, and Tegarden, 2002). The system is defined as the "Appointment System", which suggests the authors are referring to the automated information system. However, the patient does not directly interact with the computerized system, and thus should not be shown. Instead, the appointment scheduler, who acts on behalf of the patient, should be the primary actor. This example illustrates what Lilly (2000) refers to as the #1 Use Case pitfall observed in practice—an undefined or inconstant system boundary.

There is technically no ontological discrepancy because UML 1.4 defines a system construct. However, the system construct is omitted in the UML 1.5. Thus, from a practical standpoint, this may rise to the level of *construct deficiency*.

The Actor Construct

A UML actor appears to have a fairly straight-forward correspondence to a kind (class) in the BWW ontology. An actor represents a set of things in the environment that have common properties, are outside of the system under discussion, and interact with the system in a specific way (Booch et al., 1999). Regnell and Davidson (1997) state that an actor is an entity, human or non-human, that communicates with the system to achieve certain goals. These descriptions are consistent with the ontological definition of a thing (or kind).

However, the UML explicitly defines an actor as a *role,* and a role, in BWW terms, is a *property* of a thing, not a thing itself (Weber and Zhang, 1991). Constantine and Lockwood (2000) note that in most other contexts, actors are not semantically equivalent to roles. Instead, actors take on roles; the role is a part played by an actor in a certain context. They argue that UML would be more accurate to use the term role instead of actor, where a role is "a relationship between a user and a system and is defined by a set of characteristic needs, interests, expectations, behaviors, and responsibilities" (Constantine and Lockwood, 2000: 3). The UML actor is an example of *construct overload* because the one UML construct is used to represent both a BWW thing and a BWW property of a thing.

BWW Construct	**Use Case Construct**	**Observations**	**Ontological Discrepancy?**
Thing	**System**	A BWW system is a composite thing. A Use Case Diagram models a system thing with a symbol that is optional.	Possible construct deficiency
	Actor	An actor is described both as a role (a BWW property), and as a thing external to the system.	Construct overload
Property of a thing	**General-ization**	UML's generalization between actors violates the "a kind of" semantics of BWW generalization. Generalization between use cases (BWW processes) violates the BWW definition of subkind.	Construct overload
	Assoc-iation	Association represents a BWW mutual property of an actor and the system, but is shown as a mutual property of a role and a process, which has no ontological counterpart.	Construct overload/ excess
Dynamics of a thing	**Use Case**	A use case represents a set of BWW processes. The execution of a use case scenario changes the state of the system from one stable state to another.	None
	Normal Flow of Events	Normal flows in Use Case Specifications represent BWW transition law properties that govern the system's transformation from one stable state to another.	None
	Alternate Flow of Events	Alternate flows represent BWW transition law properties that govern the system's transformation from one stable state to another.	None
	Pre-condition	Preconditions in Use Case Specifications represent BWW state law properties that restrict the values of one or more properties of the system	None
	Post-condition	Post-conditions represent BWW state law properties because they restrict the values of one or more properties of the system.	None
	Include & extend	Include and extend relationships may represent a type of BWW aggregation, or they may represent constructs to support reusability that have no counterpart in the BWW model.	Construct overload or excess
System Environ-ment	**Actor**	The BWW system environment corresponds to the actors that interact with the system.	None
System Structure	**NA**	BWW system components have no correspondence to Use Case constructs.	Construct deficiency
Decompo-sition	**NA**	BWWsystem composition and level structure have no correspondence to Use Case constructs.	Construct deficiency

Table 3. Ontological Evaluation of Use Case Grammatical Constructs.

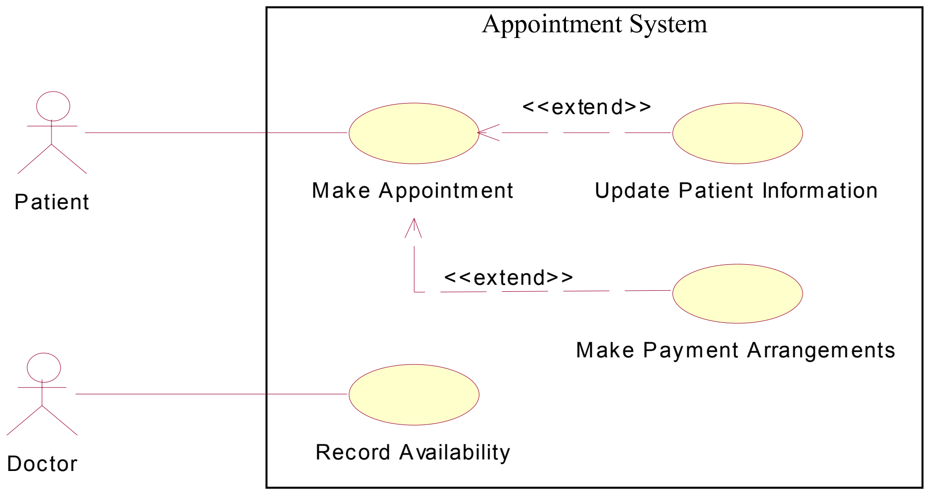


Figure 2. Use Case Diagram with Poorly Defined System Boundary (Dennis et al., 2002)

The Generalization/Specialization Construct

Generalization in UML is "the taxonomic relationship between a more general element (the parent) and a more specific element (the child) that is fully consistent with the first element and that adds additional information" (OMG, 2003: 3-86). Generalization is used in Class Diagrams to depict superclass-subclass relationships where the subclass is semantically "a kind of" the superclass. This corresponds to the BWW superkind and subkind, and is a well-known construct in conceptual-modeling grammars such as Entity-Relationship Diagrams (e.g., Elmasri and Navathe, 1994).

UML 1.5 specifies that generalization may be applied to actors and use cases as well as to classes (OMG, 2003: 3-86). The first problem with this is that generalization between use cases does not make sense from an ontological perspective. Generalization in the BWW model is a relationship between kinds of things, and a use case is not a kind of thing.

The second problem pertains to the specific UML definition of generalization between actors: "A generalization relationship from an actor A to an actor B indicates that an instance of A *can communicate with* the same kinds of use-case instances as an instance of B" (OMG, 2003: 3-99, emphasis added). To say that A can communicate with the same use cases as B is not necessarily equivalent to saying that A is a kind of B. Furthermore, as mentioned earlier, UML actors are defined as roles, which are BWW properties of things. Generalization between properties is not ontologically accurate.

This is another example of *construct overload.* The UML generalization construct in Class Diagrams corresponds to a BWW *subkind relationship* between kinds of things. The generalization construct is also used in Use Case Diagrams to specify a relationship between properties (roles) and between processes (use cases). The generalization construct in Use Case Diagrams corresponds more closely to a BWW *law* than to BWW generalization. A BWW law is a property of a thing that constrains or specifies relationships among other properties (Wand, 1996).

The Association Construct

The UML association construct is used in both Class Diagrams and Use Case Diagrams. In a Class Diagram, an association represents a relationship between instances of one or more kinds. This is consistent with a BWW *mutual property*, which is a property that can be defined only in the context of two or more things (Opdahl and Henderson-Sellers, 2002).

In a Use Case Diagram, an association is a relationship between an actor and a use case that means the actor has a goal and *interacts with* or *communicates with* the system in a specified way to accomplish the goal. There are some ontological problems with this definition. First, the UML association symbol has broad semantics when used in a Class Diagram and very narrow semantics when used in a Use Case Diagram. The "interacts with" semantics do not correspond to any construct in the BWW model. This indicates either an excessive construct in the Use Case grammar, or a missing construct in the BWW ontology.

The second problem is that the UML association is another example of *construct overload.* Association represents a mutual property between *kinds* of things in Class Diagrams but in Use Case Diagrams, association links an actor (a kind of thing or a property of a thing) to a use case (a process). This is more consistent with the definition of a BWW *transformation law* than a BWW mutual property. A transformation law is a property that constrains the events that can occur to a system. In Use Case Diagrams, the association construct represents a constraint on whom (e.g., the customer) can invoke or be involved with a particular change to the system (e.g., placing an order).

The Use Case Construct

Opdahl and Henderson-Sellers (2002) describe an instance of a use case as the performance of a sequence of actions, where each of these actions is a *BWW-event*. A sequence of BWW-events constitutes a *BWW process*. So, "UML…use case instances represent a BWW-process in the proposed system thing or in a subsystem thereof. UML-use case class represents a group of such BWW-processes" (p. 50). The execution or instantiation of a use case changes the state of the system from one stable state to another. The specification of a use case explains the lawful transformations that may occur during the execution. We found no ontological discrepancy with respect to the use case construct.

Constructs in the Use Case Specification

The Use Case Specifications define certain properties of the system. In particular, the Flows of Events specify sequences of permissible actions that may occur as part of the Use Case process. The Normal Flow represents transformations that occur when everything goes well. An Alternate Flow represents transformation that may occur at a certain branching point from the Normal Flow (e.g., when something goes wrong in the Normal Flow). Both of these constructs represent BWW-*transformation law* properties of the system.

We find no ontological discrepancy with regard to these constructs. However, we note they are fairly informally-represented constructs. The Use Case Specification grammar does not clearly indicate how to represent the transformation laws, other than to provide general guidelines such as, "use 3 to 9 steps" in the Normal Flow (Cockburn 2001) and "write present tense verb phrases in active voice" (Rosenberg and Scott 2001). Constantine and Lockwood (2000), for example, note that the Use Case Specifications are typically and strictly sequential and do not handle optional, flexible, or iterative execution of steps particularly well. We do not consider this a construct deficiency because there are other grammars and UML diagrams that represent these aspects of a system process (e.g., Activity Diagrams).

Preconditions and post-conditions are two other constructs typically included in Use Case Specifications. These constructs correspond to BWW *state law properties* because they specify the stable states of the system prior to and upon completion of a Use Case. We did not identify any ontological discrepancies with respect to pre- or post-conditions, except again, to note that the representation of these constructs is fairly informal.

The Include and Extend Constructs

The UML <<include>> and <<extend>> relationships describe certain types of interaction between use cases. Opdahl and Henderson-Sellers (2002) state that <<include>> and <<extend>> are "very high level UML constructs that… [have] no explicit counterpart in the BWW-model. The most useful interpretation might be to regard <<extend>> and <<include>> as subtypes of BWW-whole-part relations, but this must be considered further" (p. 59). We agree that the <<include>> and <<extend>> constructs are reminiscent of aggregation. For example, Figure 1 shows that the *Place Order* Use Case includes the *Pay for Order* Use Case, which means that *Pay for Order* is a logical sub-part of *Place Order.* Similarly, the <<extend>> relationship implies that in some circumstances, *Create My Account* will be a logical sub-part of *Place Order*.

However, we believe that the interpretation of <<include>> and <<extend>> as special forms of aggregation is problematic for at least two reasons. First, the UML already defines grammatical constructs for aggregation in the context of Class Diagrams. So to add a new construct for aggregation on Use Case Diagrams would constitute *construct redundancy*. Second, aggregation is understood as a whole-part relationship between things or kinds of thing. To apply an aggregation construct to use cases (ontological processes) would constitute *construct overload*.

To further complicate the issue, some authors argue that included Use Cases represent reusable "chunks" of functionality rather than a sequence of actions that produces an "observable result of value to an actor" (Lee and Xue, 1999). In this case, included Use Cases are less important for conceptual modeling purposes (i.e., representing the problem domain) than for design purposes, and have no counterpart in the BWW ontology. This would constitute *construct excess*.

Constantine and Lockwood (2000) note that both included and extending use cases may provide reusable functionality. They raise concerns about the multiple meanings and uses of the <<extend>> construct. First, <<extend>> may represent a conditional inclusion, where the base use case references the point in the sequence at which the conditional use case occurs. For example, in Figure 1, there would be a specific point in *Place Order* where certain conditions would cause the *Create My Account* use case to execute. Second, <<extend>> may represent an alternative or exceptional case whose point of occurrence is not known or is unpredictable (e.g., canceling, or resetting an order). Constantine and Lockwood argue that the former definition of <<extend>> would be more accurately defined as a specialized type of the <<include>> relationship.

In sum, the <<include>> and <<extend>> constructs do not have a clear ontological mapping or foundation, which may explain some of the complaints in practice about organizing and structuring use cases with these constructs (e.g., Korson, 1998, Lilly, 2000).

BWW Systems

A BWW-system is a *composite thing* made up of interacting component things. A system exists within an *environment*, which consists of things that interact with the system but are not part of the system. The *structure* of a system shows the interactions between components within the system and the interactions between things in the environment with the system.

The *system environment* is represented in a Use Case Diagram by the actors that are outside of the system boundary. This is clear from the definition of actors as external to the system. Actors interact with a system by initiating use cases that cause the state of the system to change. We find no ontological discrepancy with respect to the system environment, so long as the (optional) system boundary is defined.

The Use Case modeling grammar does not include constructs to adequately represent the system as a composite thing, or the structure of the system. Use Case modeling grammar does not support the representation of subsystems within a system or of the environment of a subsystem being "inside" the composite system.

The UML constructs packages, <<include>>, and <<extend>> may address system composition to some extent. The UML package construct is a generic grouping of elements in the grammar. A package could then, theoretically, be used in Use Case Diagrams to show the components within a given system. However, we have seen very few examples of Use Case Diagrams where the system's component packages are shown along with interactions between actors and the relevant component packages. The <<include>> and <<extend>> constructs in Use Case Diagram do, in some sense, represent whole-part relationships between use cases. However, as explained earlier, this interpretation is problematic, and even if it were not, it is not clear how an included or extending use case relates to a component of a system. Thus, Use Case modeling grammar is ontologically *deficient* with respect to the composite nature of a system and the structure of a system.

Decomposition

Decomposition is a key construct of the BWW ontology that is almost completely absent from the Use Case grammar (Opdahl and Henderson-Sellers, 2002). A decomposition of a system is a set of subsystems where each element of the system is included in at least one of the subsystems in the set (Wand and Weber, 1990). In a Use Case Diagram, decomposition would be supported if every use case within the system could be clearly represented as belonging to a component (subsystem) of the system. As noted earlier, however, the Use Case modeling grammar is deficient with respect to representing components of a system, and so, by necessity, is deficient in its ability to show the use cases within those components.

Several authors have noted the difficulties in creating Use Case models at different levels of abstraction, particularly of relating system-level Use Cases to business-level Use Cases (Berard, 1998, Fowler, 1998, Korson, 1998, Lilly, 2000). Cockburn (2001) relates a comment from a colleague: "I have yet to experience to my satisfaction a full-fledged story of business use cases that unfold into system use cases…I have not seen a clean connection from the business use case to system use cases" (p. 158). Comments like these reflect, in part, the notion that the Use Case modeling grammar is *deficient* with respect to system decomposition.

IMPLICATIONS AND CONCLUSIONS

Use Case models have practical and ontological strengths. The simplicity of the grammar promotes communication about the problem domain and the system requirements between analysts and users. Use Cases are designed to illustrate a high-level dynamic view of a system and, with the exception of the ontological ambiguity surrounding inclusion and extension, the grammar supports the dynamic view well.

However, the grammar has several ontological weaknesses. First, there are construct deficiencies, the most notable of which is the lack of emphasis on the system construct, system structure, and system decomposition. Future research is needed to investigate the extent to which these deficiencies lead to one or more of the hypothesized problems:

(1) Client-analyst communication problems due to unclear or mis-specified system boundaries.

(2) Analyst-designer communication problems, for similar reasons, and also because of the inability to clearly show how elements of a Use Case Diagram at one level of abstraction are related to another, more detailed level of abstraction.

Second, we identified several examples of construct overload, particularly with respect to the UML actor and generalization constructs. We hypothesize that this construct overload may lead to the following problem:

(3) Difficulty creating Class Diagrams when Use Case Diagrams are used as one source of input. The actor, association, and generalization constructs are relevant to both diagrams, but have subtly different ontological meanings in each context.

Lastly, the <<include>> and <<extend>> constructs are poorly understood from an ontological perspective. We hypothesize that this may lead to both communication and modeling difficulties:

(4) Modeling difficulties when these constructs are used to fill the deficiencies in the Use Case grammar with respect to system structure and decomposition.

(5) Client-analyst communication difficulties when included or extending use cases represent reusable functionality (a design goal) rather than a faithful representation of the problem domain.

REFERENCES

1. Berard, E. (1998). "Be Careful With Use Cases," The Object Agency, Inc. Accessed 2/1/2004 at http://www.toa.com/pub/use_cases.htm.
2. Booch, G., J. Rumbaugh, et al. (1999). The Unified Modeling Language User Guide, Addison Wesley.
3. Bunge, M. (1979). Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems. Boston, MA, Reidel.
4. Cockburn, A. (2001). Writing Effective Use Cases. Boston, Addison Wesley.
5. Constantine, L. L. and L. A. D. Lockwood (2000). "Structure and Style in Use Cases for User Interface Design." Accessed 2/1/2004 at http://www.foruse.com/articles/structurestyle2.htm.
6. Dennis, A., B. H. Wixom, and D. Tegarden. (2002). Systems Analysis and Design: An Object-Oriented Approach with UML. New York, John Wiley & Sons, Inc.
7. Dobing, B. and J. Parsons (2000). "Understanding the Role of Use Cases in UML: A Review and Reesearch Agenda." *Journal of Database Management*, 11 (4): 28-36.
8. Elmasri, R. and S. Navathe (1994). Fundamentals of Database Systems. Redwood City, The Benjamin/Cummings Publishing Company, Inc.
9. Fowler, M. (1998). Use and Abuse Cases. *Distributed Computing*, April: 1-2. Accessed 2/1/2004 at http://www.martinfowler.com/distributedComputing/abuse.pdf.
10. Jacobsen, I., G. Booch, and J. Rumbaugh. (1999). The Unified Software Development Process. Boston, Addison Wesley.
11. Korson, D. T. (1998). The Misuse of Use Cases (Managing Requirements). *Object Magazine*. Accessed 2/1/2004 at http://www.korson-mcgregor.com/publications/korson/Korson9803om.htm.
12. Lee, J. and N.-L. Xue (1999). "Analyzing User Requirements by Use Cases: A Goal-Driven Approach." *IEEE Software* (July/August): 92-101.
13. Lilly, S. (2000). How to Avoid Use-Case Pitfalls. *Software Development Magazine*. Accessed 2/1/2004 at www.sdmagazine.com/print/documentID=11235.
14. Malan, R. B., Dana (2001). Functional Requirements and Use Cases. White paper from Bredemeyer Consulting, Inc. Accessed 5/1/2004 at http://www.bredemeyer.com/pdf_files/functreq.pdf.
15. OMG (2003). Unified Modeling Language (UML), Version 1.5. Accessed 2/1/2004 at http://www.omg.org/technology/documents/formal/uml.htm.
16. Opdahl, A. L. and B. Henderson-Sellers (2002). "Ontological Evaluation of the UML Using the Bunge-Wand-Weber Model." *Software Systems Model*, 1: 43-67.

17. Regnell, B. and A. Davidson (1997). From Requirements to Design with Use Cases - Experiences from Industrial Pilot Projects. *Proceedings of the 3rd International Workshop on Requirements Engineering - Foundation for Software Quality (REFSQ)*, Barcelona, Spain.
18. Rosenberg, D. and K. Scott (2001). Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example. Reading, Mass., Addison-Wesley.
19. Schneider, G. W., Jason P. (2001). Applying Use Cases: A Practical Guide. Upper Saddle River, Addison-Wesley.
20. Wand, Y. (1996). "Ontology as a Foundation for Meta-Modelling and Method Engineering." *Information Science and Software Technology*, 38: 281-287.
21. Wand, Y. and R. Weber (1990). "An Ontological Model of an Information System." *IEEE Transactions on Software Engineering*, 16 (11): 1282-1292.
22. Wand, Y. and R. Weber (1993). "On the Ontological Expressiveness of Information Systems Analysis and Design Grammars." *Journal of Information Systems*, 3: 217-237.
23. Wand, Y. and R. Weber (1995). "On the Deep Structure of Information Systems." *Information Systems Journa*l, 5: 203-223.
24. Wand, Y. and R. Weber (2002). "Research Commentary: Information Systems and Conceptual Modeling -- A Research Agenda." *Information Systems Research*, 33 (4): 363-376.
25. Weber, R. and Y. Zhang (1991). An Ontological Evaluation of NIAM's Grammar for Conceptual Schema Diagrams. *Proceedings of the Twelfth International Conference on Information Systems*, New York.