Anatomy of the Unified Enterprise Modelling Ontology

Andreas L. Opdahl

Department of Information Science and Media Studies,
University of Bergen, NO-5020 Bergen, Norway
Andreas.Opdahl@uib.no

Abstract. The Unified Enterprise Modelling Language (UEML) aims to become a hub for integrated use of enterprise and information systems (IS) models expressed using different languages. A central part of this hub is an extendible ontology into which modelling languages and their constructs can be mapped, so that precise semantic relations between the languages and constructs can be established by comparing their ontology mappings. The paper presents and discusses ongoing work on reformulating the UEML ontology as an OWL2 DL ontology, the Unified Enterprise Modelling Ontology (UEMO).

Keywords: Ontology, ontological analysis and evaluation, Unified Enterprise Modelling Language (UEML), Unified Enterprise Modelling Ontology (UEMO), OWL2, description logic.

1 Introduction

The Unified Enterprise Modelling Language (UEML) supports *precise semantic definition* of a wide variety of enterprise and IS-modelling languages. It aims to use the definitions to also facilitate *integrated use of models* expressed in those languages [1]. The aim is an important one because information and software technologies are becoming increasingly driven by models, making interoperability between modelling languages and models a helpful step on the way to achieving interoperability between model-driven information and software systems.

To facilitate integrated use of models expressed in a wide variety of languages, the language definitions must be made semantically interoperable. UEML approaches this problem through a structured approach to describing enterprise and IS-modelling constructs in terms of an evolving ontology [2, 3]. So far, 130 constructs from a selection of 10 languages have been mapped into the ontology, although with varying degrees of precision. Whereas the idea of using an ontology to describe and integrate modelling languages is not new in itself, UEML describes and integrates the semantics of modelling constructs in a novel way that combines (1) a systematic, fine-grained approach to describing the semantics of modelling constructs with (2) a systematic approach to structuring and evolving the underlying ontology. UEML thus goes further than other ontology-based approaches to enterprise model interoperability (e.g., [4, 5]) because it is complemented by an extensive framework for systematically describing modelling

M. van Sinderen and P. Johnson (Eds.): IWEI 2011, LNBIP 76, pp. 163–176, 2011.
© IFIP International Federation for Information Processing 2011

constructs and because it has been explicitly designed to evolve and grow over time without becoming overly complex.

It is the common ontology at the heart of UEML that is the focus of this paper. On the meta-ontology (or structure) level it distinguishes itself from comparable approaches by simultaneously (1) promoting states and transformations to first-order concepts alongside things/classes and their properties, (2) providing better support for complex properties, (3) treating relations between things and classes as a type of mutual (or relational, shared) property of things/classes alongside intrinsic properties and (4) considering laws as another type of property of things/classes. On the ontology (or content) level it is distinct (1) by being the first middle-level ontology dedicated to enterprise and IS modelling in general, (2) by being explicitly grounded in Mario Bunge's philosophical ontology [6, 7] and (3) by offering particularly precise and elaborate dynamic and systemic concepts. Because of its grounding in Bunge's ontology and its adaptation to the information systems field (e.g., [8]), the common UEML ontology is ontological in both the philosophical and computer-science senses, although its mathematically formal underpinnings have been less developed so far. A set of OCL-constraints were presented in [9] and later extended and re-written in Prolog [10, 1]. But, beyond that, the UEML ontology has not been formalised so far.

This paper therefore presents a first formalisation of UEML's central ontology concepts by reformulating its classes, properties, states and transformations using OWL2 DL [11]. The purpose of the resulting *Unified Enterprise Modelling Ontology (UEMO)* is threefold. Firstly, we want to contribute towards a more precise UEML, to which the formalisation is a direct contribution. Secondly, we want to make UEML supported by formal reasoning approaches and tools. Although the old UEML ontology was represented in OWL, it did not leverage the full potential of OWL DL as a specification and reasoning language and did not explore the stronger expressiveness of OWL2. Thirdly and finally, we want to be able to show that the core of UEML has nice decision problems, i.e., that it is sound, complete and tractable with respect to many of its anticipated uses.

The rest of the paper is organised as follows. Section 2 presents the Unified Enterprise Modelling Language (UEML). Section 3 presents the backbone of the Unified Enterprise Modelling Ontology (UEMO). Section 4 outlines how UEMO can be used to facilitate interoperability between modelling constructs. Section 5 discusses the results. Finally, Section 6 concludes the paper and suggests paths for further work. Of course, a conference-length paper such as this can only explain a selection of UEMO's most important concepts. Several of our definitions have therefore been simplified because they rely on concepts that are not explained in the paper. Development versions of UEMO are available on http://www.uemlwiki.org/.

2 Theory

Construct description in UEML: UEML describes a modelling language mainly in terms of its modelling constructs. For each construct, both its syntax and semantics are described. The intended semantics of a modelling construct is described in a structured way according to the following six parts (see, e.g., [12]):

1. *Instantiation level:* A modelling construct may be used to represent either individual things (the *instance* level), classes of things (the *type* level) or *both* levels.
2. *Modality:* A modelling construct (or part thereof) may represent either a *fact* about or someone's *belief* about, *knowledge* of, *obligation* within, *intention* for a domain, and so on (in addition the model itself can have yet another modality, e.g., it may represent a possible or wanted future situation).
3. *Classes of things:* Regardless of instantiation level and modality, a modelling construct will represent one or more things (if it is instance level) or classes of things (if it is type level).
4. *Properties of things:* Most modelling constructs will also represent one or more properties that this or these thing(s)/class(es) *possess*. The properties may be *complex*, having other properties as *sub-properties*. In UEML, some complex properties even constrain their sub-properties and their values. Such properties are called *laws* [6, pp. 77-80].
5. *States of things:* Some behavioural modelling constructs represent particular *states* in their things or classes. States are *defined* in terms of a thing's properties by a *state constraint* that *restricts* these properties' values.
6. *Transformations of things:* Behavioural modelling constructs may even represent *transformations* of things/classes from a *pre-* to a *post-state*. Transformations are described by the properties that *define* the pre- and post-states and by a *transformation function* that *prescribes* changes to these properties' values.

Instead of mapping modelling constructs one-to-one with concepts in an ontology, UEML thereby describes each modelling construct as a *scene* of interrelated *roles* that are played by ontology concepts, so that the roles are either *classes/things* (item 3 above), their *properties* (item 4), their *states* (item 5) or their *transformations* (item 6). The roles are interrelated so that classes/things *possess* properties (that characterise the classes); properties *define* states; transformations have *pre-* and *post-states*; state constraints *restrict* states; transformation functions *prescribe* transformations; and by taxonomical/hierarchical relations we will explain later. The scene can be described in further detail by cardinality constraints on the relations between roles; by equivalence and/or disjointness axioms on roles; and by other types of constraints [3].

For example, a scene that describes the Class construct in UML would have a *"class"* role that describes the class of things that UML-Class is intended to represent. Because UML-Class is a very general modelling construct, the "class" role is played by Anything, which we will see is the most general of all classes in UEMO. The scene would also comprise a "name" role that describes the name property that has been assigned to the class and zero or more "attribute" and "operation" properties to describe its attributes and operations, each of them played by a precisely defined ontology property. Further roles would be used to describe associations, including aggregation/composition, and generalization relationships between UML-Classes.

Description logic: Description logic (DL) is a family of knowledge representation languages that are well suited for automated reasoning [13]. The ***SHOIN*** and ***SROIQ*** [11] variants of description logic correspond roughly to the ontology representation languages OWL and OWL2, respectively, so that OWL classes correspond to DL concepts and OWL object properties correspond to DL roles. There are even DL features that correspond to OWL datatype properties, but we will not use them here.

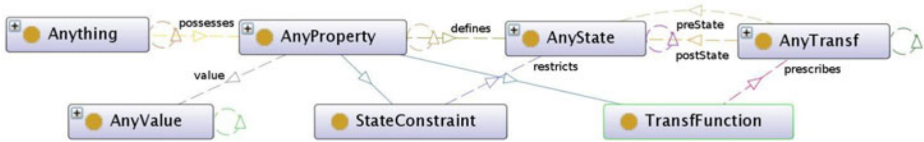

Fig. 1. High-level OWL2 classes that show the structure of UEMO

Description logics can be considered a fragment of 1. order predicate calculus, but with nicer decision problems. [13] and [14] offer introductions to basic DL notation and reasoning.

3 The Unified Enterprise Modelling Ontology (UEMO)

Overall structure: UEMO's concepts are partitioned into *classes of things*, *properties*, *states* and *transformations*, as in the UEML ontology. In addition, UEMO introduces *values* of properties. These five types of ontology concepts are disjoint but interrelated, so that classes of things *possess* properties; properties have *values* and *define* states; and transformations have *pre-* and *post-states*. Furthermore, *state constraints* and *transformation functions* are sub-types of properties that *restrict* states and *prescribe* transformations, respectively The resulting ontology structure is shown in Figure 1. Hence, UEMO has the same structure as the scenes that describe individual modelling constructs, so that each scene can be considered an excerpt from UEMO, possibly with added role names, tighter cardinalities and other constraints.

UEMO defines each concept as an OWL2 *class* (or description logic *concept*) and its interrelations as OWL2 *object properties* (or description logic *roles*) as follows:

$$
\begin{aligned}
&\text{Anything} \equiv \exists\text{possesses.AnyProperty} \sqcap \forall\text{possesses.AnyProperty} \\
&\text{AnyProperty} \equiv \exists\text{belongsTo.Anything} \sqcap \forall\text{belongsTo.Anything} \\
&\text{StateConstraint} \equiv \text{AnyProperty} \sqcap \forall\text{restricts.AnyState} \sqcap (=1\ \text{restricts}) \\
&\text{TransformationFunction} \equiv \text{AnyProperty} \sqcap \forall\text{prescribes.AnyTransformation} \sqcap (=1\ \text{prescribes}) \\
&\text{AnyState} \equiv \forall\text{restrictedBy.StateConstraint} \sqcap (=1\ \text{restrictedBy}) \sqcap \\
&\qquad\qquad \exists\text{definedBy.ConstrainedProperty} \\
&\text{AnyTransformation} \equiv \forall\text{prescribedBy.TransformationFunction} \sqcap (=1\ \text{prescribedBy}) \sqcap \\
&\qquad\qquad \forall\text{preState.MutableState} \sqcap (=1\ \text{preState}) \sqcap \forall\text{postState.AnyState} \sqcap (=1\ \text{postState}) \\
&\text{AnyValue} \equiv \forall\text{valueOf.ValuedProperty} \\
&\text{Anything} \sqcap \text{AnyProperty} \sqcap \text{AnyState} \sqcap \text{AnyTransformation} \sqcap \text{AnyValue} \sqsubseteq \bot
\end{aligned}
$$

Restrictions like "$\forall$restricts.AnyState $\sqcap$ (=1 restricts)" are used instead of the conciser "(=1 restricts).AnyState" to limit the ontology to ***SHIN*** expressiveness (e.g., [11], which, however, discuss slightly more powerful DL variants), which is supported by both "OWL1" and OWL2, thus giving access to a broader selection of reasoners and other tools.

Additional *taxonomy relations* organise the ontology concepts into five taxonomies. (1) Classes may *specialise* other classes. The root of the class taxonomy is Anything. (2) Properties may *precede* other properties, so all things that possess a property, such as "being-human", necessarily possess its precedents too, such as "being-alive".

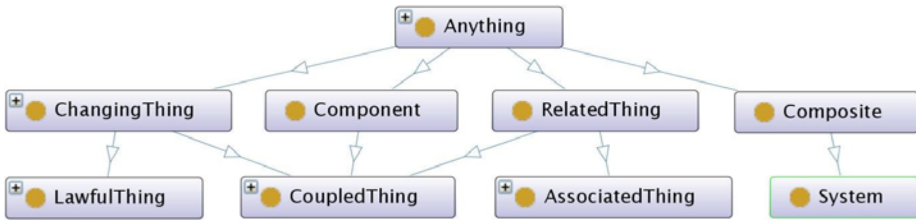

Fig. 2. Top-level classes in UEMO

The root of this taxonomy is AnyProperty. (3) States may *refine* other states (*OR*-decomposition), with AnyState at the root of the taxonomy. (4) Transformations may *elaborate* other transformations (*OR*-decomposition), with AnyTransformation as taxonomical root. (5) Values may *extend* other values. The root of this taxonomy is AnyValue. The five root concepts were shown Figure 1, which also depicted StateConstraint and TransformationFunction as important sub-types of AnyProperty. UEMO comprises several *hierarchical relations* in addition to the taxonomical ones: properties may be *sub-properties* of complex ones; states may be *regions of* composite states (*AND*-decomposition); transformations may be *components of* parallel transformations and *steps in* sequential ones (two ways to *AND*-decompose transformations). We now present each taxonomy in some detail.

Class taxonomy: According to [8], "[A] class is a set of things that possess a common property", where things and their properties are the most basic concepts in Bunge's ontology [6]. Anything is the root of the class taxonomy, so the Anything class in our OWL2 DL reformulation subsumes all the other class concepts in UEMO. Immediately below Anything are ChangingThing and RelatedThing along with Composite and Component (Figure 2). ChangingThing is characterised by possessing at least one *mutable property*, whereas RelatedThing must possess some *relation*, which is a shared (or mutual) property. Composite and Component are both characterised by possessing a *part-whole relation*, in which Composite plays the role of 'whole' and Component the role of 'part'. Composites and Components are not RelatedThings because part-whole relations are ontologically different from regular relations (shared/mutual properties) between other things.

ChangingThing ≡ Anything ⊓ ∃possesses.SomewhatMutableProperty
RelatedThing ≡ Anything ⊓ ∃possesses.Relation
Composite ≡ Anything ⊓ ∃possessesAsWhole.PartWholeRelation
Component ≡ Anything ⊓ ∃possessesAsPart.PartWholeRelation

The definitions of Composite and Component illustrate how we introduce *sub-roles* (through owl:subPropertyOf axioms on object properties), such as *possessesAsWhole* ⊑ possesses and *possessesAsPart* ⊑ possesses, of the *possesses* role to indicate more specific roles that UEML's properties may play in relation to their things/classes. For example, without sub-roles, it would have been difficult to formally distinguish Composite from Component. It would also have been impossible to limit the current UEMO to ***SHIN*** expressiveness. We will encounter more sub-roles later.

According to Bunge [6], a CoupledThing is one that interacts with one or more other things so that their histories of states and events depend on one another. Together, these things form a System. Hence, a CoupledThing is both a RelatedThing, a ChangingThing and a Component in a System. In addition, there are LawfulThings (similar to *natural kinds* [6, p. 143]) that possess law properties, which we will say more about later. We have to omit many other UEMO classes, such as the different types of active and executing things and resources, which have been included in the ontology either because they are needed directly to describe modelling constructs as part of the UEML work or indirectly to make other UEMO concepts clearer.

In addition to the named classes, we can use description logic expressions to introduce *anonymous classes* (and *anonymous properties, states* and *transformations*). Such a class can be used to define modelling constructs just like named classes, but does not contribute to making the ontology unwieldy. If it turns out to be useful over time, it can be named and included in the ontology later. For example:

$$\begin{aligned}\text{UnrelatedThing} &\equiv \text{Anything} \sqcap \forall \text{possesses.}(\text{IntrinsicProperty} \sqcup \text{PartWholeRelation})\\ &\equiv \text{AnyThing} \sqcap \neg \exists \text{possesses.Relation} \equiv \neg\text{RelatedThing}\\ \text{UnchangingThing} &\equiv \text{Anything} \sqcap \forall \text{possesses.CompletelyImmutableProperty}\\ &\equiv \neg\text{ChangingThing}\end{aligned}$$

Property taxonomy: In Bunge's ontology [6], properties belong to things and characterise classes. According to [8], “[A] property is modelled via an *attribute* function that maps the thing into some value.” Because AnyProperty is the root of the property taxonomy, the AnyProperty class in our OWL2 DL reformulation subsumes all the other property concepts in UEMO, where subsumption between property concepts has been defined to correspond to Bunge's property precedence, i.e., that all things that possess a property necessarily possess its precedents too [6]. Immediately below AnyProperty in the taxonomy is IntrinsicProperty, Relation and PartWholeRelation (Figure 3). IntrinsicProperty belongs to a single thing only. Relation belongs to more than one thing, but is not a part-whole relation. PartWholeRelation belongs to a whole thing (the Composite) and its part thing (the Component).

$$\begin{aligned}\text{IntrinsicProperty} &\equiv \text{AnyProperty} \sqcap \forall \text{belongsTo.Anything} \sqcap (=1\ \text{belongsTo})\\ \text{Relation} &\equiv \text{AnyProperty} \sqcap \forall \text{belongsTo.RelatedThing} \sqcap (\geq 2\ \text{belongsTo}) \sqcap\\ &\quad \neg \exists \text{belongsToWhole.Composite} \sqcap \neg \exists \text{belongsToPart.Component}\\ \text{PartWholeRelation} &\equiv \text{AnyProperty} \sqcap\\ &\quad \forall \text{belongsToWhole.Composite} \sqcap (=1\ \text{belongsToWhole}) \sqcap\\ &\quad \forall \text{belongsToPart.Component} \sqcap (=1\ \text{belongsToPart}) \sqcap\\ &\quad \forall \text{belongsToPartOrWhole.Component} \sqcap (=2\ \text{belongsToPartOrWhole})\\ \text{belongsTo} &\equiv \text{possesses}^{-1}\end{aligned}$$

Here, the belongsToPartOrWhole role is introduced so we can assert that the Component and the Composite are different things. Because OWL2 DL prohibits role disjunction, this role has been derived using SWRL [15]:

```
belongsToPart(?c1, ?c2) → belongsToPartOrWhole(?c1, ?c2)
belongsToWhole(?c1, ?c2) → belongsToPartOrWhole(?c1, ?c2)
```

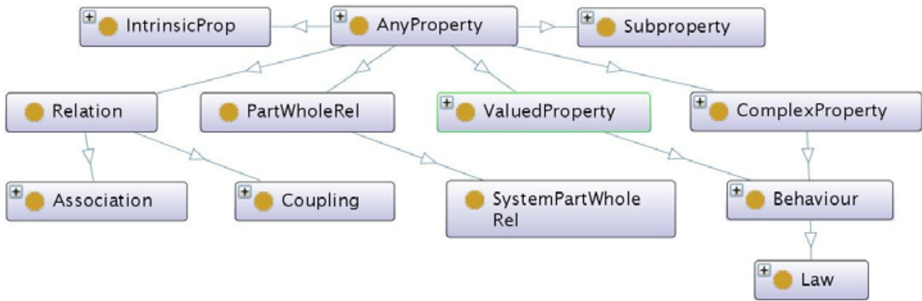

Fig. 3. Top-level properties in UEMO

UEMO thereby circumvents OWL2 DL limitations by using SWRL and its extension SQWRL [16], which allows sets and bags to be used in rules.

These three successors of AnyProperty, i.e., IntrinsicProperty, Relation and Part-WholeRelation, are disjoint, or mutually exclusive, in OWL2 terms, so that no property can be preceded by more than one of them. But they are not incompatible, meaning that the same thing can possess several of them at the same time.

An Association relates (non-coupled) AssociatedThings, whereas a Coupling relates CoupledThings. A SystemPartWholeRelation relates a CoupledThing to its System just like a Component is related to a Composite. A ValuedProperty has a specific *value*, whereas a ComplexProperty has one or more other properties as *sub-properties*. A Behaviour is a Valued- and ComplexProperty that describes either a state (when it is a StateConstraint) or a transformation (when it is a Transformation-Function). A Behaviour that is naturally or socially *enforced* is a Law. Hence, StateLaws are enforced StateConstraints and TransformationLaws are enforced TransformationFunctions, defined along these lines:

$$\text{Law} \equiv \text{Behaviour} \sqcap \forall \text{constrainedSubproperty.LawfullyConstrainedProperty}$$
$$\text{StateLaw} \equiv \text{Law} \sqcap \text{StateConstraint} \sqcap \forall \text{constrainedSubproperty.LawfullyConstrainedProperty}$$
$$\text{TransformationLaw} \equiv \text{Law} \sqcap \text{TransformationFunction} \sqcap \exists \text{manipulatedSubproperty.LawfullyManipulatedProperty}$$

Of course, there are many property concepts we cannot discuss here, including more specific types of behaviours and laws. For example, UEMO has socially assigned properties, such as Name, which is an association between a Namegiver and a NamedThing. Information and SocialLaws are other examples of assigned properties. UEMO also has concepts for Mutable- and ImmutableProperties, which come in both strong (e.g., CompletelyImmutableProperty) and weak (e.g., SomewhatMutable-Property) forms, because a property can change in many different ways, i.e., it can be dropped by its thing, it can have its value changed, it can drop a sub-property if it is complex or be dropped by its superior if it is a sub-property.

State taxonomy: According to [8], a state is "[T]he vector of values for all attribute functions of a thing" at a particular time, where an attribute function describes a property by mapping the thing to some value. AnyState is the root of the state taxonomy. Hence, the AnyState class in our OWL2 DL reformulation subsumes all the other

state concepts in UEMO, where subsumption between state concepts has been defined to correspond to *OR*-decomposition of states.

A state in UEMO is either mutable or immutable. A MutableState is defined in terms of at least one SomewhatMutableProperty, whereas an ImmutableState is defined only by CompletelyImmutableProperties.

MutableState ≡ AnyState ⊓ ∃definedBy.SomewhatMutableProperty
ImmutableState ≡ AnyState ⊓ ∀definedBy.CompletelyImmutableProperty

UEMO states are also either stable or unstable. A StableState is *restrictedBy* a StateLaw, whereas an UnstableState is *restrictedBy* a StateViolation property. Like StateLaw, StateViolation is a Behaviour (specifically, a StateConstraint). But, whereas a StateLaw is naturally or socially *enforced*, a StateViolation is only socially *sanctioned*. UnstableState refines MutableState, because the thing must eventually return to a stable state.

StableState ≡ AnyState ⊓ ∀restrictedBy.StateLaw ⊓ (=1 restrictedBy)
UnstableState ≡ MutableState ⊓ ∀restrictedBy.StateViolation ⊓ (=1 restrictedBy)

Transformation taxonomy: According to [8], a transformation of a thing "is a mapping from a domain comprising states to a co-domain comprising states." AnyTransformation is the root of the transformation taxonomy. The AnyTransformation class in our OWL2 DL reformulation therefore subsumes all the other transformation concepts in UEMO, where subsumption between transformation concepts has been defined to correspond to *OR*-decomposition of transformations.

A SelfTransformation in a thing only manipulates the thing's own properties, whereas an ExternalTransformation manipulates at least one Relation property that the thing shares (possesses mutually with) another thing. A Destabilising transformation takes the thing from a Stable- to an UnstableState and a Stabilising takes it back. A Destabilising is always an ExternalTransformation, because nothing destabilises itself, i.e., there are no Destabilising SelfTransformations.

SelfTransformation ≡ AnyTransformation ⊓ ¬∃manipulatedProperty.Relation
ExternalTransformation ≡ AnyTransformation ⊓ ∃manipulatedProperty.Relation
Destabilising ≡ ExternalTransformation ⊓ ∀prescribedBy.Destabilising ⊓
    (=1 prescribedBy) ⊓ ∀preState.StableState ⊓ (=1 preState) ⊓
    ∀postState.UnstableState ⊓ (=1 postState)
Stabilising ≡ AnyTransformation ⊓ ∀prescribedBy.StabilisingLaw ⊓ (=1 prescribedBy) ⊓
    ∀preState.UnstableState ⊓ (=1 preState) ⊓ ∀postState.StableState ⊓ (=1 postState)

A SequentialTransformation is composed of two or more TransformationSteps, whereas a ParallelTransformation is composed of two or more TransformationComponents. UEMO defines both non-sequential (single-step Firings) and sequential (multi-step Executions) transformations for describing behavioural constructs.

SequentialTransformation ≡ AnyTransformation ⊓
    ∀sequenceOf.TransformationStep ⊓ (≥2 sequenceOf)
TransformationStep ≡ AnyTransformation ⊓ ∃stepIn.SequentialTransformation
ParallelTransformation ≡ AnyTransformation ⊓
    ∀composedOf.TransformationComponent ⊓ (≥ 2 composedOf)
TransformationComponent ≡ AnyTransformation ⊓ ∃componentOf.ParallelTransformation

Value taxonomy: Bunge's ontology [6] does not account for values directly, but treats properties as dichotomous (either possessed by the thing or not). Instead of valued properties such as a “has-age” property that maps to values like “25” and “50”, Bunge therefore uses properties such as “has-age-of-25” and “has-age-of-50”. UEMO offers valued properties because they are simpler to use. No generality is lost, because valued properties (“property-name” = “value”) can trivially be transformed into dichotomous ones (“property-name-of-value”).

AnyValue is the root of the value taxonomy, so that the AnyValue class in our OWL2 DL reformulation subsumes all the other value concepts in UEMO, where subsumption between value concepts has been defined to cover both regular subsetting and something we call augmentation (adding new components to tuples). A Set has other values as elements, whereas a Tuple has other values as components. We name inverse roles of values by adding the suffix -Of, e.g., valueOf ≡ value^{-1}, componentOf ≡ component^{-1} and elementOf ≡ element^{-1} etc.

$\text{AnyValue} \equiv \forall \text{valueOf.ValuedProperty}$
$\text{Set} \equiv \text{AnyValue} \sqcap \forall \text{element.AnyValue} \sqcap \neg \exists \text{component.AnyValue}$
$\text{Tuple} \equiv \text{AnyValue} \sqcap \forall \text{component.AnyValue} \sqcap (\geq 1\ \text{component}) \sqcap \neg \exists \text{element.AnyValue}$

The basic idea is that certain sub-types of values are Constraints that describe States, whereas other sub-types of values are Functions that describe Transformations. However, we have so far only covered transformations that are simple mappings from pre- to post-states, not transformations where inputs arrive and outputs depart at different times, with some outputs possibly being produced before all inputs have been consumed. A fuller definition of transformation functions along the lines discussed, e.g., in [17] has to be left for further work.

4 Using UEMO

The preceding section has formulated UEMO as an OWL2 DL ontology with ***SHIN*** expressiveness (e.g., [11]). While using UEMO to facilitate interoperability between models expressed using different languages remains work in progress, this section suggests how UEMO can facilitate describing and comparing modelling constructs semantically.

Describing modelling ***constructs:*** To describe modelling constructs in terms of UEMO, the ontology must be extended with an additional OWL class (DL concept) for ModellingConstructs and a new OWL object property (DL role) that map ModellingConstructs to the OntologyConcepts they *represent*:

$\text{OntologyConcept} \equiv \neg \exists \text{represents} \sqcap$
$\quad (\text{Anything} \sqcup \text{AnyProperty} \sqcup \text{AnyState} \sqcup \text{AnyTransformation} \sqcup \text{AnyValue})$
$\text{ModellingConstruct} \equiv \exists \text{represents.OntologyConcept} \sqcap \forall \text{represents.OntologyConcept}$

ModellingConstruct formalises the earlier concept of *scene*, so that each role in the scene is an OntologyConcept that the ModellingConstruct *represents*. Sub-roles of the represents role are used to distinguish between the different roles of the scene. For example, the *“class”* role in the scene that describes the Class construct in UML is

accounted for by the DL-role *representsClass* $\sqsubseteq$ represents. In consequence, UML-Class can be described as follows (leaving out association, aggregation/composition, generalisation and a few other details for now):

$$\begin{aligned} \text{UMLClass} \equiv\ & \text{ModellingConstruct} \sqcap \\ & \exists \text{representsClass.Anything} \sqcap (=1\ \text{representsClass}) \sqcap \\ & \exists \text{representsName.Name} \sqcap (=1\ \text{representsName}) \sqcap \\ & \forall \text{representsAttribute.(IntrinsicProperty} \sqcup \text{AssignedProperty)} \sqcap \\ & \forall \text{representsOperation.FiringLaw} \sqcap \forall \text{representsAssociation.Relation} \sqcap \cdots \end{aligned}$$

Further axioms can be introduced for a modelling construct, e.g., to constrain the *relations* between the roles in its scene or their *cardinalities*. The internal consistency of a modelling construct description thereby becomes a *concept satisfiability problem* (e.g., [13]). For UMLClass, this problem has the following form, where ***T*** is the set of terminological axioms (the TBox) for ontology concepts and modelling constructs in UEMO:

$$\boldsymbol{T} \not\models \text{UMLClass} \equiv \bot$$

Comparing modelling ***constructs:*** We approach detailed comparison of modelling constructs as a sub-role matching problem. The above example introduced UMLClass with the sub-roles *representsClass*, *representsName*, *representsAttribute* etc. We now want to compare UMLClass to another ModellingConstruct, GRLGoal, which has sub-roles such as *representsAgent*, *representsTarget* and *representsGoal* [18]. One possible matching of sub-roles is between *representsClass* (of UMLClass) and *representsGoal* (of GRLGoal), which are restricted as follows by their respective modelling constructs:

$$\cdots\ \exists \text{representsClass.Anything} \sqcap (=1\ \text{representsClass}) \cdots \qquad \text{(by UMLClass)}$$
$$\cdots\ \exists \text{representsGoal.Behaviour} \sqcap (=1\ \text{representsBehaviour}) \cdots \qquad \text{(by GRLGoal)}$$

We match the two sub-roles by giving them the same name (ignoring possible name clashes for now), e.g., *representsClassAndGoal*. As a result, the conjunction UMLClass' $\sqcap$ GRLGoal' of the renamed concepts UMLClass' and GRLGoal' contains this combined restriction:

$$\begin{aligned} \cdots\ & \exists \text{representsClassAndBehaviour.(Anything} \sqcap \text{Behaviour)} \sqcap \\ & (=1\ \text{representsClassAndBehaviour}) \cdots \end{aligned}$$

We compare the UMLClass and GRLGoal constructs by investigating all possible matchings of UMLClass sub-roles with GRLGoal sub-roles, including combinations where some or all sub-roles of either construct remain *unmatched*. The result will be a large number of candidate matches, each of which combines sub-roles of UMLClass with sub-roles of GRLGoal in a different way. Fortunately, most candidate matches can be immediately discarded, because they contain self-contradictory role restrictions, i.e., restrictions whose conjunction is not satisfiable. In the above example, UMLClass' $\sqcap$ GRLGoal' can be safely discarded because Anything (a UEMO-class concept) and Behaviour (a UEMO-property concept) are disjoint by definition. In other cases, it is the number restrictions or other restrictions on the renamed sub-roles that are self-contradictory. The above test for internal consistency of modelling constructs can be used to eliminate candidate matches too:

$\boldsymbol{T} \not\models$ UMLClass' ⊓ GRLGoal' ≡ ⊥

We expect that most candidate matches generated by brute-force combination of sub-roles can be immediately discarded because they are not satisfiable. The much smaller set of satisfiable matches must be considered further by other means, most likely involving human inspection and assessment, which can possibly be aided by automatic *ontology classification* that arranges the remaining candidates in a more easily explored subsumption hierarchy. The top match of this hierarchy would be the least restrictive candidate, the one that does not match *any* sub-roles of the two constructs, whereas each leaf would be a candidate that is not restricted further by any other candidate. The search for the best candidate can proceed bottom- up and breadth-first from the leaves of the subsumption hierarchy. The selected best match can be written on the form UMLClass* ⊓ GRLGoal* so that the information represented by UMLClass* and not by GRLGoal* and vice versa can be written

InformationLostFromUMLClassToGRLGoal ≡ UMLClass* ⊓ ¬GRLGoal*
InformationMissingFromUMLClassToGRLGoal ≡ ¬UMLClass* ⊓ GRLGoal*

These two concepts describe, respectively, the information that will be lost and the information that is missing and must somehow be provided when translating from a UMLClass to a GRLGoal (of course, they also describe the missing/lost information when translating back from GRLGoal to UMLClass).

Comparing modelling ***languages:*** Not accounted for here is the complementary problem of aligning modelling languages construct-wise, identifying which constructs or groups of constructs in one language that correspond most closely to the constructs or groups in the other, as a necessary preparation for detailed construct comparison.

Facilitating cross-language interoperability: Further work should explore how to facilitate cross-language model-to-model translations based on the detailed construct matchings described here. One approach is to store model elements expressed in one language as OWL individuals and then use complex SPARQL and/or SQWRL queries to retrieve them as model elements expressed in another language. Another strategy is to use construct matches to generate QVT or other transformations, along the lines suggested in [19] which, however, does not build an extensive ontology. To adopt their approach, UEMO must first be extended to account for intended modelling-language and -construct *syntax* in addition to *semantics*.

5 Discussion

UEMO is still evolving and currently comprises 225 OWL classes (or DL concepts), although this number is somewhat inflated because it explicitly defines many *anonymous concepts* that may not be needed in the production version of the ontology. Most of the OWL classes represent UEMO classes and properties, with fewer representing UEML states and transformations so far. Compared to earlier versions of the UEML ontology, many new UEMO properties have been introduced to more precisely describe mutability and immutability, transients and persistence, assignments, complex properties, behaviours and laws. UEMO restricts the OWL classes with 567 subclass and 42 disjointness axioms and connects them with 96 object properties (or DL roles)

that are in turn restricted by 97 sub-property (owl:subPropertyOf, aka Bunge-precedence) and 257 other axioms.

The work has shown that a large part of UEMO can be expressed in OWL2 DL and, so far, even in the relatively inexpressive ***SHIN*** sub-language [11], making a wider range of reasoners and other tools available, because ***SHIN*** is supported by both "OWL1" and OWL2. In addition, S(Q)WRL [15, 16] has been used to express certain additional constraints. Unfortunately, these "externally expressed" restrictions thereby become out of reach for DL-based reasoners, and further work must consider how they can be best used to reason about modelling languages and constructs. Two other groups of very general constraints seem infeasible to express even in S(Q)WRL, because they may require modal and/or temporal axioms. One group comprises UEMO concepts for transients and persistence and for certain types of mutability. Another includes [6] definitions of couplings and of systems. Further work should attempt to describe as many of these constraints as possible "inside" OWL2 DL, investigating, e.g., whether the modal/temporal axioms may at least have *implications* that can be expressed in DL form.

The present work has contributed both to making UEMO more precise and to supporting it with automatic reasoning tools. It has also indicated that several of the possible uses of UEMO have nice decision problems. UEMO also has the potential to become simpler than the old UEML ontology by exploiting more of OWL's native features. Firstly, it is prepared for using XML-namespaces where the old ontology used elaborate naming schemes. Secondly, where the old ontology introduced "association classes" to account for *role names* and *cardinality constraints*. UEMO uses OWL's built-in sub-role and number restrictions to the same effect. Thirdly, whereas the old ontology represented ontology concepts as OWL *individuals*, UEMO represents its concepts as OWL *classes*. One advantage is that they are thereby better supported by DL reasoning tools, which tend to solve decision problems on the concept (or class) level. Another is that the ontology is thus prepared for representing the semantics not only of modelling constructs, but also of model elements, which can be mapped either to OWL classes in UEMO (e.g., for a particular UML-Class) or to the OWL individuals that instantiate the classes (e.g., for a particular UML-Object).

Space prevents us from discussing several other important features of UEMO, such as the possibility of *parametric definitions* that use place holders (such as <Property> and <Value> below) to define powerful *generic concepts* like these:

$$\text{PossessesProperty}\langle\text{Property}\rangle \equiv \text{AnyState} \sqcap \exists \text{definedBy.Property}$$

$$\text{PossessesPropertyValue}\langle\text{Property, Value}\rangle \equiv \text{AnyState} \sqcap \exists \text{definedBy.}(\text{Property} \sqcap \exists \text{value.Value})$$

6 Conclusion and Further Work

The paper has outlined the Unified Enterprise Modelling Ontology (UEMO), which supersedes the *common ontology* of the Unified Enterprise Modelling Language (UEML [1]). UEMO goes further than other ontology-based approaches to enterprise model interoperability (e.g., [4, 5]) because it offers an extensive framework for systematically describing modelling constructs in fine detail and because it has been explicitly designed to evolve and grow over time without becoming overly complex

(through the five taxonomies). It is an ontology in both the philosophical and computer-science senses, and the paper has emphasised the latter side. It has formulated UEMO in OWL2 DL with ***SHIN*** expressiveness, meaning that it so far remains also in "OWL1" DL form. The paper has also outlined potential uses of UEMO as a computer-science ontology and discussed its further development. The paper has thereby contributed both to making UEMO more precise and to supporting it with automatic reasoning tools.

UEMO has already grown large, and a short paper like this can only present a selection of its concepts and features. Further work must present UEMO in fuller detail as an ontology both in the philosophical sense (e.g., grounding its concepts clearly in Bunge's ontology) and the computer-science sense (e.g., defining its concepts in description logic form and detail their use by automated reasoners and other relevant tools). Further work is also needed to extend the ontology with more precise concepts for states and transformations and to properly validate it. The present version of UEMO has already been extensively validated through iterative development, by using several automated reasoners and by cross-checking with earlier ontology versions. But additional validations are needed that use UEMO to describe and support interoperability between existing modelling languages.

Acknowledgements. The author is indebted to all the researchers, assistants and students who contributed to the Domain Enterprise Modelling in Interop-NoE, in particular Giuseppe Berio, Mounira Harzallah and Raimundas Matulevičius.

References

1. Anaya, V., Berio, G., Harzallah, M., Heymans, P., Matulevičius, R., Opdahl, A.L., Panetto, H., Verdecho, M.J.: The Unified Enterprise Modelling Language – Overview and Further Work. Computers in Industry 61(2) (2010)
2. Opdahl, A.L.: A Platform for Interoperable Domain-Specific Enterprise Modelling Based on ISO 15926. In: EDOC 2010 Workshop Proceedings. IEEE CS Press, Los Alamitos (2010)
3. Opdahl, A.L.: Incorporating UML Class and Activity Constructs into UEML. In: Trujillo, J., Dobbie, G., Kangassalo, H., Hartmann, S., Kirchberg, M., Rossi, M., Reinhartz-Berger, I., Zimányi, E., Frasincar, F. (eds.) ER 2010. LNCS, vol. 6413, pp. 244–254. Springer, Heidelberg (2010)
4. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In: Wang, J., Whittle, J., Harel, D., Reggio, G. (eds.) MoDELS 2006. LNCS, vol. 4199, pp. 528–542. Springer, Heidelberg (2006)
5. Ziemann, J., Ohren, O., Jäkel, F.-W., Kahl, T., Knothe, T.: Achieving Enterprise Model Interoperability Applying a Common Enterprise Metamodel. In: Doumeingts, G., Müller, J., Morel, G., Vallespir, B. (eds.) Enterprise Interoperability New Challenges and Approaches. Springer, London (2007)
6. Bunge, M.: Treatise on Basic Philosophy. Ontology I: The Furniture of the World, vol. 3. Reidel, Boston (1977)
7. Bunge, M.: Treatise on Basic Philosophy. Ontology II: A World of Systems, vol. 4. Reidel, Boston (1979)

8. Wand, Y., Weber, R.: On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. Journal of Information Systems 3, 217–237 (1993)
9. Opdahl, A.L., Henderson-Sellers, B.: Template-Based Definition of Information Systems and Enterprise Modelling Constructs. In: Green, P., Rosemann, M. (eds.) Ontologies and Business System Analysis, vol. ch. 6. Idea Group Publishing, USA (2005)
10. Mahiat, J.: A Validation Tool for the UEML Approach. Master thesis, University of Namur (2006)
11. Horrocks, I., Kutz, O., Sattler, U.: The Even More Irresistible SROIQ. In: Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning, KR 2006, pp. 57–67 (2006)
12. Opdahl, A.L.: The UEML Approach to Modelling Construct Description. In: Doumeingts, G., Müller, J., Morel, G., Vallespir, B. (eds.) Enterprise Interoperability – New Challenges and Approaches. Springer, Berlin (2007)
13. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in Description Logic. In: Brewka, G. (ed.) Principles of Knowledge Representation and Planning, pp. 193–238. CSLI Publications, Stanford (1996)
14. Nardi, D., Brachman, R.J.: An Introduction to Description Logics. In: Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.) The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
15. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B. and Dean, M. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission (May 21, 2004)
16. O'Connor, M.J., Das, A.: SQWRL: a query language for OWL. In: OWL – Experiences and Directions Workshop Series (2009)
17. Harel, D., Rumpe, B.: Modelling Languages: Syntax, Semantics and all that Stuff (or, What's the Semantics of "Semantics"?). Technical Report, Technische Universität Braunschweig (2004)
18. Matulevičius, R., Heymans, P., Opdahl, A.L.: Comparing GRL and KAOS using the UEML Approach. In: Gonçalves, R.J., Müller, J.P., Mertins, K., Zelm, M. (eds.) Enterprise Interoperability II – New Challenges and Approaches. Springer, Heidelberg (2007)
19. Roser, S., Bauer, B.: Automatic Generation and Evolution of Model Transformations Using Ontology Engineering Space. J. Data Semantics 11, 32–64 (2008)