Journal of the Association for Information Systems

Volume 6 | Issue 1 Article 2

1-29-2005

# An Ontological Analysis of Use Case Modeling Grammar

Gretchen Irwin
*Colorado State University, USA*, gretchen.irwin@colostate.edu

Daniel Turk
*Colorado State University, USA*, Dan.Turk@colostate.edu



Recommended Citation
Irwin, Gretchen and Turk, Daniel (2005) "An Ontological Analysis of Use Case Modeling Grammar," *Journal of the Association for Information Systems*, 6(1), .
DOI: 10.17705/1jais.00063
Available at: https://aisel.aisnet.org/jais/vol6/iss1/2

Journal of the Association for Information Systems

RESEARCH ARTICLE

# An Ontological Analysis of Use Case Modeling Grammar*

**Gretchen Irwin**
College of Business
Colorado State University
Gretchen.Irwin@colostate.edu

**Daniel Turk**
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 complete, consistent, and integrated set of use cases. We argue that there is a need for more rigor and consistency in the grammatical constructs used in use case modeling. Toward this end, we present a theoretically- and practice-based assessment of use case modeling constructs, and make recommendations for future research to improve and strengthen this technique.*

**Keywords**: use cases, Unified Modeling Language (UML), ontology, design theory

---

* Kalle Lyytinen was the accepting senior editor for this paper. Andreas Opdahl and Dave Tegarden were reviewers for this paper.

## Introduction

Use case models have, for many organizations, become a *de facto* model for representing the functional requirements, or externally observable behavior, of an information system (Dobing and Parsons, 2000). Simply put, a use case diagram shows "*who* does *what* with the system, for what *purpose"* (Malan and Bredemeyer, 2001). Use case diagrams have been adopted as part of the Unified Modeling Language (UML), an object-oriented modeling language standard of the Object Management Group (2003a, 2003b). These diagrams are based on a simple and intuitive grammar. They are often supplemented by use case specifications, which provide considerably more detail about the interactions between the system and its users in a structured, narrative-style format (e.g., Cockburn, 2001; Schneider and Winters, 2001).

The popularity, simplicity, and ease of reading use cases, however, belie the difficulty of *creating* them. The "perceived informality of use cases lulls people into a false sense of security" regarding the model's correctness and completeness (Berard, 1998: 9). Cockburn (1997) says, "Use cases are wonderful but confusing". A number of practitioners discuss the pitfalls of use cases observed on systems development projects (Bittner, 2000; Cockburn, 2001; Fowler, 1998; Lilly, 2000; Rosenberg and Scott, 2001; Achour et al., 1999). 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 proposed solutions. Two theoretical analyses of UML raise some concerns about use cases but do not address the issues most often cited in practice (Dobing and Parsons, 2000; Opdahl and Henderson-Sellers, 2002). Field studies generally confirm the observations from the practitioner literature, but provide little theoretical explanation for the problems or the proposed solutions (Achour et al., 1999; Hertzum, 2003; Regnell and Davidson, 1997).

We use information system (IS) design theory (Walls et al., 1992; Markus et al., 2002) to set the context for a rigorous evaluation of use case modeling grammar that explains the problems cited in practice and identifies issues in need of empirical investigation. An IS design theory includes three major components: (1) a set of goals, or user requirements; (2) a product or set of design artifacts hypothesized to meet the goals; and (3) a set of development practices to produce the artifacts that will achieve the goals. A design theory is *prescriptive* in that it describes *how* to create or produce the desired artifacts, which makes "the design process more tractable for developers by focusing their attention and restricting their options" (Markus et al., 2002: 180). Design theories also utilize *kernel theories*—theories from other disciplines—to inform the development practices and design artifacts prescribed by the theory, and to generate testable research hypotheses.

IS design theories vary in their scope and level of detail. For example, Walls et al. (1992) and Markus et al. (2002) propose design theories for the development of "vigilant" IS and "emergent knowledge" IS, respectively. These theories are comprehensive in that they cover the development of a particular type of IS from initiation through implementation. They also include fairly detailed design practices. The systems development life cycle (SDLC) is another comprehensive design theory, but it provides only general guidelines for the development of traditional transaction processing IS. Relational database theory (Codd, 1970), on the other hand, provides formal and detailed rules, but is much more narrowly focused that the other theories.

This paper focuses on use cases, which are artifacts prescribed by several IS design theories that emphasize an object-oriented, incremental, and iterative development process, such as the Unified Process (Jacobsen et al., 1999). Use case modeling is a mini-design theory in that it is a component of other design theories that encompass the full SDLC. The emphasis of this paper is on the design *product* rather than the design *process* or *practice*, because the grammatical constructs for building the product are well-articulated and better understood than the processes for creating the models. In design-theory terms (Walls et al., 1992), our emphasis is on the following.

- The primary *goal* of use case models, during systems analysis, is to create a conceptual model of the observable behavior of the system under discussion. The goal of a conceptual model is to faithfully represent the "real world" being modeled by the information system, which, in turn, supports communication between developers and users, improves analysts' understanding of a domain, and provides a starting point for designers (Wand and Weber, 2002).
- The design *artifact* under discussion is the grammar used to produce use case diagrams and use case specifications. This grammar is believed to meet the goal of conceptual modeling, primarily because it is a simple grammar that is easily understood by users, analysts, and designers.
- We use ontology, specifically Bunge, Wand, and Weber's ontology (Wand and Weber, 1990, 1995), as the *kernel theory* to evaluate the ability of the modeling grammar to meet the goal of use case modeling in systems analysis.
- We generate a set of *hypotheses* on use case modeling grammar based on the evaluation of the grammar using BWW ontology as our kernel theory.

Our analysis shows that many of the problems cited in practice correspond to ontological weaknesses in use case modeling grammar, and also indicates other areas that are theoretically problematic and worth empirical investigation. We also use the analysis to assess several extensions or variations on use case modeling that have been proposed in the literature.

## Use Case Models

Developers employ use case models primarily to capture the functional requirements of an information system by focusing on *usage situations,* the tasks that users want to accomplish with an information system. In this context, a use case model is a conceptual model that articulates the required behavior of a system in non-technical, implementation-independent terms. Figure 1 shows a use case diagram for an online order processing system. The figure illustrates the grammatical constructs in use case diagrams, which are specified in the UML version 1.5 (OMG, 2003a) and are defined in Table 1.[1]

Figure 1 shows that the *Web Customer* is a role played by external users of the online system who interact with it to achieve the goals *Place Order*, *Find Item*, and *Create My Account*. Each of these use cases represents one logical, user-oriented task that the

[1] We used UML version 1.5 (OMG 2003a) as the basis for defining use case diagramming constructs. Unless otherwise noted, the definitions in version 1.5 have been compared to and found to be consistent with definitions in the UML 2.0 (OMG 2003b).

system must support. Furthermore, Figure 1 states that in order to successfully accomplish *Place Order*, the user must also accomplish *Find Item* and *Pay for Order. Find Item* and *Pay for Order* represent sub-goals of *Place Order* (Cockburn, 1997). The *Pay for Order* use case requires interaction with another external party, *Payment Authorizer*, in order to accomplish its goal. Finally, Figure 1 states that under certain conditions, placing an order is extended by the functionality represented by *Create My Account.*

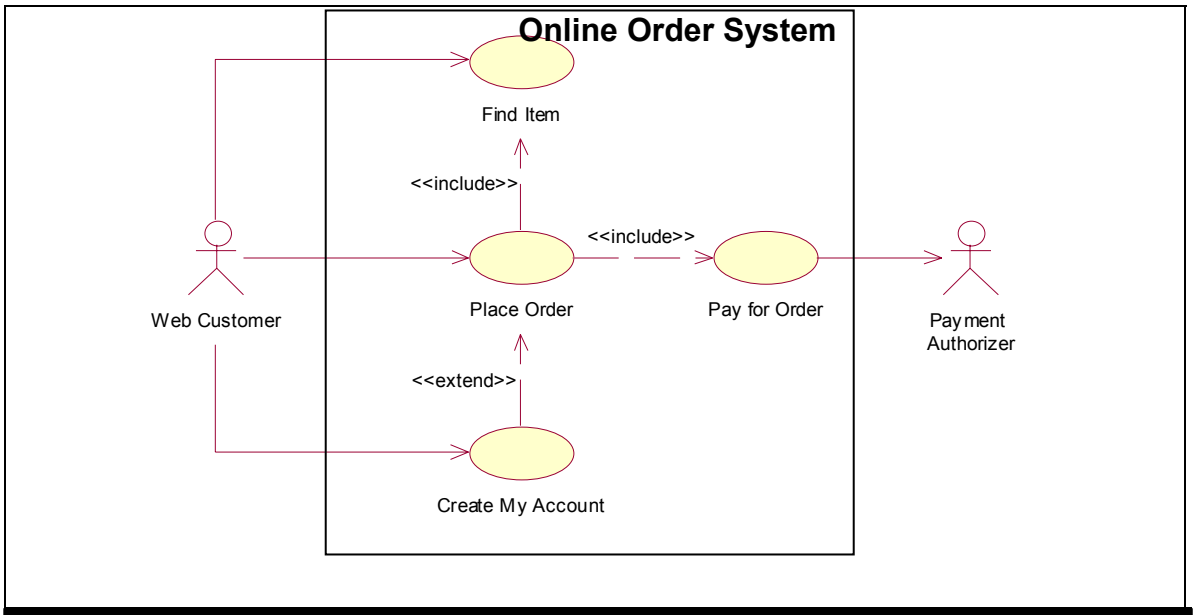
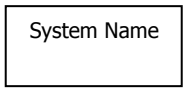



**Figure 1. Use Case Diagram for an Online Order System**

Use case diagrams are often supplemented by textual use case descriptions, which provide a more complete and detailed view of individual use cases. These descriptions are not a part of the UML, and there are many variations to choose from (e.g., Ambler, 2001; Cockburn, 2001; Schneider and Winters, 2001). Use case descriptions are closely related to scenarios, which have been used for over two decades in human-computer interaction and other disciplines (Jarke et al., 1998; Antón and Potts, 1998). Use cases and scenarios have been applied in so many ways that authors have developed classification schemes in order to clarify which variant is under discussion (Cockburn, 1997; Achour et al., 1999; Antón and Potts, 1998).

Scenarios and use cases are both descriptions of "concrete system behavior," usually for systems "that do not yet exist in a tangible, usable form" (Antón and Potts, 1998: 219). Beyond that, scenarios and use cases can be differentiated by purpose, content, life cycle, and form, among other dimensions (Achour et al., 1999; Antón and Potts, 1998). *Purpose* refers to why the use cases are created, and includes requirements specification, user interface design, test case design, or business process modeling. *Content* refers to what is described in the use case and at what level of abstraction. For instance, if the purpose is requirements specification, the content will emphasize the interactions between users and the system. The system may be treated as a black box

**Table 1. Grammatical Constructs for Use Case Diagrams**

| Construct | Representation | Definition(s) |
|---|---|---|
| System |  | A system is a "top-level subsystem in a model" or an "organized array of elements functioning as a unit" (OMG, 2003a: 16). |
| Use Case |  | A "coherent unit of functionality provided by a system" (OMG, 2003a: 3-96). A description of a "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 (OMG, 2003a: 3-97). An "entity (human or non-human) external to the system under development, that communicates with the system in order to achieve certain goals" (Regnell and Davidson, 1997: 1). |
| Association |  | A relationship that represents "the participation of an actor in a use case" (OMG, 2003a: 3-97), or communication between the actor and the use case. |
| Include |  | A relationship between two use cases that shows that an instance of one use case (the base use case) will also contain the behavior specified by another use case (the included use case) (OMG, 2003a). |
| Extend |  | A relationship between two use cases that shows that an instance of one use case (the base use case) "may be augmented (subject to specific conditions…) by the behavior" specified by another use case (the extending use case) (OMG, 2003a: 3-98). The extending use case specifies "alternative, conditional or exceptional courses of interaction that can alter or extend the main flow of events embodied in some base case" (Constantine and Lockwood, 2000: 20). |
| General-ization |  | A relationship between two use cases or between two actors. "A generalization relationship from use case C to use case D indicates that C is a specialization of D" (OMG, 2003a: 3-98). "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, 2003a: 3-99), in other words the child actor (A) can play the same roles as the parent actor (B) (p. 2-131). |

whose internal structure is hidden, or as a white box whose structure is at least partially visible. Content may also be described in instance-specific terms, using real user names (e.g., John), or in more abstract terms, using entity type names (e.g., customer). *Life cycle* refers to whether the scenarios are transient or persistent. If scenarios are persistent, then they must be maintained and managed with some degree of traceability across the systems development life cycle. Finally, *form* refers to the format, style, and degree of formality of the scenario. The most common form is a semi-formal narrative description based on a template or table structure (Achour et al., 1999).

We focus on use cases for the purpose of conceptual modeling during systems analysis. In this context, the system is treated as a black box, the use cases are expected to be persistent across the life cycle, and the form is semi-structured, textual, and template-based. We also assume each use case is written in a non-instance-specific manner. This last point is consistent with the main distinction between use cases and scenarios: one use case "bundles many possible scenarios together, each of which represents a single path through the use case" (Antón and Potts, 1998: 228). In this context, we found most of the published templates to include the following constructs: name, actor(s), normal flow of events, alternate flows, preconditions, and postconditions. We define these constructs in Table 2 and illustrate them with an example in Figure 2. We consider these constructs to be part of the use case specification grammar.

## Problems in Practice

Given the many variations of use cases advocated in the literature, and the relative infancy of use cases as a standard component of systems analysis, it is not surprising that there is some confusion and difficulty in using this technique in practice. The problems cited in the practitioner and academic literature fall into three categories: (1) specifying which system is being modeled; (2) determining the level of detail to include in the model; and (3) transitioning to and tracing between models. Each of these is described below.

Regarding the system being modeled:

- Use case diagrams often lack a clearly defined system boundary (Lilly, 2000), which in turn leads to mis-specified actors and use cases that mix business processes, system requirements, and implementation details.

Regarding the level of detail:

- "Very few software practitioners have an adequate sense of the level of detail that should be associated with a given use case." Some use cases are so ambiguous and abstract that they are "practically useless" (Korson, 1998), while others are so detailed that "even the slightest change to requirements will cause them to be rewritten" (Berard, 1998: 10).
- Among those who create use case models for conceptual modeling and functional requirements specification, there are inconsistencies over whether to include technology details. Several authors reference screen designs and other technology details in their use cases, while others write technology-free use cases (Dobing and Parsons, 2000).

**Table 2. Grammatical Constructs for Use Case Specifications**

| 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 & Postconditions | Preconditions are constraints assumed to be true before the use case begins. Postconditions are conditions guaranteed to be true after the use case finished. |

Use Case: Place Order

Actor: Web Customer

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

Preconditions: Customer is at the web site and wants to order one or more items.

Normal Flow of Events:

1. The use case begins when the Customer chooses an item and adds it to the shopping cart.
2. The System displays the shopping cart with the customer's name, the added item 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 chooses to check out.
6. The System displays the applicable sales taxes, shipping charges, and 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:

-- Any point between steps 1 and 6 (inclusive), the Customer may cancel the order, and the use case ends.

-- Any point between steps 1 and 6 (inclusive), the Customer may save his/her shopping cart, 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, he/she enters a quantity of zero, and the System displays the updated shopping cart and subtotal.

2c. If the System does not recognize the Customer, the System asks the Customer to initiate the Create My Account use case.

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

- Analysts tend to get "lost in levels" when writing use cases because the goals and interactions of a use case can be "unfolded into finer- and finer-grained goals and interactions" (Cockburn, 2001: 61).
- "Scenario explosion" occurs when analysts get bogged down in the possible variations of a use case (Cockburn, 2001). Specifying *included* and *extending* use cases may help, but this often leads to "abuse by decomposition" as the number of very fine-grained included and extending use cases increases (Fowler, 1998). In

either situation, these projects produce an unwieldy amount of use case documentation (Korson, 1998).
- Users and analysts "like to apply the black box – white box principle… [however,] in many projects, different levels of detail and different concerns are mixed up in the same scenario's description. The issue covers the definition of levels, as well as ensuring that these levels are respected when authoring scenarios" (Achour et al., 1999: 11).

Regarding the transition to and traceability between models:

- There is a wide semantic gap between a textual use case description and other, more formal representations of system behavior, such as the UML Sequence Diagram (Antón and Potts, 1998). This gap makes it challenging to maintain traceability between different models of the same scenarios—or between the same scenario at different levels of detail (p. 229).
- It is unclear whether or to what extent use cases provide a good basis from which to move forward to other conceptual models, particularly the UML class diagram (Dobing and Parsons, 2000). "The lack of integration between use cases and class models raises questions about the value of use cases in an object-oriented modeling approach" (p. 32).
- It is unclear how to structure a large collection of use cases and their relationships to other analysis and design models, and to support changes to the models over time. Practitioners are concerned that the value of use cases may be offset by the cost of maintaining and integrating use cases on large scale projects (Achour et al., 1999; Hertzum, 2003).

Most of the problems cited above emphasize finding the "right" level of abstraction for a given system boundary, and organizing use cases at different levels of abstraction in a way that supports traceability and integration within and across models. Many authors recommend creating use cases at multiple levels of abstraction, including, for example, business-level use cases and system requirements-level use cases (Fowler, 1998; Korson, 1998; Achour et al., 1999). At the same time, the authors note the difficulty in organizing and managing multiple levels within the confines of the current use case modeling grammar and techniques. This difficulty stems, in part, from the fact that use cases were originally used to model only software application boundaries, and were not intended to model various system boundaries in an integrated fashion.

Figure 3 depicts two system boundaries in one diagram. The figure uses the UML notation, but does not conform to the UML rules for use case diagramming, and so is not a valid use case diagram. The business level is represented by system boundary (a) in the figure. If the Medical Office is the system being modeled, then a use case corresponds to a business process, and the primary actors are those external to the business, such as Patient. If the organization is treated as a white or transparent box, then we might also show the actors *inside* the organization that participate to accomplish the external actor's goal, such as the receptionist helping to fulfill the patient's goal of making an appointment. If the system under discussion is a particular application, such as the appointment scheduling application shown as boundary (b), then a use case corresponds to a specific user task, and actors may be internal or external to the business depending on whether they directly interact with the application.

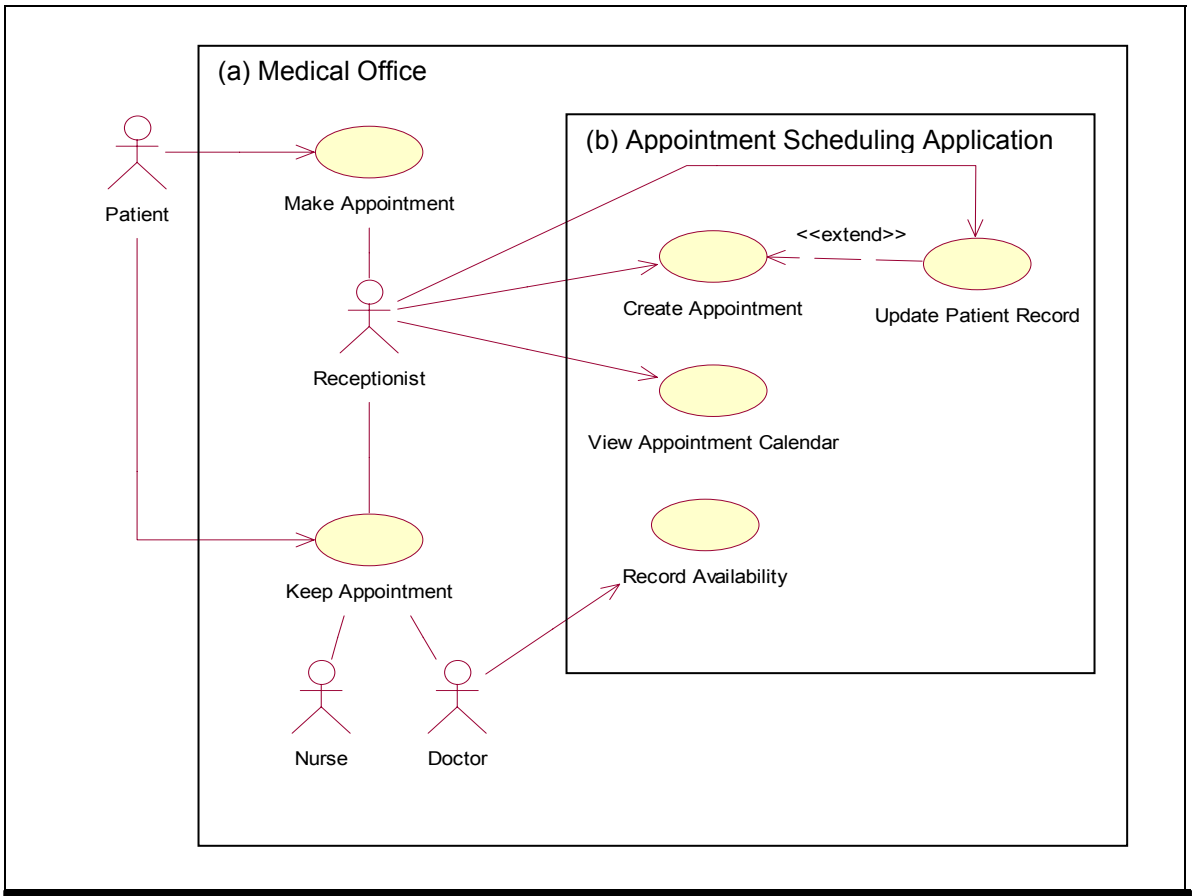



**Figure 3. Business- and Application- Level Boundaries**

## Ontological Framework

Conceptual models capture knowledge about both static and dynamic phenomena in the domain of interest (Wand and Weber, 2002). We want to evaluate the use case modeling grammar to determine the extent to which the grammar is capable of modeling the real-world phenomena for which it was designed or is most often used to represent. The question, then, is which kernel theory is most appropriate to evaluate a conceptual modeling grammar?

There are different theoretical perspectives that address important areas of conceptual modeling and have been used to evaluate conceptual modeling grammars and techniques (see Wand et al., 1995 and Wand and Weber, 2002 for a review). For instance, Parsons (1996) uses classification theory from cognitive science to identify the constructs and rules that a conceptual modeling grammar needs in order to be consistent with the way humans represent and organize knowledge about a domain. Wand and Weber (1990) use ontology to evaluate how well conceptual modeling grammars support the representation of the semantics of an application domain. They specifically use and extend Bunge's ontology (1977, 1979) to create the BWW (Bunge-Wand-Weber) model. These theories and others have been shown to be complementary rather than competing (Wand et al., 1995).

We base our analysis on the BWW model because it is well formalized in the context of systems and has been successfully used to evaluate several conceptual modeling grammars, including NIAM (Weber and Zhang, 1991), the object model (Wand, 1996), OML (Opdahl and Henderson-Sellers, 2001), entity-relationship and data flow diagrams (Wand et al., 1999), and the UML (Opdahl and Henderson-Sellers, 2002). The BWW model draws upon prior work in ontology, which is a "well-established theoretical domain within philosophy dealing with models of reality, that is, the nature of the real world" (Rosemann and Green, 2002: 78). Bunge describes his ontology as "a "hypothetical-deductive system rather than a stray opinion or an unsystematic set of opinions. In particular, an ontological theory is a theory that contains and inter-relates ontological categories… [e.g., thing, property, law, change, space-time]… and invites the formulation of hypotheses" (Bunge, 1977: 11-12).

The BWW ontology articulates the constructs that a conceptual modeling grammar should have in order to "create a 'faithful' representation" of real-world domains (Wand and Weber, 2002: 366), such as banking, university admissions, product sales, or customer service.[2] The BWW model, in brief, 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. We describe the fundamental constructs of the BWW ontology in Table 3 below. For a more detailed explanation of these constructs, see Wand (1996), Wand and Weber, (1990), and Rosemann and Green (2002).

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; i.e., the grammatical construct has multiple ontological meanings. For example, if one construct in the use case grammar has multiple ontological meanings, then the specific ontological meaning intended by the use case author may not be clearly conveyed to the stakeholder using the model. This may lead to communication breakdowns, which, in turn, may lead to requirements definition or design problems.
2. *Construct redundancy* occurs when one ontological construct maps to more than one grammatical construct; i.e., two or more grammatical constructs have the same ontological meaning. This redundancy may not be a problem if the specific nuances of the grammatical constructs are well understood. Construct redundancy may cause problems if the parties involved infer that different grammatical constructs have different ontological meanings.

**Table 3. BWW Ontological Constructs**

| Construct | Definition |
|---|---|
| Things | This is the elementary construct of the ontology. The real world is made up of things (e.g., person, product, company). A *composite thing* consists of two or more component things that are related (Wand, 1996: 282). A *class* or *kind* is a set of things that possess a common set of properties. A class may be a *subclass* or *subkind* if it has at least two properties—one from its superkind and one that is a property only of the subkind (Weber and Zhang, 1991). |
| Properties of things | Things have properties. Properties can be *mutual* to other things (e.g., a person is employed by a company). A *law* is a property that constrains the values of other properties or that specifies relationships among properties (e.g., such as a law connecting salary and tenure). The values of the properties of a thing at a specific point in time comprise the *state* of the thing (Wand, 1996: 282). |
| Dynamics of a thing | "…every change is tied to things and every thing changes" (Wand, 1996: 282). For a thing to change means that its state must change, or, in other words, one or more of its property values must change. Every change can be modeled as an *event* that transforms a thing from one state to another. Transformations are described in terms of *transition laws* which define the permissible states of a thing. |
| Systems | A system is a composite thing based on the notion of *interaction*. The components of a system interact, which means that "at least one of them can affect the states the other traverses in time. The ability of two things to interact is a mutual property of the two things... The *composition* of a system is the set of components of the system… The *environment* of the system comprises things not in the system that interact with components of the system. The *structure* of the system is the set of interactions that exist among components in the system and between components of the system and things in the environment" (Wand, 1996: 282). |
| Decomposition | A decomposition is "a set of subsystems of a system where…every element in the composition of the system is included in at least one of the subsystems in the set…and…each element in the structure of the system is included in at least one of the subsystems in the set" (Wand and Weber, 1990: 1287). A good decomposition is possible only if the grammar allows the following to be modeled: "For a given set of external (input) events at the system level, all induced events in every subsystem (which includes the system) must be specified" as events (Weber and Zhang, 1991: 76) |

[2] There are other ontologies aside from Bunge's. However, Wand and Weber (1995) chose to extend Bunge's because it is closely relates to system concepts that are used in information systems development and because it is more mature and better developed than other ontologies.

3. *Construct excess* occurs when a grammatical construct has no counterpart in the ontology. Construct excess may point to a grammatical construct that is meaningless, or to a void in the ontology. If the construct is meaningless, then it should be replaced with a grammatical construct that has meaning in the ontology. If the ontology is incomplete, then it should be expanded to cover the grammatical construct.
4. *Construct deficiency* occurs where an ontological construct has no corresponding grammatical construct. Construct deficiency leads to the problem of not being able to express certain information in the grammar. In this case, the grammar must be extended so that these things may be expressed.

We use the BWW model to analyze the grammar for use case diagrams and use case specifications as defined in Tables 1 and 2, respectively. Our analysis builds on the work of Opdahl and Henderson-Sellers (2002), who used the BWW model to make a broad evaluation of the UML. One of the contributions of their work was to provide a clear ontological definition of UML grammatical constructs. Our evaluation complements and extends their work in two ways. First, we delve more deeply into use case diagramming constructs. Second, we include the constructs for use case specifications, which, though not part of UML, are often used to supplement the diagrams. Our evaluation does not directly include other UML constructs, such as those included in class diagrams, except to the extent that some use case constructs are also used in other UML diagrams. It is important to note that use case models are intentionally deficient in some respects because they are not meant to model all aspects of a system. We do not fault use case diagrams for being deficient in ontological areas that they are not intended to model and that other models may cover. Instead, we focus on potential deficiencies that directly relate to the purpose of use case models.

## Ontological Analysis

Table 4 provides a summary of the comparison of BWW constructs to use case modeling constructs. The following sections discuss the comparisons in Table 4 and identify the ontological discrepancies we found.

### *The use case construct*

The UML describes a use case as “a piece of behavior” or “a service the entity [system] provides to its users” (OMG, 2003b: 2-136) and states that use case diagrams are “primarily used to define the behavior of an entity, like a system or a subsystem, without specifying its internal structure” (ibid: 2-129). More specifically, a use case is the “specification of a *sequence of actions*…that a system (or other entity) can perform” (OMG, 2003a: 17, emphasis added). Clearly, the use case construct represents a dynamic rather than structural aspect of the system.

Opdahl and Henderson-Sellers (2002) describe the correspondence between use cases and BWW dynamic constructs as follows: an instance of a use case represents the performance of a sequence of actions, and each of these actions is an ontological

**Table 4. Comparison of BWW Ontological Constructs and Use Case Constructs**

| BWW Construct | Corresponding Use Case Grammatical Construct(s) | Ontological Discrepancy? |
|---|---|---|
| **Thing and kind (class)** | The **system** construct in a use case diagram represents a *BWW composite thing*, but is "optional" and not given much attention in the UML specification. | None |
| | **Actor** is defined as a type of classifier in the UML metamodel, which corresponds to a BWW kind, but as a role or facet of a thing in the UML specification, which corresponds to a *BWW property*. | Construct overload |
| | Generalization between actors corresponds to *BWW* **kind/subkind** relationship but violates the "a kind of" semantics that applies to subkinds and kinds. Generalization between use cases (BWW processes) conflicts with BWW generalization which applies to kinds, not processes. | Construct overload &/or excess |
| **Property of a thing** | **Association** in class diagrams corresponds to a *BWW mutual property* of two things (kinds). In use case diagrams, association corresponds to a *binding mutual property* of an external entity and the system. But it is drawn as a link between a BWW *kind or property* (actor) and a BWW *process* (use case), which is not consistent with BWW properties. It may also correspond to a transformation law. | Construct overload |
| **Dynamics of a thing** | **Use case** is described as a set of actions that changes the state of the system, which corresponds to a *BWW process*, but the UML metamodel defines a use case as a type of classifier, which corresponds to a *BWW kind.* | Construct overload |
| | **Normal flows, alternate flows,** and **postconditions** correspond to BWW *transition law properties*. | None |
| | **Preconditions** represent BWW *state law properties*. | None |
| | **Include** & **extend** relationships may represent a form of BWW aggregation, or they may represent constructs to support reusability that have no counterpart in the BWW model. | Construct overload or excess |
| **Systems** | **System environment** consists of actors, outside the system boundary, that interact with the system. | None |
| | **System structure** would be shown by the associations between actors and the components system, and between components of the system. However, it is unclear how to model system components and relationships between actors and components in use case modeling grammar. | Construct deficiency |
| **Decompo-sition** | **System composition**, **level structure** – these concepts appear to be entirely absent. | Construct deficiency |

event that changes the state of the system or subsystem. A sequence of these events constitutes a *BWW process*. In UML, each use case *instance* represents a BWW-process in the proposed system thing and a use case "represents a group of such BWW-processes" (p. 50). The 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 of the process. Thus, Opdahl and Henderson-Sellers. (2002) find that the UML use case construct has a solid correspondence with BWW dynamic constructs.

However, the UML metamodel defines a use case as a thing rather than as a behavioral property (transformation law) of a system thing. Figure 4 shows a subset of the UML 2.0 metamodel relevant to our discussion. A use case is defined as a subclass of Classifier, and a classifier is an element that has behavioral and structural features and that may participate in relationships. We have no argument with the use case construct being defined as a thing in the sense that it is an element of the modeling grammar (i.e., a subkind of Element in Figure 4). However, the fact that the use case is further defined as a kind of thing with structural and behavioral features, rather than as a behavioral feature of a system thing, may be ontologically incorrect.

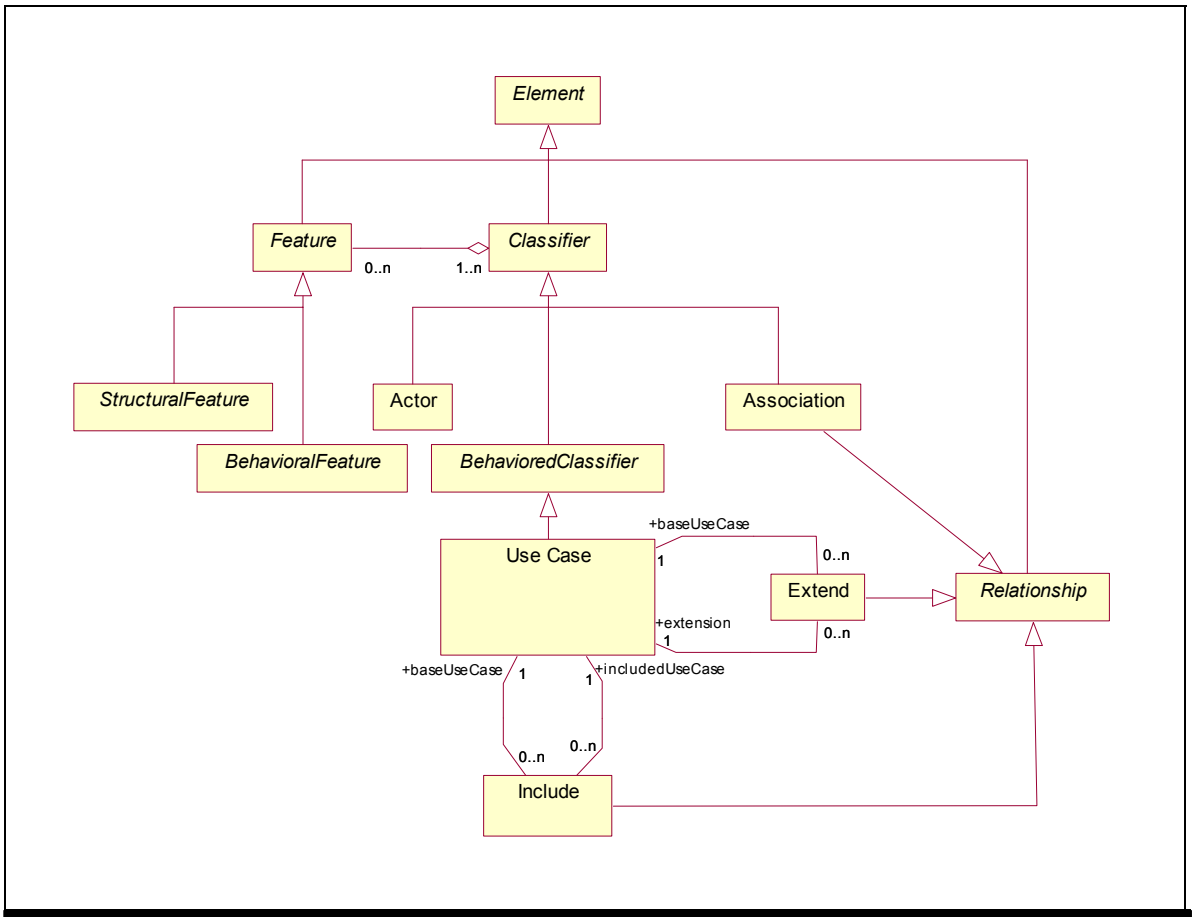



**Figure 4. Class Diagram of (part of) the UML Metamodel**

Furthermore, the UML states that, "instances of use cases and instances of actors interact when the services of the entity are used" (OMG, 2003b: 2-129). This implies that use cases and actors are things in the sense that they both send and receive messages. But a use case itself does not send or receive messages—objects within the system thing send and receive messages.[3]

The use case construct is ontologically overloaded because it is defined both as a thing and as a behavior of a thing, which are two ontologically separate constructs. This may explain some of the problems noted in practice, particularly difficulties in correctly specifying use cases (Bittner, 2000) and mapping from them to class diagrams that focus exclusively on things and their properties (Dobing and Parsons, 2000).

### *The actor construct*

In some respects, the UML actor corresponds to a *kind* 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. The UML specification states that actors represent parties outside of the system that interact with it, and that an instance of an actor "is a specific user" (OMG, 2003b: 2-135). The UML metamodel shown in Figure 4 defines actor as a kind of classifier (BWW kind) that has structural and behavioral properties. Other conceptual modeling languages, such as the OML (Opdahl and Henderson-Sellers, 2001), clearly define actor as a kind of external object that corresponds to a BWW kind.

On the other hand, an argument can be made that an actor represents a property of a thing, rather than a thing in its own right. The UML states that an actor represents a *coherent set of roles* that users can play with respect to the system, and, further, that an actor [instance] is "not necessarily a specific physical entity but merely a particular facet (i.e., 'role') of some entity that is relevant to…its associated use cases" (OMG, 2003a: 512). Thus, role may be an attribute of an entity in the sense that, for example, "project manager" is an attribute of employee. The term "role" has been defined elsewhere as "a *relationship* between a user and a system…[that] is defined by a set of characteristic needs, interests, expectations, behaviors, and responsibilities" (Constantine and Lockwood, 2000: 3, emphasis added). From this perspective, a role is a BWW *mutual property* of an external thing and a system thing. For example, "member" is a mutual property of a person and a health club because it depends on the existence (and interaction) of two things, a person and the health club (Wand et al., 1999). The role construct in other conceptual modeling grammars has been compared to a BWW property (Weber and Zhang, 1991).

---

[3] Every UML construct is a kind of thing because they are all elements of the modeling language. Every construct in UML is thus a subclass of the abstract superclass Element. Our point is in how various UML constructs are specialized from Element. Some constructs are clearly ontological subclasses of Classifier, such as Class. Others are not kinds of classifier, such as relationships and features. It is not clear why the Use Case construct is defined as a subclass of Classifier, rather than as a different kind of Element that more closely reflects its ontological meaning, such as BehavioralFeature.

The UML actor is an overloaded construct because the UML specification defines it in ways that represent both BWW kinds and BWW properties. This may explain some of the problems noted in practice with creating class diagrams from use case models and tracing between and integrating models (Antón and Potts, 1998; Dobing and Parsons, 2000).

For example, consider an order processing system. Should the actors be named for the role they play or the entity they instantiate? The actors might be named customer, sales clerk, and sales manager. Job titles such as "clerk" and "manager" are commonly used in published use case examples (e.g., Dennis et al., 2002; Kenworthy, 1997; OMG, 2001; Rosenberg and Scott, 2001; Schneider and Winters, 2001). If all of these actors can initiate the *Place Order* use case, we could show three separate actors, each with an association to *Place Order*. However, a simpler and more precise model would show that there is really one role—Order Taker—that initiates this use case. Customers, salespeople, and sales managers are entities that can play the role of Order Taker. However, this more precise way of speaking is "nonstandard in the use case world" (Cockburn, 2001: 57).

The naming of actors on a use case diagram may also influence the identification of classes on the class diagram. Use case models are often used as a starting point for creating the class diagram, and many authors argue that use case models should, in fact, drive the entire systems development effort (Jacobsen et al., 1999). If actors are perceived as things, the tendency may be to identify them as candidate classes. If, however, they are perceived as roles, they may be modeled as attributes or relationships (i.e., BWW properties). For example, the use case diagram in Figure 5 shows a Project Manager actor. If project manager is a thing, then it would be a good candidate class for a class diagram. But if it is a role, it might be better modeled as a "manages" relationship between an employee (thing) and a project (thing), or as the value of a role attribute of an employee.

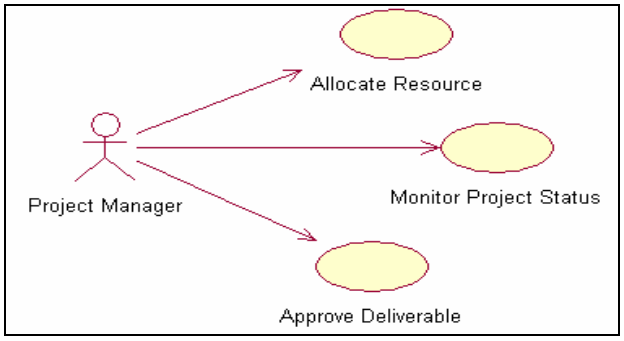



**Figure 5. Actor as Thing or a Role?**

### *The system construct*

A use case diagram is a model of one thing, namely, a *BWW system thing* (or a component of a system thing). The system in a use case diagram may represent an organizational system (e.g., a department or division) or a software application, or something in between, as we showed in Figure 3. In UML, 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. The UML definition of

system is consistent with the BWW definition, and thus there is technically no ontological discrepancy.

However, the UML states that the system construct is *optional* and downplays its importance relative to all other use case modeling constructs (OMG, 2003b, OMG, 2003a). There is very little discussion of the semantics or use of the system construct in any of the use case sections of the UML specifications. Given that the UML does not emphasize the importance of specifying the system boundary, it is not surprising that many published examples of use case diagrams omit the system construct (e.g., Rosenberg and Scott, 2001). In other cases, it is shown but poorly defined (Lilly, 2000).
As an example, consider the diagram in Figure 6 of a patient appointment system (Dennis et al., 2002). According to the authors' description, patients interact with the receptionist to make appointments, and presumably, doctors interact directly with the software system to record their availability for appointments. The system is defined as the "Appointment System." This might refer to either the *software application* for appointment scheduling or the larger *information system* that includes the application. If the system boundary is the software application, then the patient actor should not be shown. The patient does not directly interact with the application. Instead, the receptionist or appointment scheduler, who acts on behalf of the patient, should be shown as the primary actor of the *Make Appointment* use case.

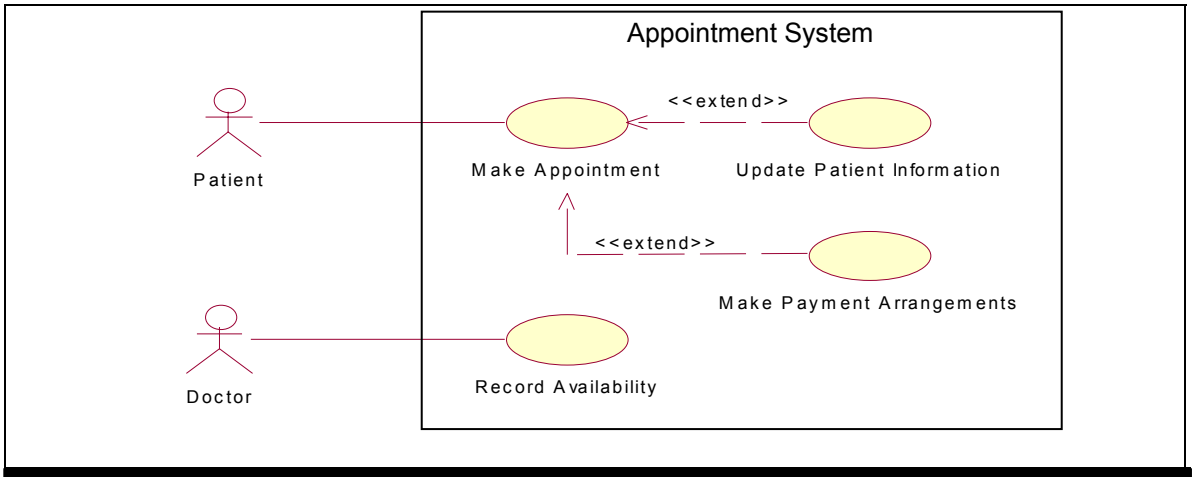



**Figure 6. Use Case Diagram with Poorly Defined System Boundary**

On the other hand, if the system boundary is the appointment scheduling *information system*, there is a different problem. An information system is a type of work system, which in turn, is a system "*in which* human participants and/or machines perform a business process using information, technology, and other resources to produce products and/or services *for internal or external customers*" (Alter, 2002: 6, emphasis added). According to this definition, the receptionist and doctors are participants *in* the system, interacting with the application and other resources to provide a service *for* the patient, who is the external customer of the system. Thus, if the use case diagram models the information system, it should *not* show the receptionist or doctor actor, because this would violate the definition of an actor as an entity in the system's environment, and the UML does not support "internal" actors. The patient is the appropriate primary actor, because the patient is the customer being served by the information system.

Figure 6 illustrates what Lilly refers to as the most common use case pitfall observed in practice—an undefined or inconstant system boundary (Lilly, 2000). The problem stems, in part, from the fact that use case diagrams were first used to model software applications, as black boxes, so that the system boundary was clear, and actors were the direct users of the software. But when use case diagrams are used to model information systems or organizational systems, the tendency seems to be to treat the system as a white box and show participants that are both internal and external to the system. The use case modeling grammar does not permit internal actors, since that would violate the definition of an actor as something that is in the *environment* of the system.

### *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 of thing. 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; Wand et al., 1999). For example, an employee *works for* a department, a student *enrolls in* a class, or a child is *legally dependent on* a parent.

In a use case diagram, an association represents a specific type of relationship; namely that an actor *initiates* a use case, or that an actor *interacts with* the system to accomplish the goal of the use case. Thus, the UML association construct has broad semantics when used in a class diagram and narrow semantics when used in a use case diagram. Association in use case diagrams corresponds to a *BWW binding mutual property.* A binding mutual property depends on two things (the actor and the system) and "implies that some changes in one thing [the actor] are related to… changes in the other thing" (Wand et al., 1999: 503). For example, if the customer actor interacts with the Place Order use case, the state of both the customer and the system will be changed (e.g., the customer has a balance owed, and the system has a new order to fill).

The fact that association in use case models has narrower semantics (i.e., a binding mutual property rather than a nonbinding mutual property) than in class diagrams is an example of ontological excess that may be a source of ontological ambiguity. However, the focus of use case diagrams is on who interacts with the system and for what purposes. Thus, we believe the distinct semantics of association in the use case modeling context can be fairly easily clarified.

The more problematic aspect of the association construct in use case diagrams is that the construct *overloads* the ontological definition for class diagrams. In class diagrams, an association links two kinds of thing, such as an employee class and a department class. In use case diagrams, association is drawn as a link between an actor and a use case. In the UML metamodel, both actors and use cases are treated as kinds of thing because they are subkinds of classifier. As stated earlier, however, the UML specifications define actor and use case in ways that are ontologically ambiguous and conflicting. If association in use case diagrams is viewed as a link between a BWW property and a BWW process—a view that is more consistent with the UML definitions of these constructs—then this overloads the definition of association in class diagrams.

In Figure 5, the precise semantics of the association to *Allocate Resource* is as follows: an entity (BWW thing), in the role of *Project Manager* (BWW property of a thing), interacts with the *system* (BWW thing) in a way that is specified by the *Allocate Resource* use case (BWW process). These semantics are more consistent with the definition of a BWW *transformation law* than with a mutual property. A transformation law is a property of a thing—in this case, the system—that constrains the events and transformations that can occur to a system. In use case diagrams, the association construct represents a constraint on who (e.g., the project manager) may invoke or initiate a particular change to the system (e.g., allocating a resource to a project).

The ontological ambiguity surrounding the association construct does not correspond directly to the problems cited in practice. It may be that creators and users of use cases and UML models clearly understand the different meanings of the association construct depending on the diagram in which the construct is used. If, however, these differences are not clear, we would expect problems to arise in the interpretation of and/or mapping between use case models, class diagrams, and other UML models.

### *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, 2003a: 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 kind/subkind relationship, and is a well-known and commonly used construct in conceptual modeling grammars such as Entity-Relationship Diagrams (e.g., Elmasri and Navathe, 1994).

In UML, generalization may be applied to actors and use cases as well as to classes (OMG, 2003a: 3-86). Generalization between use cases (ontological processes) is not clearly consistent with BWW generalization between kinds and subkinds. If use cases are conceptualized as things, as in the UML metamodel, then generalization is fine. If, as we argued earlier, use cases are conceptualized as processes, then the ontological meaning is ambiguous. A subkind, by definition, is described by all of the attributes of the superkind plus one or more attributes specific to the subkind. But according to BWW, processes, or transformation laws, are not things in and of themselves and thus do not themselves have properties or attributes (Wand et al., 1999). Instead, a process or transformation law is an algorithm that changes the state of one or more things. This ontological perspective differs from an object-oriented perspective, where we often create process-type objects with behavior and attributes.

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, 2003a: 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. The ontological ambiguity about actors as things versus properties compounds the problem.

Figure 7 shows an example from the UML 2.0 specification (OMG, 2003a). The Customer actor is a generalization of the Administrator actor because the administrator can communicate with the same use cases as the customer. However, we would not say

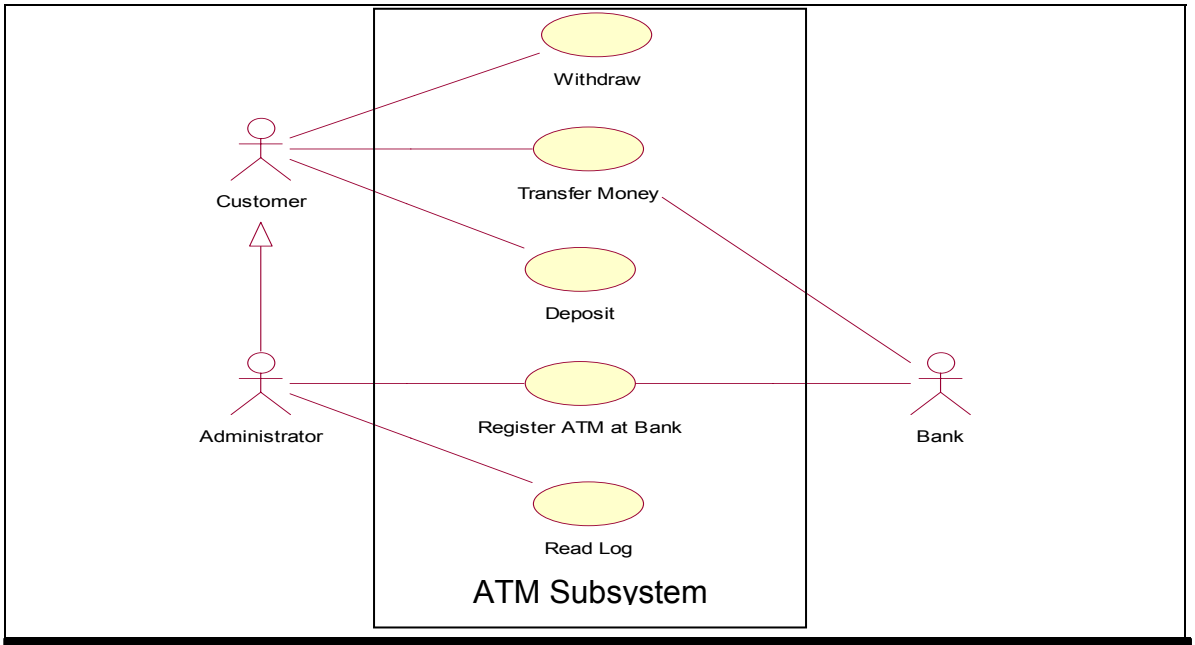



**Figure 7. Actor Specialization (from OMG, 2003a: Figure 404, p.520)**

that an administrator is semantically *a kind of* customer. Furthermore, there may be attributes of customer (e.g., account number, balance owed) that do not apply to administrators. This would clearly violate the ontological definition of a subkind. We may want to say that an administrator may *play the role of* or *act on behalf of* a customer, and in that capacity, initiates the *Transfer* use case, but this is not the same as generalization.

The UML generalization construct is ontologically overloaded. In class diagrams, it corresponds to a BWW *kind/subkind relationship*. In use case diagrams, it represents a relationship between actors or use cases that may or may not be equivalent to the kind/subkind semantics, depending on whether actors and use cases are viewed as kinds of a thing. Generalization in use case diagrams may correspond 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). UML generalization between actors states a constraint on which roles may be related to, or played by, other roles.

The ontological problems with generalization in use case models do not directly correspond to the problems cited from practice. It may be that the different interpretations of generalization in use case models are subtle and thus not problematic. Or, the generalization construct may not be as frequently used in practice as other use case constructs. However, to the extent that generalization between actors *is* used, we would expect to see some difficulty in mapping to class diagrams, where generalization is a commonly used construct with clear semantics. And we would expect users of these models to have difficulty interpreting the different meanings of generalization across these diagrams.

### *Constructs in the use case specification*

The text-based use case specifications define certain properties of the system. In particular, the **Normal Flow of Events** is a grammatical construct that specifies one sequence of permissible actions or transformations that may occur as part of the use case process. The Normal Flow represents the transformations that occur when everything goes well. The **Alternate Flows** construct represents other permissible transformations that may occur at different branching points from the Normal Flow (e.g., when something goes wrong in the Normal Flow). Both of these constructs represent BWW-*transformation law* properties of a system thing.

We find no ontological discrepancies with regard to these constructs. We note, however, that there are some practical problems because these constructs are informally represented and defined as compared to other use case modeling constructs. Use case specification authors do not clearly indicate how to represent the transformation laws, other than with general guidelines such as, "use 3 to 9 steps" in the Normal Flow (Cockburn, 2001) or "write present tense verb phrases in active voice" (Rosenberg and Scott, 2001). Constantine and Lockwood (2000) note that the 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 more formally represent these aspects of a system process (e.g., Activity Diagrams, Sequence Diagrams).

**Preconditions** and **postconditions** are two other constructs typically included in use case specifications. Preconditions correspond to BWW *state laws* because they specify the stable states of the system prior to execution of a use case (Opdahl and Henderson-Sellers, 2001; Opdahl and Henderson-Sellers, 2002). Postconditions correspond to BWW *transformation laws* that govern the allowed changes of system state (Opdahl and Henderson-Sellers, 2001; Opdahl and Henderson-Sellers, 2002). We did not identify any ontological discrepancies with respect to pre- or postconditions, except again, to note that the representation of these constructs in the text-based specifications is fairly informal.

In practice, problems have been noted with respect to writing use cases, although ontologically the constructs in these specifications are fairly straightforward. Most of the problems revolve around how detailed and comprehensive the normal and alternate flows should be (Korson, 1998; Berard, 1998; Cockburn, 2001), and on when to decompose one specification into included or extending use cases (Fowler, 1998; Bittner, 2000).

### *The Include and Extend Constructs*

The UML <<extend>> and <<include>> constructs describe two relationships between use cases. The <<extend>> construct "consists of a condition, which must be fulfilled if the extension is to take place, and a sequence of references to extension points in the base use case where the additional behavior fragments are to be inserted" (OMG, 2003b: 2-13). This definition corresponds to a *BWW transformation law* in the sense that it specifies the circumstances in which the system will undergo a particular state change. The UML also explains that an extending use case "typically defines behavior that may not necessarily be meaningful by itself" (OMG, 2003a: 515). In other words, an extending use case may not be a complete set of actions that provides an "observable result of value" to an actor. Other authors argue that included use cases are also

fragments that do not represent users' goals (Lee and Xue, 1999; Constantine and Lockwood, 2000). Thus, included and extending use cases may not be "pure" use cases and may be better understood as BWW transformation laws, in the same sense that an Alternate Flow of Events is a transformation law.

However, some authors argue that included and extending use cases are used primarily to facilitate reuse (Constantine and Lockwood, 2000; Lee and Xue, 1999; OMG, 2003a). If this is the case, then <<include>> and <<extend>> may not be needed for conceptual modeling purposes and would be examples of *construct excess* in that they have no explicit counterpart in the BWW ontology.

Another interpretation of <<include>> and <<extend>> is that they specify aggregation relationships. Opdahl and Henderson-Sellers (2002) refer to these constructs as "very high level UML constructs that… [have] no explicit counterpart in the BWW-model. The most useful interpretation might be to regard… [them] as subtypes of BWW-whole-part relations, but this must be considered further" (p. 59). For example, the use case diagram in 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 in Figure 1 implies that in some circumstances, *Create My Account* will be a logical sub-part of *Place Order*.

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 has grammatical constructs for aggregation that are fairly well defined in the context of class diagrams and in the UML metamodel (Barbier et al., 2003). To add a new construct for aggregation on use case diagrams would constitute *construct redundancy*. Second, aggregation is defined as a whole-part relationship between things or kinds of thing, with specific characteristics such as emergent properties and transitivity (Barbier et al., 2003; Wand and Weber, 1990), which have not been articulated with respect to processes. If use cases are viewed as things, as in the UML metamodel, then there is no ontological problem. However, as argued earlier, if use cases are viewed as ontological processes, as in the UML definitions, then this constitutes *construct overload*.

A variation on the aggregation interpretation is that <<include>> and/or <<extend>> are constructs for representing functional decomposition (Constantine and Lockwood, 2000). The UML 2.0 states that the "include relationship allows hierarchical composition of use cases…within one diagram" (OMG, 2003a: 518). From an ontological perspective, a process may be decomposed into its constituent actions and events. Thus, there is no ontological problem with decomposing a use case into finer levels of detail. However, many authors express concern that functional decomposition detracts from and violates the principles of object-oriented modeling (Dobing and Parsons, 2000). Fowler (1998), for example, refers to use case "abuse by decomposition," which is the excessive use of <<include>> and <<extend>> to functionally decompose a system into small, atomic units. Other authors note the "explosion" of use cases when these constructs are used to decompose high level use cases into more focused tasks, and the difficulties of managing and organizing the use cases in this situation (Bittner, 2000; Korson, 1998; Lilly, 2000).

In sum, the <<include>> and <<extend>> constructs do not have a clear ontological mapping or foundation. This may partially explain the complaints about how these constructs are used in practice. We would expect to see problems in creating use case

models with these constructs and in moving from use case models to other analysis and design models, depending on how the constructs are interpreted (e.g., transformation laws, whole-part relationships, functional decomposition).

### *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 system's environment. This ontological description of a system does not mention the terms "goal" or "boundary," which are often mentioned in the context of use case models. In the BWW model, a goal is the "preferred" state that results from a transformation law property of the system and a boundary is the dividing line between a system composition (what's inside the system) and the set of relevant external events (what happens outside the system that affects the system) (Wand, 1996: 283). The boundary or scope of a model is defined by "choosing a specific set of interactions…that exist between the system and its environment" (ibid: 284).

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 (or one of its components) to change. We find no ontological discrepancy with respect to the system environment. We do, however, note that it may be difficult to correctly identify the actors in the environment if the system boundary is not clearly defined or explicitly represented (a problem that was described earlier). As shown in Figure 3, the system boundary in a use case diagram may represent an organization, an information system within the organization, or a particular software application. When this boundary is clearly defined, the actors that exist in the system environment are clear as well. For example, if the system boundary represents an entire organization, then customers and suppliers are outside of that boundary, whereas employees are not.

The use case modeling grammar does not include constructs to adequately represent the system as a composite thing, or the *structure* of the system. This may be because the purpose of use case diagrams is to show the observable behavior of a system without showing its internal structure (OMG, 2003a). Perhaps the ontological notion of system structure is beyond the scope of use case models; however, we do not believe this is the case. The UML states that when a system is modeled as a hierarchical set of subsystems, "the system can be specified with use cases at all levels" (OMG, 2003b: 2-137). The ability to have an integrated set of use cases across structural levels or components of a system does seem to be within the intent and scope of use case models.

Figure 3 shows an organizational system along with its component systems and the interactions between actors and the system and/or component system. This conceptual diagram shows the different levels of system composition and structure that we would like to represent with the use case modeling grammar. However, Figure 3 is not a use case diagram, and 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. It does not provide mechanisms for easily and clearly describing systems hierarchically or at multiple levels of granularity.[4]

There are some UML constructs that might address system structure, such as packages, generalization/specialization, <<include>>, and <<extend>>. The UML package construct is a generic grouping of elements in the grammar. A package could then be used in use case diagrams to represent each of the system boundaries shown in Figure 3. 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 diagrams do, in some sense, represent aggregation 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.

The use case modeling grammar is ontologically *deficient* with respect to the composite nature of a system and the structure of a system. This may partially explain some of the complaints from practice, particularly about how to organize and manage large sets of use cases and how to evolve use cases from analysis to design.

### *Decomposition*

Decomposition is a key construct of the BWW ontology that is, according to Opdahl and Henderson-Sellers (2002), almost completely absent from the use case modeling grammar. 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.

Figures 8 and 9 illustrate the decomposition problem with a simple example of a Health Club and one of its information systems. The system in Figure 8 is the Health Club enterprise; in Figure 9 it is the Health Club's membership information system. The membership information system is a component of the Health Club, although there is no grammatical construct that conveys this. Furthermore, it is not a "good" decomposition in the sense that some of the information stated in Figure 8 is lost in Figure 9.

For example, the *Join Health Club* use case shown in Figure 8 (business system) is fulfilled in part by the *Open New Membership* use case in Figure 9 (information system), even though the use cases have different names and are initiated by different actors. There is no grammatical construct in either the use case diagram or the use case specification to represent this link across different system levels.

---

[4] This is not the same as the traditional notion of functional decomposition, where one unit of functionality is broken down into its components. We are referring to a set of inter-related use case diagrams, each of which defines a specific system (subsystem) boundary and treats that system (subsystem) as a black box, but where the Use Cases and actors on one diagram are related to Use Cases and actors on another diagram.

The UML 1.5 (OMG, 2003a) mentions decomposition of use cases from a system-level view to two or more subsystem views. A use case specifying one element, e.g., a system, may be "refined into a set of smaller use cases, each specifying a service of … [an element/subsystem] contained in the first one" (p. 2-137). The use case of the whole is superordinate to the refining use cases, which are subordinate. "The functionality specified by each superordinate use case is completely traceable to its subordinate use cases" (ibid). However, it is not clear which grammatical constructs support this decomposition and traceability across system-subsystem levels.

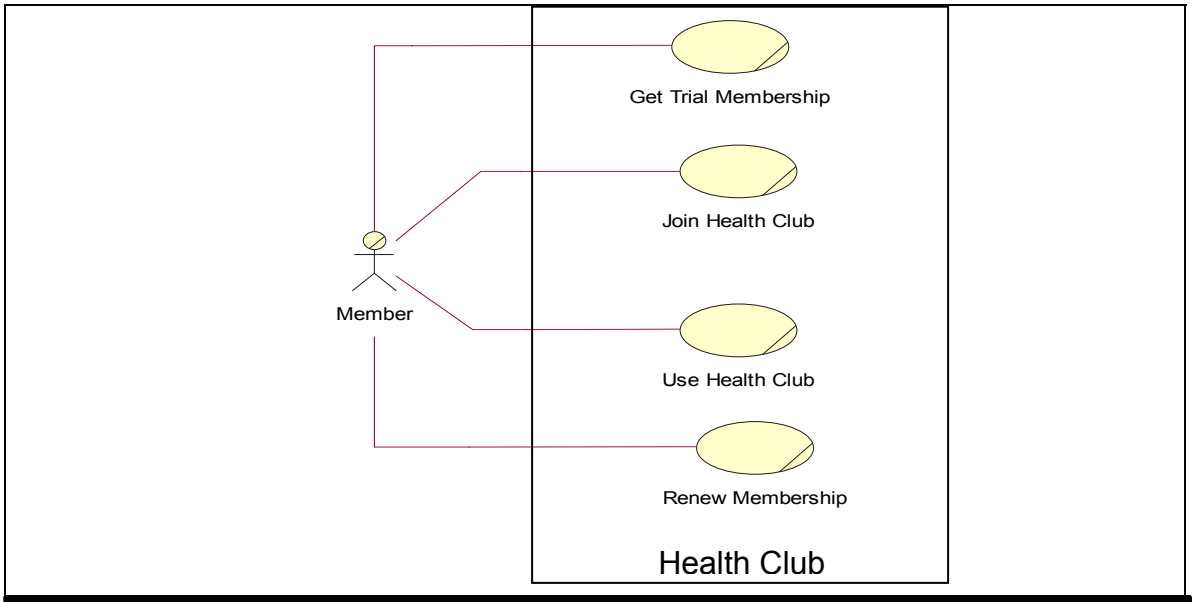



**Figure 8. Use Case Diagram for Health Club Enterprise**

The UML package construct is a possibility for representing decomposition, since it may represent subsystems within systems. However, the package construct is not discussed in the UML specifications with respect to use case diagrams, and does not have well-defined semantics that would support linking across multiple levels of use case diagrams or across use case diagrams and class diagrams. Rather, UML packages are used to broadly organize groups of models.

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.

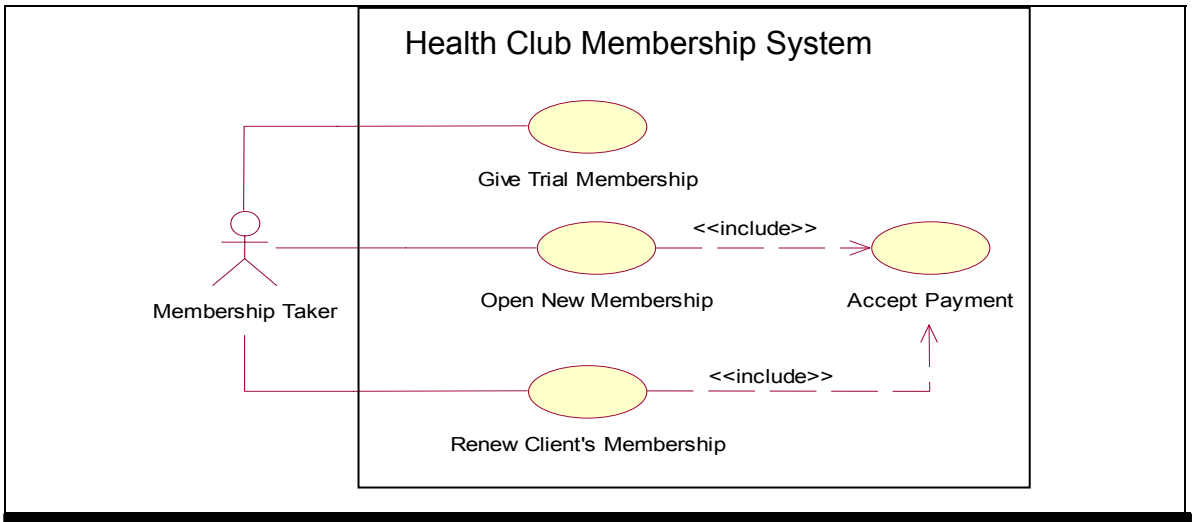



**Figure 9. Use Case Diagram for Health Club Membership System**

## Implications for research and practice

The previous section identified several ontological discrepancies in the use case modeling grammar. Before discussing the implications of these discrepancies, we must make two points. First, the relevance of our analysis depends on the purpose of the use case models in a given situation. Bodart et al. (2001) distinguish between a *presentation* model and a "true" conceptual model. The former is used to present a high level view of a domain to facilitate communication and "surface level" understanding, so simplicity in the modeling grammar is more important than semantic robustness and accuracy. On the other hand, the goal of a true conceptual model is to "provide an accurate, complete representation of someone's or some group's perception of the semantics underlying a domain" (p. 386). Thus, the ontological discrepancies in use case modeling grammar may be acceptable for presentation models but problematic for conceptual models.

Second, there are some ontological deficiencies and ambiguities in the use case modeling grammar that extend beyond the scope and purpose of the models. For example, use case models are not suitable for modeling the things in a system and their properties, but this was never the intent of these models. Thus, we limit our discussion to weaknesses that we believe are within the scope and intent of use case modeling.

The ontological discrepancies in use case modeling grammar have implications for both practice and research. For practitioners, the discrepancies help explain the problems identified in practice. They provide a basis for evaluating extensions and alternatives to use case modeling grammar. For researchers, they help to enrich the design theory of which use cases are a significant part. The ontological discrepancies provide the basis for developing hypotheses about where the use case models will or will not meet the goals of conceptual modeling.

Table 5 lists the hypotheses we derived from the ontological analysis. H1 states the general hypothesis that conceptual modeling grammars with fewer ontological discrepancies will fare better than grammars with more ontological problems. Hypotheses H2 through H9 describe specific behaviors and modeling errors we expect to see, given the problems in the use case modeling grammar. Future research is

needed to investigate these hypotheses. Controlled experiments and verbal protocol studies can illuminate the extent to which the ontological problems lead to cognitive problems and/or specific task performance degradation. Field studies can be undertaken to assess the practical impact of the ontological problems and some of the proposed solutions to those problems.

### *Construct deficiencies*

The use case modeling grammar is ontologically deficient with respect to representing system structure and system decomposition. The UML specification states that use cases may be used for both systems and their subsystems such that use cases on a subsystem level are completely traceable to the system level (OMG, 2003b). However, it does not specify which use case modeling constructs represent such a decomposition of the system or the relationships between use cases or actors at different levels of abstraction.

Ontological deficiencies are problematic in practice if the analysts try to represent the ontological constructs that are lacking in the grammar. How do analysts represent system structure and decomposition and what errors occur because of the deficiency in the modeling grammar? We expect that when analysts stay within the confines of the use case modeling grammar, there will be problems tracing between use case diagrams at different levels of abstraction (Hypothesis 2(a)). This, in turn, may lead to requirements definition problems (Hypotheses 2(b) and (c)).

On the other hand, analysts may try to "go beyond" the strict definition of the grammatical constructs in order to represent system structure or decomposition. In this case, communication problems may be expected, because the existing constructs will be overloaded or new constructs will be created that are unfamiliar to users and designers (Hypotheses H3 and H4).

We believe that the lack of support for system structure and decomposition in the grammar is a significant shortcoming. Many analysts advocate building use case models at different levels of abstraction (e.g., Fowler, 1998; Korson, 1998; Achour et al., 1999). For example, Cockburn (2001) states that a use case on one level should relate to one or more use cases on the levels "above" and "below" it. Each use case on a system-level diagram should be linked to one or more use cases on the business level that answer the question, "*Why* is this system-level use case needed?" Conversely, each use case on the business-level diagram should be linked to one or more system-level use cases that answer the question, "*How* will the new system support this business-level use case?" Such a linkage between different levels provides a degree of traceability that is important to large systems development projects, and is consistent with the ontological definitions of system, system structure, and decomposition.

| Table 5 Research Hypotheses | |
|---|---|
| **H1:** | A conceptual modeling grammar with fewer ontological discrepancies will support the goals of conceptual modeling better than a grammar with more ontological discrepancies. |
| **H2:** | When use case models are created at different levels of abstraction and they adhere to the existing grammar, one or more of the following will occur:<br>(a) Elements on a diagram at one level of abstraction will not be traceable to elements on a diagram at another level.<br>(b) The models will exhibit errors of scope creep.<br>(c) The models will exhibit errors of missed functionality. |
| **H3:** | Use case models that explicitly represent system structure or decomposition do so by:<br>(a) overloading existing constructs in the use case modeling grammar (e.g., the extend, include, or generalization constructs);<br>(b) adding new constructs (e.g., stereotypes) to the use case models; and/or<br>(c) adding supplemental documentation to the use case models. |
| **H4:** | Users and analysts will make incorrect or inconsistent inferences about system requirements based on use case models, particularly when generalization, <<include>>, and <<extend>> are used. |
| **H5:** | Analysts will use the association construct in use case diagrams to represent semantics other than “interacts with” (e.g., “sends output to,” “receives data from”). |
| **H6:** | Analysts will use the generalization construct in use case diagrams to represent semantics other than “a kind of” (e.g., “plays the role of,” “acts on behalf of”). |
| **H7:** | Analysts who use use case models to “drive” class diagram creation will have:<br>(a) Difficulty mapping from actors to classes, subclasses, or attribute values;<br>(b) A tendency to assign actors “class” names versus role names;<br>(c) Difficulty mapping use cases to classes, relationships, and/or behaviors.<br>(d) A tendency to create use cases that focus on a single class rather than a task/goal.<br>(e) Difficulty mapping associations and generalizations on use case diagrams to associations and generalizations on class diagrams. |
| **H8:** | Included use cases will be poorly understood by users when their purpose is to represent reusable functionality. |
| **H9:** | Analysts will have difficulty determining whether/how to model extending and included use cases in other UML behavioral models. |

To overcome this deficiency, "practitioners have developed elaborate schemes for planning and managing the production and co-evolution of scenarios [or use cases] at different levels of detail" (Antón and Potts, 1998: 229). Some work in this area uses the goal construct to structure use cases within and across different levels of abstraction (e.g., Lee and Xue, 1999; Cockburn, 1997; Achour et al., 1999; Antón and Potts, 1998; Regnell and Davidson, 1997). The goal-driven approach begins with an actor (e.g., a customer) with a goal (e.g., Place Order). The goal causes the actor to request that the system fulfill one of its responsibilities (i.e., represented by the Place Order use case). The system sets its goal to accomplish Place Order, and this may, in turn, lead it to set one or more sub-goals that may be fulfilled by other use cases or actors. This process helps structure use cases for one system boundary. The CREWS project takes the goal-driven approach further to organize use cases (scenarios, in this case) into a hierarchy of inter-related levels (Achour et al., 1999).

Other authors have created specific constructs for the use case specification grammar to represent different levels of abstraction. Cockburn (2001) uses two symbols, a house and a box, to differentiate between a use case defined at the enterprise level and the system level, respectively. He also describes several levels of detail that may be applied to either the enterprise or system scope, and each of which has a unique graphical symbol: (1) the *summary* level; (2) the *user goal* level; and (3) the *subfunction* level. Similarly, WirfsBrock and Schwartz (2002) mention *summary*, *core*, *supporting*, and *internal* levels of detail for writing use cases, and specify the level of detail as an explicit construct in their use case specification. And many authors differentiate between *essential* and *real* use cases, where the former is appropriate for requirements modeling and the latter for design specification (e.g., Constantine and Lockwood, 2000; Dennis et al., 2002). However, it is unclear whether or how these constructs enable a use case defined at a summary or essential level to be linked to one or more use cases at a user goal, supporting, or real level.

Another alternative is to create a new type of model for showing the linkages of use cases across levels of abstraction. De Cesare et al. (2003), for example, create a matrix that lists business use cases, shows which of these are to be automated by a new information system, and, for each automated business use case, lists the associated system use cases. The matrix provides the "missing link" between different levels.

Field studies and reports from practitioners emphasize problems of poorly defined or mixed-up system boundaries (Lilly, 2000) and problems organizing and structuring large sets of use cases at different levels of detail (Cockburn, 2001; Korson, 1998). Future research is needed to evaluate whether and to what extent proposals such as those described above may alleviate these problems. The goal-oriented approaches have some ontological merit as a way to organize and inter-relate use cases, since goals represent desirable ontological states of a system. Another option is to specify new constructs to represent levels of detail (decomposition) to fill the void in the use case modeling grammar, or to revise the current definitions and/or usages of constructs such as inclusion, extension, and generalization.

### *Construct overload*

The UML actor, use case, generalization, association, <<include>>, and <<extend>> constructs are ontologically *overloaded*. We expect construct overload to be a problem because the multiple interpretations of a construct may hinder a common understanding

of a domain between analysts and clients (Hypothesis H4). The UML association and generalization constructs are used in both use case diagrams and class diagrams, but their definitions are not entirely consistent across both contexts. Thus, we expect analysts will use generalization and association in use case diagrams to represent semantics other than what is precisely defined in the UML specification (Hypotheses H5 and H6). We also expect construct overload to cause modeling problems when use cases are the starting point for other UML models that include the same constructs (Hypothesis H7). This may be particularly problematic for actors and use cases, which may or may not be perceived as "things" to represent as classes on a class diagram.

We have not seen any approaches to address the ontological ambiguity for the use case association, <<include>>, <<extend>>, or generalization constructs. However, there are two approaches that may help clarify the ambiguity of the actor construct.

First, Constantine and Lockwood (2000) view actors as ontological *things* that may assume one or more roles (ontological properties), and they use a diagram called a "User Role Map" to show which actors play which roles. This may be a way to more clearly relate roles to entities, or to reconcile organizational roles (e.g., job titles) to system-related roles (i.e., "true" use case roles). For example, a User Role Map could show that Sales Associate, Sales Manager, and Customer can all play the role of Order Taker. To coin a phrase from Cockburn (2001), the User Role Map could show which actors in which roles are allowed to "act on behalf of" other actors in other roles, such as a system-level sales associate acting of behalf of the "ultimate" business-level actor, i.e., the customer. This could then pave the way for better integrating use case diagrams at different levels of abstraction by allowing the reader to reconcile actors on one Diagram with actors on another. Figure 10 below shows an example of what a User Role Map might show.

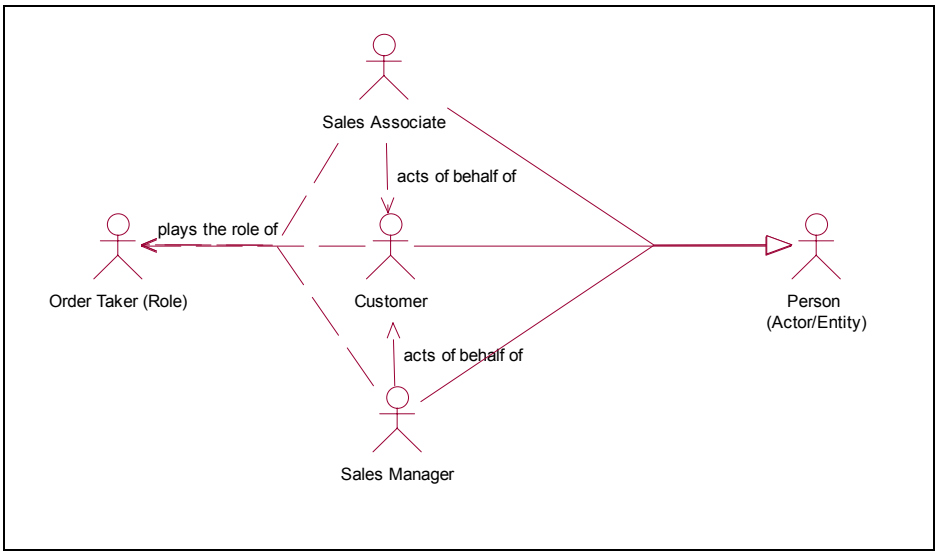



**Figure 10. Sample User Role Map**

Second, Parsons and Wand (2000) describe a conceptual modeling approach, based on classification theory, that may also help clarify the ambiguity surrounding the actor construct. They discuss the "multiple classification problem" where one thing may

simultaneously be an instance of more than one kind, or where there are overlapping subtypes (Bodart et al., 2001). For example, an individual may be an engineer, a customer, and a shareholder. In use case modeling, the individual plays the role of engineer, customer, and/or shareholder, and each role corresponds to an actor. In a class diagram, each role corresponds to a class, and the challenge for current conceptual modeling grammars is to show that an individual is an instance of all three classes, even when the classes are not necessarily related to each other by generalization/specialization. Parsons and Wand propose a "layered approach" that separates modeling of instances (BWW things) from any particular classification (BWW kinds and subkinds). Thus, their approach may also provide a way to reconcile actors to classes (kinds) in a class diagram.

### *Construct excess*

We identified an example of construct excess with respect to the <<include>> and <<extend>> relationships. Field studies and practitioner reports indicate that these constructs are problematic in systems development projects. Based on our analysis, we expect the value of these constructs to be unclear to users and analysts, particularly when they are used for design (e.g., reuse) rather than conceptual modeling purposes (Hypothesis H8). Given the ambiguity of the <<include>> and <<extend>> constructs, analysts may use them to address deficiencies in the grammar, such as system decomposition (Hypothesis H3). Furthermore, the ambiguity of the constructs may lead to difficulties when mapping from use case models to other behavioral models, such as Sequence Diagrams and Activity Diagrams (Hypothesis H9).

More work is needed on the ontological value of these two constructs. They may, as Opdahl and Hendersen-Sellers (2002) argue, be best understood as a type of ontological aggregation. In this case, the precise semantics of aggregating ontological processes needs to be specified, as this is not necessarily the same as aggregation between kinds of things. We believe that the work mentioned earlier on goals (Antón and Potts, 1998; Achour et al., 1999; Cockburn, 1997; Lee and Xue, 1999; Regnell and Davidson, 1997) may have implications for defining and using these constructs in a manner that may address the system decomposition deficiencies mentioned earlier. The inclusion and extension constructs also need further investigation to understand their value for representing problem domains versus representing reusable "chunks" of system functionality. If their primary purpose is the latter, then these constructs should be considered an ontological excess, albeit an excess that serves a potentially useful purpose in the transition from analysis to design.

## Conclusions

This paper has examined use case modeling as a component of information system design theories (Walls et al., 1992) that advocate iterative, incremental development, such as the Unified Process (Jacobsen et al., 1999). We evaluated the extent to which use case modeling grammar is able to produce artifacts—diagrams and descriptions—that meet the goals of conceptual modeling. The primary goal of a conceptual model is to faithfully model the part of the "real world" that will be represented in a future information system. We used the BWW ontology as the theoretical basis for our evaluation.

Our analysis shows that use case models have some practical and ontological strengths. The simplicity of the grammar is important since it 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, in many respects, the use case modeling grammar supports the dynamic view quite well. Use cases represent the ontological transformations that will occur in a system, typically triggered by the actor associated with the use case, and the use case name reflects the goal to be accomplished at the end of the transformation. The use case specification details the visible steps in the transformation process.

There are, however, several areas where the use case modeling grammar falls short. First, the grammar is ontologically incomplete with respect to representing the system structure or decomposition. There are no clearly-defined constructs for representing systems at different levels of detail such that no information is lost between levels. Second, the definitions of actor, use case, association, and generalization are ontologically overloaded, or at best, ambiguous and imprecise. Finally, the <<include>> and <<extend>> constructs are problematic. It is unclear that they are necessary or useful for conceptual modeling purposes, and if they are, their semantics overlap with other UML constructs, such as aggregation.

Use case modeling is a widely-used tool for requirements analysis and is often the model which “drives” the entire systems development effort (Jacobsen et al., 1999). Thus, inaccuracies and misunderstandings in a use case model may have significant ripple effects into systems design and implementation. We proposed nine hypotheses (Table 5) regarding the specific modeling and communication problems that may appear in systems development projects where use case models are a primary tool for representing functional requirements. We encourage future research to test these hypotheses so that the impact of the ontological discrepancies may be assessed and better understood. Future research is also needed to evaluate whether other UML constructs (e.g., packages, aggregation) may address the use case modeling problems and to develop extensions to the UML grammar that address the problems.

## *References:*

Achour, C. B., M. Tawbi, and C. Souveyet (1999) “Bridging the Gap Between Users and Requirements Engineering: The Scenario-Based Approach,” *CREWS Report Series 99-07,* CRI Universite de Paris 1 – Sorbonne. http://sunsite.informatik.rwth-aachen.de/CREWS/reports99.htm.

Ambler, S. (2001) "The Object Primer: Introduction to Techniques for Agile Modeling," *A Ronin International White Paper*. http://www.ronin-intl.com/publications/objectPrimerAgileModeling.pdf.

Antón, A. I. and C. Potts (1998) “A Representational Framework for Scenarios of System Use,” *Requirements Engineering,* 3, pp. 219-241.

Alter, S. (2002) *Information Systems: Foundations of E-Business*, Pearson Education, Inc., Upper Saddle River, NJ.

Barbier, F., B. Henderson-Sellers, A. L. Parc-Lacayrelle, and J.-M. Bruel, (2003) “Formalization of the Whole-Part Relationship in the Unified Modeling Language,” *IEEE Transactions on Software Engineering,* 29**,** pp. 459-469.

Berard, E. (1998) "Be Careful with Use Cases," The Object Agency, Inc. http://www.toa.com/pub/use_cases.htm#s4.

Bittner, K. (2000) "Why Use Cases are not Functions," *The Rational Edge*, December, www.therationaledge.com/content/dec_00/t_ucnotfunctions.html.

Bodart, F., A. Patel, M. Sim, and R. Weber (2001) "Should Optional Properties be Used in Conceptual Modeling? A Theory and Three Empirical Tests," *Information Systems Research*, 12(4): 383-405.

Booch, G., J. Rumbaugh, and I. Jacobson (1999) *The Unified Modeling Language User Guide,* Reading, MA: Addison Wesley.

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

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

Cockburn, A. (1997) "Goals and Use Cases," *Journal of Object-Oriented Programming* (10)5, pp 35-40.

Cockburn, A. (2001) *Writing Effective Use Cases,* Boston, MA: Addison Wesley.

Codd, E. (1970) "A Relational Model for Large Shared Data Banks," *Communications of the ACM,* 13 (6): 377-387.

Constantine, L.L., and L.D. Lockwood (2000) "Structure and Style in Use Cases for User Interface Design," www.foruse.com/articles/structurestyle2.htm.

De Cesare, S., M. Lycett, and R.J. Paul (2003) "Actor Perception in Business use Case Modeling," *Communications of the Association for Information Systems* (12), pp 223-241.

Dennis, A., B.H. Wixom, and D. Tegarden (2002) *Systems Analysis and Design: An Object-Oriented Approach with UML*, New York, NY: John Wiley & Sons, Inc.

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, pp 28-36.

Elmasri, R., and S. Navathe (1994) *Fundamentals of Database Systems*, (2nd edition ed.) Redwood City, CA: The Benjamin/Cummings Publishing.

Fowler, M. (1998) "Use and Abuse Cases," *Distributed Computing*, April, pp. 1-2.

Hertzum, M. (2003) "Making use of scenarios: a field study of conceptual design," *International Journal of Human-Computer Studies,* 58**,** pp. 215-239.

Jacobsen, I., G. Booch, and J. Rumbaugh (1999) *The Unified Software Development Process,* Boston, MA: Addison Wesley.

Jarke, M., X.T. Bui, and J.M. Carroll (1998) "Scenario Management: An Interdisciplinary Approach," *Requirements Engineering,* 3, pp. 155-173.

Kenworthy, E. (1997) "Use Case Modeling: Capturing User Requirements, http://www.zoo.co.uk/~z0001039/PracGuides/pg_use_cases.htm.

Korson, D.T. (1998) "The Misuse of Use Cases (Managing Requirements)," *Object Magazine*, (5). http://www.korsonmcgregor.com/publications/korson/Korson9803om.htm.

Lee, J., and N.-L. Xue (1999) "Analyzing User Requirements by Use Cases: A Goal-Driven Approach," *IEEE Software,* (July/August), pp 92-101.

Lilly, S. (2000) "How to Avoid Use-Case Pitfalls," *Software Development Magazine*, January, www.sdmagazine.com/print/documentID=11235.

Malan, R., and D. Bredemeyer (2001) "Functional Requirements and Use Cases," Architecture Resources for Enterprise Advantage, Bloomington, IN. http://www.bredemeyer.com/pdf_files/functreq.pdf.

Markus, M. L., A. Majchrzak, and L. Gasser (2002). "A Design Theory for Systems That Support Emergent Knowledge Processes." *Management Information Systems Quarterly* 26(3): 179-212.

OMG (2001) "Unified Modeling Language (UML), Version 1.4," Object Management Group, Inc., Needham, MA. http://www.omg.org/cgi-bin/doc?formal/01-09-67.

OMG (2003a) "Unified Modeling Language (UML), Version 1.5," Object Management Group, Inc., Needham, MA. http://www.omg.org/technology/documents/formal/uml.htm.

OMG, (2003b) "UML 2.0 Superstructure Final Adopted Specification (ptc/03-08-02)," Object Management Group, Inc., Needham, MA. http://www.omg.org/cgi-bin/doc?ptc/2003-08-02.

Opdahl, A. L. and B. Henderson-Sellers (2001) "Grounding the OML Metamodel in Ontology," *Journal of Systems & Software,* 57**,** pp. 119-143.

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

Parsons, J. (1996) "An Information Model Based on Classification Theory," *Management Science,* 42**,** pp. 1437-1453.

Parsons, J. and Y. Wand (2000) "Emancipating Instances from the Tyranny of Classes in Information Modeling," *ACM Transactions on Database Systems,* 25, pp. 228-268.

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*, Barcelona, Spain www.tts.lth.se/Personal/bjornr/Papers/REFSQ97.pdf

Rosemann, M., and P. Green (2002) "Developing a Meta Model for the Bunge-Wand-Weber Ontological Constructs," *Information Systems,* 27: 75-91.

Rosenberg, D., and K. Scott (2001) *Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example,* Reading, MA: Addison-Wesley.

Schneider, G., and J. Winters (2001) *Applying Use Cases: A Practical Guide*, (2nd edition) Upper Saddle River, NJ: Addison-Wesley.

Walls, J. G., G. R. Widmeyer, and O. El Sawy. (1992). "Building an Information System Design Theory for Vigilant EIS." *Information Systems Research* 3(1): 36-59.

Wand, Y. (1996) "Ontology as a Foundation for Meta-Modelling and Method Engineering," *Information Science and Software Technology,* (38), pp 281-287.

Wand, Y., D.E. Monarchi, J. Parsons, and C. Woo (1995) “Theoretical Foundations for Conceptual Modelling in Information System Development,” *Decision Support Systems,* 15**,** pp. 285-304.

Wand, Y., V.C. Storey, and R. Weber (1999) “An Ontological Analysis of the Relationship Construct in Conceptual Modeling,” *ACM Transactions on Database Systems,* 24, pp. 494-528.

Wand, Y., and R. Weber (1990) "An Ontological Model of an Information System," *IEEE Transactions on Software Engineering* (16)11, pp 1282-1292.

Wand, Y. and R. Weber (1995) “On the Deep Structure of Information Systems,” *Information Systems Journal,* 5, pp. 203-223.

Wand, Y., and R. Weber (2002) "Research Commentary: Information Systems and Conceptual Modeling -- A Research Agenda," *Information Systems Research* (33)4, pp 363-376.

Weber, R., and Y. Zhang (1991) "An Ontological Evaluation of NIAM's Grammar for Conceptual Schema Diagrams," *Twelfth International Conference on Information Systems*, New York, NY, pp. 75-82.WirfsBrock, R. and J.A. Schwartz (2002) “Tutorial #9: The Art of Writing Use Cases,” In proceedings of *OOPSLA:* The *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, & Applications,* Seattle, Washington.

## *About the Authors*

**Gretchen Irwin** is an Assistant Professor of Computer Information Systems in the College of Business at Colorado State University. Dr. Irwin received her Ph.D. and M.S. in Information Systems from the University of Colorado, Boulder, and was on the faculty at the University of Auckland, New Zealand, prior to her position at CSU. Her research focuses on evaluating the usability, productivity, and quality claims associated with software process improvements, such as the Unified Modeling Language, object-oriented technology and software reuse. Dr. Irwin has published in the *Journal of MIS, Communications of the ACM,* and *Human-Computer Interaction*.

**Daniel Turk** received a M.S. degree in computer science from Andrews University in 1988 and a Ph.D. degree in business administration (computer information systems) from Georgia State University in 1999. He is currently an Assistant Professor in the Computer Information Systems department at Colorado State University in Fort Collins, Colorado. His research interests are in the areas of computer networking, object-oriented systems, software engineering, business- and system-level modeling, software development process modeling, the value of modeling, process improvement. He is a member of the IEEE and the ACM, and has published papers in *IEEE Transactions on Software Engineering*, *L’Objet*, *The Journal of Systems and Software*, *Information Technology & Management*, and the *International Journal of Human Computer Studies*.

Journal of the Association for Information Systems
ISSN: 1536-9323