

Semantic Annotations for Modelling Language Interoperability

Andreas L. Opdahl

Department of Information Science and Media Studies,
University of Bergen, NO-5020 Bergen, Norway
+47 – 5558 4140

Andreas.Opdahl@uib.no

ABSTRACT

Interoperability between model-driven software technologies can become easier to achieve if the models and modelling languages used are made interoperable too. One way to achieve interoperability is to capture the semantics of modelling language constructs and model elements by mapping them to semantic models, such as ontologies. The paper proposes a set of extensions for XML-based modelling language specifications. The extensions have the potential to work equally well for model representations. The purpose is to support fine-grained and structured semantic annotations of languages and, eventually, of models. The proposal is inspired by the Semantic Annotations for WSDL and XML (SAWSDL) recommendation and on the Unified Enterprise Modelling Language (UEML). The proposal is exemplified using the Action and ActionExecution constructs from UML 2.2.

Categories and Subject Descriptors

H.0 [Information Systems]: General – *information systems modelling, enterprise modelling, semantic annotation, interoperability.*

General Terms

Documentation, Standardization, Languages, Theory.

Keywords

Enterprise modelling, information systems modelling, software modelling, web-service modelling, ontology, SAWSDL, Unified Enterprise Modelling Language, UEML, Unified Modeling Language, UML.

1. INTRODUCTION

The last remaining technical and syntactical roadblocks to software interoperability are gradually being removed by advances such as heavy-weight [20] or RESTful [3] web services. In consequence, semantic roadblocks to interoperability are

coming to the fore. At the same time, software and information systems are becoming increasingly model-driven, as witnessed by the OMG's model-driven architecture [9] and related initiatives such as SoaML [8, 10]. Model-driven technologies are good news for semantic interoperability because, when the semantics of software systems have been expressed as models, those semantics can be harnessed to make the software systems themselves interoperable. For example, if two web services are both model-driven, their models can be used to generate semantic descriptions of the web services' operations and their input and output data types. The semantic descriptions can in turn be used to make the web services interoperate using semantic web-service technologies such as OWL-S [16] or WSMO [21] or their successors.

Unfortunately, the models used to drive software and information systems today seldom provide the semantic precision needed by OWL-S, WSMO and similar semantic technologies. The different languages used are usually not semantically interoperable either. We want to amend this situation by facilitating semantically-precise, integrated use of software and information system (IS) models across modelling languages. For this to happen, the modelling languages must be made semantically precise and interoperable too. We therefore propose a set of extension attributes and elements for XML-based modelling language specifications that allow precise descriptions of the semantics of modelling-language constructs through mappings to semantic models, such as ontologies.

The annotations we propose extend and are fully compliant with the Semantic Annotations for WSDL and XML recommendation (SAWSDL [19]). Hence, our proposal does not specify a new language or ontology for representing semantics. Instead it proposes annotation mechanisms by which concepts in existing ontologies can be referenced from within existing XML-based language specifications. The proposal places only weak restrictions on the specifications that will be annotated and on the ontology that will be referred to. Whereas the paper mostly discusses the language level, it is likely that the extensions we propose will be usable and useful at the model level too, where model elements will map to both ontology concepts and their instances.

The rest of the paper is organised as follows. Section 2 presents the *background* of our work, the Unified Enterprise Modelling Language (UEML [1]) and the SAWSDL standard for semantic annotations. Section 3 explains how modelling languages and constructs are *described* in UEML. Section 4 proposes our new

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WIMS'11, May 25-27, 2011 Sogndal, Norway

Copyright © 2011 ACM 978-1-4503-0148-0/11/05... \$10.00

language annotations and exemplifies them using the Action and ActionExecution constructs from UML activity diagrams. Section 5 discusses the proposal, before Section 6 finishes the paper with conclusions and paths for further work.

2. BACKGROUND

2.1 UEML

The Unified Enterprise Modelling Language 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. 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. The most central parts of the UEML approach are: 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 [1].

A central idea behind UEML is that the semantics of individual modelling constructs are described by mapping them into a *fine-grained* and *well-structured* ontology. The ontology is finely grained to support precise reasoning about semantic correspondences between modelling languages and their constructs. And it is well structured to ensure that it remains manageable as the number of modelling languages and constructs continues to grow. As a result, the semantic mappings into the ontology become fine-grained and well-structured too, using a technique that will be presented in Section 3 and which is the starting point for our proposal in Section 4.

2.2 Semantic annotation

Semantic annotation (or semantic mark-up) enriches information resources with additional semantic information, typically by referencing external semantic models, such as ontologies. One type is SAWSDL [19] and WSDL-S [18], which annotate information in WSDL with ontologies. Another type is OWL-S [16], SWSF [17] and WSML [21], which offer dedicated ontology languages for semantic web services. Hence our focus in this paper is on standards for semantic annotations related to *software services*. While we acknowledge that there are many other useful approaches to semantic annotation, we choose this starting point because software is becoming increasingly packaged and presented as services. Using similar semantic annotations for software and information systems languages and models and for software services thereby makes life-cycle management simpler.

Our proposal is based on Semantic Annotations for WSDL and XML Schema (SAWSDL [19]), which describes how semantic annotations can be added to WSDL documents to give semantics to its operations and input and output message types. For example, SAWSDL provides categorization information that can be used when publishing the web service in a registry and during web service discovery and composition. In addition, SAWSDL defines an annotation mechanism for specifying the data mapping of XML Schema types to and from a semantic model or ontology. These mappings can be used during invocation, in particular to

mediate between web services. SAWSDL's annotation mechanism is independent of which language the semantic model or ontology is expressed in, whether RDF, OWL or another format. It is also independent of the particular language used to express data mappings. And although it is primarily intended for WSDL documents and XML schema, we will see that it is easily transferable to other XML-based specifications.

To accomplish semantic annotation, SAWSDL defines extension attributes and elements that can be applied both to WSDL and to XML Schema. SAWSDL defines the following annotation mechanisms [19]:

- “[A]n extension attribute, named *modelReference*, to specify the association between a WSDL or XML Schema component and a concept in some semantic model. It is used to to annotate XML Schema type definitions, element declarations, and attribute declarations as well as WSDL interfaces, operations, and faults”. The *sawSDL:modelReference* attribute thereby becomes the basic mechanism for mapping XML-elements to ontology concepts. The value of a *sawSDL:modelReference* attribute is a list of URIs that identify concepts in one or more existing ontologies.
- “[T]wo extension attributes, named *liftingSchemaMapping* and *loweringSchemaMapping*, that are added to XML Schema element declarations and type definitions for specifying mappings between semantic data and XML”. These attributes are used when the semantic mapping of an XML element depends on its attributes and child elements. The values of the *sawSDL:liftingSchemaMapping* and *sawSDL:loweringSchemaMapping* attributes are lists of URIs to resources that contain schema mappings expressed, e.g., using XSLT, SPARQL or XQuery, although SAWSDL does not specify a particular mapping language. The *liftingSchemaMapping* attribute maps from XML-element to ontology concepts and the *loweringSchemaMapping* maps back again, so the two should correspond.
- An extension element, named *sawSDL:attrExtensions*, allows any attribute and provides “a general mechanism for adding extension attributes where attribute extensibility is not allowed, but element extensibility is allowed.” This element can be added to a specification to allow *sawSDL:modelReference* and other extension attributes where they would otherwise be prohibited (because some XML elements only allow a restricted set of pre-defined attributes).

SAWSDL allows multiple semantic annotations, so that the same XML elements can be annotated with pointers to different, perhaps competing, ontologies. According to [19], “[m]ultiple schema mappings are interpreted as alternatives whereas multiple model references all apply. SAWSDL does not specify any other relationship between them.”

3. DESCRIBING SEMANTICS

This section will describe construct description in UEML and review the structure of the ontology that has resulted from the UEML work.

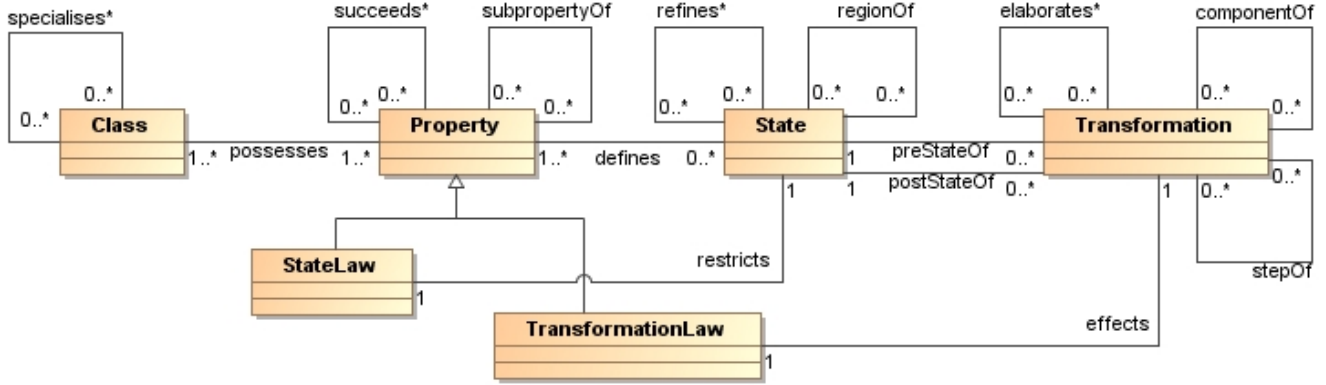


Figure 1. Structure of the UEML ontology (simplified, from [12]).

3.1 Construct description

UEML describes the semantics of individual modelling constructs by mapping them into their ontologically atomic parts by answering the following six questions [1, based on 14]:

- “Which instantiation levels is the construct intended to represent?” A modelling construct is used to represent either the *instance* or *type* level or *both*. For example, UML-Class and UML-Object are very similar both in structure and semantics, but one represents types, the other instances.
- “Which modality (or mode) is the construct intended to represent?” Some model elements may not assert objective facts, but instead state human intentions: wanting something to be the case or happen, permitting or prohibiting something, knowing something etc. Most modelling constructs represent facts, but some, such as goal modelling constructs, represent someone's intention.
- “Which things or class(es) of things is the construct intended to represent?” Most modelling constructs somehow represent one or more things (if it is *instance level*) or classes of things (if it is *type level*). 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 things or classes thereof.
- “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* (that belong to only one thing/class) or *relations* (properties that are *mutual* to several things/classes). Some properties may be *complex*, having other properties as *subproperties*. Some complex properties are even *laws* that impose structural (state laws) or behavioural (transformation laws) restrictions on their subproperties.
- “Which states is the construct intended to represent?” Some modelling constructs are used to represent a *state* in one or more things or classes of things. The state law

that *restricts* the state has properties of those things/classes as its subproperties.

- “Which transformations is the construct intended to represent?” Some constructs are used to represent a *transformation* of one or more things or classes of things from one state to another. The transformation law that *effects* the transformation has properties of those things/classes as subproperties. The transformation takes the thing/class from a *pre-* to a *post-state* defined by the subproperties.

The idea behind the six questions is that, whereas many existing approaches to language semantics map modelling constructs one-to-one with concepts in an ontology, most modelling constructs in fact represent a small *scene* (or *state of affairs*) played by several (classes of) things and their properties, perhaps along with states and transformations, each of them playing a particular *role*.

Example: We will illustrate construct description with an example from UML 2.2, whose superstructure specification is available in XMI (XML-based Model Interchange) format from OMG's web site. An *Action* in UML 2.2 is “a named element that is the fundamental unit of executable functionality. The execution of an action represents some transformation or processing in the modeled system, be it a computer system or otherwise” [11], and it “represents a single step within an activity, that is, one that is not further decomposed within the activity.” The *Action* construct has five attributes of its own: sets of *input* and *output pins*, a *context* and a *pre-* and a *postcondition*. It also inherits a *name*. To incorporate the UML-Action construct into UEML, its semantics is described by answering the six questions as follows:

1. *Instantiation level:* UML-Action is *type level* because it represents an action of a UML-Classifier (which belongs on the type level and constitutes the action's context).
2. *Modality:* UML-Action is *factual* because it represents an actual action in the context of a model (although the model itself can have another modality, e.g., it may represent a possible or wanted future situation).
3. *Thing/classes of things:* A UML-Action represents a single class of *executor things* that perform the action.

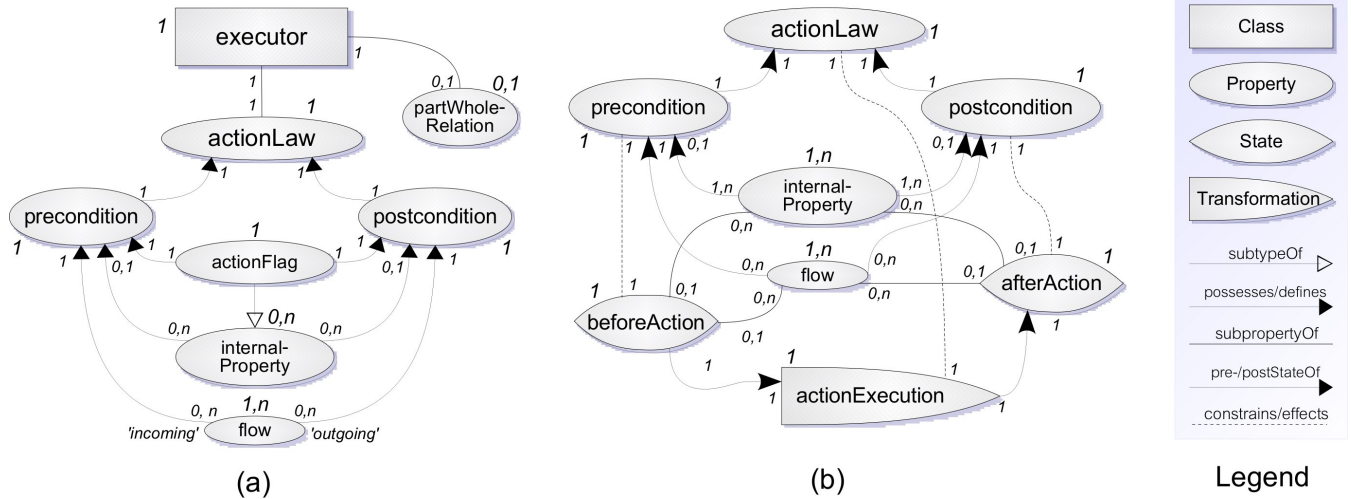


Figure 2. Illustration of UML-Action and ActionExecution described using UEML (simplified, based on [12]).

4. *Properties of things/classes:* A UML-Action has a *name* and it represents a number of '*incoming*' and '*outgoing*' flows of the executor thing. Each flow may have a subproperty that represents its *content* at a particular time (either the content property is the exchanged data or signal or the content is a *mutual* property that relates the flow to an information or material resource). The UML-Action also represents zero or more *internalProperties* of the action and an *actionFlag*, which indicates whether it is executing or not at a particular time. The UML-Action may also represent a *partWholeRelation* to its execution context, if it is a different thing from the executor. UML-Action also represents an *actionLaw* that constrains the contents of its '*incoming*' and '*outgoing*' flows, its *internalProperties* and *actionFlag*. The actionLaw even has two *sublaws* (subproperties that are laws), so that the *precondition* sublaw is satisfied when the action starts and the *postcondition* sublaw when it ends.
5. *States of things/classes:* UML-Action is not a behavioural construct, so its description does not contain states. A UML-ActionExecution, however, extends UML-Action with a *before-* and an *afterState*. The beforeState is constrained by the precondition of the action and the afterState by its postcondition.
6. *Transformations of things/classes:* UML-Action does not represent transformations either, but UML-ActionExecution represents an *actionExecution* transformation from the before- to the afterState, effected by the actionLaw. □

Hence, instead of mapping modelling constructs one-to-one with concepts in an ontology, UEML describes each modelling construct as a scene where the roles are played by things/classes and their properties, perhaps along with states and transformations.

Example: The roles in the UML-Action scene are played by executor, actionLaw, precondition, '*incoming*' flow etc. as already described. The different roles are related according to the meta model in Figure 1 as shown in Figure 2a (omitting a few properties, including names). The executor class possesses an actionLaw and, possibly, a partWholeRelation to its context (which is, however, not represented directly by the UML-Action).

The two sublaws of actionLaw constrain the '*incoming*' and '*outgoing*' flows of the action. Subproperty relations are transitive, so the precondition may constrain not only the '*incoming*' flow, but also its contents, and correspondingly for postconditions and the overall actionLaws. Also, property possession distributes over subproperty relations, so only the most compound property, the actionLaw, is explicitly possessed by the executor. Figure 2b indicates how to extend 2a with before- and afterStates and an actionExecution transformation. A few less relevant roles from Figure 2a have been omitted from this figure [12]. □

A fuller description of UML-Actions and other UML constructs (although based on UML 2.0, not version 2.2) is found in [2], which presents a comprehensive OWL repository that currently describes 130 modelling constructs in terms of more than a hundred ontology concepts.

3.2 Ontology structure

Each of the concepts in the scenes of Figure 2 are mapped into the common *UEML ontology*, whose structure is determined by the last four questions of the previous section. The ontology is thereby partitioned into four interrelated taxonomies:

1. Its classes are organised in a conventional generalisation hierarchy, in which a more specific class *specialises* one or more less specific ones.
2. Its properties form a precedence hierarchy (*succeeds*), so that, for example, *being-alive* precedes *being-human*, because everything that possesses the latter property also possesses the former. Its state and transformation laws form sub-hierarchies of this property hierarchy.
3. Its states form a state hierarchy so that a more specific state *refines* less specific states.
4. Its transformations are arranged in a transformation hierarchy, where a more specific transformation *elaborates* one or more less specific ones.

The concepts are described by attributes, such as a unique name and a precise definition. In addition to the relations that define the four taxonomies, there are several hierarchical relations between concepts within each taxonomy. We have already mentioned the

relation between (complex) properties and their subproperties (*subpropertyOf*). There is a similar relation between compound states and their region states (*regionOf*). Between composite transformations and their components there are two relations, one for sequential (*stepOf*) and one for parallel decomposition (*componentOf*). Concepts from different taxonomies are also interrelated. For example, classes are related to the properties they *possess*; properties are related to the states they *define*; transformations have *pre-* and *post-states*, state laws *restrict* states and transformation laws *effect* transformations. Figure 1 shows the UEML ontology structure as a UML class diagram (from [1]). The division into distinct, but interrelated taxonomies makes it possible to evolve the ontology over time without increasing complexity more than necessary as new concepts are added. New classes, properties, states and transformations will always belong to a clearly identifiable taxonomy and will always have a clearly identifiable location inside it. Because the scenes that describe individual modelling constructs in UEML have the same structure as the UEML ontology they map into, Figure 1 also shows the structure of these scenes.

4. LANGUAGE ANNOTATIONS

This section presents our proposal for basic semantic annotation, based on the extensions proposed by SAWSDL, before we discuss semantic annotation structure and restrictions on the annotations.

4.1 Basic annotations

Reference type attributes: Our proposal is to annotate XML-based language specifications using UEML's approach to semantic descriptions. The annotations we propose are fully SAWSDL-compliant, although we apply them to XML specifications in general, and XMI in particular, not only to WSDL definitions and XML Schema. Our starting point is the new optional attribute *ueml:referenceType*, which is used together with a *sawSDL:modelReference* attribute to show whether the reference is to a *class*, *property*, *stateLaw*, *transformationLaw*, *state* or *transformation*.

The following example shows how an excerpt of the XMI-version of the UML 2.2 superstructure specification is annotated to describe that a UML-Action maps into the ontology class of *executing things*, the so-called *executor* class.

```
<ownedMember xmi:type="cmof:Class"
  xmi:id="Actions-BasicActions-Action"
  name="Action" ...
  sawSDL:modelReference="http://.../classes/executor"
  ueml:referenceType="class">
  ...
</ownedMember>
```

This attribute is optional because the ontology concept it refers to may already be clearly identifiable as a class, property, state law, transformation law, state or transformation in the ontology. For example, the UML-Action in the above example was described by mapping it into the common UEML ontology that has resulted from the UEML work and which clearly separates between ontology classes, properties, stateLaws, transformationLaws, states and transformations. Although the *referenceType* can often be inferred from the ontology, it can still be included in the annotations for clarity.

SAWSDL allows the value of a *modelReference* attribute to be a *list* of *anyURLs*, so that an XML element can be annotated semantically with the URLs of *several* ontology concepts. This is common because UEML describes the semantics of modelling constructs as scenes played by *multiple* roles that is each mapped to an ontology concept. The referenced ontology concepts can even belong to different ontologies, opening for multiple parallel definitions of modelling constructs.

The value of the *referenceType* attribute is therefore defined as a *list* of types, as shown in the following XML schema excerpt:

```
<xs:attribute name="referenceType"
  type="listOfReferenceType" OPTIONAL/>
<xs:simpleType name="listOfReferenceType">
  <xs:list itemType="referenceType"/>
</xs:simpleType>
<xs:simpleType name="referenceType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="class"/>
    <xs:enumeration value="property"/>
    <xs:enumeration value="stateLaw"/>
    <xs:enumeration value="transfLaw"/>
    <xs:enumeration value="state"/>
    <xs:enumeration value="transf"/>
  </xs:restriction>
</xs:simpleType>
```

Role names and explanations: To make multiple semantic annotations for the same specification element easier to understand, we introduce two more optional properties. The *roleName* attribute is used to give each semantic annotation (each URI in the value of a *modelReference* attribute) a unique descriptive name, and the *explanation* attribute is used to provide a textual explanation for the annotation. The following XML-schema excerpt defines the two attributes, leaving out the detailed types for now:

```
<xs:attribute name="roleName"
  type="listOfRoleName" OPTIONAL/>
<xs:attribute name="explanation"
  type="listOfExplanation" OPTIONAL/>
<xs:simpleType name="listOfRoleName">
  <xs:list itemType="xs:string"/>
</xs:simpleType>
<xs:simpleType name="listOfExplanation">
  <xs:list itemType="xs:string"/>
</xs:simpleType>
```

The following example extends the previous one, mapping UML-Action not only to its *executor* class, but also to its top-level *actionLaw* property:

```
<ownedMember xmi:type="cmof:Class"
  xmi:id="Actions-BasicActions-Action"
  name="Action" ...
  sawSDL:modelReference=
    "http://.../classes/executor,
    http://.../props/actionLaw"
  ueml:referenceType="class, prop"
  ueml:roleName="executor, actionLaw"
  ueml:explanation="..., ..."
  ...
```

```
</ownedMember>
```

Extension elements: Not all XML-based notations allow additional attributes to be added to their element types, e.g., WSDL 1.1. For such notations, SAWSDL offers *element extension* as an alternative, so that a *sawSDL:attrExtensions* element is added as a child of the specification element to be annotated. The *attrExtensions* element has a *modelReference* attribute and an optional *referenceType* attribute as defined above. The following definition repeats an earlier one, but using an extension element:

```
<ownedMember xmi:type="cmof:Class"
  xmi:id="Actions-BasicActions-Action"
  name="Action" ...>
  ...
  <sawSDL:attrExtensions
    sawSDL:modelReference="http://.../classes/executor"
    uuml:referenceType="class"
    uuml:roleName="executor"
    uuml:explanation="..." />
  ...
</ownedMember>
```

Attribute-extension elements should not be used unless needed because the language specification does not allow attribute extensions.

Local element attributes: The standard interpretation is that an attribute extension refers to its immediate parent element in the specification. But not all XML schemas allow element extensions everywhere in the specification. If it is not possible to make the attribute-extension element an immediate child of its specification element, a *ueuml:localElement* attribute can be used to point to the right element, as shown in the following example:

```
<sawSDL:attrExtensions
  sawSDL:modelReference="http://.../classes/executor"
  uuml:referenceType="class"
  uuml:localElement="Actions-BasicActions-Action" />
```

The *localElement* attribute can be defined as follows.

```
<xs:attribute name="localElement"
  type="xs:string" OPTIONAL/>
```

Attribute extension files: A final solution, when extension elements are not allowed anywhere in the language specification, is to place all the semantic annotations in a separate XML-document of *sawSDL:attrExtensions* elements. However, a separate file should only be used when it is the only alternative.

4.2 Adding semantic structure

Basic semantic structure: In UEML, the concepts in a scene that describes the semantics of a modelling construct are related. The different annotations that describe the semantics of a modelling construct in an XML-based specification must be related in the same way. In many cases, the exact relations can be derived from the structure of the ontology. For example, when a construct from a hypothetical domain-specific language has been annotated with a reference to an ontology class (“ProductionMachine”), to an ontology property (“IsWorking”) and to two states (“WorkingState” and “FailureState”), and nothing else, the relation between the two annotations is obvious: the property

must belong to the class and it must define the two states, which are states of the class. But if we add a fourth annotation with reference to an ontology transformation (“Breakdown”), it becomes less clear which is the pre- and which is the post-state of the transformation. Presumably, the “Breakdown” concept referred to in the ontology will have as its pre-state the concept referred to by the “WorkingState” annotation and as its post-state the concept referred to by the “FailureState”. But we may not always be able to derive the exact semantic relations between a group of annotations from the ontology.

We therefore introduce additional attributes for describing relations between annotations according to the ontology structure depicted in Figure 1. The attributes are defined as follows:

```
<xs:attribute name="possesses"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="defines"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="preStateOf"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="postStateOf"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="restricts"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="effects"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="specialises"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="succeeds"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="subPropertyOf"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="refines"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="regionOf"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="elaborates"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="componentOf"
  type="listOfRelations" OPTIONAL/>
<xs:attribute name="stepOf"
  type="listOfRelations" OPTIONAL/>
```

Each attribute has a list of one or more *relations* as value, where a relation is a ‘:’-separated pair of role names occurring as values of *ueuml:roleName* attributes. The following example extends our earlier one with a *possesses* relation between the *executor* and its *actionLaw* property, as shown in Figure 1a:

```
<ownedMember xmi:type="cmof:Class"
  xmi:id="Actions-BasicActions-Action"
  name="Action" ...
  sawSDL:modelReference=
    "http://.../classes/executor,
    http://.../props/actionLaw"
  uuml:referenceType="class, prop"
  uuml:roleName="executor, actionLaw"
  uuml:possesses="executor::actionLaw">
  ...
</ownedMember>
```

It is not always necessary to specify relations such as this. In the UML-Action example, it can in principle be inferred that the `actionLaw` has to be possessed by the executor because all properties must be possessed by a class in the same scene and the executor is the only one available.

Matching syntactical and semantic structure: Usually, the language specification defines a detailed structure for each of its modelling constructs. For example, a typical modelling construct in the XMI-version of the UML superstructure is defined by a group of *ownedMember* elements (because they are considered “members” of UML), which in turn have several children each, such as *ownedAttribute*, *ownedRule* and *ownedComment* elements. Some of the children are not shown in diagrams, such as comments, but others have visual counterparts, such as the *role names* and *cardinality constraints* on Associations, or at least visual ramifications, such as the attributes that connect *Actions* to their *Input-* and *OutputPins*. Some of the children have identifiable semantics of their own, such as the *cardinality constraints* and “*pin attributes*” just mentioned, but not the comments and *role names*.

In order to tie semantic annotations as closely as possible to language structure, *sawSDL:modelReference* and *uEML:...-attributes* should be placed as near as possible to the XML-elements they annotate. The best is to have the extension attribute directly to an existing specification element. If this is not possible, it should be added to a *sawSDL:attrExtension* element that is a child of the specification element. Otherwise, the *sawSDL:attrExtension* element should have a *uEML:localElement* attribute that points to the specification element.

The following example shows how the *preconditions* of UML-Actions are defined by a child element that has its own semantic-annotation attributes:

```
<ownedMember xmi:type="cmof:Class"
  xmi:id="Actions-BasicActions-Action"
  name="Action" ...
  sawSDL:modelReference=
    "http://.../classes/executor, http://.../props/actionLaw"
  uEML:referenceType="class, prop"
  uEML:roleName="executor, actionLaw"
  uEML:possesses="executor::actionLaw">
  ...
  <ownedAttribute xmi:type="cmof:Property"
    xmi:id="...-Action-localPrecondition"
    name="localPrecondition" ...
    upper="*" lower="0" ...
    sawSDL:modelReference=
      "http://.../props/precondition"
    uEML:referenceType="class, prop"
    uEML:roleName="precondition"
    uEML:subpropertyOf=
      "actionLaw::precondition">
    ...
  </ownedAttribute>
  ...
</ownedMember>
```

As a result, the semantic mappings can be used to establish precise cross-language semantic correspondences not only between modelling constructs, but also between their parts. The

two levels of correspondences will play different roles when facilitating integrated model use across languages:

- Coarser correspondences between modelling constructs will assist, e.g., in deciding which modelling constructs in one language that correspond to which constructs in another. For example, it can assist in determining, for a detailed *activity* construct in some domain-specific language (DSL, e.g., [5]), which subtype of UML-Action that is most similar.
- The finer correspondences between parts of modelling constructs will aid moving information between instances of a pair of constructs, e.g., transforming an instance of one construct (in one language) into an instance of another construct (in the other language). For example, it can aid in translating the relations and attributes correctly to their closest matches, if any, in some subtype of UML-Actions. Specifically, we plan to use the detailed mappings to derive transformations expressed in standard mapping languages for moving model information across modelling-language boundaries.

4.3 Constraints

Although we have attempted to propose maximally lenient annotations, not all combinations of extensions are allowed. Figure 2a has already shown the cardinality constraints on the construct level for the elements that instantiate UML-Action, the Action nodes in activity diagrams. In addition to the usual cardinalities for relations between the roles in a scene, it presents cardinality constraints for the roles themselves. The Rule Interchange Format (RIF [6]) is the obvious way of including these and other constraints in XML-based specifications, introducing dedicated constraint attributes as follows:

```
<ownedMember xmi:type="cmof:Class"
  xmi:id="Actions-BasicActions-Action"
  name="Action" ...
  sawSDL:modelReference=
    "http://.../classes/executor, http://.../props/actionLaw"
  uEML:referenceType="class, prop"
  uEML:roleName="executor, actionLaw"
  uEML:possesses="executor::actionLaw"
  uEML:constraint="...URIs..."
  uEML:constraintLanguage="RIF-recommendation">
  ...
</ownedMember>
```

Figure 2b shows the corresponding cardinality constraints for UML-ActionExecutions. Here, the cardinalities of *internalProperty* have changed because *actionFlag* is no longer drawn separately. There can also be constraints on the element level, which constrain instances of Action elements, i.e., actions that belong to instances of an action's Classifier. An element-level constraint can be language-wide, which means that it constraints all the instances of one type of element in a language, or it can be element-specific, which means that it constrains the instances of a single element only. Whereas language-wide element-level constraints are defined once for all instances of a modelling construct, an (element-)specific element-level constraint is specified as an attribute of the model element.

Example: A construct-level cardinality constraint in UML is that a model element that instantiates an Association in a class

diagram must always connect two or more Class elements. An element-level cardinality constraint is that an Association element may have minimum and maximum cardinality constraints on its own for each of its LinkEnds, which restrict the UML Links that instantiate the Association. This constraint is in part language-wide and in part element-specific. It is element-specific because the actual minimum and maximum cardinalities are specific to each Link and its ends. It is language-wide because it is otherwise similar for all Links/-Ends, i.e., it always states that the number of Objects connected to a Link end must be between the minimum and the maximum cardinality. It is a language-wide constraint with element-specific parameters. □

In addition to cardinality constraints, there are identity constraints, which describe whether construct roles must, may or cannot be described by the same instances. One type of identity constraint is already built into UEMML: classes, properties, states and transformations are mutually exclusive on all levels. Also, state and transformation laws are mutually exclusive both with one another and with non-law properties. Identity constraints can apply to the same three levels as cardinalities, i.e., they can be construct or element level and, at the element level, they can be either language-wide or element-specific (or a combination).

This concludes our walk-through of constraints on construct descriptions in UEMML. There are other constraints than cardinality and identity constraints too, of course, in particular behavioural constraints, which we leave for further work.

5. DISCUSSION

Supporting the model level: A natural continuation of the present work is to support semantic annotations of models and model elements in addition to modelling languages and their constructs. Because model elements instantiate modelling constructs, the same annotation approach can be used. The difference is that, whereas modelling constructs are mapped to ontology *concepts* (at the “type level”), model elements will be mapped *either* to ontology concepts *or* to *individuals* (at the “instance level” that instantiate the concepts). For example, UML-Action is a modelling construct that maps to the type level. UML-Object, on the other hand, maps to the instance level. Model elements of both types will be described by scenes like the one in Figure 2. But the roles in the UML-Action element scene will map to (“type-level”) ontology concepts (that are sub-concepts of the ones mapped to by the scene for the UML-Action construct). The roles in the UML-Object element scene will map to (“instance-level”) ontology individuals (that are instances of the concepts mapped to by the scene for the UML-Object construct).

The resulting model annotations can potentially be used to establish correspondences between elements of software and information systems models in the same way language annotations describe and integrate constructs of the corresponding languages. Whereas new languages are only incorporated once in a while, models will need to be checked in and out all the time. For the model-level to be well supported by UEMML, it must therefore be quick and easy to semantically annotate new software and information systems models. Because model elements instantiate modelling constructs, the basic annotation structure is already given by the language annotations. The critical question is how to (semi-)automatically map the roles in the scene describing

a model element to the correct ontology concepts and individuals in the ontology.

Towards a distributed UEMML: Our proposal can be understood as a logical continuation of the UEMML initiative. Although UEMML promotes distributed evolution of language descriptions and ontologies, it is so far supported by centralised technology in the form of an OWL repository and associated tools. The present proposal attempts to distribute and open the UEMML work in several ways. It prepares for supporting semantic annotations in terms of other ontologies than the current UEMML ontology. It uses open standards such as SAWSDL, XMI and, implicitly, MOF in addition to the current OWL-based UEMML. It thus opens up for use of additional types of repositories and tools. At the same time, it shifts focus further away from another original intention of UEMML, to be a common language for modelling enterprises. Hence it is possible that, in the future, the name “UEMML” should be reserved for a more specific technology that centres around an actual common modelling language, perhaps a language synthesised from the modelling constructs incorporated into the current UEMML framework and defined and made extensible and interoperable by the current UEMML framework. The wider-scoped annotation framework presented in this paper should evolve in parallel with, but largely independent of, such a more language-centred “UEMML”. The initiative taken in [13] correspondingly separates out the common UEMML ontology as a Unified Enterprise Modelling Ontology (UEMO) that can evolve independently of the rest of UEMML.

Supporting integrated model use: An immediate ambition of our work – and indeed of the UEMML work it is inspired by – is to support integrated use of enterprise models, information systems models and software and service models that are expressed in different languages. We expect that the structured and fine-grained mappings we propose can be used to support cross-language services such as consistency checking between models, automated reflections of changes made to one model into another model and model-to-model translation. As part of the UEMML work, we have already tested our approach on smaller examples, but automated support is needed for more extensive studies. We have already suggested one approach in the previous section, that of using our semantic mappings to derive transformations expressed in existing mapping languages. In further work, we want to extend and adapt to UEMML the ideas proposed in [15], which outlines an ontology-based approach to cross-language model transformation, but which complements our work by paying less attention to structured and precise language description and to ontology evolution. [4] presents an approach to semi-automatically annotating (“lifting”) language metamodels with semantic mappings to ontologies, based in part on a catalogue of lifting patterns. Like us, the authors are concerned with the quality and structure of the underlying ontology, but they appear more optimistic about the prospects of semi-automatic semantic language lifting. We agree strongly that tool-supported approaches to lifting must be investigated, but our experience with incorporating a wide variety of languages into UEMML suggests that modelling-language lifting must remain a predominantly manual process, at least in the near future.

Achieving spontaneous semantic interoperability: A longer-term ambition of our work is to support spontaneous semantic interoperability between software services on the emerging web

of services. We expect that our mappings can be used to map, not only modelling languages and models, but also software interfaces and data, such as web-service operations and their input and output data types, into a common ontology. From a model-driven architecture perspective, we expect these mappings to be derivable in part from the mappings of the software models used to generate the services. We expect to be able to use the mappings to identify and exploit semantic relations between running software systems and services.

6. CONCLUSION

We have presented a proposal for annotating XML-based modelling-language specifications semantically. Our proposal is inspired by and compliant with the SAWSDL recommendation [19]. It is based on and continues the UEMML initiative [1, 2] while opening it up towards complementary standards and towards a more distributed evolution of UEMML. Although the focus in this paper has been on modelling languages, our proposal is intended to support semantic annotation of models too. The paper has also extended the scope of the UEMML work from its focus on enterprise and IS languages so far to also consider languages for software modelling, although much work remains in this area.

The longer-term goal of our work is to make model-driven software systems – such as software services on the web – able to interoperate whenever needed and with minimal human intervention, even though they were not intentionally developed with this in mind, and even when they were developed on and for different platforms. We see the present proposal as a step towards a semantic software interoperability platform, which is able to re-engineer interoperability into model-driven software and services on demand and with minimal human intervention.

7. REFERENCES

- [1] Anaya, V., Berio, G., Harzallah, M., Heymans, P., Matulevičius, R., Opdahl, A.L., Panetto, H. and Verdecho, M.J. 2010. The Unified Enterprise Modelling Language – Overview and Further Work. *Computers in Industry* 61(2). Elsevier.
- [2] Berio, G., Opdahl, A., Anaya, V. and Dassisti, M. 2005. DEM1: UEMML 2.1. Interop-NoE Deliverable, www.interop-noe.org.
- [3] Fielding, R.T. 2000. Representational State Transfer (REST). Chapter 5 in *Architectural Styles and the Design of Network-based Software Architectures*, Ph.D. Dissertation, University of California, Irvine.
- [4] Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W. and Wimmer, M. 2006. Lifting metamodels to ontologies - a step to the semantic integration of modeling languages. In *Model Driven Engineering Languages and Systems*, LNCS 4199, pp. 528-542, Springer, Berlin.
- [5] Kelly, S., Lyytinen, K., and Rossi, M. 1996. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE Environment, Proceedings of CAiSE'96, 8th Int. Conference on Advanced Information Systems Engineering, LNCS 1080, pp. 1-21. Springer, Berlin.
- [6] Kifer, M. 2008. Rule Interchange Format: The Framework. In *Web Reasoning and Rule Systems*, LNCS 5341, pp. 1-11. Springer, Berlin.
- [7] OMG MDA. 2003. MDA Guide. Version 1.0.1, <http://www.omg.org/cgi-bin/doc?omg/03-06-01>, June.
- [8] OMG. 2008. Service oriented architecture Modeling Language (SoaML) Specification for the UML Profile and Metamodel for Services (UPMS). Revised Submission, OMG document: ad/2008-11-01; available at: <http://www.omg.org/docs/ad/08-11-01.pdf>.
- [9] OMG. 2007. OMG Model Driven Architecture. <http://www.omg.org/mda/>. Accessed 2010-02-05.
- [10] OMG. 2009a. SoaML Wiki. <http://www.omgwiki.org/SoaML/doku.php>
- [11] OMG. 2009b. UML 2.2 superstructure (PDF and XML versions). Object Management Group. www.omg.org.
- [12] Opdahl, A.L. 2010. Incorporating UML Class and Activity Constructs into UEMML. In *Advances in Conceptual Modeling – Applications and Challenges, Proc. ER 2010 Workshops*. Trujillo, J. and Dobbie G. (eds.), LNCS 6413, pp. 244-254. Springer, Berlin.
- [13] Opdahl, A.L. 2011. Anatomy of the Unified Enterprise Modelling Ontology. In *Proc. Int. Workshop on Enterprise Interoperability*. Stockholm/Sweden (March 2011).
- [14] Opdahl, A.L. and Henderson-Sellers, B. 2004. A Template for Defining Enterprise Modelling Constructs. *J. Database Management* 15(2).
- [15] Roser, S. and Bauer, B. 2008. Automatic Generation and Evolution of Model Transformations Using Ontology Engineering Space. *J. Data Semantics* 11:32-64.
- [16] W3C. 2004. OWL-S: Semantic Markup for Web Services. W3C Member Submission, 22. November 2004. <http://www.w3.org/Submission/OWL-S/>.
- [17] W3C. 2005a. Semantic Web Services Framework (SWSF) Overview W3C Member Submission 9 September 2005. <http://www.w3.org/Submission/SWSF/>.
- [18] W3C. 2005b. Web Service Semantics – WSDL-S. W3C Member Submission 7 November 2005. Version 1.0. <http://www.w3.org/Submission/WSDL-S/>.
- [19] W3C. 2007. Semantic Annotations for WSDL and XML Schema. W3C Recommendation 28 August 2007. <http://www.w3.org/TR/sawSDL/>.
- [20] W3C. 2009. Web Services Description Language (WSDL 2.0). <http://www.w3.org/TR/wsdl20/>.
- [21] WSMO. 2009. ESSI WSMO working group. Web Service Modeling Ontology (WSMO). <http://www.wsmo.org/>.