

# The Unified Enterprise Modelling Language – Overview and Further Work

Victor Anaya\*, Giuseppe Berio\*\*, Mounira Harzallah\*\*\*, Patrick Heymans\*\*\*\*, Raimundas Matulevičius\*\*\*\*, Andreas L. Opdahl\*\*\*\*\*, Hervé Panetto\*\*\*\*\* and Maria Jose Verdecho\*

\* CIGIP, Universidad Politecnica de Valencia, Spain; (e-mail: {vanaya, mverdecho}@cigip.upv.es).

\*\* LabSTICC, University of South Brittany, France; (e-mail: Giuseppe.Berio@univ-ubs.fr)

\*\*\* LINA, University of Nantes, France; (e-mail: Mounira.Harzallah@univ-nantes.fr)

\*\*\*\* PReCISE, Computer Science Faculty, University of Namur, Belgium; (e-mail: {phe, rma}@info.fundp.ac.be)

\*\*\*\*\* Dept. of Information Science and Media Studies, University of Bergen, Norway; (e-mail: Andreas.Opdahl@uib.no)

\*\*\*\*\* CRAN, Nancy-University, CNRS, France; (e-mail: Herve.Panetto@cran.uhp-nancy.fr)

**Corresponding author:** Andreas L Opdahl <Andreas.Opdahl@uib.no>, Department of Information Science and Media Studies, University of Bergen, P.O. Box 7802, NO-5020 Bergen, Norway; Phone: +47 5558 4140; Fax: +47 5558 9149.

# The Unified Enterprise Modelling Language – Overview and Further Work

**Abstract:** The Unified Enterprise Modelling Language (UEML) aims at supporting *integrated use of enterprise and IS models* expressed using different languages. To achieve this aim, UEML offers a *hub* through which modelling languages can be connected, thereby paving the way for also connecting the *models* expressed in those languages. This paper motivates and presents the most central parts of the UEML approach: a structured path to *describing enterprise and IS modelling constructs*; a *common ontology* to interrelate construct descriptions at the semantic level; a *correspondence analysis approach* to estimate semantic construct similarity; a *quality framework* to aid selection of languages; a *meta-meta model* to integrate the different parts of the approach; and a *set of tools* to aid its use and evolution. The paper also discusses the benefits of UEML and points to paths for further work.

## 1. Introduction

Emerging information and communication technologies are becoming increasingly *model-driven*. An important driving force behind the model-driven trend is to provide ICT solutions that are both *adaptable* and *integrated* at the same time. One idea is that model-driven system can easily be *adapted* at the model level, so that a change to the system model automatically effects a corresponding change of the running system, without requiring resource-demanding and error-prone manual re-design and re-implementation. Another idea is that two model-driven systems can be easily *integrated* at the model level, so that a mapping from one system model to another automatically configures middleware (such as enterprise application integration software) that integrates the running systems, again without human intervention. For example, a customer-relationship management (CRM) system in an enterprise may in the future be driven by a set of models that represent, e.g., the relevant human roles and types of products. And the same enterprise may possess a product-line management (PLM) system driven by a set of models that represent, e.g., the products and their features, product variants and the dependencies between them. Having model-driven systems for CRM and PLM is highly beneficial because it makes the systems easier to change whenever new requirements are identified. In addition, the two systems become easier to integrate when the representations of products in the two system models are kept complete and consistent.

But the above scenario works best for systems that are driven by models expressed in *interoperable* languages, meaning that it is possible to compare and otherwise relate the models automatically. When model-driven information systems are driven by models expressed using non-interoperable languages, the models easily become *inconsistent* with one another as they evolve, because they cannot be automatically compared or otherwise related to one another. This will make it much harder to integrate the running systems they drive. Model-driven technologies

may thus end up *reinforcing* – rather than alleviating – existing interoperability problems. In the above scenario, the modelling languages that drive the CRM and PLM systems may not be interoperable. For example, the CRM models may be expressed in a language with many specialised modelling concepts for people (as customers, users, salesmen etc.) in addition to products, whereas the PLM models are expressed in another modelling language that is particularly rich on concepts for expressing product features, variants and dependencies. This makes it difficult to ensure that products and their variants are represented in compatible ways in the two systems: every change to how products are represented in the CRM system must be immediately reflected in the PLM system and vice versa, but it becomes hard to provide tool support for this process. Manual model updates are slow and error prone and, as more model-driven systems are added to the picture, the number of dependencies between models grows faster than linearly. It is not easy for the enterprise to standardise its use of modelling languages either, because it may cut the enterprise off from the newest model-driven systems that are driven by the richest, most specialised models. The situation creates a need for theories, technologies and tools that allow information systems to be adapted and evolve, each driven by the most suitable languages for their purposes and context, while allowing the information systems and the models that drive them to be used in an integrated manner.

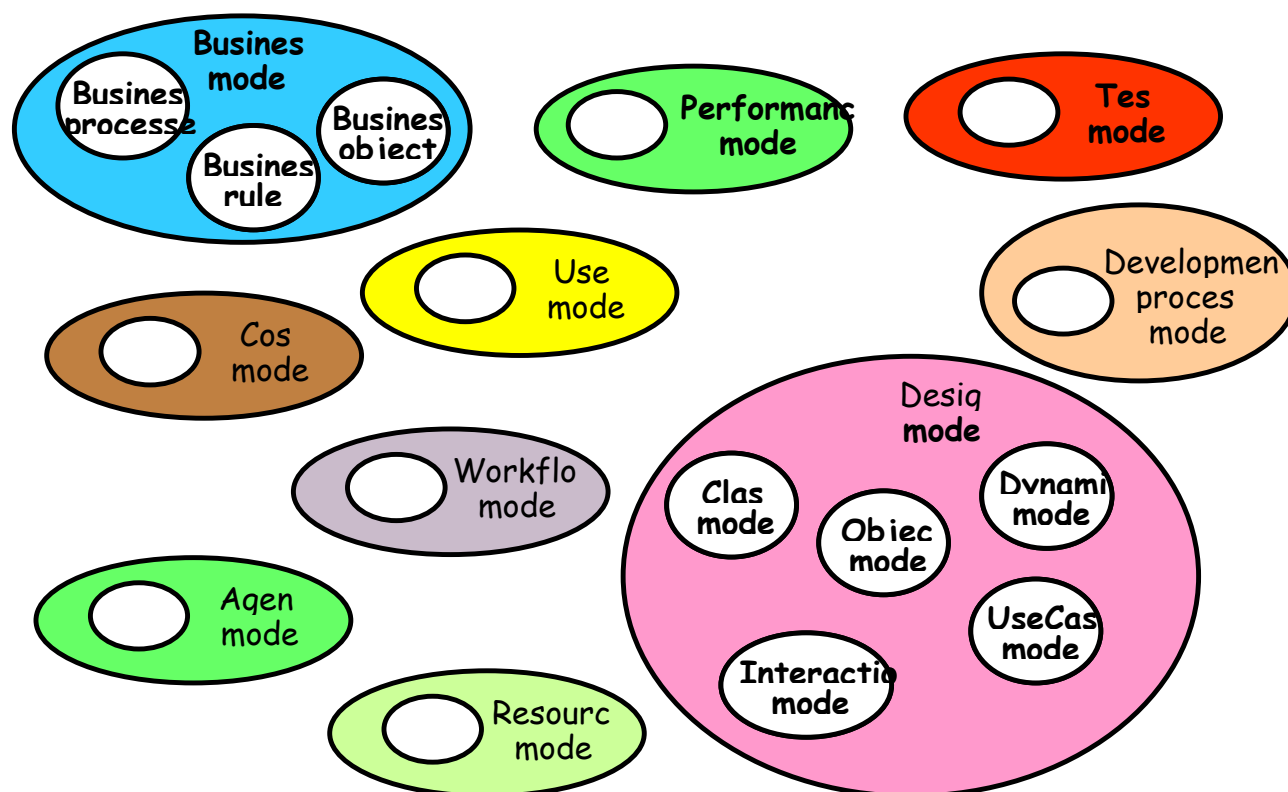


Figure 1. A selection of model types commonly used in modern enterprises, each of them ideally requiring dedicated modelling languages.

The Unified Enterprise Modelling Language (UEML) refers to an on-going attempt to develop theories, technologies and tools for *integrated use of enterprise and IS models* expressed in different languages. By this we mean keeping

the existing models as they are and, in addition, establishing correspondences between them in an explicit and usable way. Examples of useful services are consistency checking, automatic update reflection and model-to-model translation *across modelling language* boundaries. Figure 1 shows a selection of model types commonly used in modern enterprises for a wide range of purposes. Each of them requires modelling languages with different properties. UEML is intended as a *hub* for connecting these and other types of languages, eventually along with models expressed in those languages. To achieve this aim, UEML comprises:

- a structured approach to *describe enterprise and IS modelling constructs* syntactically and semantically,
- an *evolving common ontology* to interrelate the semantics of different modelling constructs,
- a *correspondence analysis approach* that uses the common ontology to identify and quantify semantic similarities between modelling constructs,
- a *quality framework* to define and evaluate the quality of enterprise and IS modelling languages in order to aid language selection for specific purposes,
- a modular *meta-meta model* to organise the overall UEML framework and
- a *set of tools* to aid its use and evolution.

The purpose of this paper is to present an overview of UEML and to discuss paths for further work. The paper is organised as follows: Section 2 presents UEML's *background* and its *vision*. Section 3 explains how languages and constructs are *described* in UEML, whereas Section 4 shows how descriptions of constructs are tied together by a *common ontology*. Section 5 discusses how *correspondences* between languages and constructs can be established and used, e.g., to support model-to-model translation across languages. Section 6 shows how enterprise and IS modelling languages are *classified and selected* in UEML according to specific goals. Section 7 presents the *meta-meta models* that hold the UEML framework together, whereas Section 8 reviews the various *prototype tools* supporting its use and evolution. Section 9 discusses UEML in its present state, before Section 10 concludes the paper and offers paths for further work.

## 2. Background

The idea of a Unified Enterprise Modelling Language first emerged during the ICEIMT'97 conference (Goossenaerts, Gruninger, Nell, Petit & Vernadat 1997), with the aim of providing an underlying formal theory for enterprise modelling languages. A major motivation was the "Tower of Babel" situation that hindered proliferation of enterprise modelling in industry (Vernadat 2002). The first development version of a unified enterprise modelling language was delivered by the *UEML Thematic Network* (UEML-TN 2002-2003), funded by the EU's FP5 (Jochem 2002, Panetto, Berio, Benali, Boudjlida & Petit 2004, Mertins, Knothe & Zelm 2004, Berio, Anaya & Ortiz 2004). UEML development has since continued within the Interop-NoE (2003-2007) Network of Excellence, funded by

EU's FP6, which produced two further development versions of UEMML, 2.0 and 2.1 (Berio, Opdahl, Anaya & Dassisti 2005a, 2005b, 2006).

The following scenarios illustrate the UEMML vision:

- *Exchanging information contained in enterprise and IS models* across modelling language boundaries. This is the central motivation behind UEMML, which explains its focus on *interoperability between modelling languages* as a prerequisite for integrated use of the models that are expressed in those languages.
- *Creating new problem- and/or domain-specific methods* by combining elements from existing modelling techniques. UEMML aims to make it easier to combine modelling languages and associated techniques and tools depending on the problem at hand, an ambition resembling that of *method engineering* (Brinkkemper, Saeki & Harmsen 1998, Henderson-Sellers, Gonzalez-Perez, Serour & Firesmith 2005, Ralyé & Rolland 2001). In particular, UEMML aims to support *local tailoring/adaptation* of languages and constructs to fit local practices and needs, possibly producing new *domain-specific languages* as a result.
- *Systematic, quality-driven, reuse of existing enterprise and IS modelling languages*. Combining techniques and tools across modelling languages has the side benefit of making the languages available for the domains where they are most suited, without limitations posed by modelling tools and other technologies.
- *Defining a core language for enterprise and IS modelling*. As UEMML stabilises, it may become possible to extract a core set of modelling constructs to use as the starting point for a new enterprise/IS modelling language. Such a *UEMML core language* should be composed of those constructs that have proven most useful for practical, integrated model use. However, the core language scenario should be understood as a longer term objective that is beyond the scope of this paper.
- *Facilitating a web of languages and of models* is another long-term objective. Whereas much research and development effort has gone into techniques and tools for integrated management of structured data (e.g., relational database theory) and of semi-structured data (e.g., XML and other web technologies), there is a lack of theory and technology for integrating information resources in the form of diagrammatic *models*. UEMML could contribute to growing a *web of languages and of models* in a way that resembles the touted semantic web of semi-structured data (Berners-Lee, Hendler & Lassila 2001).

Although UEMML is intended as a hub for connecting different languages and different models expressed in those languages, it should not necessarily be the only means of making enterprise and IS languages and models interoperable. Other theories, technologies and tools may be better suited for certain integration needs and should possibly be made usable alongside UEMML.

### 3. Language and Construct Description

UEMML facilitates integrated model use by making *semantic correspondences* between the modelling constructs of different languages clear. Making the languages interoperable is seen as a first step towards also making the models expressed in those languages interoperable. A central part of UEMML is therefore a standard, integrative and evolvable approach to *describing enterprise and IS modelling constructs*. By *standard* we mean that it provides a

structured path to describing modelling languages, diagram types and constructs. By *integrative* we mean that, as soon as the languages, diagram types and constructs have been described in the UEMML way, they have also become prepared for assessment of semantic correspondences, possibly across languages. And by *evolvable* we mean that UEMML will be able to grow and adapt by incorporation and modification of additional modelling languages and constructs without becoming overly complex and thus unmanageable.

The descriptions of individual modelling constructs are particularly important, because it is this level that connects different modelling languages. Hence construct descriptions are more complex than descriptions of languages and diagram types. Specifically, in UEMML, two distinct descriptions need to be made for each construct (Opdahl 2006):

- *Presentation* (or *concrete syntax*), which deals with the presentation of the modelling construct as part of model diagrams or in serialised form, e.g., in an XML file.
- *Representation* (or *semantics*), which accounts for which enterprise phenomena the construct is intended to represent (in particular covering *reference*, a central aspect of *semantics*).

Whereas a construct can have many presentations, it can have only one representation. This paper will focus on the representation part, which has so far been more developed than presentation.

In UEMML, semantics is described by a *representation mapping* of each modelling construct into a *common ontology*, based on earlier work by Opdahl & Henderson-Sellers (2004, 2005). The UEMML uses *separation of reference* to break individual modelling constructs into their *ontologically atomic parts*, along the following six axes:

1. *Which class(es) of things is the construct intended to represent?* Most modelling constructs somehow represent one or more *classes of things*. Even when the *primary* purpose of a construct is to represent certain properties, states or transformations, the construct implicitly also represents a property of, state of or transformation in, *one or more classes of things*. (A transformation may correspond either to an atomic event or a complex process.)
2. *Which properties is the construct intended to represent?* Most modelling constructs somehow represent one or more *types of properties*, which may either be *intrinsic properties* (belonging to only one thing) or *relationships* (properties that are *mutual* to several things). Some intrinsic properties are *laws* that restrict other properties. Even if the primary purpose of a construct is to represent classes, states or transformations, it represents classes, states or transformations that involve *one or more types of property*.
3. *Which states is the construct intended to represent?* Some modelling constructs are intended to represent a more or less restricted state in one or more classes of things. The *state law* that restricts the state can be described in terms of the properties of those classes. Whereas most modelling constructs represent one or more properties and, at least, one or more classes, not all constructs are intended to represent a state.
4. *Which transformations is the construct intended to represent?* Some constructs are intended to represent a simple or complex transformation of one or more classes of things from one state to another. The *transformation law* that effects the transformation can be described in terms of the states of those classes.

Again, not all constructs are intended to represent a transformation. Although some constructs are apparently not intended to represent behaviour at all, other constructs represent particular *states*, *transformations*.

5. *Which instantiation levels is the construct intended to represent?* A modelling construct represents classes, properties, states and transformations at either the instance or type level or both.
6. *Which modality (or mode) is the construct intended to represent?* We usually think of enterprise and IS models as *assertions of facts* about a domain, e.g., assertions that something is or is not the case in the enterprise. But some model elements may instead state that someone *wants* something to be the case, or that someone is not *permitted* to do something, or that someone *knows* something is the case, or that something *will be* the case some time in the *future*.

Hence, whereas the two first axes deal with *structure*, the next two deal with *behaviour*. Together, these four axes describe the semantics of a modelling construct by describing a *state of affairs*, or a *scene*, played by several classes, properties and, perhaps, states and transformations together. The final two axes supplement the scene with information about the construct's intended *use*, i.e., its instantiation level and modality/mode.

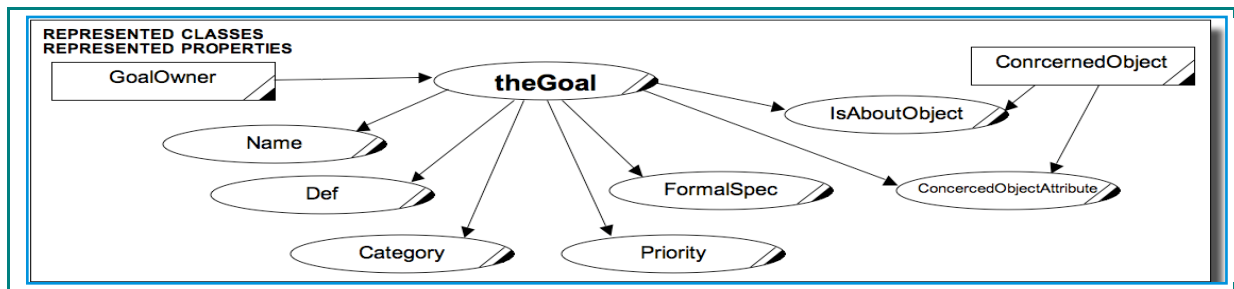


Figure 2. A break-down of KAOS' *Goal* modelling construct (van Lamswerde & Willemet 1998).

**Example.** We will use the *Goal* constructs from the GRL (Yu 1997) and KAOS (van Lamswerde & Willemet 1998) languages as our running example. Figure 2 shows how the *KAOS.Goal* construct is broken down into its semantically atomic parts using separation of reference (Matulevicius, Heymans & Opdahl 2006, 2007a, 2007b). A *Goal* in KAOS represents a pair of things belonging to different classes. The *goalOwner* is a thing that wants the goal to be achieved, and the *concernedObject* is the thing, usually a complex system, that the goal is about. A *KAOS.Goal* also represents a few properties of these two things. One such property is *theGoal* itself. This property belongs to the *goalOwner* and describes those states of the *concernedObject* in which the *KAOS.Goal* is achieved. The goal property is described further in terms of its subproperties that, e.g., attribute a *Name* to *theGoal* and indicate which object the goal is about (i.e., the *concernedObject*). Because KAOS.Goals are considered structural in this example, there is no behavioural part with represented states and transformations. The instantiation level is either *type* or *instance* and the modality is *intentional* from the *goalOwner* side and *obligational* for the *concernedObject*. □

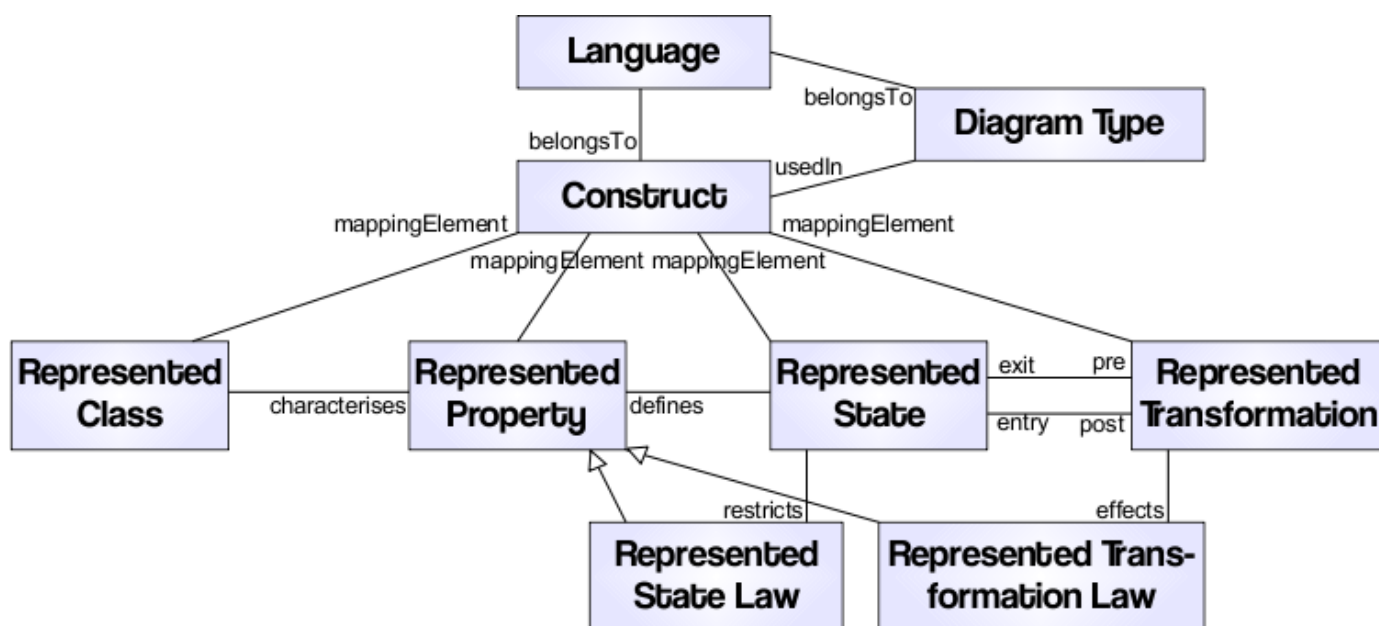


Figure 3: The main classes of the UEML representation meta-meta model, used to describe the semantics of modelling constructs.

The UML class diagram in Figure 3 shows the key concepts used to describe modelling languages and constructs in UEML. The upper part of the diagram depicts *modelling languages*, along with their *diagram types* and *modelling constructs*. The lower part shows how each individual construct is described by a *scene* of interrelated classes, properties, states and transformations.

#### 4. The Common Ontology

To tie modelling-construct descriptions together, UEML uses a common ontology into which the represented classes, properties, states and transformations of each construct are mapped. The common ontology thereby comes to interrelate the construct descriptions at the semantic level.

The UEML ontology is organised into four *taxonomies*: The classes in the ontology are organised in a conventional *generalisation hierarchy*. Properties, on the other hand, have their places in a *precedence hierarchy*, where a property precedes another if every thing that possesses the second property necessarily also possess the first. (For example, *associated-with* precedes *having-content*, because everything that is *having-content* is also *associated-with* that content.) There are similar generalisation hierarchies of states and of transformations too. Classes, properties, states and transformations – including the state and transformation laws – all have *attributes*. For example, they all have unique names and there are cardinality constraints and role names on the associations between classes and properties.

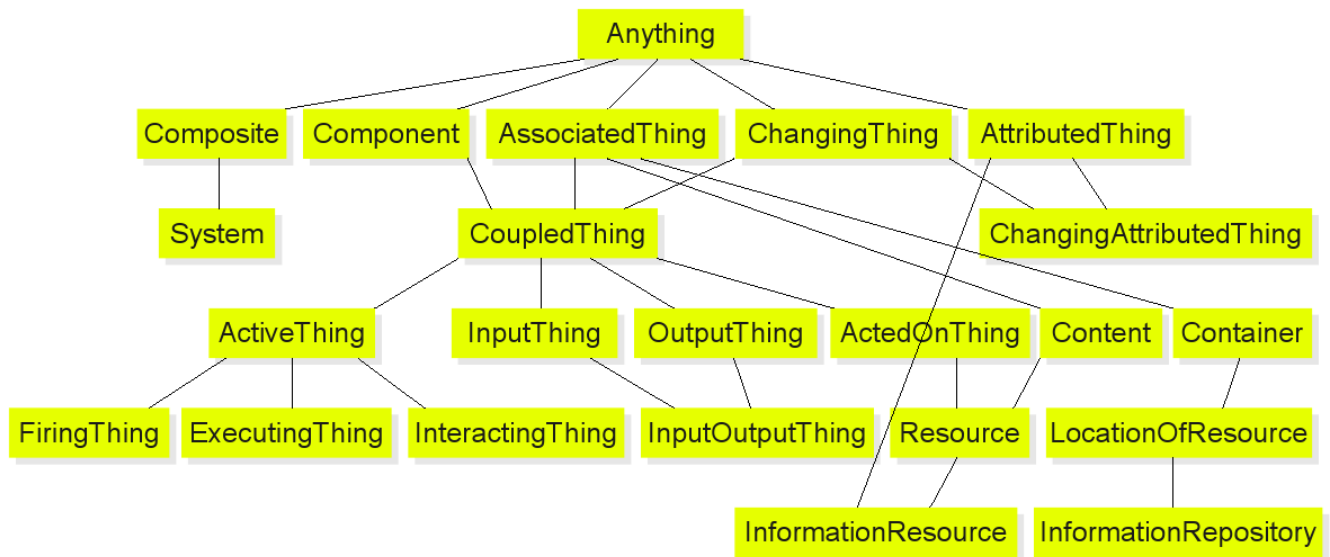


Figure 4. The current UEMML taxonomy of classes, which has evolved from Bunge's ontological model (Bunge 1977, 1979) and which will keep growing as more modelling languages and constructs are added to UEMML.

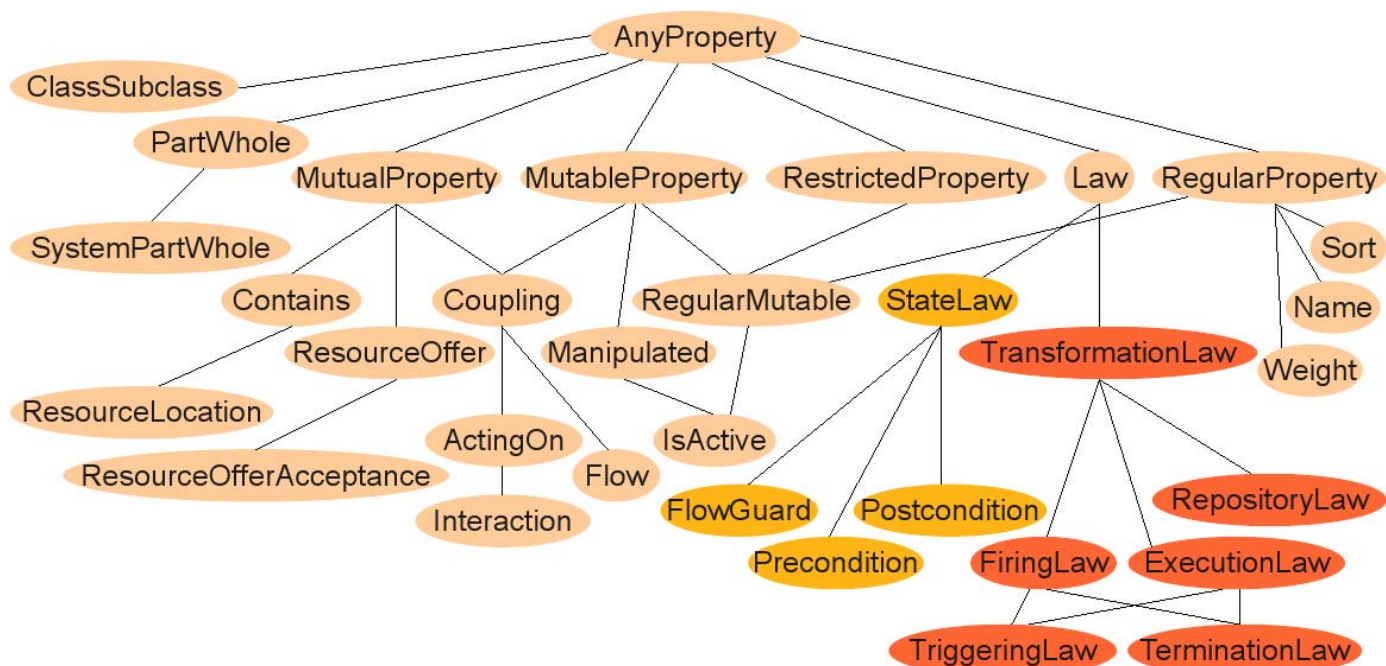


Figure 5. The current UEML taxonomy of properties, which has evolved from Bunge's ontological model (Bunge 1977, 1979) and which keeps growing along with the class taxonomy in Figure 4.

**Example.** Figures 4 and 5 show the current UEML taxonomies of classes and of properties respectively. The two taxonomies, and the corresponding ones for states and transformations, are not static, but will keep evolving as more languages and constructs are incorporated into UEML.

In Figure 4, the most general class is **Anything**, used to describe constructs that are intended to represent any type of thing. The subclasses of **Anything** are more specific. **Composite** is used to describe constructs that represent things that are composed of parts, and its subclass **System** is used for constructs whose parts affect one another's behaviour. The **Component** class is used to represent constructs that represent parts of **Composite** things, whereas the **AssociatedThing** is used for constructs that represent things that are somehow related to one another. **ChangingThing** is used to describe constructs that represent dynamic behaviour, whereas **AttributedThing** is used to describe constructs that represent things to which humans have assigned particular properties, such as a name or sort. These more specific classes are further subclassed.

Figure 5 shows the corresponding taxonomy of properties. The most generic property is **AnyProperty**, which has 7 immediate successors. Of these, a **PartWhole** relation associates a **Composite** thing with its **Components**. A **MutualProperty** associates two or more **AssociatedThings**. A **MutableProperty** is characteristic of a **ChangingThing** etc. **StateLaws** and **TransformationLaws** are more specific properties that are used to describe modelling constructs with dynamic behaviour in further detail. Hence, although it is now shown in the figures, each property in Figure 6 is characteristic of a class in Figure 5. For example **Anything**, the most general class, is characterised by the most generic property, **AnyProperty**. □

The UEML ontology was first populated with a set of initial classes, properties, states and transformations derived directly from Mario Bunge's ontological model (Bunge 1977, 1979) and the Bunge-Wand-Weber representation model of information systems, the so-called BWW model (Wand & Weber 1988a, 1988b, 1993, 1995). Since then, it has evolved and grown as new constructs have been added. Currently, UEML incorporates a selection of academic and industrial modelling languages, such as ARIS (Dossogne & Jeanmart 2007), BMM (Tu 2007), BPMN (Dossogne & Jeanmart 2007), coloured Petri nets, GRL (Dallons, Heymans & Pollet 2005, Heymans, Saval, Dallons & Pollet 2005, Matulevičius, Heymans & Opdahl 2006, 2007a, Tu 2007), IDEF3 (Harzallah, Berio & Opdahl 2007), ISO/DIS 19440, KAOS (Matulevičius, Heymans & Opdahl 2006, 2007a, 2007b), UEML 1.0 and selected diagram types from UML (OMG-UML 2009), version 2.0. In consequence, the most general concepts in the common ontology are *ontologically committed*, in the sense that they have grown out of Bunge's ontology and the BWW model, whereas the more specific ones have emerged through language and construct analyses.

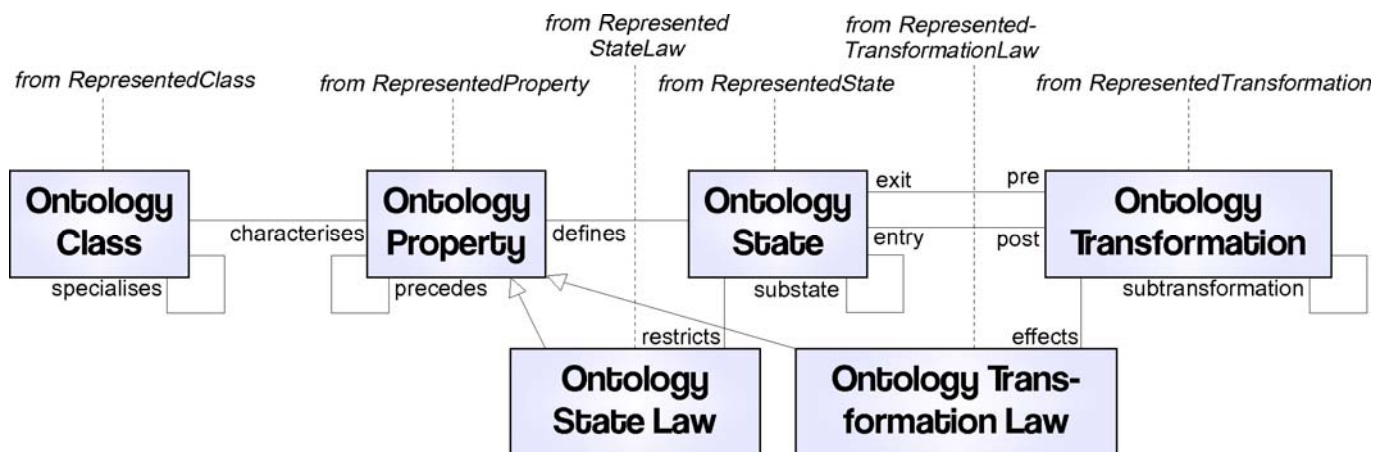


Figure 6: The main classes of the common UEML ontology, into which construct descriptions are mapped.

The four taxonomies are interrelated, as shown in the UML class diagram in Figure 6, which is based on the earlier work of Opdahl & Henderson-Sellers (2004, 2005). According to this model, ontological classes are related to the properties that *characterise* them. Properties are related to the states they *define*. States are in turn *entered* and *exited* by transformations. Certain types of properties are laws that restrict other properties. State laws *restrict* states, whereas transformation laws *effect* transformations. The resulting organisation of the UEML ontology as four distinct, but interrelated taxonomies makes it possible to evolve the ontology over time without increasing complexity more than necessary. New classes, properties, states and transformations will always have a clearly identifiable location where they can be added to the appropriate taxonomy. And because they are placed in taxonomies that have grown out of Bunge's ontology and the Bunge-Wand-Weber models, these ontological concepts have been given precise meanings in terms of what they represent in the For every construct incorporated into UEML, each represented class, property, state and transformation is mapped into an ontology concept in the ontology. Figure 6 therefore structurally resembles the lower part of Figure 3.

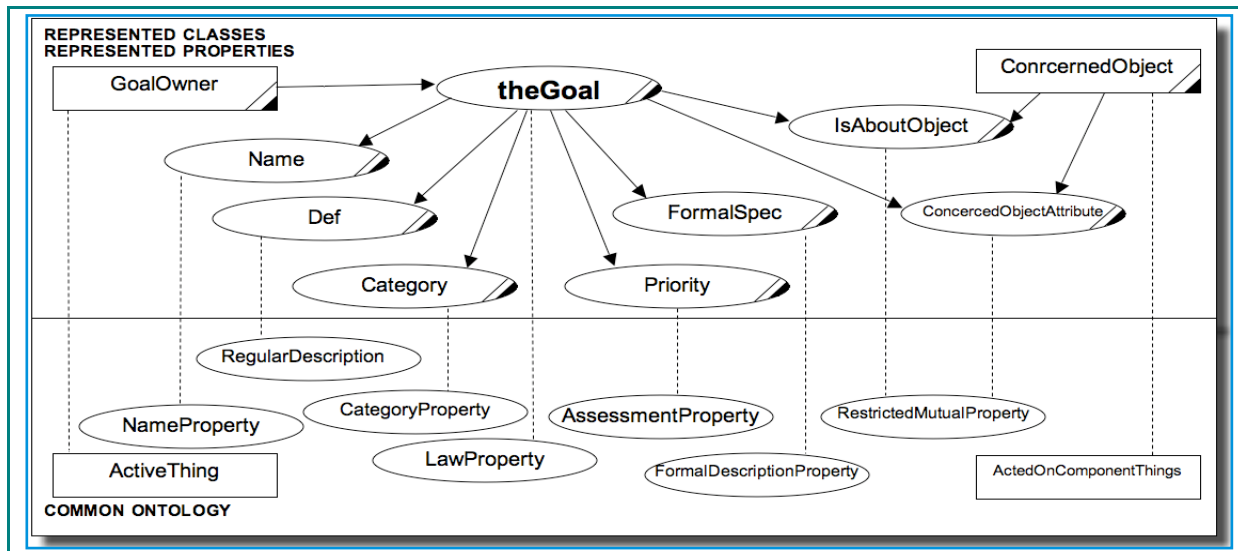


Figure 7. The ontological concepts used to describe the *KAOS.Goal* construct are mapped into the four taxonomies of the common UEMML ontology.

**Example.** Figure 7 also shows how the classes and properties represented by a *KAOS.Goal* are mapped to classes and properties in the common UEMML ontology. The *goalOwner* is an *ActiveThing*, meaning that it is capable of affecting the history of another thing (in this case the *concernedObject*). The *concernedObject*, on the other hand, is both an *ActedOnThing* and a *ComponentThing*, i.e., every *concernedObject* belongs to both classes. It is an *ActedOnThing* because its history is influenced by the *goalOwner*. It is also *ComponentThing*. The property *theGoal* is both a *LawProperty*, meaning that it constrains the values of other properties, and a *ComplexProperty*, meaning that it has sub-properties (some of which it constrains as a *LawProperty*). □

## 5. Language and Construct Correspondences

To support integrated use of models, UEMML must offer ways to exploit the mappings to *identify and manage correspondences* among language constructs and among model elements. Correspondences between any pair of constructs can be examined by comparing their mappings into the common ontology. If two modelling constructs are identical, they will map into the exact same ontology concepts. If two modelling constructs do not overlap at all, they will map into concepts that are not closely related in their respective taxonomies. However, the most common situation will most likely be where the modelling constructs map into some common ontology concepts, into some concepts that are closely related and into some that are not.

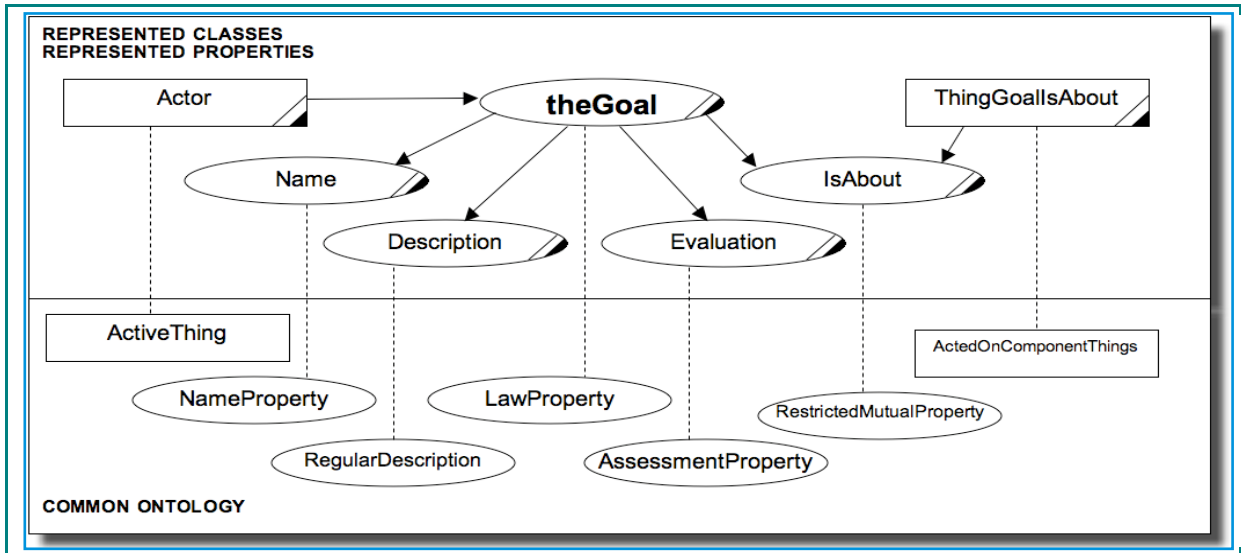


Figure 8. The ontological concepts used to describe the *GRL.Goal* construct are mapped into the four taxonomies of the common UEML ontology.

Table 1. Detailed comparison of the *GRL.GOAL* and *KAOS.Goal* modelling constructs via their mappings into ontological concepts.

	<b>GRL.goal</b>	<b>KAOS.goal</b>
<b>Transforms</b>	Name	Name
	Description	Def
<b>To consider</b>	Actor	GoalOwner Agent
	Evaluation	Priority
	IsAbout	IsAboutObject IsAboutAttribute
<b>Undefined</b>		FormalDef = ""
		Category = ""

**Example.** Figure 8 shows how the *Goal* construct from another goal-oriented language, this time GRL (Yu 1997), is broken down in the same manner as *KAOS.Goal* and similarly mapped to classes and properties in the common UEML ontology. Comparing Figures 7 and 8 via their mappings into the same ontological concepts provides us with a precise description of exactly how *GRL.Goal* and *KAOS.Goal* are similar and exactly how they differ. Table 1 compares the two based on their construct descriptions, showing how the UEML descriptions points towards automatic translations from GRL to *KAOS.Goals*. As shown in Figure 9, the ontology mappings indicate that the *Name* of a *GRL.Goal* maps directly into its *KAOS* counterpart, and similar for *GRL.Goal Descriptions*, which map to *KAOS.Goal Defs*. The ontology mappings also suggest that *GRL.Actor* may correspond to both *KAOS.GoalOwner* and *KAOS.Agent*, but that this cannot be established from the ontological mappings alone. Further work will have to investigate how reasoning techniques based on language structure and other information can be used to identify even more precise language correspondences. Finally, the comparison indicates that the *FormalDef* and *Category*

attributes of a *KAOS.Goal* are not covered specifically by a *GRL.Goal* and cannot be provided when a KAOS model is derived from a GRL model alone. Accordingly, these attributes would be lost if a KAOS model were to be transformed into GRL. In this way, the UEMML framework even makes it clear which information is missing and lost when model information is moved across language boundaries.

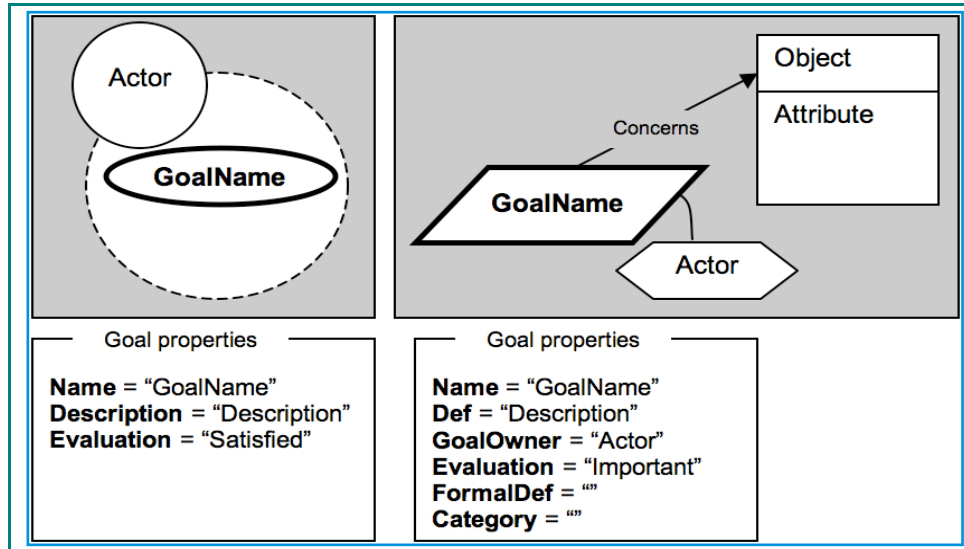


Figure 9. The comparison of ontology mappings suggests how a *GRL.Goal* can be transformed into a *KAOS.Goal* to support, e.g., model-to-model translation across modelling languages.

Three kinds of correspondences between modelling constructs have been identified as part of the UEMML framework. Each of them can be precisely formulated in terms of the ontology classes, properties, states and transformations into which the constructs in the correspondence map:

- *Equality* occurs when two or more constructs represent the exact same state of affairs, as explained in Section 3. If two constructs are equal, one can always replace the other without loss of information, e.g., for model-to-model translation.
- *Containment* occurs when the state of affairs represented by one construct has the state of affairs represented by another as a part. When one construct contains several others, the former can be replaced by a combination of the others during model-to-model translation.
- *Generalisation* occurs when one modelling construct represents a state of affairs that generalises the state of affairs represented by another. When one construct generalises another, the general construct can replace the special one in a model-to-model translation (with some loss of information), but the inverse replacement is only appropriate under specific circumstances.

Of course these simple kinds of correspondences are not independent. For example, constructs that are equal will trivially contain and generalise one another. There are also *complex correspondences*, e.g., when one construct represents a state of affairs that *generalises a part* of the state of affairs represented by another, thus combining

containment and generalisation. There are also *overlapping* constructs, each of which contains part, but not all, of the other. However, a complete typology of correspondences and how they combine stills needs to be worked out.

Correspondences are also characterised by different *degrees of precision*. For example, it is possible to only take into account how each construct is mapped into ontology concepts, ignoring how the concepts are *related* within the construct description. More precise correspondences can be identified by taking into account both ontology concepts and the relations between them, but ignoring the *roles* that the concepts may play in the relations. Finally, both the ontology concepts, the relations between them and the roles played can be taken into account.

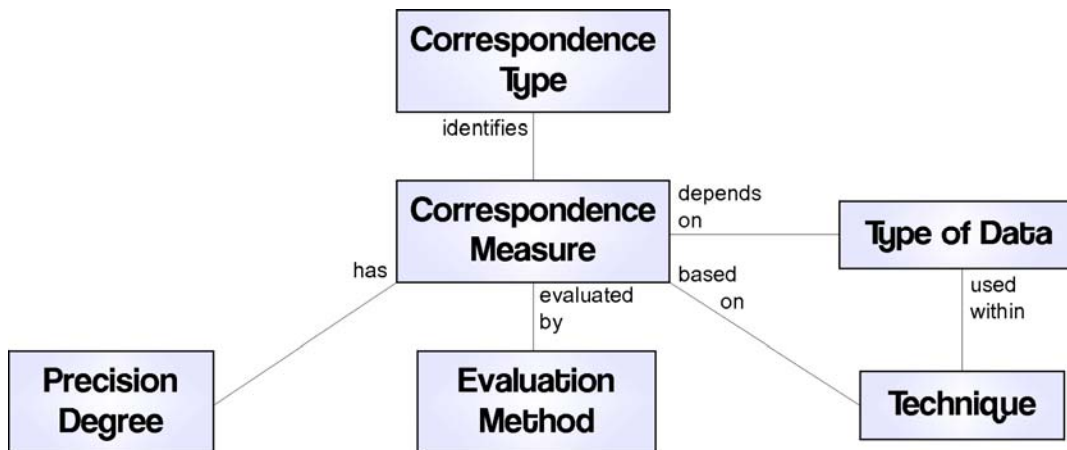


Figure 10: The main classes of the UEML correspondence analysis framework, used to identify and manage correspondences between modelling constructs.

Because correspondences are generally ways to assess to what extent constructs are similar or dissimilar (under a precision degree), it might be possible to characterise correspondences by using correspondence measures (CM). In this sense, a CM is a function:

$$CM: UEML_C \times UEML_C \rightarrow \square \square$$

In this function,  $UEML_C$  represents the set of constructs incorporated into UEML. CM results from explicit selections ranging on the following five parameters: correspondence type, precision degree, technique, type of data and evaluation method, as shown in Figure 10. The form of the function is related to the *correspondence type* to be identified (e.g. equality). There are three well-known forms of function that can be used, i.e., Jaccard, Recall and Precision (Gower & Legendre 1986). The *type of data* taken as input by the function is varying depending on the *precision degree* required. Type of data constrains the specific *technique* that can be used for effectively evaluating function results as well (using, e.g., structure-based, graph-based or attribute-based techniques) (Lin 1998, Rodríguez & Egenhofer 2003, Blanchard, Kuntz, Harzallah & Briand 2006). Finally, the measure should be meaningful. To this end, an *evaluation method* should be defined to evaluate the measurement accuracy. According to Budanitsky (1999), accuracy of the measure can be evaluated by comparing the measurement results with

correspondences found in three alternative and distinct ways: 1) theoretical investigation 2) human judgement and 3) knowledge about a particular application.

Correspondence measures can also be used to *validate* the representation mappings and the common ontology, when correspondence measures derived automatically from the common ontology is compared to expert estimates of the same correspondences. Deviations indicate that the representation mapping for a construct is wrong and/or that there are weaknesses in the common ontology. There may be concepts missing from the common ontology, or there may be taxonomical relations between ontology concepts missing, e.g., a missing generalisation relation from a sub- to a superclass. If left undetected, missing taxonomical relations can lead to redundancies in the common ontology when the same subclass is added several times because it cannot be retrieved as a specialisation of its superclass. In this way, correspondence measures can also aid *elimination of redundancy* in the common ontology.

Correspondence measures as representative of correspondences are useful as *high-level guides* for model-to-model translation and other cross-language services. The representation mappings and common ontology provide the details for how to translate between modelling constructs belonging to different languages, as soon as the pair of modelling constructs to translate between have been decided. But it offers less help with deciding which constructs in one language to translate into which constructs in another. The correspondence measures can help by suggesting, for each construct in a language, which constructs in the other language that are most suitable as targets for, e.g., translation, leaving the final choice to be made by the model manager. When the construct-to-construct correspondences at the language-level have been established in this way, the representation mapping and common ontology are there to support the detailed construct-level mappings.

## 6. Language Quality Framework

Together, the representation mappings, common ontology and correspondence measures contribute towards integrated use of models expressed in different languages. But there is also a need to select suitable languages to include in the UEML in the first place. For example, to quickly enrich the common ontology, it may be better to incorporate a much used and relatively complete language early on than a narrower language used only by specific communities. Later, when using UEML, there is also a need to select suitable languages for particular purposes among the many available. UEML therefore includes a *language quality framework* (Anaya, Berio & Verdecho 2007) that aids language selection by:

- defining the *concept of quality* of a modelling language;
- supporting *methodical, goal-dependent evaluation* of the quality of enterprise and IS modelling languages.

The current quality framework has adapted and extended the SEQUAL quality framework (Krogstie 1998, 2005), which provides a model of the quality of models, later extended to also account for the quality of languages. SEQUAL identifies 8 *quality types* for characterising what quality is: physical quality, empirical quality, syntactic quality, semantic quality, perceived semantic quality, pragmatic quality, social and organisational quality. For example, *semantic quality* is the correspondence between the model and the domain. SEQUAL also identifies

several *types of appropriateness*, each indicating a language aspect that must be considered when assessing whether a language is appropriate for a particular purpose (Krogstie 1998, 2005). For example, *comprehensibility appropriateness* reflects the ease with which the language its model can be understood by a certain audience. In SEQUAL, each quality type is related to one or more appropriateness types and vice versa. For example, domain appropriateness is used to assess physical and semantic qualities. Therefore, the different types of appropriateness provide the context for evaluating the related quality types.

In addition to SEQUAL, the UEMML quality framework has been inspired by two additional quality frameworks: Moody's framework (2003) and ISO/IEC 9126 international standard for assessing software product quality (ISO/IEC 2001). These two frameworks have been adapted and aligned with SEQUAL's appropriateness types through a generalisation hierarchy (Berio, Opdahl, Anaya & Dassisti 2005b).

The resulting *appropriateness types* in UEMML's quality framework remain too general to allow concrete evaluations (Anaya, Berio & Verdecho 2007). Therefore, the framework also covers *requirements* and *criteria*. Requirements are collected from users (actors or experts), asking them how enterprise modelling should contribute towards enterprise integration and interoperability, based on a requirements base established in the previous UEMML Thematic Network (UEMML-TN 2002-2003). Criteria are the *operational*, or *measurable*, counterparts of requirements. Each criterion can in turn be related to one or more *appropriateness types*, making it clear to which quality types the criterion contributes. The framework provides two complementary ways of collecting data for evaluating criteria. The *language template* is used to gather general and factual information about a language, such as its notations and meta models, whereas the *language-evaluation questionnaire* comprises both questions derived from current criteria and an associated glossary (Verdecho & Matulevičius 2007).

The framework also covers *language descriptions*, which cover, e.g., a language's owner and version; *goals*, which are aggregations of criteria for the purpose for evaluating language quality; *metrics-for-goal*, which are selected metrics relevant to a specific goal (metrics are needed to perform criteria assessment); *metric evaluations*, which are specific evaluations (for instance, a value) of a single metric on a specific language; *combined metrics evaluations*, which are combined evaluation of several metrics evaluations for a given language and a given goal (an explicit combined metrics evaluation makes explicit how several single metrics are combined, e.g., with a weighted formula, to evaluate the quality of a language with respect to a given goal; additionally, it is useful because the same metrics evaluation can be used several times if needed).

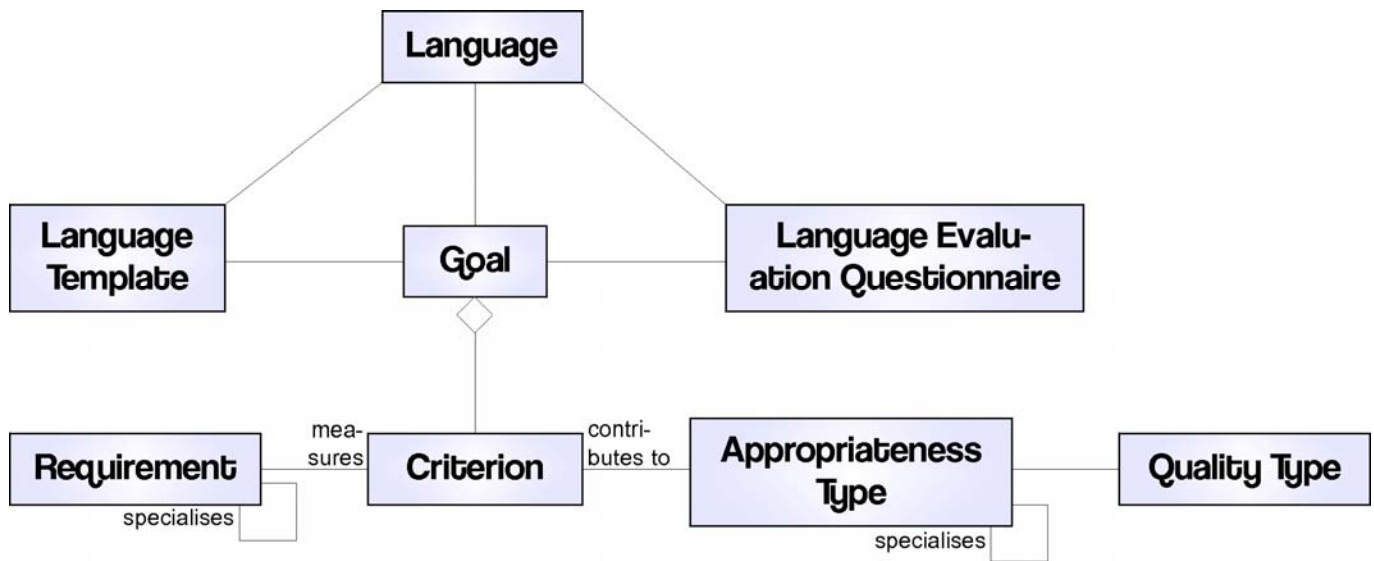


Figure 11: The main classes of the UEML language quality framework, used to classify and select modelling languages.

The UML class diagram in Figure 11 shows the key concepts used to evaluate the quality of modelling languages in UEML. The associated *quality evaluation method* gives a clear picture of how to evaluate and select one or more enterprise and/or IS modelling languages for a specific purpose. The first task is to define the goal as an aggregation of criteria and then select suitable metrics for each criterion. A list is made of languages to be evaluated. The language template is used to collect factual information about each language, and the language-evaluation questionnaire is used to collect subjective opinions. Hence, whereas only a single filled-in language template is needed for each language, multiple filled-in questionnaires from language users are usually needed. Once the selected criteria are assessed by using selected metrics and storing these assessments as metrics evaluations, combined metrics evaluations are calculated and stored. Finally, languages must be suitably selected based on the results stored as combined metrics evaluations. Before its use, an enterprise may undertake a customisation of the quality framework: This simply means to define additional requirements, appropriateness types, criteria and metrics.

Table 2. Examples of criteria for language appropriateness, along with their descriptions.

Nº	Criterion	Language quality appropriateness	Description
6	Formal semantics	Technical actor appropriateness	There is a formal description of the semantics of each construct so a model cannot be misunderstood (from a mathematical point of view)..... Alternative representations are also ruled out.
7	Graphical representation	Comprehensibility appropriateness	The language possesses a graphical representation of each construct so it can be easily used by users.
8	Widely used / supported by the scientific / industrial community?	Organisational appropriateness	Influence the impact of INTEROP researches and the acceptance of the work performed on UEMML

**Example.** Table 2 provides examples of of criteria for language appropriateness, along with their descriptions.

## 7. Meta-Meta Models

The UML class diagrams of the language and construct description approach (Section 3), of the common ontology (Section 4), of the correspondence analysis approach (Section 5) and of the quality framework (Section 6) are all *meta-meta models*. They are meta-meta models because models of modelling languages are meta models and because Figures 3, 6 and 10-11 are models of how to model aspects of modelling languages (thus of how to model meta models). The UML diagrams are intended as illustrations only. For example, Figures 3 and 6 do not show attributes and omit several association classes and abstract classes. More detailed meta-meta models can be found in (Opdahl 2006).

Whereas the representation mappings connect Figures 3 and 6, the meta-meta models of the correspondence analysis and language quality frameworks in Figures 10-11 are currently connected to Figure 3 only through the *language description* in Figure 6. Further work should establish a single combined, yet modular, meta-meta model that covers all constituents of the UEMML framework, the *overall UEMML meta-meta model*.

The meta-meta models, in particular the construction description approach and the common ontology in Figures 3 and 6, serve the same purpose for UEMML as the OMG's MetaObject Facility (OMG-MOF 2009) does for UML. However, the two meta-modelling approaches differ in two ways: in how they deal with semantics and in how coarsely they describe modelling constructs. At the semantic level, the UEMML meta-meta model distinguishes between the ontological concepts of things, properties, states and transformations, which are considered ontologically fundamental for enterprise and IS models. In contrast, the OMG's MOF makes weaker ontological assumptions, representing modelling constructs in terms of meta classes (MOF::Classes) along with their properties,

associations and association roles. In consequence, MOF meta models tend to focus first on representing syntactical information about languages. To the extent that problem-domain semantics are also represented, they are grafted onto the primary (abstract and/or concrete) syntactical meta-model structure. Indeed, MOF does provide some built-in semantics, but these are semantics for MOF-based repositories, i.e., semantics required for creating objects according to MOF specifications, *not* problem-domain semantics that describe which phenomena in the problem domain a modelling construct is intended to represent. This contrasts UEML's language and construct descriptions, which *only* deal with problem-domain semantics. At the semantic level, UEML and MOF are thus complementary: they focus on distinct aspects of languages, the first on the ontological problem-domain semantics, the second on repository semantics.

At the construct-description level, UEML's use of *separation of reference* leads to descriptions of modelling constructs that are more *fine grained* than most MOF descriptions. Indeed, Opdahl & Henderson-Sellers (2005) argue that the quality of MOF's meta models is compromised by the many *referentially redundant* modelling constructs that result from using MOF, where a referential redundancy occurs when multiple modelling constructs or model elements represent the *same* thing or property in the problem domain. The UEML meta-meta model, in contrast, has been explicitly designed to discourage referential redundancies, leading to language descriptions that better support activities such as consistency checking, automatic update reflection and reusing model contents across modelling languages and notations.

## 8. Tools

UEML is supported by a set of prototype tools realised using a selection of existing technologies. There are currently five tools in the set:

- *UEMLBase Repository* is a Protege-OWL realisation of the representation and ontology meta-meta models of Figures 3 and 6, translated into OWL.
- *UEMLBase Editor* is an emerging set of Eclipse GMF-based editors for browsing and updating the contents of the UEMLBase repository.
- *UEMLBase Manager* is a Java-plugin for Protege-OWL that provides merging, reporting and other housekeeping functions for the repository.
- *UEMLBase Verifier* is a set of Prolog rules and a Prolog rule checker that support formal verification of the contents in the UEMLBase repository, for example to check cardinality constraints and ensure that the construct descriptions are concrete. Prolog was chosen instead of newer technologies, such as SWRL, because of its high availability, robustness and general versatility (Mahiat 2006).
- *UEMLBase Analyser* is another Java-plugin, which uses the repository to compute similarity measures between UEMLBase constructs based on the meta-meta model in Figure 10. It thus paves the way for consistency checking, automatic update reflection, model-to-model translation across languages, as well as other integrated model uses.

Each tool strives to be consistent with the meta-meta models of Figures 3, 6 and 10-11, although they realise more specific implementation models, such as OWL, Eclipse EMF, Java classes and Prolog facts. Hence, the meta-meta models is used to support interoperability within the UEML tool set.

## 9. Discussion

The paper has presented the main constituents of the UEML and explained how they are related. Languages, possibly selected with the aid of the quality framework, are described using separation of reference according to Section 3. The descriptions of the states of affairs are then mapped into the common ontology of Section 4. It thereby becomes possible to establish correspondences between different constructs in terms of their mappings into the common ontology as in Section 5. The selection of modelling languages is guided by the quality framework of Section 6. In the long term, the most used and useful concepts in the common ontology can be used to form a *core UEML language* for enterprise and IS modelling. In the long term, UEML could also contribute towards developing a *web of languages and of models* in a way that resembles the touted semantic web of semi-structured data (Berners-Lee, Hendler & Lassila 2001), which is currently emerging in areas such as *e-science* and *e-government* (Shadbolt, Hall & Berners-Lee 2006).

From an initial set of around 25 concepts taken more or less directly out of Bunge's ontology and the BWW model, the common UEML ontology has grown to comprise 110 concepts. Most of them have resulted from analyses of individual modelling constructs using separation of reference. (A few initial higher-level remain to organise and structure the four taxonomies.) As part of the Interop-NoE work, 130 constructs from the following 10 languages have been mapped into this ontology: ARIS, BMM, BPMN, GRL, IDEF3, ISO/DIS 19440, KAOS, coloured Petri nets, UEML 1.0 and selected diagram types from UML 2.0. However, they are not all described in equal detail and none of them are yet fully validated. The languages, constructs, mappings and ontology have all been stored in the UEMLBase Repository, supported by the Editor, Manager, Verifier and Correspondence Analyser tools.

The standardised approach to language and construct description has turned out to have several advantages, in particular at the modelling construct level. The structured descriptions become complete, consistent, cohesive and, thus, more learnable and understandable. It therefore becomes easier to compare them to one another. The structured approach also offers systematic and detailed advice on how to proceed when analysing individual language constructs. It encourages highly-detailed construct description, which leads to languages that are integrated at a fine level of detail. It supports ontological analysis in terms of *particular* classes, properties, states and events, and not just in terms of the *concepts* in general.

Instead of starting from scratch by building a new integrated modelling language, UEML leverages existing investments in enterprise modelling-language design and in their supporting tools by integrating existing languages with one another through a common hub. This contrasts the UML approach for software modelling, which built a new language with a limited set of diagram types from the bottom up (although the new language was extensible through stereotyping, tags and constraints, and it was based on selected existing notations like use cases). We

suggest that enterprise and IS modelling and software modelling may need different integration strategies because the types of phenomena (things, properties, states and transformations) to can be part of an enterprise or information system is virtually unlimited, calling for fundamentally open-ended language integration approaches. Software modelling also deals with a highly complex domain, but one that is in the end limited by the evolution of software technology. Software modelling may therefore be better served by a closed, but potentially also more tightly integrated modelling language built from the bottom up.

The UEML framework has *positive network externality*, in the sense that incorporating an additional construct or language becomes:

- *more valuable* the more constructs and languages that have already been incorporated, because the additional language becomes interoperable with a larger number of other languages;
- *less costly* because reusing an enriched common ontology and existing representation mappings provide good reference examples and because the cost of maintaining tools and infrastructure can be shared by more UEML users.

Similar positive network externality effects can be expected at the *model level* beside the language level discussed here, although this needs to be explored in further work. Indeed, the most extensive benefits can be expected at the model level, where UEML will enable integrated use of models expressed in different languages. Taking an analogy from traditional architecture and construction, it is likely that many of our most beautiful and useful buildings would have been impossible to design and build had different and semantically unconnected languages been used for all the parts of the building and stages of the building process.

Early experience with the construct description approach indicated that it was difficult to use because it was based on a novel, unconventional way of thinking about the semantics of modelling constructs. It was sometimes hard to find the appropriate classes, properties, states and events in the common ontology to use when describing a construct. Also, it was sometimes hard to determine exactly which part of a language that constitutes a modelling construct. As part of the Interop-NoE, tools and tutorials were developed that have seemingly resolved many of these problems. Also, early drafts of the common ontology have become available along with exemplary representation mappings. As a result, the first draft of several of the most recent language incorporations could be made by students with little direct supervision.

The language quality framework is also helpful for users that need to select the most suitable language for a practical setting. The framework captures the users' perspectives on the setting in the form of user requirements, and guides the use of requirements for evaluating languages for that particular setting.

## **10. Conclusion and Further Work**

UEML is an ambitious, long-term effort that will require several years of cooperation between academia and industry. The overall challenge for further work is to extend the theory and tools developed by the Interop-NoE

network to support *practical integrated use of models and languages*. In order to make UEML a working hub for exchanging information contained in models across modelling languages, further research must therefore focus particularly on how industrial enterprise and IS models can be automatically converted into UEML form and back to conventional model form, but in a different language. Although several paper-and-pencil trials have demonstrated the feasibility of our ideas (Berio, Opdahl, Anaya, Dassisti, 2005b; Matulevičius, Heymans, Opdahl, 2007; Harzallah, Berio, Opdahl, 2007), detailed methods for integrated model use still need to be developed and implemented.

For UEML-supported integrated model use to be tested in large-scale, realistic settings, the common ontology and representation mappings must be verified, validated and improved. The current ontology and mappings have been contributed by several Interop-NoE research teams working in a distributed manner. The most immediate challenge is to improve the ontology and mappings in two directions. Firstly, the Editor and Verifier tools are being extended and improved. Secondly, the Correspondence Analyser tool is used to compare correspondences calculated from the common ontology and the representation mappings with correspondence estimates provided by human experts. The comparisons are used to identify weaknesses in the representation mappings. For example, when two constructs are considered similar by human experts, but not by the Correspondence Analyser, the reason might be that one or more ontology concepts have been duplicated. Accordingly, when the Analyser, but not the human experts, deem two constructs similar, the reason may be weaknesses in the generalisation hierarchies in the ontology. In this way, verification not only supports improving the representation mappings but also controls the quality of the common ontology.

As for the overall UEML framework, an obvious path for further work is to connect the meta-meta models for language and construct description and for the common ontology with the one for the quality framework. Also, the combined meta-meta model must be extended to account for the presentation part of language and construct description and for construct correspondences. In addition to tying together the overall approach, this work can be expected to reveal further possibilities, such as deriving quality and appropriateness metrics for languages, not only at the language level, but also at the construct level from the detailed UEML ontology and mappings. A further extension would be to aid users in selecting not only the most suitable languages to use in practical setting, but also which models to integrate. The users' perspectives on the model level could be captured in terms of *usage and business benefits*, suggesting how integrated use of models can address key business requirements and, most importantly, suggesting *which* models that are worth connecting using UEML. The latter point is important for assessing cost-effectiveness, as it is unlikely that UEML will be able to support fully automatic (and thus effortless) integrated use of models in the near term.

These and other possible future developments have been organised in a *UEML roadmap* comprising several research directions, each detailed by specific actions (Opdahl & Berio 2006b): 1. Language breadth – include more languages; 2. Ontological depth – refine the common ontology; 3. Ontological clarity – elaborate the common ontology language; 4. Presentation – extend the support for presentation issues; 5. Mathematical formality – define

UEML semantics formally; 6. Tool support – develop prototype tool with GUI and validation support; 7. Model management – provide support for model management in addition to language management; 8. Validation – structural and behavioural language and model validation; 9. Dissemination – make UEML known in industry and academia and as a standard; 10. Community – establish and maintain a committed and cohesive community for managing and evolving the UEML. Additional directions that deal specifically with the language quality framework are: 1. Continuing the development of the quality framework by introducing new criteria and extending the questionnaire accordingly; 2. Continuing the accommodation of existing quality frameworks by specialising appropriateness; 3. Gradually developing supporting tools based on the meta-meta model, starting from the current simple support for filling-in the questionnaire to complete functionality to define and evaluate metrics; 4. Launching use of the quality framework and especially by performing evaluations of languages for developing a core language; 5. Extending the scope of the quality framework to cover *model quality* in addition to language quality using a corresponding overall approach. For example, more specific quality frameworks can be used to systematically introduce new appropriateness measures and to specialise existing ones. The roadmap still needs to be extended to account better for correspondence analysis.

The UEML approach may even be useful outside enterprise and IS modelling, e.g., for software modelling. Significantly, none of the central components of the UEML framework are specific to enterprise modelling. They may therefore be used for a wider set of modelling domains.

## Acknowledgment

The authors are indebted to all the partners of Interop-NoE and, in particular, to the researchers participating in its *Domain Enterprise Modelling*. The authors are indebted to the many students who have contributed to the work, including Emmanuel Blanchard, Aurelie Dossogne, Cedric Jeanmart, Alf Harry Karlsen, Jeremy Mahiat, Christophe Tu, Torbjørn Vefring and Tomas Zijdemans. This work was partially funded by the Interop Network of Excellence and by the Interuniversity Attraction Poles Programme, Belgian State, Belgian Science Policy.

## References

- Anaya, V., Berio, G., and Verdecho, M.J. (2007). Evaluating Quality of Enterprise Modelling Languages: The UEML Solution. *Proc. I-ESA 2007*, Funchal, Portugal.
- Berio G., Anaya V., and Ortiz A. (2004). Supporting Enterprise Integration through a Unified Enterprise Modeling Language. In *Proc. of EMOI 2004 (Enterprise Modelling and Ontologies for Interoperability)*, Grundspenkis, J., Kirikova, M. (eds.), joint with CAiSE\*04, Riga Technical University, 3:165-176.
- Berio, G., Opdahl, A., Anaya, V. and Dassisti, M. (2005a). Deliverable DEM1. Publicly available at [www.interop-noe.org](http://www.interop-noe.org).

Berio, G., Opdahl, A., Anaya, V., and Dassisti, M. (2005b). Deliverable DEM2. Publicly available at [www.interop-noe.org](http://www.interop-noe.org).

Berio, G., Opdahl, A., Anaya, V., and Dassisti, M. (2006). Deliverable DEM3. Publicly available at [www.interop-noe.org](http://www.interop-noe.org).

Berners-Lee, T., Hendler, J., Lassila, O. (2001). The Semantic Web. *Scientific American Magazine* - May, 2001

Blanchard E., Kuntz P., Harzallah M. and Briand H. (2006). A Tree-Based Similarity for Evaluating Concept Proximities in an Ontology. In *Proc. 10th Conf. of the International Federation of Classification Society*, pp.3–11. Springer.

Brinkkemper, S., Saeki, M. & Harmsen, F. (1998). Assembly Techniques for Method Engineering. *Proc. CAiSE'98*, pp. 381-400. LNCS, Springer.

Budanitsky, A. (1999). Lexical Semantic Relatedness and its Application in Natural Language Processing, Technical report, Univ. of Toronto

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.

Dallons, G., Heymans, P. and Pollet, I. (2005). A Template-based Analysis of GRL, in *Proc. of EMMSAD'05 (CAiSE\*05)*, Tenth International Workshop on Exploring Modeling Methods in Systems Analysis and Design, pp. 493-504.

Dossogne A. and Jeanmart C. (2007) Evaluation of ARIS and BPMN using the UEML approach. Master thesis, University of Namur.

Goossenaerts, J., Gruninger, M., Nell, J.G., Petit, M. and Vernadat, F. (1997). Formal Semantics of Enterprise Models. In *Proc. of ICEIMT'97*, K.Kosanke and J.G Nell. (Eds.), Springer- Verlag.

Gower, J.C. and Legendre, P. (1986). Metric and Euclidean Properties of Dissimilarity Coefficients', *J. of Classification*, 3, 5–48.

Harzallah M., G. Berio, et A.L. Opdahl (2007). Incorporating IDEF3 into the Unified Enterprise Modelling Language (UEML). In *Proc. VORTE 2007*, joint with EDOC07.

Henderson-Sellers, B., Gonzalez-Perez, C., Serour, M.K. & Firesmith, D.G. (2005). Method Engineering and COTS Evaluation. In *Proc. 2<sup>nd</sup> Int W'shop on Models and Processes for the Evaluation of Off-the-shelf Components (MPEC '05)*. ACM.

Heymans, P., Saval, G., Dallons, G. and Pollet, I. (2005). A Template-Based Analysis of GRL: Book chapter, in *Advanced Topic in Database Research - Volume 5*. Idea Group Publishing.

INTEROP-NoE (2003-2007). Interop Network of Excellence. [www.interop-noe.org](http://www.interop-noe.org), 2003-2007.

ISO/IEC Standard 9126 (2001). Software product quality, International Standards Organisation (ISO). International Electrotechnical Commission (IEC).

Jochem, R. (2002). Common Representation through UEML – Requirement and Approach. In *Proc. of ICEIMT 2002*, Kosanke K., Jochem R., Nell J., Ortiz Bas A. (Eds.), Polytechnic University of Valencia, Valencia, Spain, April 24-26, Kluwer. IFIP TC 5/WG5.12.

Krogstie, J. (1998). Using a Semiotic Framework to Evaluate UML for the Development for Models of High Quality. Siau K., Halpin T., (eds) *Unified Modelling Language: System Analysis, Design and Development Issues*, IDEA Group Publishing, pp. 89-106.

Krogstie, J. (2005). Evaluating UML Using a Generic Quality Framework. *Encyclopaedia of Information Science and Technology*. M. Khosrow-Pour Editor, IDEA Group Publishing.

Lin D. (1998). An Information-Theoretic Definition of Similarity. In *Proc. 15th International Conference on Machine Learning*, pp. 296–304. Morgan Kaufmann.

Mahiat, J. (2006). A Validation Tool for the UEML Approach. Master thesis, University of Namur.

Matulevičius R., Heymans P., Opdahl A. L. (2006). Comparison of Goal-oriented Languages using the UEML Approach. In *Interoperability for Enterprise Software Applications*, Panetto H., Boudjlida N. (eds), pp 37-48. ISTE.

Matulevičius R., Heymans P., Opdahl A. L. (2007a). Comparing GRL and KAOS using the UEML Approach. In *Enterprise Interoperability II. New Challenges and Approaches*, Concalves, R. J., Muller, J. P., Mertins, K., Zelm, M. (eds.), pp 77-88. Springer-Verlag.

Matulevičius R., Heymans P., Opdahl A. L. (2007b). Ontological Analysis of KAOS Using Separation of Reference. In *Contemporary Issues in Database Design and Information Systems Development*, Siau K. (ed.), pp. 37-54. IGI Publishing.

Mertins, K., Knothe, T., Zelm, M. (2004). User oriented Enterprise Modeling for Interoperability with UEML. In *Proc. of EMMSAD'04 (Evaluating Modeling Methods for Systems Analysis and Design)*, pp.25-36, joint with CAiSE\*04, Riga – Latvia, June 7-8.

Moody D.L. (2003) Measuring the Quality of Data Models: an Empirical Evaluation of the Use of Quality Metrics in Practice. *Proc. ECIS'2003*, Naples, Italy.

OMG-MOF (2009). OMG's MetaObject Facility, <http://www.omg.org/mof/> . OMG – Object Management Group. Accessed 2009-04-16.

OMG-UML (2009). UML Resource Page, <http://www.uml.org/> . OMG – Object Management Group. Accessed 2009-04-16.

Opdahl, A.L. (2006). The UEML Approach to Modelling Construct Description. Proc. 2nd International Conference on Interoperability for Enterprise Software and Applications (I-ESA 2006).

Opdahl, A.L. and Berio, G. (2006a). Interoperable Language and Model Management using the UEML Approach. Proc. G@mma 2006 (International Workshop on Global Integrated Model Management), pp. 35-42.. ACM Press.

Opdahl, A.L. and Berio, G. (2006b). A Roadmap for the UEML. Proc. 2nd International Conference on Interoperability for Enterprise Software and Applications (I-ESA 2006).

Opdahl, A.L. and Henderson-Sellers, B. (2004). A Template for Defining Enterprise Modelling Constructs. *Journal of Database Management* 15(2).

Opdahl, A.L. and Henderson-Sellers, B. (2005). Template-Based Definition of Information Systems and Enterprise Modelling Constructs. In *Ontologies and Business System Analysis*, Peter Green and Michael Rosemann (eds.). Idea Group Publishing, 2005.

Opdahl, A.L. and Henderson-Sellers, B. (2005). A Unified Modeling Language Without Referential Redundancy, *Data and Knowledge Engineering (DKE)* 55(3). Elsevier.

Panetto H., Berio G., Benali K., Boudjlida N. and Petit M. (2004). A Unified Enterprise Modelling Language for enhanced interoperability of Enterprise Models. In *Proc. of the 11th IFAC INCOM2004 Symposium*, Bahia, Brazil, April 5-7.

Ralyté, J. & Rolland, C. (2001). An Assembly Process Model for Method Engineering. In Proc. CAiSE'01, pp. 267-283. LNCS, Springer.

Rodríguez M., Egenhofer M (2003). Determining semantic similarity among entity classes from different ontologies. *IEEE Transactions on Knowledge and Data Engineering* 15(2), pp. 442–456.

Shadbolt, N., Hall, W. and Berners-Lee, T. (2006). The Semantic Web Revisited. *IEEE Intelligent Systems*, May/June 2006.

Tu, C. (2007). Ontological Evaluation of BMM and i\* with the UEML Approach. University of Namur, Master thesis.

UEML-TN (2002-2003) Unified Enterprise Modelling Language Thematic Network. 2002-2003.

van Lamsweerde, A., & Willemet, L. (1998). Inferring Declarative Requirements Specifications from Operational Scenarios. *IEEE Transactions on Software Engineering*, 24(12), 1089–1114.

Verdecho M.J. and Matulevičius, R. (2007). Language Evaluation Questionnaire for Enterprise Modelling Languages. Unpublished.

Vernadat, F. (2002). UEML: Towards a Unified Enterprise Modelling Language. *International Journal of Production Research*, 40(17):4309-4321, Taylor & Francis Group.

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

Wand, Y. and Weber, R. (1988b). An Ontological Model of an Information System. *IEEE Transactions of Software Engineering*, 16 (11):1282-1292, IEEE Press.

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

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

Yu, E. (1997). Towards Modeling and Reasoning Support for Early-phase Requirements Engineering. In *Proceedings of the 3rd IEEE Int. Symposium on Requirements Engineering (RE'97)*, IEEE Computer Society.