A Template for Defining Enterprise Modelling Constructs

Andreas L. Opdahl, University of Bergen, Norway
Brian Henderson-Sellers, University of Technology, Sydney, Australia

ABSTRACT

The paper explains the need for a standard way of defining modelling constructs from different enterprise modelling languages and proposes a template for defining enterprise modelling constructs in a way that facilitates language integration. The template is based on the Bunge-Wand-Weber (BWW) representation model of information systems (IS) and has been used on several existing modelling languages and frameworks. It is illustrated with definitions of constructs from the Unified Modeling Language (UML). The paper focusses on modelling constructs that represent concrete problem domains, i.e., that represent materials rather than concepts, and thus focuses on the concrete parts and aspects of enterprises.

Keywords: problem domain representation; object-oriented analysis; ontological analysis and evaluation; Bunge-Wand-Weber (BWW) model; Unified Modeling Language; UML

BACKGROUND

Modelling languages and ontologies for enterprises and their information systems (IS) are becoming increasingly important. New and emerging technologies, such as enterprise application integration, enterprise content management, domain-specific languages, intelligent agents and the semantic web, all rely on models of or ontologies for enterprises. As more and more enterprise knowledge is captured in models, there is a danger that the knowledge is dispersed into many small ,isolated islands because it is represented in a variety of different modelling languages. Language standardisation alone is not sufficient to solve this problem because different modelling domains, modelling problems, user communities, business partners and model-based tools will require their own dedicated modelling languages in the future as they do today.

To ensure that knowledge captured in enterprise models can be integrated and made available throughout the organisation, it is therefore necessary to enable organisations to integrate more closely the different modelling languages they use.

Philosophical ontology offers a common ground for integrating enterprise and IS modelling languages. According to Weber (1997), philosophical ontology is the branch of philosophy that deals with theories about the nature of things in general, as opposed to theories about particular things. In the IS field, one much used philosophical ontology is the Bunge-Wand-Weber (BWW) model of information systems (e.g., Wand & Weber, 1988, 1993, 1995), which adapts Mario Bunge's (1977, 1979) comprehensive ontology to the IS field. Bunge's ontology is an example of *scientific realism*, meaning that it "identifies reality with the collection of all concrete things, [...] postulates the autonomous existence of the external world, admits that we are largely ignorant of it, and encourages us to explore it" (Bunge, 1999). It is therefore well suited for integrating modelling constructs that represent concrete problem domains, i.e., that represent materials rather than concepts. The BWW model is a set of three models, of which this paper will only use one: the *representation model*.

This paper uses the BWW representation model—called just the BWW model in this paper—as a common ground for defining enterprise and IS modelling constructs in a way that facilitates language integration. The paper thereby focusses on the concrete parts and aspects of enterprises, and proposes a *template* for defining enterprise and IS modelling constructs. By "template" we mean a standard way of defining modelling constructs by filling in standard set of "entries", some of which are complex and some of which are interrelated. Figures 1–6 will introduce the template stepwise in a series of UML class diagrams. The main idea is to provide a standard way of defining modelling constructs in terms of the BWW model, in order to make the definitions cohesive and, thus, learnable, understandable and as directly comparable to one another as possible. Another important idea is to provide a way of defining modelling constructs not only generally, in terms of whether they represent "classes", "properties" or other ontological categories, but also in terms of *which* classes and/or properties they represent, in order to make the definitions more clearly and precisely related to the enterprise. Although most of the paper is about the concrete parts and aspects of enterprises, we believe the template and other results of this paper are sufficiently general to apply to concrete problem domains in general.

The template has been developed based on practical experience from analysing, suggesting improvements to and providing precise definitions of several full-scale integrated modelling languages and frameworks, including:

- 73 constructs from the OPEN Modeling Language (OML) (Firesmith, Henderson-Sellers & Graham, 1997) in Opdahl, Henderson-Sellers & Barbier, 1999; Opdahl & Henderson-Sellers (2001) and
- 68 constructs from the Unified Modeling Language (UML) (OMG, 2001) in Opdahl & Henderson-Sellers (2002).

The template will be illustrated with definitions of constructs from the UML Version 1.4. Next, we will explain the underlying *theory* of the paper–the BWW

model–and then introduce *the template* itself, before we present *results* of using the template to define constructs from the UML. This is followed by a *discussion* of the usefulness of the template, and *conclusions* and paths for *further work*. The outcome is threefold. Firstly, the template offers a standard way of precisely defining modelling constructs and thereby integrating different modelling languages. Secondly, the template makes the BWW model more easy to use for integrating modelling languages. Thirdly, the illustration of the template with definitions of constructs from the UML is a contribution to making the UML more precisely defined.

The template was preliminarily outlined by Opdahl, Henderson-Sellers and Barbier (1999), and is presented in an extended and much refined form here.

THEORY

The Bunge-Wand-Weber representation model (e.g., Wand & Weber, 1988, 1993, 1995)—called the BWW model in this paper—has already been used to analyse and evaluate the modelling constructs of many established IS and enterprise modelling languages, including:

- dataflow diagrams (Wand & Weber, 1989),
- ER models (Wand & Weber, 1989; Weber, 1997),
- NIAM (Weber & Zhang, 1996),
- nine languages supported by the Upper CASE-toolset Excelerator (Green, 1996),
- four languages supported by the ARIS toolset for business modelling (Green & Rosemann, 1999, 2000),
- the OPEN Modelling Language (OML) (Opdahl & Henderson-Sellers, 2001) and
- the Unified Modelling Language (UML) (Evermann & Wand, 2001; Opdahl & Henderson-Sellers, 2002). The BWW model has also been used for general analyses of :

- IS design theory (Wand, 1989a),
- object-oriented modelling constructs (Wand, 1989b; Parsons & Wand, 1997),
- systems decomposition (Wand & Weber, 1990; Paulson & Wand, 1992),
- object-oriented information systems (Takagaki & Wand, 1991),
- dimensions of data quality (Wand & Wang, 1996),
- optional properties in conceptual modelling (Bodart et al., 2001),
- a two-layered information modelling approach where instances are not tied to particular classes (Parsons & Wand, 2000) and
- whole-part relationships (like UML's aggregation and composition constructs) in OO models (Barbier et al., 2000; Opdahl, Henderson-Sellers & Barbier, 2001).

The BWW model is therefore a natural starting point for a template for defining enterprise modelling constructs, although alternatives exist both in the form of general philosophical ontologies, e.g., Chisholm (1996), or special enterprise and IS ontologies, e.g., the enterprise ontology (Uschold et al., 1998) and the framework of information systems concepts (FRISCO) (Verrijn-Stuart et al., 2001). In support of the BWW model, Wand & Weber (1993) have argued that Bunge's ontology is:

1. better developed and formalised than alternative philosophical ontologies;
2. based on concepts that are fundamental

to the computer science and information systems domains; and
3. productive, in the sense that it has given useful results.

Space does not permit a full presentation of the BWW model, but this section will present the most important BWW concepts that we will use. Table 1 gives definitions of *all* the BWW concepts used in the paper.

Things and Properties: According to Bunge's ontology and the BWW model, there is a world that exists independently of human observers, and it consists of *things* that possess *properties*. Examples of BWW-things are "atoms, fields, persons, artifacts and social systems" (Bunge, 1999), whereas "properties of things (e.g., energy) changes in them, and ideas considered in themselves" are non-things (Bunge, 1999). In particular, concepts are not BWW-things.

Bunge's ontology and the BWW model also reminds us that we only know about things via *models* of things we create in our minds, and that we ascribe *attributes* to those models of things to stand for the properties we believe the corresponding things possess. In the BWW model, an attribute (that stands for a BWW-property) is represented as a *property function* of time, which maps the property onto different *property values* in a *property co-domain* for different points in time.

Properties: The BWW model distinguishes between properties in several different ways. An *intrinsic* property belongs to only a single thing, whereas a *mutual* property belongs to two or more things. (BWW-mutual properties are represented by *relationships* or similar constructs in many modelling languages.) A *whole-part relation* is a property that relates an *aggregate thing* to one of its *component things*. A *resultant* property belongs to a BWW-aggregate and is derived from one or more properties of its components, whereas an *emergent* property belongs to a BWW-aggregate but not to any of its components. A *law* property restricts other properties of the same thing. A BWW-law is either a *state law* or a *transition law*. An *individual* property (or property *of a particular*) is a specific, e.g., "being 25 years old" and "having grey hair," whereas the corresponding *general* properties are "having an age" and "having a hair color." Bunge (1977) also distinguishes between BWW-properties that are *permanent* and those that are *variable*.

BWW-properties may be *complex*, i.e., they may have other properties as *constituents*. A BWW-property *precedes* a second BWW-property if and only if:

- *either* (a) the second property is complex (or compound) and the first property is one of its constituents,
- *or* (b) a BWW-law states that all BWW-things that possess the second property must also possess the first.

According to (a), "having a ZIP-code" precedes "having a postal address" because every postal address includes a ZIP-code and, according to (b), "being a human being" precedes "being married,"

Classes: Things with a property in common form *BWW-classes*. A class contains all the things, and only those things, that possess one or more *characteristic properties* for the class. In other words, every BWW-class is defined by a non-empty set of characteristic properties of the things in the class. The most general BWW-class is the class off *all things,* which is defined by the universal property of *being able to associate* with other

things (Bunge, 1977). Because characteristic properties may be complex, it is sometimes possible to say that a BWW-class is defined by a *group of* ***characteristic*** BWW-properties. One BWW-class may be defined by a group of characteristic properties that is contained in a larger group of properties that defines a second class. We then say that the second BWW-class is a *subclass* of the first.

Coupling and Systems: A BWW-thing has time-dependent *states* that are determined by the values of the thing's property functions over time. A change of BWW-state in a thing is an *event*, hence a BWW-event can be described as a pair of BWW-states. Consecutive BWW-events form *complex events*, or *processes* if they occur in the same thing. The sequence of consecutive BWW-states undergone by a thing (or, alternatively, the sequence of consecutive BWW-events) is called its *history*. A BWW-thing *acts* on a second thing if and only if the BWW-history of the second thing would have been different had the first thing not existed. The first thing is called an *active thing*. Two BWW-things are *coupled* if and only if (at least) one of them acts on the other. BWW-couplings are caused by certain BWW-mutual properties that are said to be *binding*. A BWW-aggregate whose BWW-components are coupled is a *system*.

THE TEMPLATE

Overview

The template is used to define each modelling construct separately by filling in four types of *top-level entries,* some of which have *sub-entries*:

- The **instantiation level** entry type is used to define whether the modelling construct represents the enterprise at the *type level*, at the *instance level* or at *either level*. This is the simplest type of top-level entry.
- The **class** entry type is used to define which *class of things* (or *classes* of things) in the enterprise that the modelling construct may represent. We will see later that a modelling construct may be defined by *multiple* **class** entries, each of them with several sub-entries.
- The **property** entry type is used to define which *property* (or *properties*) in the enterprise the construct may represent. We will see that it too may be *repeated* and may have several sub-entries.
- The **lifetime** entry type is used to define whether the modelling construct represents *events* in, *states* of, *processes* in or the whole *lifetime* of one or more things.

We will now discuss each type of top-level entry separately using constructs from the UML—and sometimes from other languages—as examples. Although the UML is not primarily an enterprise or IS modelling language, it is relevant here because it is often used to represent concrete problem domains in the early stages of systems development. It is also a natural example language because it is widely known.

Instantiation Level

The first and simplest entry type is used to define the **instantiation level** of a modelling construct. The construct is at the **type** level if it represents BWW-classes (or their characteristic properties, etc.) and it is at the **instance** level if it represents BWW-things (and/or their properties,

Table 1: Basic concepts in the BWW model

BWW concept	*Concept definition*
BWW-thing	"The elementary unit in our ontological model. The real world is made up of things." (Wand & Weber, 1995)
BWW-property of a thing	"Things possess properties" (Wand & Weber, 1995). "We know about things in the world via their properties" (Weber, 1997).
BWW-complex property	A complex BWW-property consists of other properties, which may themselves be complex.
BWW-property function	"A property is modeled via a function that maps the thing into some value" (Wand & Weber, 1995). A BWW-property function represents how some BWW-property changes over time. BWW-property functions are also called *state functions* (Weber, 1997) or *state variables* (Parsons & Wand, 1997).
BWW-property co-domain	"The set of values into which the function that stands for the property of a thing maps the thing" (Weber & Zhang, 1996).
BWW-class of things	"A set of things that can be defined by their possessing a particular set of properties" (Weber & Zhang, 1996). 1) A BWW-class is *defined by* a "characteristic set" of properties. 2) All groups of BWW-properties that are possessed by at least one BWW-thing define a BWW-class.
BWW-subclass of things two or more things	"A set of things that can be defined via their possessing the set of properties in a class plus an additional set of properties" (Weber & Zhang, 1996). (Hence, a BWW-subclass is itself a BWW-class.)
BWW-intrinsic property of a thing	"A property that is inherently a property of an individual thing" (Wand & Weber, 1995).
BWW-mutual property of	"A property that is meaningful only in the context of two or more things" (Wand & Weber, 1995).
BWW-state of a thing	"The vector of values for all property functions of a thing" (Wand & Weber, 1995).
BWW-state law of a thing	A property that "[r]estricts the values of the property functions of a thing to a subset that is deemed lawful because of natural laws or human laws" (Wand & Weber, 1995).

Table 1, Continued: Basic concepts in the BWW model

BWW-event in a thing	"A change of state of a thing. It is affected via a transformation (see below)" (Wand & Weber, 1995).
BWW-process in a thing	"An intrinsically ordered sequence of events on, or states of, a thing" (Green, 1996). Processes may be either chains or trees of events (Bunge, 1977).
BWW-transformation of a thing	"A mapping from a domain comprising states to a co-domain comprising states" (Wand & Weber, 1995).
BWW-transformation law of a thing	"Events are governed by transformation laws that define the allowed changes of state" (Parsons & Wand, 1997). (Wand & Weber, 1995) and other papers on the BWW model instead introduce *BWW-lawful transformations*, which define "which events in a thing that are lawful". The term "transformation law" instead of "lawful transformation" is chosen here to emphasise that a transformation law — like a state law — is a property of a particular thing.
BWW-law property of a thing	"Properties can be restricted by laws relating to one or several properties" (Parsons & Wand, 1997). 1) A law is either a state law or a transformation law of a particular thing. 2) A law is either a natural law or a human law (see below.)
BWW-natural law	"Natural laws are established by nature" (Weber, 1997). For example, a law of physics.
BWW-human law	"Some laws are human-made artifacts" (Weber, 1997), i.e., they are socially constructed and enforced by humans. Events and processes may sometimes *violate* human laws, but not natural ones.
BWW-natural kind of things	"A natural kind is defined by a set of properties and the laws connecting them" (Parsons & Wand, 1997). 1) Hence, a BWW-natural kind is itself a BWW-class, but all its characteristic properties must be BWW-laws. 2) In this paper, we refer to the "subclasses" of BWW-natural kinds as *BWW–sub-kinds*.
BWW-conceivable state	"The set of all states that the thing may ever assume" (Wand & Weber, 1995).
BWW-possible state space of a thing	"[T]he space of states that are possible given our understanding of the laws of nature" (Weber, 1997).
BWW-lawful state space of a thing space of a thing	"[T]he set of states of a thing that comply with the state laws of the thing" (Wand & Weber, 1995). Hence, lawful states satisfy both human and natural state laws, whereas possible states may violate human ones.
BWW-conceivable event space of a thing	"The set of all possible events that can occur in the thing" (Weber & Zhang, 1996).

Table 1, Continued: Basic concepts in the BWW model

BWW-lawful event space of a thing	"The set of all events in a thing that are lawful" (Wand & Weber, 1995). Weber (1997) adds "[...] because (a) nature permits them to occur, and (b) there are no human laws that denote them as unlawful".
BWW-composite thing	"A composite thing may be made up of other things (composite or primitive)" (Wand & Weber, 1995). "Things can be combined to form a composite thing" (Parsons & Wand, 1997).
BWW-component thing	Any BWW-thing that is in the composition of a composite thing.
BWW–whole-part relation	The property of being in the composition of another thing or, complementary, of having another thing as a component (according to Bunge, 1977)).
BWW-resultant property of a composite thing	"A property of a composite thing that belongs to a component thing" (Wand & Weber, 1995).
BWW-emergent property of a composite thing	A property of a composite thing that does not belong to a component thing (adapted from (Wand & Weber, 1995).)
BWW-history of a thing	"The chronologically ordered states that a thing traverses in time" (Weber & Zhang, 1996).
BWW-acting on another thing, BWW-coupling of things	"A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled [...]" (Wand & Weber, 1995).
BWW-direct acting on, BWW-binding mutual property	A thing acts *directly* on one or more other things when the former thing changes a *BWW-binding mutual property* they all possess. Changing the binding mutual property is an internal event in the former thing and an external event in each of the latter things.
BWW-system of things	"A set of things is a system if, for any bi-partitioning of the set, couplings exist among things in the two subsets" (Wand & Weber, 1995). 1) A BWW-system is itself a BWW-thing. 2) BWW-system things belong to BWW-system natural kinds.
BWW-system composition	"The things in the system" (Wand & Weber, 1995), i.e., its component things.
BWW-system environment	"Things that are not in the system but interact with things in the system" (Wand & Weber, 1995).
BWW-system structure	"The set of couplings that exist among things in the system and things in the environment of the system" (Wand & Weber, 1995).
BWW-subsystem	"A system whose composition and structure are subsets of the composition and structure of another system" (Wand & Weber, 1995).

Table 1, Continued: Basic concepts in the BWW model

BWW-system decomposition	"A set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition" (Wand & Weber, 1995).
BWW-level structure	"Defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself" (Wand & Weber, 1995).
BWW-external event in a thing, subsystem or system	"An event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment of the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable" (see below) (Wand & Weber, 1995). 1) Stable and unstable states will be defined below. 2) We have not defined the subsystem-concept because we do not need it in this paper.
BWW-internal event in a thing, subsystem or system	"An event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable" (see below) (Wand & Weber, 1995).
BWW-unstable state of a thing	"A state that will be changed into another state by virtue of the action of transformation in the system" (Wand & Weber, 1995).
BWW-stable state of a thing	"A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event)" (Wand & Weber, 1995).

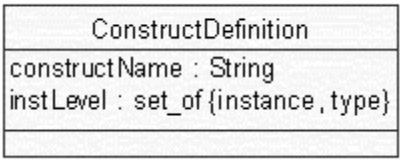
states, events, histories, etc.).

An obvious example is the distinction between UML-types at the **type** level and UML-objects at the **instance** level.[1] (Table 2 will later show template-based definitions of all the modelling constructs from the UML that were considered in this paper. As can be seen from this table, the definitions of UML-objects and –types only differ in their **instantiation level** entries.)

Some constructs can even be used to represent ***either*** level. For example, *processes* in some dialects of traditional dataflow diagrams (DFDs) can represent either *logical processes* that can occur several places in an information system — and are therefore at the type level — or *physical processes* that can occur only at a specific place — and therefore belong at the instance level.

Figure 1 shows the first part of a UML class diagram for the template, according to which a ***"ConstructDefinition"*** has a ***"constructName"*** and an ***"instLevel"*** as attributes. (In the UML, when multiplicities are not shown for attributes, the default is [1,1], so each ***"ConstructDefinition"*** has exactly one ***"instLevel"*** attribute. In this

Figure 1: A UML class diagram of the instantiation level entr y.



and later UML class diagrams, we allow attributes with *set value types* .)

Class

The second type of entry is used to define which **class** of things the modelling construct may represent. For a modelling construct at the **type** level, this means that the construct may only represent *subclasses* of the specified class. For a modelling construct at the **instance** level, this means that the construct may only represent things that belong to the specified class.

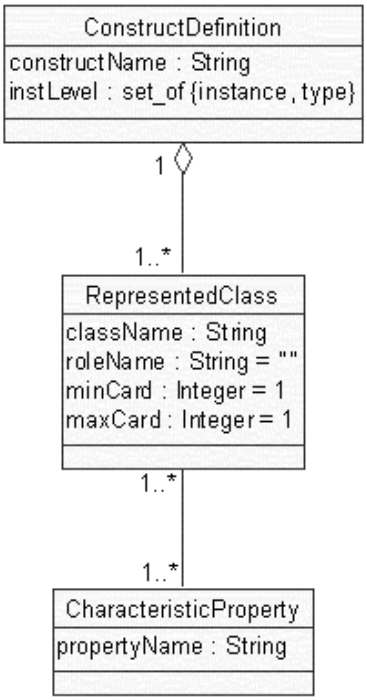
For example, at the **type** level, the UML has constructs that may only represent subclasses of the BWW-class of *"ActiveThings"*, defined by the characteristic property of *acting on other things*. The UML also has constructs that may only represent subclasses of other important BWW-classes, such as the class of *"ActedOnThings"* or the class of *"CompositeThings"*. (Figure 7 will later show a *generalisation hierarchy* of all the BWW-classes that are represented by modelling constructs from the UML.) Accordingly, at the **instance** level, the UML has constructs that may only represent *active* BWW-things, other constructs that may only represent things that are *acted on* and still other constructs that may only represent *composite* things.

These restrictions are important parts of the semantics of the modelling constructs in question. The **class** entry type is particularly important for defining domain-specific languages, which are often organised as deep specialisation hierarchies.[2] The entry type is also useful when comparing and analysing multiple models, because it makes it clear that some modelling constructs may never be used to represent the same classes or things in the problem domain.

Figure 2 extends the UML class diagram to show that a ***"ConstructDefinition"*** consists of one or more ***"RepresentedClasses"***, each of which is defined, according to the BWW model, by one or more ***"CharacteristicProperties"***.

According to Figure 2, the template allows *repeated* **class** entries for modelling constructs that may represent several different classes of things (at the **type**

Figure 2: The UML class diagram extended to show the class entry.



level) or several things of different classes (at the **instance** level). For example, for the moment, a UML-aggregation (at the **type** level) represents one *"composite (or whole) class"* and one *"component (or part) class"*.

Whenever a construct definition has more than one **class** entry, they must be distinguished by ***"roleNames"***, as shown in Figure 2. UML-aggregation is therefore defined by two ***"RepresentedClasses"***, one with the ***"roleName"*** *"whole"* and one with the ***"roleName"*** *"part"*. In this example, the two ***"RepresentedClasses"*** **represent** *different* BWW-classes (because they have different ***"CharacteristicProperties"***), but a construct definition may even have more than one ***"RepresentedClasses" for*** the same BWW-class as long as they have different ***"roleNames"***.

Even when the **class** entry is not repeated, a modelling construct may represent several subclasses of the same class in the same role (at the **type** level) or several things of the same class in the same role (at the **instance** level). In Figure 2, ***"RepresentedClass"*** therefore has minimum and maximum cardinalities, ***"minCard"*** and ***"maxCard"***, that default to 1. For example, a UML-link (at the **instance** level) represents two or more *"AssociatedThings"* and is therefore defined by a single ***"RepresentedClass"***, but one with ***"minCard"***=2 (because the link connects at least two UML-objects) and ***"maxCard"***= –1 (because –1 is used to indicate no upper limit on the number of objects.)

Properties

The third type of entry is used to define which **properties** of things the modelling construct may represent because, sometimes, different modelling constructs may represent the same class of things but not the same properties of those things. For example, in the UML the BWW-class of *"ActiveThings"* can be represented by UML-operations and UML–pre- and -postconditions, but each UML construct nevertheless represents slightly different *properties* of active BWW-things. An additional example is found in Conallen's (1999) extension of the UML for developing web applications. His extension introduces several *stereotypes* of UML-associations, one of them for representing web "hyperlinks", i.e., links that change the content of the web-browser frame that the links themselves are in, and another one for representing "target links", i.e., links that change the contents of *other* web-browser frames. These two stereotypes have identical **class** entries because they both represent pairs of web pages, but they have different **property** entries because they represent different properties of those pairs.

Figure 3 extends the UML class diagram to show that a ***"ConstructDefinition"*** also consists of zero or more ***"RepresentedProperties"***, which specialise the ***"Properties"*** that characterise ***"RepresentedClasses"***. (In Figure 2, we have therefore changed the name of the ***"CharacteristicProperty"*** class to ***"Property"*** and instead added ***"characteristic"*** as a role of the ***"Property"*** class.) The template allows ***"ConstructDefinitions"*** with *zero* ***"RepresentedProperties"*** because there are modelling constructs that may not represent properties at all. For example, general constructs like UML-object and -type do not have **property** entries because they do not themselves represent BWW-properties (although in the UML metamodel they are associated with other

Table 2 : Template-based definitions of modelling constructs from the UML.

UML construct	***Interpretation by Opdahl & Henderson-Sellers (2002)***	***'Instantiation level'*** *entry*	***'Class'*** *entry*	***'Property*** *entry*	***'Segment'*** *entry*
UML-object	BWW-thing	**'Instance'**	An instance of *'AllThings'*		**'Lifetime'**
UML-active object	BWW-thing that *acts on* other things	**'Instance'**	An instance of *'ActiveThings'*		**'Lifetime'**
UML-swimlane	BWW-thing that *acts on* other things	**'Instance'**	An instance of *'ActiveThings'*		**'Process'**
UML-actor	BWW-thing that *acts on* the proposed system thing	**'Instance'**	An instance of *'ThingsActingOnTheProposedSystem'*		**'Lifetime'**
UML-object lifeline	A segment of a BWW-history	**'Instance'**	An instance of *'ChangingThings'*		**'Process'**
UML-type	BWW-natural kind	**'Type'**	A subclass of *'AllThings'*		**'Lifetime'**
UML-supertype	BWW-natural kind that has a subkind	**'Type'**	A subclass of *'AllThings'*		**'Lifetime'**
UML-subtype	Subkind	**'Type'**	A *proper* subclass of *'AllThings'*		**'Lifetime'**
UML-generalization	Natural kind/subkind relationship	**'Type'**	A subclass of *'AllThings'* and a *proper* subclass of *'AllThings'*		**'Lifetime'**

Table 2, Cont. : Template-based definitions of modelling constructs from the UML.

UML-actor class	BWW-natural kind of things that *act on* the proposed system thing	**'Type'**	A subclass of *'ThingsActingOnTheProposedSystem'*		**'Lifetime'**
UML-active class	BWW-natural kind of things that act on other things	**'Type'**	A subclass of *'ActiveThings'*		**'Lifetime'**
UML-property [of an object]	BWW-intrinsic property [of a thing] that is *not* a law or a whole-part relation, but that can be either resultant or emergent. BWW-intrinsic *complex* property (if the UML-property has a non-primitive type)	**'Instance'**	An instance of *'AllThings'*	Any intrinsic non-law property that is not a whole-part relation	**'Lifetime'**
UML-attribute [of a class]	BWW-characteristic intrinsic property [that defines a natural kind and] that is *not* a law or a whole-part relation, but that can be either resultant or emergent	**'Type'**	A subclass of *'AllThings'*	Any characteristic intrinsic non-law property that is not a whole-part relation	**'Lifetime'**
UML-multiplicity	BWW-characteristic state law about how many BWW-properties that a thing can possess.	**'Type'**	A subclass of *'AllThings'*	Any characteristic state law about how many BWW-properties the thing can possess	**'Lifetime'**
UML-datatype	BWW-codomain of a property function	**'Type'**	A subclass of *'AllThings'*	Any codomain of a characteristic intrinsic non-law property that is not a whole-part relation	**'Lifetime'**

Table 2, Cont. : Template-based definitions of modelling constructs from the UML.

UML-operation	BWW-transformation	**'Type'**	A subclass of *'ActiveThings'*	Any characteristic complex intrinsic law consisting of one state law and one transformation law	**'Lifetime'**
UML-precondition	Subtype of BWW-characteristic intrinsic state law	**'Type'**	A subclass of *'ActiveThings'*	Any characteristic intrinsic state law	**'Lifetime'**
UML-postcondition	Subtype of BWW-characteristic intrinsic transformation law.	**'Type'**	A subclass of *'ActiveThings'*	Any characteristic intrinsic transformation law	**'Lifetime'**
UML-responsibility [of a class]	Subtype of BWW-characteristic complex law property, but with very weak semantics in the UML	**'Type'**	A subclass of *'InteractingThings'*	Any complex law	**'Lifetime'**
UML-link	BWW-mutual property of two or more things	**'Instance'**	Two or more instances of *'AssociatedThings'*	Any mutual non-law property	**'State'**
UML-link end	One or more BWW-state laws about the BWW-mutual property in the above interpretation. (The link end may also represent that there are no such state laws.)	**'Instance'**	An instance of *'AssociatedThings'*	Any intrinsic state law about a mutual property (that is represented as a UML-link)	**'Lifetime'**
UML-association	BWW-characteristic mutual property	**'Type'**	Two or more subclasses of *'AssociatedThings'*	Any characteristic mutual non-law property	**'Lifetime'**

Table 2, Cont. : Template-based definitions of modelling constructs from the UML.

UML-link object	BWW-composite thing with one or more intrinsic properties, all of whose component things possess the same mutual property	**'Instance'**	An instance of *'CompositeThings'* and two or more instances of *'ComponentThings' and 'AssociatedThings'* (that are part of the composite thing and that are associated with one another)	One or more intrinsic properties of the composite thing, two or more whole-part relations and a mutual property of all the component things	**'Lifetime'**
UML-association class	BWW-natural kind (of composite things) with one or more characteristic intrinsic properties, all of whose component kinds are defined (in part) by the same characteristic mutual property	**'Type'**	A subclass of *'CompositeThings'* and one or more subclasses of *'ComponentThings' and 'AssociatedThings'* (that are parts of the composite class)	One or more characterstic intrinsic properties of the composite class, one or more characteristic whole-part relation and a characterstic mutual property of all the component classes	**'Lifetime'**
UML-communication association	BWW-characteristic *binding* mutual property	**'Type'**	One or more subclasses of *'CoupledThings'*	A characteristic binding mutual property of all the subclasses	**'Lifetime'**
UML-aggregate	BWW-composite thing	**'Instance'**	Ab instance of *'CompositeThings'*	Any whole-part relation	**'Lifetime'**
UML-aggregate class	BWW-natural kind of composite things	**'Type'**	A subclass of *'CompositeThings'*	Any characteristic whole-part relation	**'Lifetime'**
UML-aggregation	BWW–whole-part relation where the whole is *not* a system	**'Type'**	A subclass of *'CompositeThings'* and a subclass of *'ComponentThings'*	Any characteristic whole-part relation	**'Lifetime'**

Table 2, Cont. : Template-based definitions of modelling constructs from the UML.

UML-composition, composite aggregation	BWW–whole-part relation where the whole *is* a system	**'Type'**	A subclass of *'SystemThings'* and a subclass of *'SystemComponentThings'*	Any characteristic whole-part relation	**'Lifetime'**
UML-container (1)	Subtype of BWW-thing	**'Instance'**	An instance of *'ContainerThings'*		**'Lifetime'**
UML-physical system	BWW-system composition	**'Instance'**	Two or more instances of *'SystemComponentThings'*		**'Lifetime'**
UML-state	BWW-state	**'Instance'**	An instance of *'ChangingThings'*	One or more non-law properties	**'State'**
UML-object flow state	Subtype of BWW-state of a BWW-thing when it is *acted on* by one or more other things	**'Instance'**	An instance of *'CommunicatedThings'*	A binding mutual property that is changed by a transformation law of another thing	**'State'**
UML-event	BWW-event	**'Instance'**	An instance of *'ChangingThings'*	One or more non-law properties	**'Event'**
UML-transition	BWW-transformation law that describes a single event.	**'Type'**	A subclass of *'ActiveThings'*	Any characteristic intrinsic transformation law	**'Lifetime'**
UML-guard condition	Subtype of characteristic and intrinsic BWW-state law	**'Type'**	A subclass of *'ActiveThings'*	Any characteristic intrinsic state law	**'Lifetime'**

Table 2, Cont. : Template-based definitions of modelling constructs from the UML.

UML-transition firing (or -fire)	BWW-event. BWW-process	**'Instance'**	A subclass of *'ActiveThings'*	A intrinsic transformation law and one or more non-law properties that are changed by the transformation law	**'Event'** or **'process'**
UML-action	BWW-transformation law that describes a single event	**'Type'**	A subclass of *'ChangingThings'*	A characteristic intrinsic transformation law	**'Lifetime'**
UML-action sequence	BWW-transformation law that describes a process	**'Type'**	A subclass of *'ChangingThings'*	A characteristic intrinsic transformation law	**'Lifetime'**
UML-activation	BWW-event. BWW-process (if the UML-action is a sequence)	**'Instance'**	An instance of *'ChangingThings'*	An intrinsic transformation law	**'Event'** or **'Process'**
UML-interaction	BWW-coupled event, i.e., when an event in one thing changes a BWW-binding mutual property and thereby causes an external event in another thing	**'Instance'**	An instance of *'ActiveThings'* and an instance of *'ActedOnThings'*	Any binding mutual property	**'Event'**
UML-message	BWW-binding mutual property	**'Instance'**	An instance of *'ActiveThings'* and an instance of *'ActedOnThings'*	A binding mutual property	**'Event'**
UML-sender [object]	BWW-thing that *acts on* other things	**'Instance'**	An instance of *'ActiveThings'*		**'Lifetime'**
UML-receiver [object]	BWW-thing that is *acted on* by other things	**'Instance'**	An instance of *'ActedOnThings'*		**'Lifetime'**

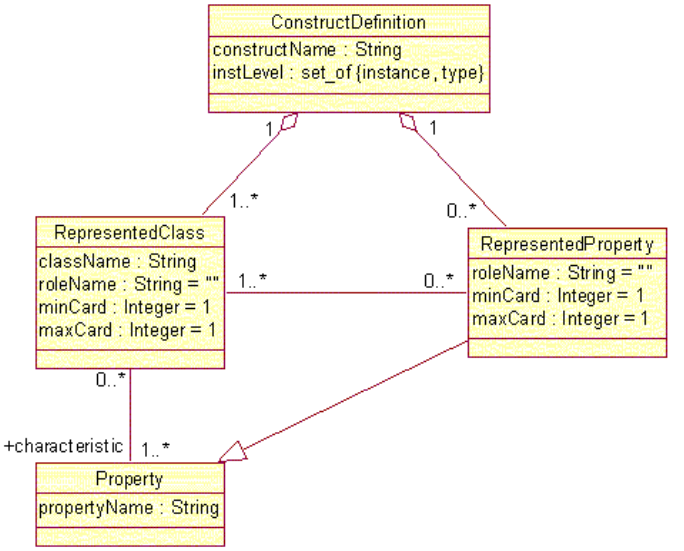
Table 2, Cont. : Template-based definitions of modelling constructs from the UML.

UML-focus of control	Sequence of BWW-unstable states in a thing	**'Instance'**	An instance of *'ChangingThings'*	One or more non-law properties	**'Process'**
UML-use case instance	BWW-process in the proposed system thing or in a subsystem thereof	**'Instance'**	An instance of *'ProposedSystemThings'* or *'SubsystemsOfTheProposedSystem'*	One or more non-law properties	**'Process'**
UML-use case class	A group of BWW-processes in the proposed system thing or in a subsystem thereof	**'Type'**	A subclass of *'ProposedSystemThings'* or *'SubsystemsOfTheProposedSystem'*	One or more characteristic non-law properties	**'Process'**
UML-extend	Subtype of BWW–whole-part relation or binding mutual property, but with very weak semantics in the UML. (We propose that UML-extend should only be used to represent whole-part relations)	**'Type'**	A subclass of *'SystemThings'* and a subclass of *'SystemComponentThings'* (proposed)	A characteristic whole-part relation (proposed)	**'Lifetime'**
UML-include	Subtype of BWW–whole-part relation or binding mutual property, but with very weak semantics in the UML. (We propose that UML-extend should only be used to represent whole-part relations)	**'Type'**	A subclass of *'SystemThings'* and a subclass of *'SystemComponentThings'* (proposed)	A characteristic whole-part relation (proposed)	**'Lifetime'**
UML-scenario	BWW-process in the proposed system thing or in a subsystem thereof	**'Instance'**	An instance of *'ProposedSystemThings'* or of *'SubsystemsOfTheProposedSystem'*	One or more non-law properties	**'Process'**

Table 2, Cont. : Template-based definitions of modelling constructs from the UML.

UML-timing mark	Element in the domain of any BWW-property function	**'Instance'**	An instance of *'ThingsThatKnowAbsoluteTime'*	Any element in the domain of an intrinsic non-law property that is not a whole-part relation	**'Lifetime'**
UML-time event	Subtype of BWW-event	**'Instance'**	An instance of *'ThingsThatTrackTime'*	Any intrinsic non-law property that is not a whole-part relation	**'Event'**

Figure 3: The UML class diagram extended to show the property entry.



constructs that do so).

However, most modelling constructs have at least one **property** entry. In the languages we have studied, many of the **property** entries we encountered were not very restrictive and might represent *any regular property*, i.e., any intrinsic non-law BWW-property that is not a whole-part relation. We expect to find more examples of more restrictive **property** entries in less general, domain-specific modelling languages.

As for classes, the template allows *repeated* **property** entries for modelling constructs that may represent several characteristic properties of classes (at the **type** level) or several properties of things (at the **instance** level). Whenever a ***"ConstructDefinition"*** has more than one **property** entry, they must be distinguished by ***"roleNames"***, as shown in Figure 3. A ***"ConstructDefinition"*** may have more than one ***"RepresentedProperty"*** that specialises the same ***"Property"***, as long as they have different ***"roleNames"***. Because the **class** entry may also be repeated, each ***"RepresentedProperty"*** is associated with one or more ***"RepresentedClass"*** to specify exactly to which class the property belongs in the definition.

Even when the **property** entry is not repeated, a modelling construct may represent the same property several times in the same role. In Figure 3, each ***"RepresentedProperty"*** therefore has a minimum and maximum cardinality, ***"minCard"*** and ***"maxCard"*** that default to 1.

In the BWW model, a property may be *mutual*, i.e., it may belong to more than one thing. In the template, the corresponding ***"RepresentedProperty"*** would be associated with more than one ***"RepresentedClass"***. In fact, this was the case in several of the earlier examples. In the web application example, both "hyperlinks" and "target links" represented (different) mutual properties, each of them belonging to two "web-page" things, one playing the role of "source" and the other

playing "target". In the UML-link example, UML-links represented a mutual property that belonged to two or more BWW-things. Finally, in the UML-aggregation example, the aggregation represented a property (the whole-part relation) that belonged both to the aggregate (whole) class and the component (part) class. The two ***"RepresentedClasses" used to define*** UML-aggregation, i.e., the "whole" and the "part" class, were therefore both linked to the same ***"RepresentedProperty"*** with ***"propertyName"*** "whole-part relation".

This concludes the discussion of **property** entries in the template. Together, the **class** and **property** entries form the core of the template. The two types of entries fit nicely with Bunge's view of the world as composed of things and properties, in terms of which the other BWW concepts are defined.

Ontological Descriptions of Properties

The BWW model has concepts that describe properties in even greater detail and that are also used in the template. Figure 4 extends the UML class diagram to show the additional attributes of ***"Properties"*** and ***"RepresentedProperties"***.

Most importantly, according to the BWW model, a ***"RepresentedProperty"*** has an attribute that defines whether the modelling construct represents (a) the *property per se* , i.e., the BWW-property itself; (b) the *property datatype*, i.e., the BWW-property co-domain; (c) a *property value*, i.e., a value in the BWW-property co-domain; or (d) some combination of these. For example, this attribute distinguishes between constructs like UML-attribute and -datatype, which are both at the **type** level and may represent any subclass of *"AllThings"* and any characteristic intrinsic property that is not a whole-part relation, because UML-attributes represent *properties per se* , whereas UML-datatypes of course represent *datatypes*. Accordingly, this attribute also distinguishes between UML-property and -value. UML-datatypes and -values are the only UML constructs we have encountered that represent *property datatypes* and *values*, but the attribute may be useful for languages with datatypes that can only be used in connection with certain modelling constructs. We have not found a use for identifying constructs that represent *BWW-property functions* explicitly, leaving this as a possible future extension.

A ***"Property"*** has an attribute that defines whether the modelling construct represents a *non-law*, a *state law* or a *transition law* according to the BWW model. For example, whereas a UML-attribute represents a non-law, a UML-operation represents a law. A ***"Property"*** that is a law is described by an ***"oclExpression"***. A ***"Property"*** also has an attribute that defines whether the modelling construct represents a *whole-part relation* or not according to the BWW model. For example, whereas a UML-association cannot represent a whole-part relation, a UML-aggregation must do so.

Figure 4 also shows the additional attributes of the ***"ClassPropertyAssociation"*** class. The first of these defines whether the ***"RepresentedProperty"*** is *intrinsic*, *non-binding mutual* or *binding mutual* with respect to a particular ***"RepresentedClass"***. To see why it is necessary to define this attribute in an association class, rather than in the ***"RepresentedProperty"*** class, consider the following hypothetical example. A

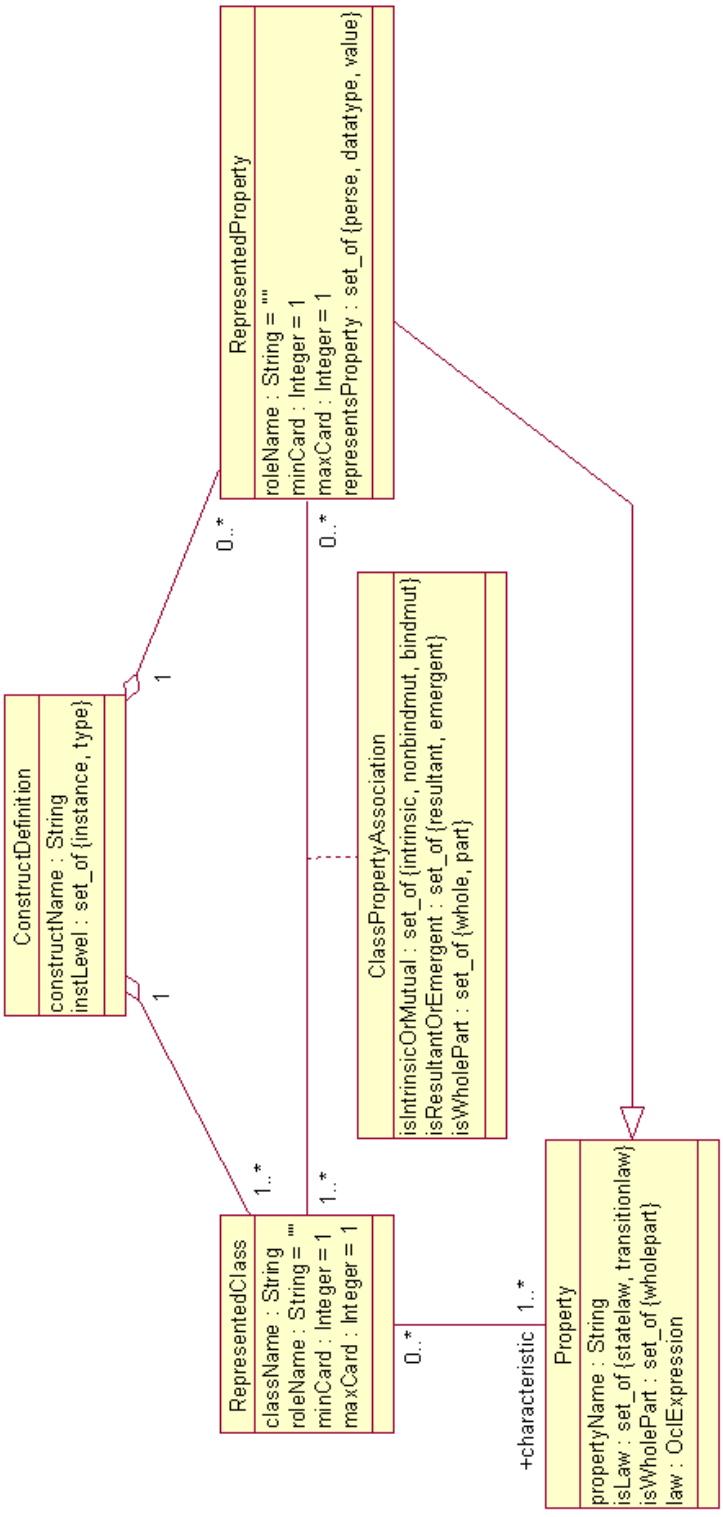


Figure 4: The UML class diagram extended to show the ontological descriptions of properties.

modelling construct represents a property that belongs to three things. Two of these things are parts of the third. The property is mutual between the two part things and is also a resultant property of the whole. In this case, the property is intrinsic with respect to the whole thing, but it is mutual with respect to the two part things. This explains why the ***"isIntrinsicOrMutual"*** attribute must be defined in an association class. Although the example is hypothetical, the situation it describes is not uncommon and it is not impossible that some modelling language may have a dedicated construct for it.

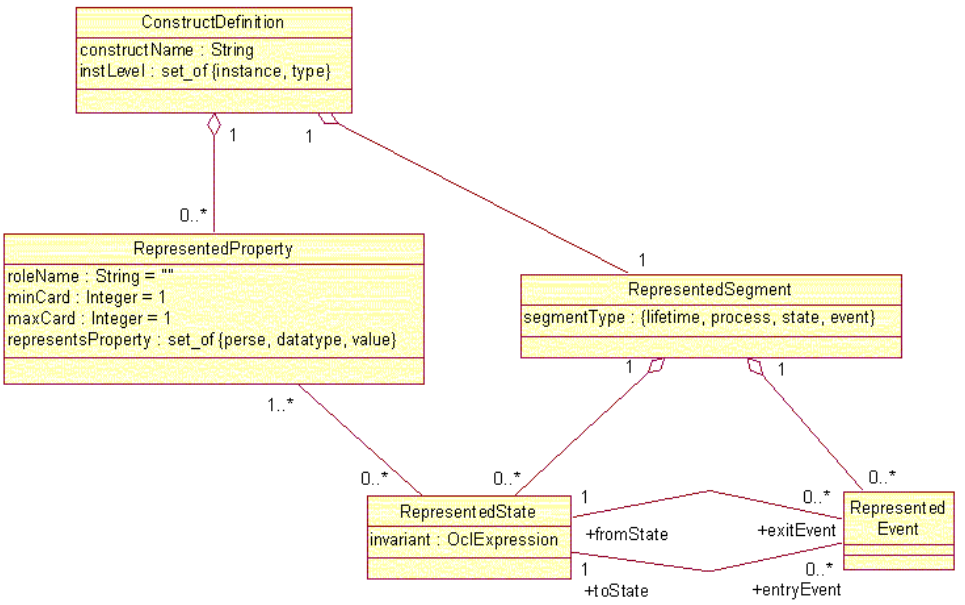
A ***"ClassPropertyAssociation"*** has another attribute that defines whether the ***"RepresentedProperty"*** is *resultant*, *emergent* or *neither* with respect to the ***"RepresentedClass"***. A final attribute applies only to ***"RepresentedProperties"*** that are whole-part relations and defines whether the ***"RepresentedClass"*** is the *whole* or the *part* in the relation or *neither*.

We need further experience with the template to determine how useful these ontologically motivated attributes of ***"Properties"*** and ***"ClassPropertyAssociations"*** are. Also, the template does not make use of Bunge's (1997) distinction between *permanent* and *variable* BWW-properties, leaving this as a possible future extension.

Lifetimes

The fourth type of entry is used to define which part of the **lifetime** of a thing that the modelling construct may represent because, sometimes, different modelling constructs may represent the same class of things and the same properties of those things but different segments of the *lifetimes* of those things. For example, one construct may represent an *event*, another a *state* and a third a *process,* although all three constructs represent the same property of the same thing. This becomes obvious when we see that constructs that are as different as UML-state and UML-

Figure 5: The UML class diagram extended to show the lifetime entry.



event have identical **instantiation level**, **class** and **property** entries. Both constructs represent the **type** level, may represent any subclass of the class of ***"ChangingThings"*** and may represent any non-law properties of those subclasses. However, they are distinguished by their **lifetime** entries.

Figure 5 extends the UML class diagram to show ***"RepresentedSegments"*** of the lifetimes of things and classes. A ***"ConstructDefinition"*** has exactly one ***"RepresentedSegment"***, which is either the whole ***"lifetime"*** of the thing or class, a ***"process"***, a ***"state"*** or an ***"event"***. ***"RepresentedSegments"*** that are ***"states"*** or ***"events"*** must also have a ***"RepresentedState"*** and/or a ***"RepresentedEvent"*** as parts. A ***"RepresentedState"*** is described by an ***"oclExpression"*** that involves ***"RepresentedProperties"***. A ***"RepresentedEvent"*** is defined in terms of its ***"from-"*** and ***"toStates"***. BWW-processes are represented as chains of ***"RepresentedStates"*** and ***"-Events".***

Building Taxonomies

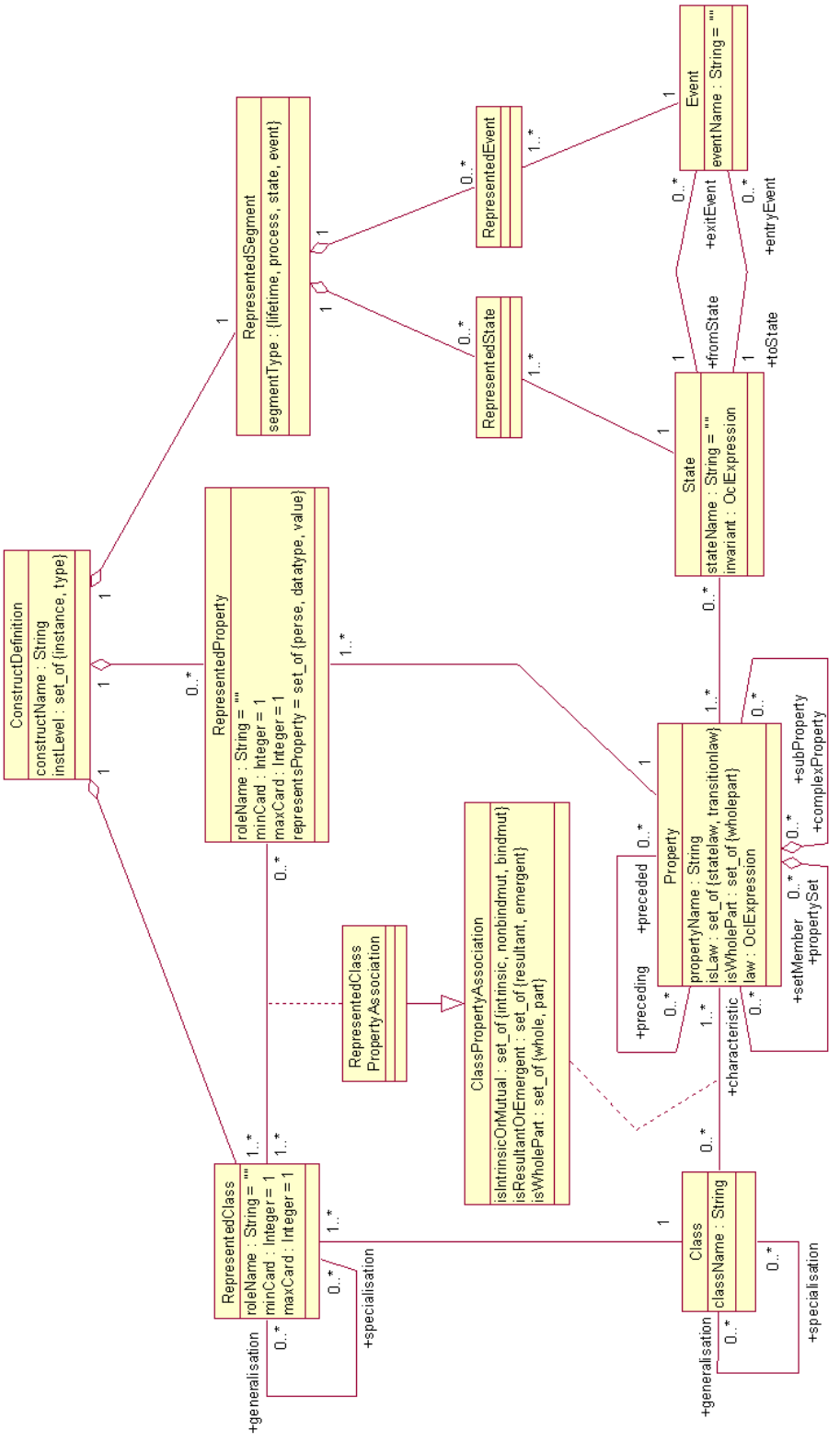
As the template is used to define an increasing number of constructs from different modelling languages, the number of ***"RepresentedProperties"*** and ***"Properties"*** used to characterise ***"RepresentedClasses"*** will grow large. In this situation, it is essential that the same classes and properties are not represented independently several times using the template. Should this happen, the template would no longer aid in identifying modelling constructs that can be used to represent the same BWW-classes and -properties

The template should therefore be supported by a tool for defining modelling constructs, a tool that would also support building and maintaining a *taxonomy* of BWW-classes and -properties. Whenever a new modelling construct is defined using the template, this tool should provide overviews of the classes and properties that have been used in previous definitions so they can be reused. Whenever the new modelling construct necessitates defining new BWW-classes or -properties, they should be entered into the taxonomy.

Figure 6 extends the UML class diagram to show the taxonomy of ***"Classes"*** and ***"Properties" that*** can be reused in ***"ConstructDefinitions"***. Because ***"Classes"*** and ***"Properties" in the taxonomy are stored independently of "ConstructDefinitions", they are now connected to "RepresentedClasses" and "RepresentedProperties"*** via *association* rather than via *generalisation/ specialisation*.

In order to make the taxonomy more easy to use, both ***"Classes"*** and ***"RepresentedClasses"*** are organised in ***"generalisation"*** hierarchies, whereas ***"Properties"*** are organised through ***"precedence"*** relations and hierarchies of ***"complexProperties"*** and ***"subProperties"***, all according to the BWW model. A ***"propertySet"*** association is defined for ***"ConstructDefinitions"*** that may represent a choice of more than one ***"Property".***

Figure 6 also shows that ***"States"*** and ***"Events"*** have been added to the taxonomy. Although less critical than classes and their properties, states and events also make the template easier to use, because the **lifetime** entries of new modelling constructs can sometimes be defined by reuse, and more useful, because the construct definitions become easier to compare.


ConstructDefinition
constructName : String
instLevel : set_of {instance, type}
RepresentedSegment
segmentType : {lifetime, process, state, event}
RepresentedProperty
roleName : String = ""
minCard : Integer = 1
maxCard : Integer = 1
representsProperty = set_of {perse, datatype, value}
RepresentedState
RepresentedEvent
State
stateName : String = ""
invariant : OclExpression
Event
eventName : String = ""
+fromState
+toState
+exitEvent
+entryEvent
RepresentedClass
roleName : String = ""
minCard : Integer = 1
maxCard : Integer = 1
RepresentedClass
PropertyAssociation
ClassPropertyAssociation
isIntrinsicOrMutual : set_of {intrinsic, nonbindmut, bindmut}
isResultantOrEmergent : set_of {resultant, emergent}
isWholePart : set_of {whole, part}
Property
propertyName : String
isLaw : set_of {statelaw, transitionlaw}
isWholePart : set_of {wholepart}
law : OclExpression
+preceding
+preceded
+setMember
+propertySet
+subProperty
+complexProperty
+characteristic
Class
className : String
+generalisation
+specialisation

RESULTS

Modelling constructs from the Object Management Group's (OMG) standard Unified Modeling Language (UML) (OMG, 2001) were used in several examples so far. In part, the examples were based on experience from analysing the UML in terms of the BWW model in (Opdahl & Henderson-Sellers, 2002) and from analysing a variant of UML, the OPEN Modelling Language (OML) (Firesmith, Henderson-Sellers & Graham, 1997), in Opdahl, Henderson-Sellers and Barbier (1999) and Opdahl and Henderson-Sellers (2001). On the one hand, the UML is similar to the BWW model because it provides constructs that match key ontological concepts such as BWW-things (UML-objects), BWW-properties (many UML-features) and BWW-classes (UML-concrete classes).[3] On the other hand, a closer look at the UML definition in (OMG, 2001) reveals numerous problems, many of which are resolved by the template. This section will present results of using the template to define constructs from the UML. Of course, the most important result of using the template on the UML, that of integrating the UML with other modelling languages, cannot be illustrated at this stage of our work (because the UML is the first language we have presented results of analysing in this detail).

The Generalisation Hierarchy of UML Constructs

An important outcome of using the template on the UML is a *generalisation hierarchy* of BWW-classes that has emerged from the **class** entries for the UML constructs we have analysed, as shown in Figure 7. This hierarchy shows which BWW-classes in concrete problem domains are recognised by the UML. Each BWW-class in Figure 7 has been annotated with a list of those UML constructs that represent the class. The lists are based on the analysis by Opdahl & Henderson-Sellers (2002), which interpreted 68 UML constructs that were relevant for representing concrete problem domains in terms of the BWW model. In this paper, 58 of these constructs have been defined using the template, as indicated in Table 2, although it is not the purpose of this paper to present our definitions in full detail. Ten UML constructs from Opdahl and Henderson-Sellers (2002) were left out because they, on closer inspection, turned out to be less relevant for modelling concrete problem domains or because they were subtypes of other constructs.[4] Also, because the UML has weak semantics in relation to concrete problem domains today, many of the definitions are interpretations and proposals that must be evaluated in further work.

Generalisation hierarchies is a new and interesting approach to analysing, evaluating and comparing modelling languages. As a contribution to the analysis of the UML, Figure 7 shows clearly which types of model elements that may *overlap* with one another and may therefore be inconsistent. As a contribution to the evaluation of the UML, Figure 7 presents a generalisation hierarchy that is an alternative to the one in OMG (2001) and to which the UML metamodel may therefore be compared.

As pointed out by Opdahl & Henderson-Sellers (2002), the modelling constructs in the UML have been defined to play several different *roles*, often at the same time, such as representing proposed software solutions, supporting the development process and matching with other modelling constructs to create a well-

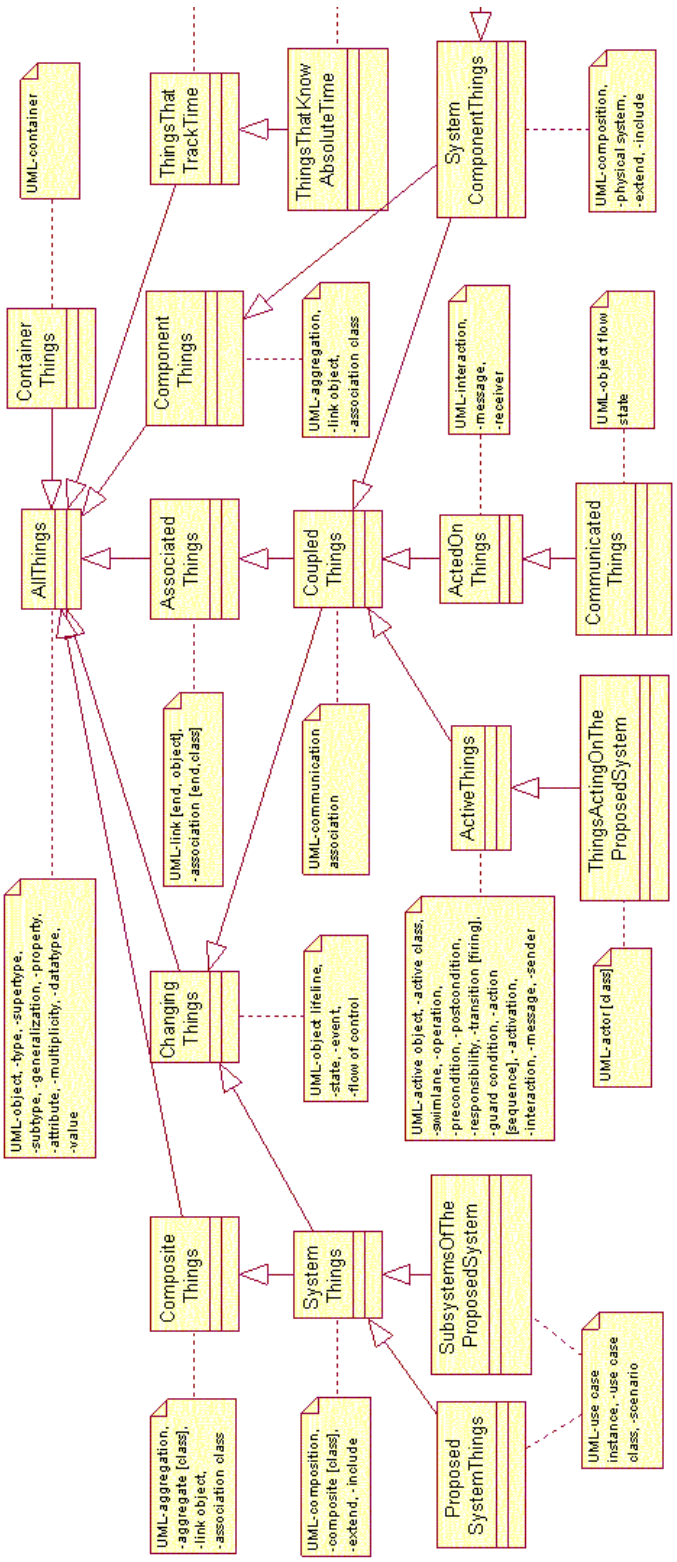


Figure 7: A generalisation hierarchy of all the BWW-classes that are represented by modelling constructs from the UML. The classes shown are the ones used to fill in the class entries for the UML constructs we have analysed.

defined, compact and tightly integrated modelling language. This attempt to satisfy many different roles at the same time has made the generalisation hierarchies in the UML metamodel very hard to comprehend. Figure 7 offers a clear-cut alternative based on the constructs' semantics in relation to concrete problem domains. As a contribution to comparing modelling languages, we have already informally compared Figure 7 to generalisation hierarchies that have emerged from other languages we have analysed, such as the OML (Opdahl, Henderson-Sellers & Barbier, 1999; Opdahl & Henderson-Sellers, 2001). These hierarchies have many similarities and point towards a common generalisation hierarchy that can serve as a common ground for semantic language integration and which can be extended gradually as more languages are defined using the template. As in the BWW model, the generalisation hierarchy is likely to be a multi-inheritance graph rather than a single-inheritance tree.

Precise Definitions of UML Constructs in Terms of Concrete Problem Domains

Among the different roles mentioned in the previous section, we have found that UML constructs are often defined in terms of the proposed software system and not in terms of the problem domain (Wand & Weber, 1989; Parsons & Wand, 1997), i.e., the definition has given priority to the role of representing proposed software solutions over the role of representing the problem domain. For example, the UML glossary (OMG, 2001, appendix B) defines a UML-action as "The *specification* of an [executable statement] that forms an *abstraction* of a [computational procedure.] An action typically results in a change in the state of the system, and can be *realized* by [sending a message] to an object or modifying a link or a value of an attribute." (The [brackets] and *emphases* are ours.) In this definition, the bracketed terms all refer to the proposed software solution and not to the problem domain. As with many constructs in the UML, the construct thereby becomes harder to use in practical modelling situations and the resulting models become less precise. Using the template, UML-action can be precisely defined in terms of concrete problem domains, as representing the **type** level, the **class** of ***"ChangingThings"***, any **property** that is a characteristic and intrinsic BWW-transformation law that describes a single event and the **lifetime** of the ***"ChangingThings"***. (It describes the **lifetime** because the things possess the transformation law for as long as they exist, even though the law itself only describes an **event**.)

In the above definition, there are even emphasised terms that all support the development process and do not refer to the problem domain. Definitions that mix software, development and problem domain issues are confusing and detrimental to the learnability and usability of the language. Mixed definitions also weaken the semantics of the language, because when unrelated terms are mixed in the same definition, the result is several incomplete definitions instead of a single coherent one. For example, a UML-object is defined using a mixture of software (our [brackets]) and problem domain (our {braces}) issues as "An [{entity}] with a well—defined {boundary} and [identity] that [encapsulates] [{state}] and [{behavior}]." This definition is incoherent, because it is not at all clear what a "well-defined boundary" or the concept of "encapsulation" (at least in the likely interpretation of it as meaning information

hiding) mean in relation to concrete problem domains. Again, this problem is avoided using the template, according to which UML-object is precisely defined as representing the **instance** level, the **class** of ***"AllThings"***, *not* representing **properties** of those things but representing their whole **lifetimes**.

For another example, the glossary defines UML-reception as a "declaration that a classifier is prepared to react to the receipt of a signal", but it is unclear whether this "declaration" is static or dynamic, because "declaration" is left undefined. The template would have avoided this ambiguity, because the **lifetime** entry explicitly requests this to be defined.

Finally, using the template helps avoiding circular definitions, which Castellani (1998) demonstrates to be common in the UML, because each construct definition involves only filling in the standard set of entries, and none of the entries allow references to other construct definitions. The **instantiation** level, **ontological** and **lifetime** entries are all specified in terms of a limited number of attributes with a limited set of possible values for each. Circularities cannot occur in **class** entries either, because ***"Classes"*** are defined only in terms of ***"Properties"*** and never refer to other ***"Classes"***. The only place where circularities can potentially occur is therefore in **property** entries, where ***"Properties"*** may refer to other ***"Properties"*** through *precedence relations*. However, precedence circularities can easily be avoided by manual checking or by simple tool support.

DISCUSSION

The main idea behind the template was to provide a standard way of defining enterprise and IS modelling constructs in terms of the BWW model, in order to make the definitions cohesive and, thus, learnable, understandable and as directly comparable to one another as possible. When all construct definitions are directly comparable, it becomes easier to translate models from one language to another. It also becomes easier to detect models and model elements, possibly expressed in different languages, that may *overlap* with one another and may therefore be inconsistent. Another important idea was to provide a way of defining modelling constructs not only generally in terms of whether they represent "classes", "properties" or other ontological concepts, but also in terms of *which* classes and/or properties they represent. As we have seen, this additional level of detail was necessary to differentiate important modelling constructs in the UML. The additional level of detail also made the definitions more clearly and precisely related to the enterprise.

Clearly and precisely defined modelling constructs better support several of the *quality features* discussed in Lindland, Sindre and Sølvberg (1994) and Krogstie, Lindland and Sindre (1995), and languages become easier to learn, comprehend and use. Also, *semantic overlaps* (Spanoudakis & Finkelstein, 1998, 1999) (or *construct redundancies* (Wand & Weber, 1993)) and semantic omissions in languages (or *construct deficits* (Wand & Weber, 1993)) can be detected more easily and more precisely, along with redundant constructs, i.e., constructs that do not refer to anything in the problem domain. At the model level, inconsistencies, conflicts and omissions in models can be detected more easily and precisely. There is also less scope for misunderstanding of the resulting requirements.

The template supplements other contributions that make the BWW model more precise and useful: 1) Wand and Weber (1995) have provided a tabular description of the main concepts in their model, from which Table 1 in this paper was derived. 2) Wand and Weber (1990) have also provided a set-theoretic formulation of the BWW model. 3) Recently, Rosemann and Green (2002) have proposed an extended ER model of the main concepts in the BWW model. These contributions each make the BWW model more precise, but they do not 1) provide an obvious *standard* way of defining modelling constructs so that different definitions are directly comparable, nor do they 2) provide a way of saying that a modelling construct represents, e.g., a *specific* class or a *specific* property.

The template has been illustrated with definitions of constructs from the UML and, thereby, also supplements other contributions that use the BWW model to analyse the UML. Evermann and Wand (2001) present ontology-based rules for using the UML to model the real world, whereas Opdahl & Henderson-Sellers (2002) use the BWW model to analyse and evaluate the UML as a language for representing concrete problem domains. However, in contrast to this paper, neither contribution addresses in detail the question of how to *define* modelling constructs in relation to the BWW model.

CONCLUSIONS AND FURTHER WORK

The paper has explained the need for a standard way of defining modelling constructs from different enterprise modelling languages and has proposed a template for defining enterprise modelling constructs in a way that facilitates language integration. The template was based on the Bunge-Wand-Weber (BWW) representation model of information systems (IS)—called just the BWW model in this paper—and was illustrated with definitions of constructs from the Unified Modeling Language (UML). The paper focussed on modelling constructs that represent concrete problem domains, i.e., on representation of materials rather than concepts. The main idea behind the paper was to provide a standard way of defining modelling constructs in terms of the BWW model, in order to make the definitions cohesive and, thus, learnable, understandable and as directly comparable to one another as possible. Another important idea was to provide a way of defining modelling constructs not only generally, in terms of whether they represent "classes", "properties" or other ontological categories, but also in terms of *which* classes and/or properties they represent, in order to make the definitions more clearly and precisely related to the enterprise. Although most of the paper was about the concrete parts and aspects of *enterprises,* we believe the template and other results of this paper are sufficiently general to apply to concrete *problem domains* in general.

An important outcome is that the template encourages thorough analyses and precise definitions of enterprise, IS and other problem domain modelling constructs and languages. In particular, it assists in identifying semantical overlaps (Spanoudakis & Finkelstein, 1998, 1999) at a detailed level between seemingly unrelated modelling constructs and their languages. The template is also useful for identifying constructs that are too complex or too vaguely defined. Thereby, the

template paves the way both for more precise and tightly integrated enterprise and IS models and for better completeness and consistency checking of models. However, we do not mean to imply that all enterprise and IS modelling languages and all models should necessarily be integrated or even defined in terms of a standard template. In many situations, e.g., to foster creativity when a new opportunity is identified or a new IS is to be conceptualised, there will be a need for modelling languages and constructs that challenge the commonly accepted ground. The important issue is that problem domain modelling languages that are integrated and defined in terms of a standard template should be available *when they are called for*. Also, we do not mean to imply that all enterprise and IS modelling languages and models should necessarily be as precisely defined as possible. Whereas in some situations, models (and thus languages) should be precise, other situations might call for less precise models and languages, e.g., to represent early ideas of the problem domain during initial development. Again, the important issue is that problem domain modelling languages should support precise modelling *when this is called for*. Further work is needed to investigate how to define languages that provide more and less precise semantics for different stages of development.

Another important outcome is the identification of the generalisation hierarchy that is inherent in the UML. This type of generalisation hierarchy is a new way of analysing and comparing modelling languages and constructs. It offers both a new way to clarify and explain the semantics of the UML and introduces a new perspective from which the UML can be constructively criticised.

The template demonstrates again the applicability and usefulness of the BWW model as the foundation for work that addresses the semantics of enterprise and IS modelling languages. The template also makes the BWW model more applicable and useful in practice. Rosemann and Green (2002) point out that although the BWW model has produced important research results, it is large and complex and therefore difficult to learn and use. The template makes the BWW model simpler to use by decomposing construct definitions into five entries that are largely independent and that each are simpler than the BWW model or the template as a whole. At the same time, the template does not deviate much from the BWW model. Although at first sight, the template does not account for all the BWW-concepts presented in other papers (e.g., Wand & Weber, 1988, 1993, 1995), it accounts for all the basic ones, so that modelling constructs defined in terms of the template should also be implicitly related to the rest of the BWW model. However, this needs to be verified in further work.

Further work is needed to validate and refine the proposal made here, both by relating the template to other ontologies and other mathematical formalisms and by using it on additional modelling languages and constructs. For example, it would be interesting to use the template to define the constructs in the ARIS language (Green & Rosemann, 1999) for business process modelling and to define intentional modelling constructs such as goals and speech acts. The current version of the metamodel has been developed to be clear and understandable but, especially when it comes to the ontological descriptions of properties, it has some redundancies that need to be sorted out.

Further work is also needed on tool support for the template. Such a tool would

assist definition of modelling constructs in terms of the five types of entries and would manage dependencies between entries. If would also assist in maintaining the taxonomy of ***"Classes"***, ***"Properties"***, ***"States"*** and ***"Events"***. The tool could also support analysis of modelling languages, e.g., by automatically generating generalisation hierarchies and identifying overlapping modelling constructs.

The template presented in this paper focusses on modelling constructs that represent concrete problem domains, i.e., that represent materials rather than concepts. Further work should extend the template to account for modelling constructs that represent social constructs and mental concepts. Also, as pointed out by Opdahl and Henderson-Sellers (2002), representing problem domains—be they material, social or mental—is only one of several roles played by enterprise and IS modelling constructs, which must also sometimes represent proposed software solutions, support modellers and software developers ,and match other modelling constructs to create a well-defined, compact and tightly integrated modelling language. The current version of the template only deals with material problem domains and offers little help with managing these additional roles.

ACKNOWLEDGMENTS

This is Contribution number 02/17 of the Centre for Object Technology Applications and Research.

ENDNOTES

[1] Following an observation made by Opdahl and Henderson-Sellers (2002), UML-class is not prominent in this paper because **UML-type**, a stereotype of UML-class in UML Version 1.4, is more specific to representing concrete problem domains like enterprises.

[2] Here, we make a distinction between *general ontology*, such as the BWW model, and *special* or *domain-specific ontology*, which is a high-level, generic and often reusable model of a problem domain. Bunge (1999) makes a similar distinction between general ontology, which "studies all existents", and special ontology, which "studies one genus of thing or process."

[3] Table 2 in Opdahl and Henderson-Sellers (2002) gives a full list of 14 ontological matches or near matches between the UML and the BWW model, whereas Opdahl and Henderson-Sellers (2001) summarise key ontological differences between OO-modelling in general and the BWW model.

[4] The 10 constructs left out were UML-class (because UML-types are more relevant, see footnote 1), UML-send and -receive, UML-action, -call, -subactivity, -synch and -final state and UML-signal and -stimulus.

REFERENCES

Barbier, F., Henderson-Sellers, B., Opdahl, A.L. & Gogolla, M. (2000). The whole-part relationship in the Unified Modeling Language: A new approach. In Halpin, T. & Siau, K. (eds.), *Unified Modeling Language: Systems Analysis, Design, and Development Issues* . Hershey PA: Idea Group Publishing (IGP).

Bodart, F., Patel, A., Sim, M. & Weber, R. (2001). Should optional properties be used in conceptual modelling? A theory and three empirical tests. *Information Systems Resear ch*, 12(4), 384–405.

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

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

Bunge, M. (1999). *Dictionary of Philosophy*. Amherst:Prometheus Books.

Castellani, X. (1998). An overview of the version 1.1 of the UML defined with charts of concepts. In P.A. Muller and Bézivin, J. (eds.), *Proc. "International Conference on the Unified Modeling Language, <<UML>>'98 — Beyond the Notation",* Mulhouse/France, June 3-.4. LNCS, Springer Verlag.

Chisholm, R.M. (1996). *A Realistic Theory of Categories: An Essay on Ontology*. Cambridge University Press.

Conallen, J. (1999). Modeling Web Application Architectures with UML. *Communications of the ACM* , 42(10). (Special Issue on UML, Booch, G. (guest ed.))

Evermann, J. & Wand, Y. (2001). Towards ontologically based semantics for UML constructs. In Kunii, H., Jajodia, S. & Solvberg, A. (eds.), *Proc. "20th International Conference on Conceptual Modeling", ER 2001, Yokohama, Japan, Nov. 27–30, 2001* .

Firesmith, D., Henderson-Sellers, B. & Graham, I. (1997). *OPEN Modelling Language — OML Reference Manual*. SIGS Books. Cambridge University Press.

Green, P.F. (1996). *An Ontological Analysis of Information Systems Analysis and Design (ISAD) Grammars in Upper CASE Tools*. PhD thesis, Department of Commerce, University of Queensland.

Green, P. & Rosemann, M. (1999). An ontological evaluation of integrated process modelling. In *Proceedings "11th Conference on Advanced Information Systems Engineering", CAiSE*99,* Heidelberg/Germany, 14–18 June 1999. Springer.

Green, P. & Rosemann, M. (2000). Integrated process modelling: An ontological evaluation. *Information Systems*, 25(2), 73–87.

Krogstie, J., Lindland, O.I. & Sindre, G. (1995). Towards a deeper understanding of quality in requirements engineering. In Iivari, J., Lyytinen, K. & Rossi, M. (eds.), *Advanced Information Systems Engineering, Proc. CAiSE*95, Jyväskylä.* LNCS 932, Springer Verlag.

Lindland, O.I., Sindre, G. & Sølvberg, A. (1994). Understanding quality in conceptual modeling. *IEEE Software*, 11(2):42–49, March.

OMG (2001). *OMG Unified Modeling Language Specification, version 1.4.* Object Management Group.

Opdahl, A.L., Henderson-Sellers, B. & Barbier, F. (1999). An ontological evaluation of the OML metamodel. In Falkenberg, E.D., Lyytinen, K. & Verrijn-Stuart, A.A. (eds.), *Information System Concepts: An Integrated Discipline Emerging*, pp. 217–232. Kluwer (IFIP 8.1).

Opdahl, A.L., Henderson-Sellers, B. & Barbier, F. (2001). Ontological analysis of whole-part relationships in OO models. *Information and Software Technology*, 43(6), 387–399.

Opdahl, A.L. & Henderson-Sellers, B. (2001). Grounding the OML metamodel in ontology. *Journal of Systems and Software*, 57(2), 119–143.

Opdahl, A.L. & Henderson-Sellers, B. (2002). Understanding and improving the UML metamodel through ontological analysis. *Journal of Software and Systems Modelling (SoSyM)*, 1(1):43–67, Springer.

Parsons, J. & Wand, Y. (1997). Using objects for systems analysis. *Communications of the ACM* , 40(12),

104–110.

Parsons, J. & Wand, Y. (2000). Emancipating instances from the tyranny of classes in information modeling. *ACM Transactions on Database Systems* , 25(2), 228–268.

Paulson, D. & Wand, Y. (1992). An automated approach to information systems decomposition. *IEEE Transactions on Software Engineering (TSE)*, 18(3), 174–189.

Rosemann, M. & Green, P. (2002). Developing a meta model for the Bunge-Wand-Weber ontological constructs. *Information Systems*, 27, 75–91.

Spanoudakis, G., Finkelstein, A. & Till, D. (1999). Overlaps in requirements engineering. *Automated Softwar e Engineering Journal*.

Spanoudakis, G. & Finkelstein, A. (1998). A semi-automatic process of identifying overlaps and inconsistencies between requirement specifications. In *Proc. "5th International Confer ence on Object-Oriented Information Systems", OOIS 98*, 405–424.

Takagaki, K. & Wand, Y. (1991). An object-oriented information systems model based on ontology. In Van Assche, F., Moulin, B. & Rolland, C. (eds.), *Object Oriented Appr oach in Information Systems*, pp. 275–296, Amsterdam:Elsevier (North-Holland).

Uschold, M., King, M., Moralee, S. & Zorgios, Y. (1998). The enterprise ontology. *The Knowledge Engineering Review*, 13.

Verrijn-Stuart, A.A. (ed.) (2001). *A Framework of Information System Concepts — The Revised FRISCO Report*. Web document, draft version.

Wand, Y. & Wang, R.Y. (1996). Anchoring data quality dimensions in ontological foundations. *Communications of the ACM*, 39(11), 86–95.

Wand, Y. & Weber, R. (1988). An ontological analysis of some fundamental information systems concepts. In DeGross, J.I. & Olson, M.H. (eds.), *Proceedings of the Ninth International Confer ence on Information Systems, Minneapolis/ USA, November 30–December 3, 1988*,213–225.

Wand, Y. & Weber, R. (1989). An ontological evaluation of systems analysis and design methods. In Falkenberg, E. & Lindgreen, P. (eds.), *Proceedings of the IFIP WG8.1 W orking Confer ence on "Information Systems Concepts: An In-Depth Analysis", Namur , Belgium* , pp. 79–107, Amsterdam:North-Holland.

Wand, Y. & Weber, R. (1990). An ontological model of an information system. *IEEE T ransactions on Softwar e Engineering (TSE)*, 16(11), 1282–1292.

Wand, Y. & Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems*, 3:217–237.

Wand, Y. & Weber, R. (1995). On the deep structure of information systems. *Information Systems Journal*, 5, 203–223.

Wand, Y. (1989a). An ontological foundation for information systems design theory. In Pernici, B. & Verrijn-Stuart, A.A. (eds.), *Office Information Systems: The Design Process*. Amsterdam:Elsevier (North-Holland).

Wand, Y. (1989b). A proposal for a formal model of objects. In Kim, W. & Lochovsky, F.H. (eds.), *Object-Oriented Concepts, Databases, and Applications* , chapter 21, pages 537–559. New York:ACM Press/Addison-Wesley.

Weber, R. & Zhang, Y. (1996). An analytical evaluation of NIAM's grammar for conceptual schema diagrams. *Information Systems Journal*, 6:147–170.

Weber, R. (1997). *Ontological*

Foundations of Information Systems. Number 4 in Accounting Research Methodology Monograph series. Coopers & Lybrand, 333 Collins Street, Melbourne Vic 3000, Australia.

Andreas L. Opdahl is Professor of Information Systems Development in the Department of Information Science, University of Bergen, Norway. Dr. Opdahl is the author, co-author or co-editor of more than 30 journal articles, book chapters, refereed archival conference papers and books on requirements engineering, multi-perspective enterprise modelling, software performance engineering and other areas. He is a member of IFIP WG8.1 on Design and Evaluation of Information Systems. He serves regularly as a reviewer for premier international journals and on the program committees of renowned international conferences and workshops. Opdahl can be contacted at postal address: Department of Information Science, University of Bergen, P.O.Box 7800, N-5020 Bergen, Norway, andreas@ifi.uib.no, http://www.ifi.uib.no/staff/andreas/.

Brian Henderson-Sellers is Director of the Centre for Object Technology Applications and Research and Professor of Information Systems at University of Technology, Sydney (UTS). He is author of eleven books on object technology and is well-known for his work in OO methodologies (MOSES, COMMA, OPEN, OOSPICE) and in OO metrics. Brian has been Regional Editor of Object-Oriented Systems, a member of the editorial board of Object Magazine/Component Strategies and Object Expert for many years and is currently on the editorial board of Journal of Object Technology and Software and Systems Modelling. He was the Founder of the Object-Oriented Special Interest Group of the Australian Computer Society (NSW Branch) and Chairman of the Computerworld Object Developers' Awards committee for ObjectWorld 94 and 95 (Sydney). He is a frequent, invited speaker at international OT conferences. In 1999, he was voted number 3 in the Who's Who of Object Technology (Handbook of Object Technology, CRC Press, Appendix N). He is currently a member of the review panel for the OMG's Software Process Engineering Model (SPEM) standards initiative and is a member of the UML 2.0 review team. In July 2001, Professor Henderson-Sellers was awarded a Doctor of Science (DSc) from the University of London for his research contributions in object-oriented methodologies