## Abstract

This paper, updated from “Requirements for Federated Modeling” (<http://www.omgwiki.org/architecture-ecosystem/doku.php?id=simf_requirements_and_use_cases> ) and is intended as a more detailed requirement set to gauge and validate various approaches to SIMF. The initial form of this paper offered possible representations for requirements. We are now focusing more on how to test if the requirement has been met, the “possible representations” are intended to be suggestive, not a final model. . Note that these requirements are not intended to be fully independent; requirements may overlap and be satisfied by the same or related features of SIMF.

This is a work in progress, please contribute!

Contents

[Abstract 1](#_Toc316891739)

[Issues with Federated Modeling 3](#_Toc316891740)

[Associations, Association Classes & Properties 3](#_Toc316891741)

[Binding 6](#_Toc316891742)

[Closed Vs. Open Descriptions 8](#_Toc316891743)

[Composite Concepts 11](#_Toc316891744)

[Composition 17](#_Toc316891745)

[Constraints 19](#_Toc316891746)

[Dominant Structure & Context 21](#_Toc316891747)

[Expressions 23](#_Toc316891748)

[Expressions about individuals, types and sets using the same predicate 25](#_Toc316891749)

[Federation Semantics 26](#_Toc316891750)

[Model Semantics 28](#_Toc316891751)

[Multiple Names for Anything 31](#_Toc316891752)

[Nested Models & Graphs 34](#_Toc316891753)

[Parameterization 37](#_Toc316891754)

[Perspective agnostic assertions 38](#_Toc316891755)

[Reification Choices 39](#_Toc316891756)

[Roles 42](#_Toc316891757)

[Scope and composition of concept definitions 44](#_Toc316891758)

[Subject of a Model 48](#_Toc316891759)

[Type/Instance 52](#_Toc316891760)

[Units & Quantities 56](#_Toc316891761)

[Views, Viewpoints & Notations 57](#_Toc316891762)

[Template 59](#_Toc316891763)

[System Stories 60](#_Toc316891764)

[The pump 60](#_Toc316891765)

[Satisfaction of requirements 63](#_Toc316891766)

## Issues with Federated Modeling

The following section enumerates some reasons models and modeling languages do not federate well today. Drawing on our own experience and observing issues with many models and meta models we have identified issues that seem to be barriers to federated modeling and suggested ways to resolve the issues. The requirements below would apply to a kernel set of concepts intended to be used to express and federate models and languages. We have also identified some key concepts that are required to facilitate these federation barriers – the combination provides a start to SIMF.

### Associations, Association Classes & Properties

Issue Scope: UML

Note: Needs review

#### Summary of issue

There is quite a bit of confusion about the relationship between “properties” and “associations”. In some languages these are the same thing, in others you can have one or the other, in still others they are different. In that the same concepts are frequently represented as either, a unifying model would solve a lot of federation problems and reduce complexity.

#### Example

1. UML uses an association to represent generalization, where as MOF uses a property. Prior to UML 2.4 these were just incompatible, in current UML they have been unified by making redundant elements.
2. Some connections between things seem to be “primitive” (like age) where as others are more complex connections (like a marriage). This is another reification distinction (see reification section). This confusion also caused the “end ownership” issues with UML-2, where the modeler has to make a commitment as to the end being “owned” by the association or the “classes”. If ownership is decoupled (see above), this issue goes away.

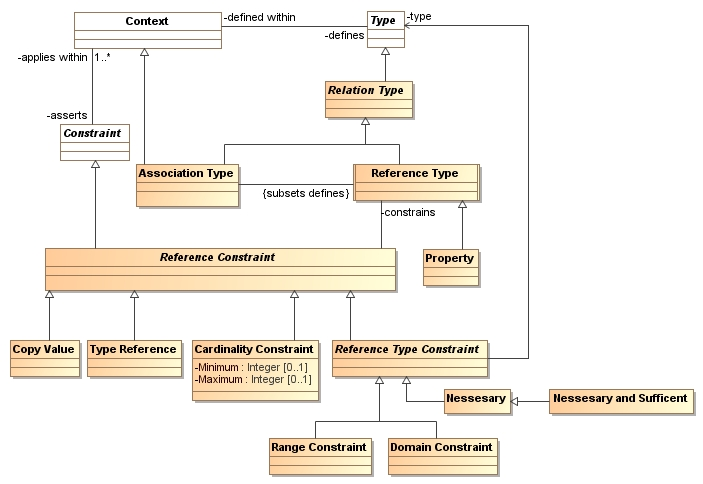
#### Other considerations

* Stand-alone properties (references that are not owned by the subject object) should be possible and not require extra baggage
* N-ary associations (properties with multiple ends) should be possible
* Associations that are a type (association classes) should be possible, allowing such associations to have properties

#### Requirement

Provide a general model that unifies properties and associations and also provides for variations in typed or untyped relations, cardinalities, etc. Provide for common constraints used in associations such as cardinality and “by value”.

#### Possible representation



The above model is believed to be general (but not complete) across UML, OWL, Programming Languages and DBMS. The meaning of the classes are as follows:

* **Relation type** – a common abstract superclass over association type and reference type. Intended to capture anything that “connects” instances of other types.
* **Association type** – a composite of multiple references which is a context defining a set of related reference types and reference constraints. While most of these will be binary, N-ary associations are also possible. Associations that contain binary references will infer inverse constraints between those references. Note that since we are not assuming instances can only have one type, an association may also be a class – making association classes possible.
* **Reference Type** – a type of reference from instances of one type to another. This can also be specialized for other connections, such as functions, properties, association ends, behavior –follows- behavior, etc.
* **Property** – a reference type representing a simple traversal from one type to another – this includes “association ends”.
* **Reference constraint** – a constraint on a reference type owned in a context. Note the context that assets the constraint does not have to be the same as the context that defines the reference, allowing an “open world” of such constraints.
* **Cardinality constraint** – a constraint on the number of the particular kind of reference from a given domain instance.
* **Reference Type Constraint** – a constraint on the domain or range of a reference. Where multiple reference type constraints exist this represents a choice of types, not that the reference must be all such types (not that this is not the same as OWL semantics). The purpose of this is to allow more flexible reuse of references without introducing arbitrary supertypes. Where the referenced instances must be of the referenced types “necessary” should be specified.
* **Necessary** – the existence of the reference requires a type of the referenced element matches “type” (OWL Semantics). Without necessary the type constraint only requires that one of the types be satisfied (type constrains specify a choice of types).
* **Necessary and sufficient** – the existence of the reference implies the type of the referenced element includes “type” – type will be added to the type of the referenced instance.
* **Range constraint** – a constraint on a possible range (source) of the reference. If many such constraints exist a choice of any of the types is allowed {note this differs from OWL so as to make the references more reusable}
* **Domain constraint** – a constraint on a possible range(target) of the reference. If many such constraints exist a choice of any of the types is allowed {note this differs from OWL so as to make the references more reusable}.
* **Implies type** – a mixin that asserts that the reference will assert the domain or range (this represents RDF/OWL semantics).
* **Copy value** –mixin to specify that referenced instance is copied when used by this reference (e.g. “by value”).
* **Type reference** – a mixin to assert that the reference references the type description, not instances of that type. This allows situations like “U.S. Congress (An instance) passes laws (a type)”.
* Others… Our expectation is that there can be any number of constraints, some specialized and some general.

Note: This is similar to the draft IMM model which contains much more detail.

Note also that any reference (like any model element) can have any number of names (see naming) and be used in any number of contexts by any number of types.

#### Example

See also:

* IMM for examples (model is slightly different but structure is sufficiently similar).
* Example in “Composition”

[TODO] Examples in SIMF model .

### Binding

Issue Scope: Common across most languages

Note: Needs review

#### Summary of issue

Most modeling languages have at least one, if not several, ways to bind values to references, roles, variables, arguments in expressions or associations.

#### Pseudo code Examples

* + A = b+1 {A is bound to b+1 – an assignment}
  + F(a) {argument of F is bound to A}
  + Pump S3556 is bound to Role P101 in system C. (See “pump” user story, below)

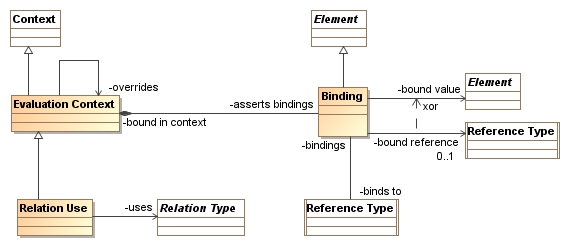
#### Other considerations

* We also require a way to bind the USE of a relation.
* Some bindings are to sets or types, others to individuals

#### Requirement

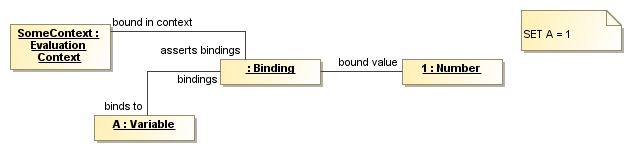
Provide for a common representation of binding that can be used across multiple languages and in different contexts within the same language.

#### Possible representation



* **Evaluation Context** – any context which can bind values – this can be a static, dynamic or temporal context. Some context can override the bindings of other context using “overrides”.
* **Binding** – the binding of a value to a reference (which can be a variable, argument, part, etc). The reference set is “binds to”. A direct reference to a value is provided by “bound value” where as a bound reference” will traverse the reference to get the value (traversing references can also call operations). The binding is valid within “bound in context”.
* **Relation Use** – The use of one relation within another. This can call an operation, use a parameterized type, use a composition, etc. based on specializations of the relation.

#### Example



In this example the variable A is set to the value 1 in the context “SomeContext” .

See expressions for additional examples.

{More examples needed}

### Closed Vs. Open Descriptions

Issue Scope: Many languages including most OMG languages. Most ontological languages are already open world which provides for open descriptions.

#### Summary of issue

The term “open description” as used here differentiates information systems that allow arbitrary information to be added from those that can only work with a fixed structure. The term “open world” has been popularized by the “semantic web” with the slogan: “anyone can say anything about anything, anywhere”. Open Descriptions are a special case of open world but are less constrained than the formal logic interpretation of open world which disallows defaults.

Most “traditional” information systems work on closed Descriptions: SQL, OO Languages, Etc. Ontologies and some applications of XML allow for a fully open definition, however there are degrees to openness – for example OO languages do, at least, allow for subclasses to be defined by anyone but they do not allow for subclasses to be added.

In a closed description world the “statements” that can be made about something are pre-defined and typically used to structure and optimized the application. This was necessary in the early days of computing due to limited resources, and is still required in some applications.

In an open description a resource could exist in some other repository, say one representing “France”. In an open description someone else, in another resource, could say something about this France just by referencing its identity (the identity used in the other repository). For example, this could be used to add population data to “France” where it was missing in the other repository. It could also be used to add new attributes to a class of countries. This kind of open work is essential for federation.

What scares many people about this idea is trust. What if “they” say something wrong! Just because anyone can say anything doesn’t mean that anyone else has to pay any attention or believe it. Trust and selection are still required. This is essentially the “web” model of information – there are a lot of webpages, but you don’t believe all of them (I hope!).

It is a fundamental tenant of our approach that model information is structured with an open description assumption and that there is a model for trust, selection and provenance.

#### Example

Class “Country” and a set of countries with country codes could be owned by the United Nations and published as data on their web site. The CIA world book could add information about these countries in another repository but references the same country identifiers as are used by the U.N.

#### Other considerations

* In many formal logics “open world” has a stricter interpretation – “In [formal logic](http://en.wikipedia.org/wiki/Formal_logic), the **open world assumption** is the assumption that the [truth-value](http://en.wikipedia.org/wiki/Truth-value) of a [statement](http://en.wikipedia.org/wiki/Statement_(logic)) is independent of whether or not it is *known* by any single observer or agent to be true.” {Wikepedia}. This strong constraint eliminates the possibility of defaults and conflict. In information modeling defaults and conflict are required so our interpretation is less strict. It should be up to the client what level of strictness and consistency is to be imposed on a set of models. The stricter interpretation is an extension to our base description. See also: <http://en.wikipedia.org/wiki/Open_world_assumption>
* Many applications, including most programming languages require a closed description (except for the programs data). It is necessary to be able to “close the description” by considering only a fixed set of models for a particular application. It should also be possible to specify certain sets as “complete” within a context such that they will not and cannot be extended, this helps in closing the world in a particular context.
* Open description systems require more sophisticated model validation in that new model elements may have an impact on other models that were on their own consistent.
* The “black diamonds” common in UML and MOF models, when interpreted to mean structural nesting of data, are antithetical to an open description in that they fix a pre-determined structure.
* An open description with the ability to augment external models requires some kind of general identity system for model elements. The URI is the assumed identity scheme: every model element will have a URI.

#### Requirement

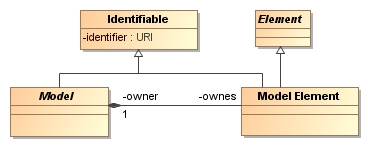
The representation of model information should assume an open description and only   
“close the description” dynamically when required by the consumer of the information. The assumption that the description of model elements is “owned” by a single model element is to be removed.

Model elements are “data” that must “be” someplace. Each such a model element is thus part of a model resource – a model resource is “data” such in a repository or file. This ownership of model information – its life-cycle and physical location and source should not be co-mingled with the semantics of composite concepts such as classes or data structures unless such information is, by logical necessity, atomic. Another way of saying this is: An individual assertion cannot be spread among separate data resources but facts about the same thing should have no requirement for being in the same resource because doing so prevents others from augmenting information out of their control.

Associations other than ownership should be used to specify the context, subject or domain of a model element – which in SIMF are independent of data ownership.

Application of the open world will eliminate some of the current UML restrictions including the tight binding and ownership of: Properties, generalization, cardinalities, etc. Associations will be added where required to substitute for the tight binding of ownership and context. Note that this eliminates the distinction of “end ownership” currently plaguing UML-2 as the ends are now owned by a model like everything else.

#### Possible representation



The above model shows that model elements are owned only by models. “Element” allows for something that is an essential and indivisible part of another model element and owned by it – but this is the exception. Each model and model element has a unique URI. Owns, in this context, is physical containment and implies delete propagation. However, note that this is one of the few places such a strong tie is made so that organization of the information ownership is orthogonal to the information’s semantics.

The open description assumption is pervasive throughout these requirements and assumed in all other sections.

##### Definition of the above model elements

* **Identifiable –** anything that can be individually identified or named.
* **Element** – any piece of information that may be found in a model.
* **Model Element** – an element that is owned directly by a model, has a unique identity (element identifier) and may be named. Note that elements that are not model elements must be nested inside of some other model element.
* **Model** – a set of model elements as asserted by some individual or authority – most often a “file”, DBMS or other computational resource or a part of such a resource.
* **Owner/owns** – the relation between a model and the set if model elements it asserts.

[TODO] Closing the world, Trust.

#### Example

Due to the pervasive nature a single example is not shown. See below.

### Composite Concepts

Issue Scope: All major modeling languages

#### Summary of issue

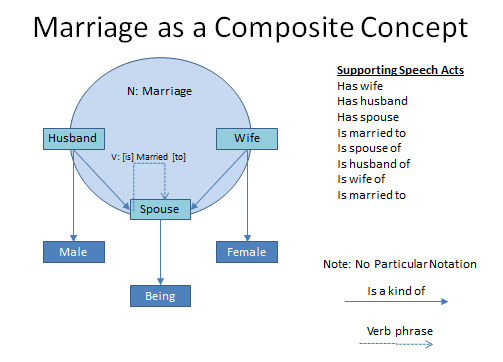
The representation of concepts (meant here in the broad sense of covering (narrow) concepts (definitions), predicates and integrity rules) should be sufficiently broad to encompass different viewpoints, representations and use cases for the same concept or set of concepts.

Concepts range from the atomic to the very broad and the method to define composite concepts from simpler ones and, likewise, to define simple concepts in the context of broader ones is essential to a scalable design.

#### Example

For example the concepts of “marriage” and “spouse” are tightly tied; in a certain context one could say that every marriage has exactly two spouses. Other interpretations of marriage allow for multiple spouses. Regardless, the concept of “spouse” is intertwined with the concept of marriage. In the EU Statistical Office (Eurostat) Marriage Use Case the “exactly two” constraint is selected as this is the case for all associated countries. Hence the concepts of marriage and spouse are clearly related. When the relationships between these composite concepts are lost we have trouble understanding what they mean.

This example is further defined on the OMG wiki, here: <http://www.omgwiki.org/architecture-ecosystem/doku.php?id=composite_concepts>

[](http://www.omgwiki.org/architecture-ecosystem/lib/exe/detail.php?id=composite_concepts&media=marriage.png)

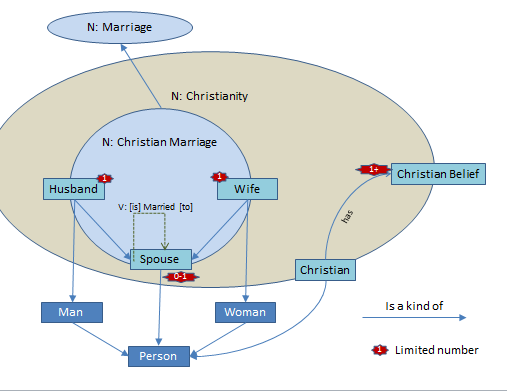
This diagram (in no particular notation) suggests that the noun concept “marriage” provides a context for a set of associated role concepts such “spouse”, “husband” and “wife”. It also provides context for verb concepts such as “has spouse” or “is married to”. That, in fact, each of these verb concepts comes from a particular perspective on marriage. Once you select a certain context or view of marriage you can describe how these various roles and verb phrases are related. Once certain of these parts of the composite concept are known, it is possible to infer the others.

The above example is intended to show how composite concepts relate to both natural language statements and information representations that are about the same underlying concept.

We can also add some more constraints, also called integrity rules in SIMF, such that a “Husband” is a “Male being” and a wife a female in the context of a Traditional European Marriage, etc.

##### Traditional Marriage

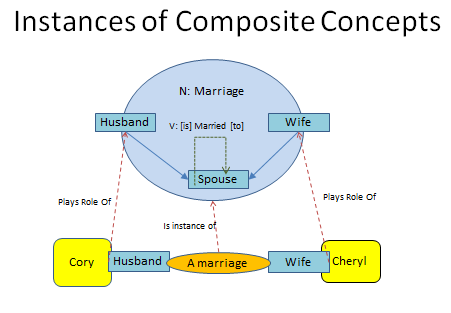
As is the subject of much political debate, some would like to “lock down” the concept of marriage. Lets call this “Traditional Marriage”. This becomes a specialized concept within some context such as Europe.

[](http://www.omgwiki.org/architecture-ecosystem/lib/exe/fetch.php?cache=&media=christian_marriage.png)

Traditional Marriage (or 'Christian Marriage') is defined within some political domains and imposes certain constraints (integrity rules) - that a spouse must be a person, and more precise that the husband must be a male and the wife a female. That a person may be a spouse zero or one time (at any moment in time) and that within a Traditional Marriage there is exactly one husband (a man) and one wife (a woman). These constrains may, of course, be represented by textual expression, diagrams, tables or other forms of formal expression. What is consistent is that they are constraints placed on (the communication about a Traditional Marriage.

**Example instance of a composite concept**

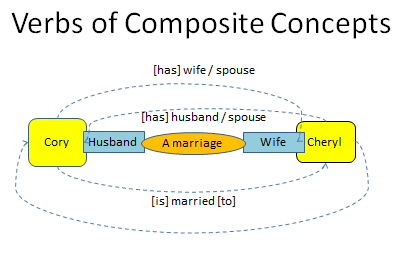
A composite concept represents a pattern, or type, that can describe instances. Hence a composite concept is a populatable construct by binding the roles to individuals. Each such instance may be considered an “assertion” or a “fact”.

[](http://www.omgwiki.org/architecture-ecosystem/lib/exe/detail.php?id=composite_concepts&media=instanceofmarriage.png)

This diagram (in no particular notation) shows how an “instance” of marriage, a particular marriage, involves individuals that “play the roles” in that composite concept. A particular Logical Information Model may capture this information as a property or a reified class. In logic the instance of a composite concept is known as a “tuple”, a limited version of a tuple is the “triple” used in RDF.

**Example of verbs associated with a composite concept**

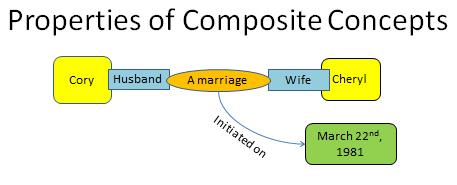
The nouns (The marriage of Ronald inaugurated on 1940-01-26) and verbs (There exists a marriage of Ronald inaugurated on 1940-01-26) are “two sides of the same coin”, just different ways to say the same thing. The verb phrases that describe the same concepts are represented by the following diagram.

[](http://www.omgwiki.org/architecture-ecosystem/lib/exe/detail.php?id=composite_concepts&media=marriageverbs.png)

Here we can see that the same individuals (Cory and Cheryl) are the subjects and objects of verb phrases about the *same marriage*. Once any combination of the above is known the rest is known.

**Example of a property of a composite concept**

It is important to capture “a marriage” as a single identifiable concept so that we can manage it, understand who asserted it and ascribe properties to it. For example, a marriage may have a property that describes when it happened. Information models that simply capture spouse properties of individuals loose this ability.

[](http://www.omgwiki.org/architecture-ecosystem/lib/exe/detail.php?id=composite_concepts&media=propertyofamarriage.png)

This diagram shows that the composite marriage concept can have properties, such as the date-time when it was initiated (when the couple was married). Such a concept could have any number of properties or associations with other concepts.

What frequently happens in information models is that these domain concepts get “flattened” into properties of a single class or type, such as a person. The model bridging relations are required to connect between the conceptual representations and the information models. The SIMF tooling should be able to help users make these distinctions while connecting the various representations.

Objectified (nominalized) constructs and variable based constructs have both their advantages and disadvantages. Hence there are user communities that prefer the one and other user communities prefer the other. Hence SIMF will provide for federation of both representations but unification of the concepts being represented.

#### Other considerations

* Composite concepts have a relationship with composition and roles
* There is a level of concept that is atomic – that is that no part of it could be known without knowing any other part. Other composite concepts can be constructed from these more atomic ones. Therefor there is some relationship between composite concepts and concept management and reference.

In many languages these noun, verb and role concepts are defined independently, indeed they are just “string names” on the ends of relations or properties. What this fails to do is provide for the connecting semantics of these different views of the *same* fact or to provide a single identity for that fact. What can we do with a SIMF understanding of composite concepts?

* We can make statements about the composite concept, such as the timeframe over which it was valid
* We can infer “inverse” and other relations between the properties in case of binaries
* We can manage and version it as one concept
* We can understand how “reified” (nominalized, objecfified) and non-reified representations of the same thing are connected.
* We can connect information model elements to the composite concept or any of the roles or verb phrases.
* We can have properties of composite concepts
* We can classify the composite concept, roles and verb phrases to better understand the semantics of each and of how each concept supports the semantic definition of the others. For example, we can classify marriage as a social arrangement among a couple of individuals that has a defined beginning and (but not always known)end.

#### Requirement

* Provide for concepts to be defined in terms of other, simpler concepts
* Provide for concepts to be defined within the context of broader composite concepts
* Provide for specialization of concepts
* Provide for a relationship between composite concepts and various sentence or representational forms
* Provide for a relationship between verb for and noun forms of expression of the same concept
* Be able to make relations to and from composite concepts

#### Tests

The marriage example is our first test: <http://www.omgwiki.org/architecture-ecosystem/doku.php?id=composite_concepts>

#### Possible representation

### Composition

Issue Scope: All major modeling languages

Note: Needs review

#### Summary of issue

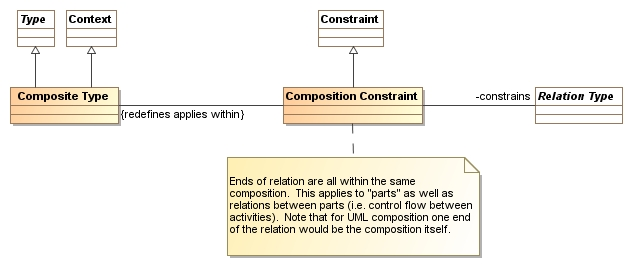
Representation of a composite along with its parts and the relationships between them is fundamental to architecture at all levels and domains, yet this basic pattern does not have a common representation. By defining a common set of composition concepts with well-defined semantics we can use it is a foundation for the many representations of composition in multiple languages and used in almost every model.

#### Reference

For this topic, please see the excellent article by Conrad Bock:

<http://www.jot.fm/issues/issue_2004_11/column5/column5.pdf>

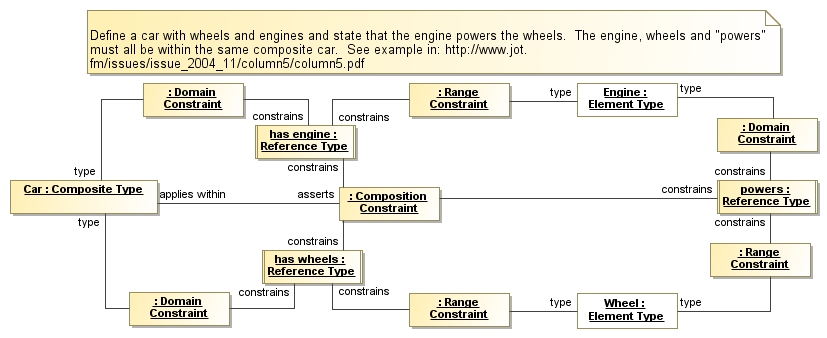
#### Possible representation



* **Composite type** –The composite type is the type of a “whole” that composites are part of based on a set of composition constraints. Note that the composites may directly or indirectly related to the whole.
* **Composition constraint** – a constraint on a relation type that all ends of the referenced relation must be within the same composition – that is they are all parts of the same thing. Note that the composition is frequently one end of these relations, corresponding to the domain semantic interpretation of the black diamond (that the thing in the world has parts).

This model of composition does not type the “parts”, it makes a constraint about the relations to the parts. So a “wheel” is an independent thing – it is its relation to a car that makes it (potentially) part of a car.

#### Example



Note: Unidirectional relations are used for simplicity in the example but the same approach works with binary associations. Note also that any part could also be a composite type, providing for recursive definition.

There are multiple concepts of composition. Specializations of the composition constraint could assert more specific forms, such as something that exists exclusively within its composite (a hole can’t exist outside of the thing it is a hole in). Mereology {http://en.wikipedia.org/wiki/Mereology } defines a full theory of composition that we could draw on as required. For example, the IDEAS framework draws on the concept of “overlap”, in that the space/time of multiple objects overlaps(have parts in common) and thus forms a composite.

In UML the “black diamond” has sometimes been interpreted to mean that the information descriptions of one thing are part of another and therefore have lifecycle dependencies. A model should be clear about its intent for composition as describing the model element dependencies or the modeled reality.

### Constraints

Issue Scope: UML

Note: Needs review

#### Summary of issue

What we can express with purpose-specific “model elements” is limited to what language designers intended. A general purpose constraint mechanism allows specific statements as to what is permitted and required. Such a general purpose constraint mechanism can then be used to “ground” the semantics of higher level concepts in a formal language.

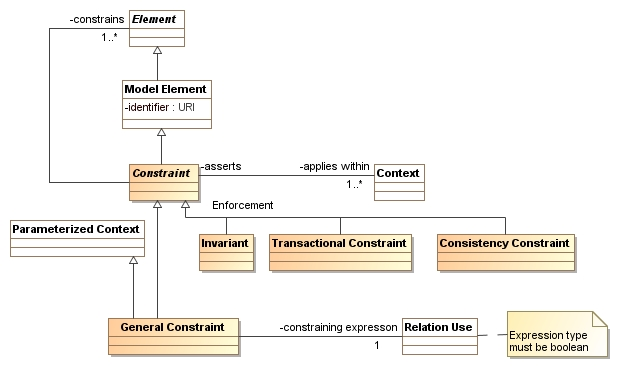
Constraint semantics that can be interpreted across a variety of models and languages provides for greater information sharing and consistency. Constraints are a required part of SIMF.

#### Other considerations

#### Requirement

Provide a general constraint model that can be extended to represent more purpose-specific constraints. Allow for different levels of enforcement of constraints.

#### Possible representation



* **Constraint** – constraint is the base class for all constraints in SIMF models. Any constraint must constrain one or more elements and is applicable within one or more context.
* **General Constraint** – a general constraint has a constraining expression that specifies the truth value of the constraint – the constraining expression must be TRUE. The type of the constraining expression must be Boolean.
* **Enforcement** – enforcement subtypes specify when the constraint will be enforced
  + **Invariant** – the constraint must hold at all times
  + **transactional constraint** – the constraint must hold after any set of model modifications
  + **Consistency Constraint** - The constraint must hold for the model to be consistent, but it may be legal to have models that exist in an inconsistent state. The expectation would be that inconsistencies would be a warming and may prevent certain actions (such as execution)

#### Example

See associations and compositions for examples.

### Dominant Structure & Context

Issue Scope: Many languages including most OMG languages (with the exception of SBVR)

#### Summary of issue

It is common to structure models into hierarchical packages that reflect the dominant decomposition of the system being modeled. The problem is that this dominant decomposition may be right for that one perspective but is not right for other uses, other views or other context that use the same information.

Even within one model it is often a poor choice to pick one decomposition – is it by the kind of element? By a functional decomposition or along project boundaries? Having to pick one organizational structure for model information is not acceptable; information should be able to be organized into multiple hierarchies or contextual dimensions as it is being defined and later when it is being used.

A more general concept of context is required, where something can be contextualized in many different ways. No one decomposition should be “primary” as primary is context dependent.

#### Example

A project I worked on recently had a package structure with elements like “phase 1” and “phase 2” – these made perfect sense as a way to manage the project and deliverables. But, to someone reading or using these models these are arbitrary divisions that just cause confusion. If I were to use one element out of this model, say a “Person” class, I couldn’t insert that person class into my model in a place it made sense – I would have to get the entire “other” model and search for things in it.

#### Other considerations

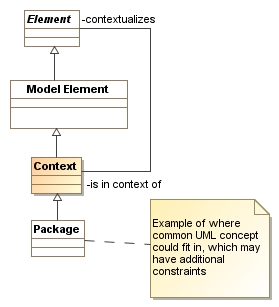
* Supporting other decompositions for information is an essential element of supporting different viewpoints. Different viewpoints will frequently require another organization of data. Sometimes this is called a “pivot”.

#### Requirement

Hierarchies are a valuable tool for organizing information but the hierarchy should not “own” the information but be a view of a selected set of elements. The hierarchy defines a context that contain names and references to model elements. A model element should be able to be “in” any number of hierarchies, packages and/or contexts.

Define a general concept of context and contextualization that can be used to organize information as well as define where certain concepts or constraints are valid.

#### Possible representation



The diagram shows that any element can be placed in any context (of course constraints can be added that limit what certain context reference). Context is then used as a packaging structure by recursively following the “contextualizes” association. The contextualizes association can be specialized to represent various contextual dimensions.

See also: Naming & Constraints

#### Example

See example under naming.

### Expressions

Issue Scope: UML & OWL

Note: Needs review

#### Summary of issue

Most forms of modeling require expressions and the structure and semantics of expressions is quite consistent. Yet, we have no fundamental expression model. The OCL model is tied to UML, yet not integrated with UML. For OWL, general purpose expressions are also missing, yet are the key component of first order logic.

#### Example

As expressions are well known, further examples may not be required.

#### Other considerations

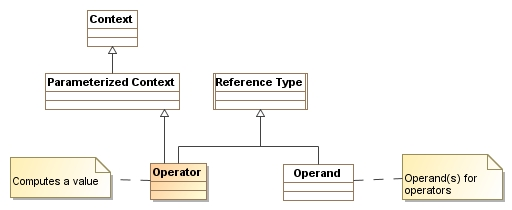
* While the general form of expressions are well understood, the operators available in various languages vary quite a bit (however there are large islands of consistency).
* Precedence impacts how a textual expression is parsed but should have no impact of an expression model.
* Expressions have two sides – the definition of expressions, including arguments, and the binding to these expressions where they are used.
* A common semantic of expressions is that they return something based on a computation involving their arguments and, sometimes, the context of the expression. Other forms of expressions state an invariant (the expression is equal to true).
* Composition semantics is also required for instances where there may be no explicit composite type.

#### Requirement

Provide a general expression model with a library of operators appropriate to different languages and types. These expressions should allow for arguments and binding to these arguments as parameters. Move as much as possible out of the “meta” layer and into a library of operators. Expressions should be able to express general constraints and invariants.

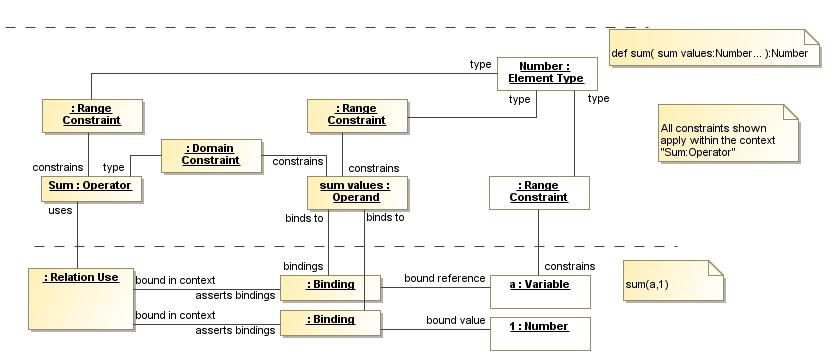
Note: We are using the term “operand” in a general sense as a synonym for argument, parameter, etc. We are using the term “operator” in the general sense to include anything that can perform expression evaluations.

#### Possible representation



We build on the relations module to add the concept of an operator. Since we already have operands (from parameterization) there is little new needed. Note that all functions are instances of this meta model so the set of possible “built in” functions is unlimited. The Model Semantics module defines a set of primitive operators based on first order logic.

#### Example



The upper part shows the definition of an operator “sum” which returns a number (based on its range) and has a set of “sum values” as operands, which must also be numbers.

The lower part shows calling the Sum function with 2 arguments: The variable A and the number 1.

### Expressions about individuals, types and sets using the same predicate

#### Summary of issue

The same predicates can be used in facts about individuals and sets( or types) of individuals. In some languages (UML, OWL) these force the creation of completely different predicates (one for the individual and one for the type), in others (SBVR) the ideas seem conflated and there is an attempt to use exactly the same predicate and pattern to talk about individuals or types (which can become semantically confused). Our requirement is that the same concepts can be used for sets and individuals but that it is clear which is being used in any particular fact or speech act.

#### Example

* “Like” can be used with respect to an individual or set in either role
  + John likes supermodels
  + Teenage boys I know like supermodels
  + Teenage boys I know like Marisa Miller
  + John likes Marisa Miller
* Foobar wasCreatedIn 2003
  + Does this refer to a particular concept of FooBar or to all FooBar
  + Forall FooBar F, F wasCreatedIn 2003, or
  + (class FooBar) wasCreatedIn 2003
  + Consider:
  + Forall people P, P wasCreatedIn 2003, or
  + (class people) wasCreatedIn 2003

#### Other considerations

* The sets defined may be static (e.g. people) or dynamic: people in my room now.

#### Requirement

* Be able to use the same basic predicate concept to refer individuals or sets in either the subject or object role
* Be able to distinguish between statements about an individual or set/type
* Be able to have static or dynamic sets fill a role in a predicate

#### Tests

See example, above.

### Federation Semantics

Issue Scope: Most major modeling languages

Note: Needs review

#### Summary of issue

Models are created by different people (or groups) at different times for different purposes. Often these different models model the same thing, related things or things that can be reused. Creating federated models and analyzing partially federated models provides value for communities, supply chain automation, systems of systems, information integration, SOA, cloud computing and other activities involving a distributed federation. Modeling should have built-in capabilities for dealing with federated models, relating those models to each other and to reference models such as SIMF.

#### Examples

See architecture ecosystem wiki here: <http://www.omgwiki.org/architecture-ecosystem/doku.php?id=use_cases>

#### Requirement

Provide relations to federate similar, equivalent and related elements across indecently conceived models that may use different structures, terminology and have different opinions and theories about the systems being modeled.

Federation semantics would include those relations defined under “model semantics” as well as:

* **Equivalence** – where the same concept is represented by different model elements. Equivalence may apply to types, relations, behaviors or instances.
  + Note OWL provides for “equivalent class”, “equivalent property” and “same as” to represent the same thing represented in different ways. However, the semantics of owl:sameAs are stronger than we would assert in that we do not want to assert that equivalent representations must be consistent (this constraint would be added in an appropriate module).
* **Aspects** and other elements found in SMOF to specify what may be and must be multiply classified.
* **Façade** – a way to specify the conditions where instances of one type may be projected as instances of another type and the relations that implement the projection. This would, for example, allow a BPMN process to “look like” a UML activity – to the extent that such a projection is possible. Projections may have information loss.
* **Inclusion & Exclusion** – a way to specify parts of models that are and are NOT to be used in another model.

Federation also assumes that there are some set of concepts that are shared between models and languages to be federated. The concepts defined in this paper are examples of such common concepts. However, common concepts must be loosely coupled and not require “buy in” to large or restrictive theories. Concepts designed for reuse should commit to as little as possible and be organized into small modules that can be used as appropriate. Such modules are expected to use and specialize each other, forming a lattice.

#### Possible representation

#### 

* **Equivalence** – specifies that all constrained elements represent the same thing or set of things.
* **Exclusion** – eliminates the constrained elements from the model . Exclusion of a context eliminates all elements contextualized by that context.
* **Façade** – A declaration that represents the source type (or a subtype of it) as the façade type using a set of bindings. Note that bindings can expand to an arbitrarily complex specification. It is expected that many facades will have information loss but that the subset is consistent across the different representations.

See also: SMOF { OMG Document Number: ad/2010-08-06} meta model extensions relating to type compatibility, a fragment of which is below.



#### Example

[TODO]

### Model Semantics

Issue Scope: Most non logical languages, including UML.

Note: Needs review

#### Summary of issue

A model is intended to mean something, to capture and communicate some facts or assertions about some subject of study. If this meaning is less clear, the model is less useful. Dictionaries attempt to address this problem for natural language, logic for ontologies. Models need a way to address the meaning of models, the model semantics.

What semantics means for a model is how the data and symbols in the model relate to the subject of the model. Informal models depend on intuition and external explanations to make this connection. Formal models use logic to help define the semantics. This does not lessen the need for clear natural language Descriptions, it provides a way to test and validate that models are correct and, in some cases, test that systems described by those models are operating correctly. Most importantly it means that the same model will mean the same thing to different people or machines interpreting that model.

Informal models still have a place, as a way to augment human to human communication. Such informal models may still use formally defined concepts – just like natural speech uses terms that are well defined in dictionaries. Informal models just “say less” about the domain, saying less is not an excuse for being unclear about what it means.

Most modeling languages, with the exception of formal logics, do not have well defined semantics. How they relate to the systems they model and even their own diagrams is often unclear. As models are used by larger teams and depended on to support automation such inconsistences are unacceptable and make the models less useful (or even dangerous). Trying to execute or federate models with unclear semantics is a recipe for error, wasted time and inconsistency. The goal of federation and the goal of well defined semantics are linked. Some models may also be directly executable in an information system – these models clearly need precise semantics.

While it is not our expectation that all modeling languages can have a strong semantic grounding, we can start along that path by providing mechanisms for grounding and making sure that SIMF concepts are formally defined.

#### Other considerations

* Systems may make additional “commitments” (assumptions and constraints) about models in a particular context, such as an execution context. These commitments may add to but should not conflict with the semantics of models they are built on. For this reason it is important that SIMF concepts make as few commitments as possible so that more specialized context can add the commitments they require. This layering of semantics, of commitments, is common to any well designed architecture.
* We don’t have to and should not define new theories for semantics; this has been done over the last few thousand years. SIMF should be grounded in at least one existing formal language and should provide the capability to extend semantic grounding to models and modeling languages. One such formal logic intended to cover a broad range of languages is “common logic[[1]](#endnote-1)”, which is a strong candidate for use as our formal grounding language.

#### Requirement

SIMF must be grounded in a proven and tooled formal logic such that models can, at minimum, be validated with that logic. In addition, SIMF will include concepts required for the semantic grounding of modeling languages defined in SIMF. There will be a precise mapping of these SIMF concepts to at least one formal logic.

The logical primitives are small in number, to quote John Sowa “all you need in the core are three primitives: 'and', 'not', and 'some'. And you also need a notation for relations, such as R(x,y,z).”. There are probably a few more like types and numbers – but the list is limited.

It must be possible to define a simplified model element that logically “expands” to its formal definition without requiring that the full formal definition be rendered in every model.

#### Possible representation

Semantic representation will be part of the fully worked out model, this is only a starting point.

What we have in mind is making sure we have an expression form that can cover CL and OCL, which may not be as bad as it sounds because I’m going to put all the functions in a library, not in meta classes.

Note, see: <http://www.jfsowa.com/talks/clintro.pdf> from John Sowa for one approach to base semantics.

The expression module is the basis for logical operators. Operators expected to be part of semantic representation modules include:

* Logical operators: and, or, xor, not, implication, true, false
* Universal and existential quantifiers
* Sets, subsets and types
* Set and type union and intersection

A meta operator (i.e. “that”) represented as statements about models (see

* Nested Models & Graphs)
* Numbers and numeric operators (+, -, \*, /, > <)

Note that base semantic operators (such as “and”) do not have a special place in the meta model. All operators are at the same “level” and may be included in any language or model that recognizes them.

[TODO] Our expectation is that SIMF model and these operators would be grounded in common logic <http://en.wikipedia.org/wiki/Common_logic>), “IKL” extension (<http://nrrc.mitre.org/NRRC/Docs_Data/ikris/IKRIS_Evaluation_Report_31Dec06.doc>) and common logic ontologies.



The above fragment from common logic represents foundational operators. “Sentence” in this model would be “Operator” in SIMF model.

#### Example

[TODO] Example from first order logic to be rendered as model instances

### Multiple Names for Anything

Issue Scope: Many languages including most OMG languages (with the exception of SBVR)

#### Summary of issue

Many languages demand a single name for an element – in UML this is known as a “Named Element”. Such an element has a single text name and must be in a namespace which defined the context for that name and element. The name, concept and context are tightly coupled.

In reality things and concepts have multiple names – names differ according to context, user community and natural language. Binding a single name with a single model element prevents a proper representation of the same thing used in different context and different communities. As a result redundant models are frequently created or “modeling hacks” to simulate multiple names through dependencies. In UML this is done using “element import” for some named elements (those in a package), in addition the UML “named element” is often not named at all! So the model is inconsistent – what it is really saying is that it is an element that may have a name – but most many named elements have no name at all.

#### Example

In a “person” model I could have called a class “Person” or “Individual” or “Human Being”. If I were to use this model of a person with another that made some other choice the model would be inconsistent. I have no way to properly notate the “synonym set[[2]](#footnote-1)” that would be appropriate to name this concept.

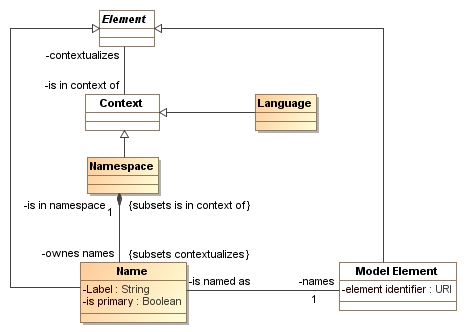
#### Other considerations

* Most formal languages require unique names where as natural languages do not, both conditions should be supported. Some OO languages allow for multiple names within a context based on signatures.
* Multiple natural language terms may be assigned to the same concept in the same context – for example terms in English, French and Japanese. Terms should be identifiable within their natural language context as well as their topical context.
* Identifiers may not be limited to textual – icons and other symbols could mark a concept.
* It is useful for something to have a preferred name within a context so that such a name can be displayed. This is also required for compatibility with single name languages.
* Some languages have user hostile restrictions on names, such as not allowing spaces.

#### Requirement

Provide for things and concepts to have multiple names in models. These names should be able to be assigned in different context and use different natural languages. SIMF should place few restrictions on names – not requiring them to be unique or textual. Additional constraints should be available to restrict names within certain languages as appropriate. There should be a way to return the preferred name within a context or set of context.

#### Possible representation



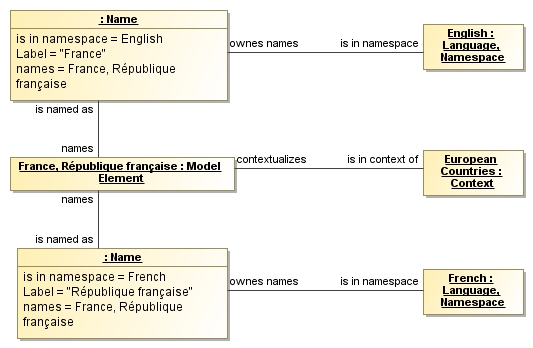
The above model fragment shows that any model element can be identified by any number of symbols contextualized by any number of context. One such context is the natural language (i.e. English, German).

Note that as discussed in “open world” naming and ownership of the identified model element are completely separate – a name could be added for anything, anywhere.

##### Definition of the above model elements:

* **Name** – a context specific name for any model element. The name “names” the element and is in a single name space. Another namespace may have the same or different names for the same thing. One name for an element may be marked as primary within a namespace to serve as the name returned for that element.
* **Namespace** – a context that owns a set of names.
* **Language** – A context representing a human or computer language or dialect thereof {more detail will be required here}.

#### Example



The above instance example shows a model element with two names: “France” and “*République française*” each in the context of their respective human languages and the concept “France” is in a context of “countries”.

Note: For easy representation names will be shown in their normal UML syntax within this document. Should UML support multiple names they could be simply separated by a comma as are other lists in UML model elements.

### Nested Models & Graphs

Issue Scope: UML, RDF & OWL

Note: Needs review

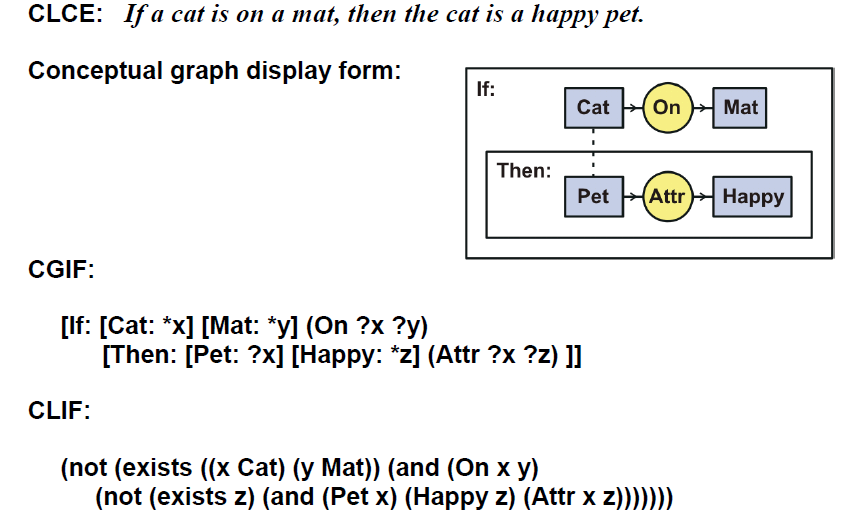
#### Summary of issue

Saying very simple things requires that we say things about other information, like if it is true or who authored it. So it is necessary to be able to reference one model or model part from another.

#### Example

The following example is from: <http://www.jfsowa.com/talks/clintro.pdf>, By John Sowa.

The example is how to say a very simple thing in various ways:



Note that what is required is to “enclose” a set of statements and talk about those statements as an argument to other statements. This ends up as either references to external model segments or nested model segments *that are arguments to other statements*. Arguments are the parameters (roles) to any parameterized structure (see parameterization).

Here is the same example in SQL:

DEFINE VIEW Happycat AS

SELECT Supportee

FROM Supports

WHERE Supportee IS IN

              (SELECT Object

               FROM ObjectType

               WHERE Type = "Cat")

AND

             Supporter IS IN

              (SELECT Object

               FROM ObjectType

               WHERE Type = "Mat")

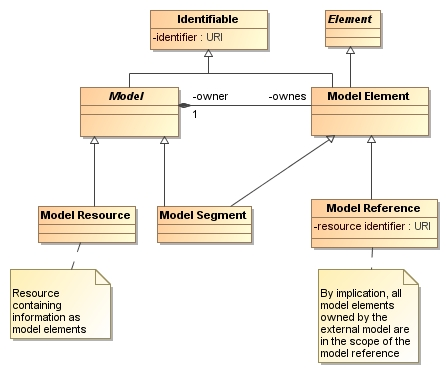
#### Other considerations

* Such nesting goes beyond typical “first order logic” (FOL) but is common in modeling and natural language.
* Nested elements are also known as a “block” in many programming languages. However this is not the same as the concept of a block in SysML which is a logical, not lexical, construct.

#### Requirement

Models should be able to reference or nest other model segments as arguments. The referenced or enclosed segments should be able to contain an arbitrarily large set of other model elements. Referenced segments should be able to be local or external.

#### Possible representation



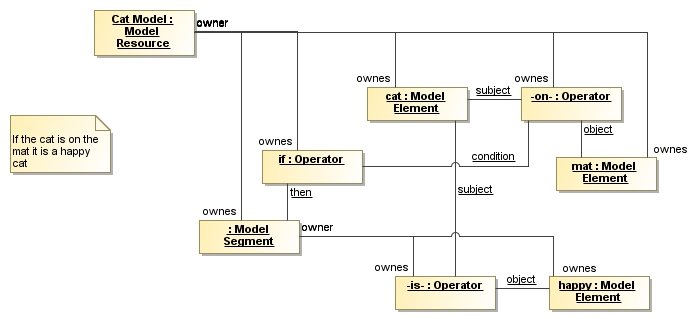
The primary model elements are:

* **Model** – a set of model elements that represent axioms relating to a subject area.
* Model Element – an axiom where the axiom contained in and asserted within the context of a mode.
* **Model Resource** – an identifiable container for a set of model elements – usually a file or resource within a repository. The model resource is a representation of the physical resource which is orthogonal to the logical structure of the model but may still be references by another model.
* **Model segment** - allows a model to be nested inside of another model or an external model to be referenced. A model resource is actual “data” in a file or repository. A model segment is nested inside such a model resource. A model reference imports the model elements of an external model.
* **Model Reference** – a reference to an external model that imports the axioms of that external model.

Since model segments and references are model elements, they may be used as the subject of relations in the model, operators such as “and”, “or” and qualification are those common to first order logic.

**See also**: This presentation from Pat Hayes: <http://www.slideshare.net/PatHayes/blogic-iswc-2009-invited-talk>

#### Example



Above is the “cat” model as nested models and model elements.

### Parameterization

Issue Scope: All major modeling languages

Note: Needs review

#### Summary of issue

Parameterization provides for defining and using patterns of structure or behavior. Such structures frequently need parameters (also called arguments or operands). Parameters are used for function arguments, parameterized types, compositions and general patterns.

#### Examples

* Define f(number: x) {f is parameterized with the argument X}
* Template classes

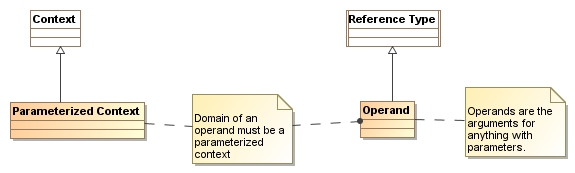
#### Requirement

Since it is common that cross-domain and cross system interactions are done with parameterized functions and/or data the understanding of parameterization between these systems is required.

Provide a general representation for parameterizations that can unify concepts across models and languages.

Parameterization should also be able to represent composite concepts, such that complex concepts can be “built up” from simpler ones. Most models of higher level languages will then use these composite concepts instead of complex patterns of primitive concepts, yet the composite concepts can be understood in terms of their composite representation. This requirement builds on composition and biding.

#### Possible representation



Parameterization builds on relations. Reference type is specialized to represent the operand of anything that is parameterized.

#### Example

See expressions

{TODO add more examples}

### Perspective agnostic assertions

Issue Scope: All major modeling languages

#### Summary of issue

The same statements (or facts or assertion) may be stated from the perspective of any involved role. The core representation of an statement/assertion/fact should allow for all such statements within the same assertion identity. It should be the same identity since such assertions may be the subject of other relations, such as when it is true or who asserted it. In, for example, OWL the use of “inverse” creates corresponding triples but these are then often hard to relate to the asserted triple.

#### Example

* The following are the same assertions
  + “Joe likes sue” is the same assertion as
  + “sue is liked by joe” and the same as
  + a table with columns “entity” and a column for “likes” with “Joe” and “Sue” in each column, respectively.
* The following are the same assertions
  + Cory is the husband of Cheryl
  + Cory has the wife Cheryl
  + Cheryl has the husband Cory
  + There is a marriage where Cory is the husband and Cheryl is the wife.

#### Other considerations

* Both simple properties and relations have can be expressed from multiple perspectives
* There are noun and verb variants of the same assertions

#### Requirement

* The representation of a assertion should be independent of the multiple representations of that assertion
* Both noun and verb expressions of the same assertion should be expressible
* All perspectives on the same assertion should have the same identity

#### Tests

See the above examples

#### Possible representation

Not yet

### Reification Choices

Issue Scope: All known languages

Note: Needs review

#### Summary of issue

There is a class of concepts that involve connections between entities where the concept can sometimes be defined as a relation and at other times defined as an entity class. When a relation is defined as a class it is called “reification”. Reification typically is used when the relation needs to have some kind of identity supporting more than two roles, properties of the relation or tracking provenance of the relation.

In different models the same concepts will sometimes be specified using relations and sometimes be reified as types with a set of relations. While these two representations look very different they represent the same thing.

When using or extending an existing model it can be difficult if a relation has not been reified and you require some reification feature. Alternatively reified relations add complexity to what seems like it should be a simpler model. Since there is no direct way to connect the reified and non-reified relation semantically, separate models become disjoint where they should converge.

Having two different ways to represent the same thing leads to complexity and stovepiping and should be reduced as much as possible. Our goal is to eliminate the need for the distinction.

#### Example

In UML generalization is represented as a class so as to support some specialized features like “generalization sets”. In MOF (prior to 2.4) generalization was defined as a property – the argument being that this is simpler. This divergence in the models was a conscious choice made for (overly?) pragmatic reasons. The result of this and similar issues was a divergence in the UML and MOF meta models for years.

Modelers are constantly making “reification” choices like this and sometimes refactoring when the choice was incorrect or inappropriate when the model is used in a new context.

#### Other considerations

* Properties of properties are not uncommon and sometimes used to define UNITS.
* For provenance to work properly each assertion in the model, including each link, should be traceable back to its source – this implies some kind of identity, which implies reification.
* It is sometimes desirable to put conditions on relations – this is a major part of SBVR. Such conditions require a reified association. For example: Don’t sell Wine on Sunday.
* Some semantic models, such as the IDEAS framework used in DoDAF-DM2 depend on a semantically rich hierarchy of relations where it is common to have relations between relations. Once relations are “first class”, relations between relations and relation specializations become natural and simplify the model (See DM2 example, below).
* Reification can apply to behavioral as well as structural relations. Behavioral reification is common in OO and is usually spotted by a set of classes having a “run” or “execute” method.

#### Requirement

Structure relations and associations such that there is no need for the modeler to make a reification choice. Allow properties and relations to and from any association. This should be accomplished without the complexity inherent in manually reifying a relation. By building this into the infrastructure the cost of this generality can be minimized, however it does have a cost.

#### Possible representation

See the representation under associations and properties.

#### DM2 Example

The diagram above is from DoDAF-DMA using the IDEAS framework expressed in a UML profile. Things to note:

1. Associations (in green) can be between entities (purple) or other associations
2. Associations have well defined semantics, these are expressed as their association type – things like “overlap”, temporal whole/part, “Before/After”, etc..

While the UML representation of IDEAS is a bit odd, the above concepts are valid and useful. See also: <http://en.wikipedia.org/wiki/IDEAS_Group>

### Roles

Issue Scope: All major modeling languages

#### Summary of issue

The concept of a “role” is pervasive in how we think about the word; “President of the united states” (POTUS) is a role, at this moment it is filled by “Barack Obama”. There are different word senses of role, a dictionary definition is: “The function assumed or part played by a person or thing in a particular situation.”

As essential as this concept is most modeling languages do not have a first-class concept of role or a theory of roles. Roles are sometimes modeled as classes, associations or properties.

#### Example

Cory is a parent, he has 2 children: Chelsea and Connor.

In one respect it can be said “Cory is a parent”. This states a type or status about Cory.

In another respect it can be said “Cory is the parent of Chelsea” and “Cory is the parent of Connor. In this respect Cory is taking on a parent role with respect to the relation to each child.

You could also say “Cory has been a parent since 1992”. Here we are asserting a property of “parent” across all the children.

#### Other considerations

* Roles can be placeholders in functions
* While roles can be “encapsulated” in some kind of type or relation, the role concepts they represent can be found in multiple relations.
* Roles can be filled by individuals or sets
* Roles are typically constrained by one or more types
* The number of times an entity can play a particular role may be constrained or may be open.
* Roles can behave like types and be subject to subtype/supertype relations, unions, intersections, etc.
* What roles an entity plays may change over time

#### Requirement

* Identify and provide explicitly support for role concepts
  + As a part of a relation
  + As a type of an individual
  + Role as a variable in a function
* Provide for binding of roles
  + To individuals
  + To types
* Provide for type relations between roles

#### Tests

* Represent marriage as defined in <http://www.omgwiki.org/architecture-ecosystem/doku.php?id=composite_concepts>
* Represent Cory and his children with properties about each child relation (such as when the child was born) as well as properties about the role in general (how many children one has).
* Represent roles that may be and are filled by both individuals and sets/types
* Represent constraints on how many times an entity may play a role (assume a person may be the awardee of a Nobel prize only once)
* Show how roles are related to systems and compositions

#### Possible representation

#### Example

### Scope and composition of concept definitions

Issue Scope: Pervasive

#### Summary of issue

In the conceptual domain we have concepts that we use in multiple context and multiple variations. In information structures and even ontologies properties and relations in particular are specializations of these general concepts, combining multiple dimensions and concepts into one.

Within SIMF it should be possible to define concepts in a way that corresponds with the conceptual dimension and to then construct fact or concept representations out of combinations of these concepts, recognizing their similarities and differences.

We will use the term “definitional dimension” to mean a particular facet of a facts description, such as its subject, predicate, timeframe, etc.

#### Example

Consider the simple concept of “weight”, obviously weight applies to any physical object (ignoring weight Vs. mass issues for now). It is common to have properties that specify weight:

Person {

hasWeight:Int;

}

Or we may be a bit more specific:

Person {

hasWeight: Kilograms;

}

* Connor weighs 105 pounds
* Connor weighed 95 pounds last visit to the Dr.

But what if we were capturing the weight that person should be:

Person{

goalWeight: Range { minWeight, maxWeight};

}

* I should weigh between 180 and 200 pounds

Or, perhaps a set of weight measurements:

Person{

Measure: WeightMeasurement {

hasWeight: Kilograms;

timeTaken: DateTime;

}

}

Or,

* Today I measured Cheryl’s weight as 167 pounds

The concept of “weight” may be used in a number of different contexts:

* Asserted weight
* Measured weight
* Intended weight
* Expected weight
* Etc.

It may also be relative to different times:

* Past,
* present or
* future.

Weight may also apply to an individual, a type or a set:

* John weighs more then 100 pounds
* All my friends weigh more than 100 pounds

Weight may be expressed in different forms

* Integers
* Reals
* Tags for hard or soft ranges (e.g. a lot, a little)
* Ranges
* Min and max

Finally we should be able to understand both discrete values and ranges

* Jan weighs 100 pounds
* Paul weighs between 140 and 160 pounds
* Fred weighs at least 200 pounds

Our concern is that the concept of “weight” should be defined once in terms of its real domain meaning (at least once within any CDM) as should other dimensions such as the assertion context or timeframe. It should then be possible to define a particular property or relation by combining these concepts. So when we make up terms such as “hasWeight” we are combining 2 concepts: that some subject “has” some value, implying an assertion in the current timeframe and that the value they have is a weight. When we define the type of this property properly, we would also identify the unit; which could be fixed (LBS) or variable with respect to each instance (any unit of weight).

#### Other considerations

* These domain concepts should normally have dictionary definitions and be able to be related to these. For example wordnet:: •S: (n) weight (weight%1:07:00::) (the vertical force exerted by a mass as a result of gravity)
* Consider that things like “has” may be more like the “type of the line” than the predicate.
* In ISO 11179: Name parts consist of discrete terms. The terms in this annex are derived from administered items in the MDR metamodel described in ISO/IEC 11179-3. These are: object class terms, property terms, representation terms, and qualifier terms. These “name parts” correspond to these different dimensions of a fact representation.
* The pattern that would relate “weight” to its subject and the definitional dimensions would be the same pattern as would apply to any quantity and many other relations. These patterns should also be reusable.
* David Frankel has written on this topic, see: <http://www.bptrends.com/publicationfiles/02-10-COL-MDA%20Journal%202010-02%20Semantic%20Metadata-Frankel-v01-cap.pdf>

#### Requirements

* Concepts should be able to be defined in terms of their domain semantic without being bound to a particular fact representation, speech act or structure.
* Specific facts, fact types, representations and speech acts should be able to be constructed from or related to multiple definitional dimensions.
* In many cases the name phrases of facts or fact components should be able to be determined from the model. Likewise, dimensions describing a fact should, in many cases, be able to be determined from a phrase and an understanding of the language syntax.
* This compositional approach to concept definitions should allow the automated determination of similarity and differences between definitions
* Provide for definitional dimensions that include but are not limited to: the kind of value (e.g. weight), timeframe, data representation, individuals sets or types, assertions vs measurements vs predictions vs goals, etc.

#### Tests

* Find models with phrases for concept/fact names (e.g. hasWeight)
  + It should be possible to relate these phrases to the constituent terms and concepts that form the phrase and relate these to dictionary definitions of the concepts
* Compare “weight” as a property with weight for the same individual expressed as a measurement.
* Express a rule that the last measured weight is the asserted weight for an individual
* Express that the FAA considers all people on an airplane to weigh 170lbs
* Express that a weight may be expressed as a number, a range or a number with an error
* Express a model and instance where the unit is known (e.g. KG) with one where each instance may express the unit. Show how these are related.

### Subject of a Model

Issue Scope: All major modeling languages

Note: Needs review

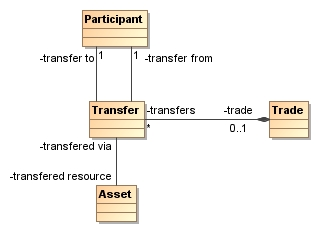
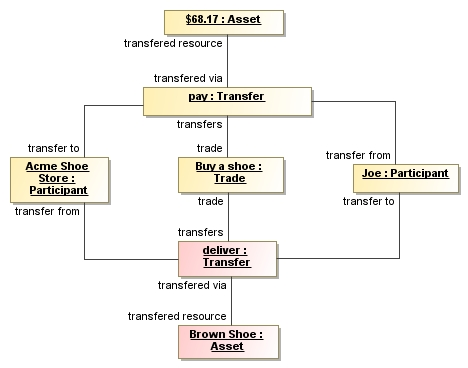
#### Summary of issue

Models can be “about” anything – the “real world”, software, physical systems, information, process, etc. Some models are about things that exist and others about things we make up. Some models may even represent elements of more than one subject – such as a business and software system within that business.

There is no way in any modeling language we are aware of to specify the subject of a model. Without knowing the subject of a model the semantics of that model are unclear.

#### Example

Consider a simple language for modeling trades, based in this model:

A trade is defined as a set of transfers of assets between participants. Lets also look at an instance of this model describing “Joe” buying brown shoes from “Acme Shoe Store” for $68.17.

If this is a model of the full business transaction that happens in the “real world” the transfer of the shoes makes sense. However, if this is a model of a computer system supporting the shoe store they may or may not want to include the physical delivery in that model (this part is shaded in red). So a system that is just concerned with payments may only have the “pay” transfer. If we are modeling the full life-cycle of the system, including the business and software mode, both models are relevant. By understanding the context and subject of the model we can better understand how to interpret it. We would like some way to say what the subject of a model is – is it the business process of buying shoes or the payment model in a cash register supporting the business model? What is the relationship between these models and how do we make the connections? How do we know what the model is a model of?

How an information management system relates to a model of the “real world” is a matter of policy for that information system and brings in concepts such as an “open world” and information completeness. A family of modeling languages should allow for these different policies. These policies frequently introduce complexities in models as well as divergent representation s of the same thing.

#### Other considerations

* A model may have a mixed set of subjects modeled – for example business models and systems models are often used together.
* Some models are intended to be precise while others are illustrative – these differences should be made explicit
* A primary source of error is confusion over models of “the real world” and information or computer artifacts representing “the real world” (or a proposed real world). There should be standard relations between these different models.
  + Models of “things” and information about things should be clearly so labeled.
* Models frequently have “as is” and “to be” components, these should also be dimensions captured in the subject of the model.

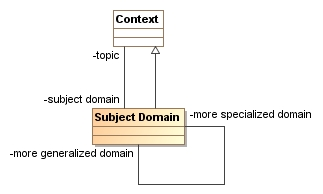
#### Requirement

There should be a way to specify the “domain of discourse” of a model or a part of a model. Domains categorize the topics of models according to the subject of a model.

There should be a relationship defined from model elements at various levels to elements that define the subject’s domain. These subject domains may form a hierarchy. The hierarchy should include items in the physical world, various conceptions or theories about the physical world (such as processes), agreements, processes, information, system designs, etc.

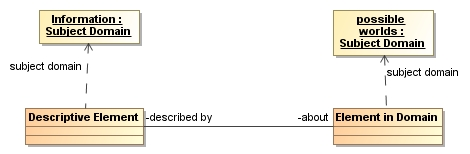
In addition, define a relationship between model elements representing “the real world” (such as people and processes) and information or computer artifacts representing that world (such as forms, tables or java classes).

#### Possible representation



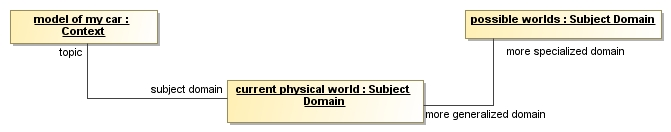
The model, above, shows how the domain of a topic could be specified by enumerating a set of instances that define the domains of interest that can form a hierarchy.

It is also necessary to be able to project from a domain model to an information model based on policies and constraints, this is the essence of “MDA” (model driven architecture). Such projections would be handled as any other “view”, see the sections of federation semantics and views.

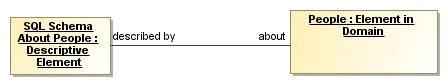


The above model fragment defines two classes for describing “elements in domain” (things in the real or a possible world) and “descriptive elements” (data about elements in the domain such as SQL tables, classes or pieces of paper). While the descriptive elements are also “in the world”, what is described by these two types is fundamentally different. The “about” relation then connects models of descriptions to models of things in the world.

#### Example



This example shows a model topic “model of my car” that has a subject domain of the “current physical world” This means that the model of my car is intended to represent the real world at a particular time (current at the time of the models publication). This example also shows that the “current physical world” is a specialization of the more general concept of any possible world (real, virtual or imagined at any time).



The above fragment shows the relationship between an SQL schema about people and an “ontological” model of people in the “real world”. The SQL schema models information elements in a DBMS. The ontology models people. Instances of the SQL Schema (rows in the DBMS) are about instances of the ontology (people). Note that not all ontologies model the real world, some ontologies model information, just like the SQL schema.

### Type/Instance

Issue Scope: UML-2

#### Summary of issue

One of the most basic concepts of modeling, many logics and of most software languages is that of “types”. The concept of types only makes sense in the presence of “instances” of those types (We will discuss what we mean by instance, below). However important types are; it is instances that are the reason for defining types.

UML Note: Oddly, UML does not represent instances, it represents “instance specifications” which suggests that anything in the model is somehow removed from “real instances” more than it would be in any model. Instances are not the “real world” in any information system, but symbols for and axioms about the real world as we perceive it. But, there is no reason for the complexity and level of indirection in “instance specification”. The MOF has regular instances, why not UML? Any object or data type is an instance of something.

The concept of type is best described as a “predicate”; something TRUE about the instance. I could have a type named “open invoice” and as long as I can test that something is an invoice and that it is open, I can determine if something is an open invoice. The OO concept of class doesn’t test instances, it creates them and through the language design guarantees that something created as an invoice stays an invoice. So an OO class is a factory for instances where the predicate will always be true – the predicate is that the instance is created by the class.

Ontology languages (like OWL) allow you to *assert* that an instance is of some type, it will also infer a type if it is required by the logic. But in all these cases we have instances that, one way or another, satisfies the predicate of the type.

The Type/Instance issue is two-fold. First, SIMF should have a simple and direct concept of “instance” (or object) without the indirection of UML –note that MOF already works this way, as do all ontological languages. The second is that there is a very direct relation between types and instances – every instance has a set of types it satisfies and every type has a set of instances that is its extent (but our knowledge of those extents may be incomplete). The type/instance relation is simply set membership {note – this needs to be validated}. How that membership is determined varies.

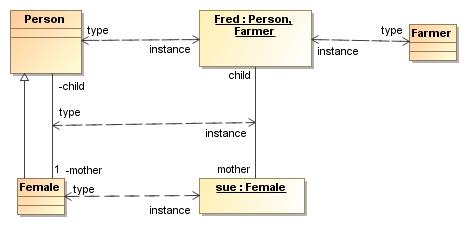
Different types will have different ways to specify their predicate and different ways to create/infer or assert type membership – but the basic relation between types and instances is simple and basic. Type/instance and the relation between them are SIMF concepts.

Drawing on SIMF concepts of type/instance we should also recognize that anything can have a type:

* OO Objects have a type – in this document we call these classes
* Links between elements can have a type – we call these properties or associations
* Actions or anything that happens can have a type – we call these behaviors or processes
* Numbers can have a type – we call these units or data types. We also type numbers based on their degree and precision into types like Integer and Real.
* Enumerations are types where the set of instances of a type is specified, a closed set.
* Systems (natural or artificial) can have a type – we call these designs, theories or “code”.
* Roles can have a type and be the type of the individual playing the role

The type/instance relation is a fundamental concept linking anything we can model.

#### Example



The above diagram shows multiple type/instance relationships:

* Person<>Fred
* Farmer<>Fred
* Female<>sue
* Mother/child association <> link between Fred and Sue

Note: The dual dependencies marked “type/instance” are a way to diagram the type of an instance that is in the UML meta model but not shown graphically on a UML diagram.

Note that everything in a model is an instance of something. Note also that “Farmer” could be a subtype of person, but perhaps farmer was created in another model and doesn’t know about person – farmer and person are potentially overlapping types.

For an example of type/instance relation applied to behaviors, see the paper from Conrad Bock {<http://www.omg.org/cgi-bin/doc?ad/09-08-05>}.

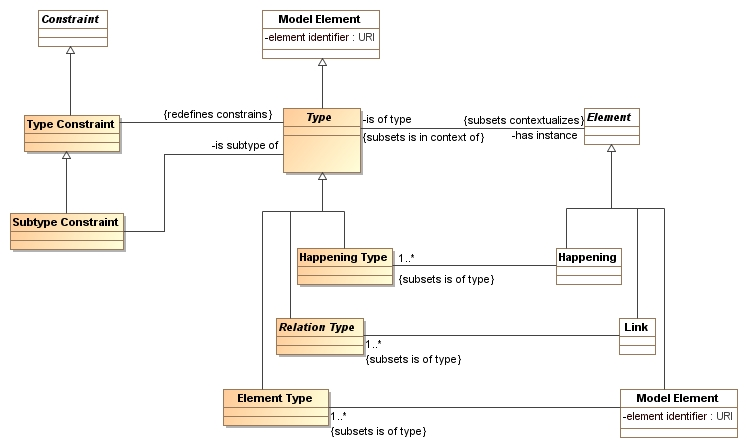
#### Other considerations

* The term “instance” somewhat implies the OO concept of an object that is created by a class and may be “stuck” with that class and only that class. The more general meaning is that an instance is anything that satisfies the types predicate. It is this more general word sense that we are using. Unfortunately there doesn’t seem to be another generally accepted term for the more general concept of instance, so we are using instance.
* UML also speaks about “parts” in a composite structure or collaboration as “instances”. We think this is incorrect, these are not instances. They are roles in a composition pattern. A class does not contain a set of “instances” it contains roles (or slots) where instances can play a role in instances of the composite type. It is a pattern of types, not instances. Composition will be covered in another section.
* Instances may be instances of multiple types and this may change over time. Of course, many languages constrain this capability. The type system should be able to specify permissible combinations of types, equivalent types and required combinations of types (see SMOF).
* “Powertypes” are types of types, or metatypes, that have a common supertype.

#### Requirement

SIMF should provide for a general concept of types and instances and apply the type/instance pattern to objects, data types, associations, properties and behaviors. Provide a basic mechanism for describing constraints on types.

#### Possible representation



The diagram above shows the basic type/instance relation as well has how it is specialized for:

* Element type as the type for Elements (things),
* associations as the types for links and
* happening types as the type for anything that happens. Behaviors are things that are done by an actor and would be a subtype of a “happening”.

The subtypes of “element” shown here are constrained to have the corresponding metatype as indicated by the subset relations {note that we are using a modeling convention of introducing association constraints using subsets (Known as “restrictions” in OWL) – this constrains an existing association, it doesn’t define a new one.}

A type constraint applies to anything that constrains the set of instances of the type. The subtype constraint defines the general concept of a subtype – that the set of instances of the subtype are a subset of the set of instances of the supertype.

Note that this type instance model says nothing about how the types are defined. While there will be some common concepts for type specification there must also be a lot of opportunity for specialization as different languages have different ways to do so. Common type specification concepts will be expanded in the fully worked out model. Further down the hierarchy would be classes as type for “OO Objects” as instances and classes as types.

Note also that each of these basic metatypes could correspond to common UML shapes for a basic modeling capability while not implying anything is “OO”.

### Units & Quantities

Issue Scope: All major modeling languages

Note: Needs review

#### Summary of issue

Modeling languages have inherited primitive numeric types from computer assembly languages. This loses crucial domain content in the form of units and quantities. Numeric in models should use units, not “int” or “real”. Of course it is permissible that there be constraints on the size and precision of numbers, but units should be the primary semantic of a numeric reference.

#### Example

Car {

Current Speed: miles per hour; // Instead of current speed : double

}

Container {

Capacity: Liquid Measure;

}

#### Other considerations

* In some cases the unit of a property is fixed {i.e. millimeters}, in other cases only a family of unit is known and the specific unit chosen is specified in the data {i.e. Capacity: Liquid Measure}. The modeling paradigm should allow for both fixed units and related families of units.

#### Requirement

Provide a general model for units and quantities that issufficiently usable and intuitive as to overtake use of int/float/doubles, etc.

#### Possible representation

See SysML for a general treatment of units and quantities.

### Views, Viewpoints & Notations

Issue Scope: Most major modeling languages

Note: Needs review

#### Summary of issue

Models represent information about the real or a possible world. Different information about the same things may be relevant to different stakeholders. In addition, different stakeholders make require different ways to structure information and different terminology. What binds all of these views together is that they are all about various aspects of the same things.

Most models are only usable in the context and viewpoint in which they are created, limiting their usefulness and ability for repurposing, federating and analyzing the information in those models.

Our goal is that model information may be sourced from and projected onto multiple views corresponding to multiple viewpoints, relative to different stakeholders and communities.

#### Example

See architecture ecosystem wiki here: <http://www.omgwiki.org/architecture-ecosystem/doku.php?id=use_cases>

#### Requirement

Provide for the ability to project a model rendered in one language to a model rendered in another. This requirement builds on other requirements such as common concepts (as defined here), model semantics, federation semantics and projection onto concrete syntaxes.

#### Possible representation

See the draft diagram definition RFP response (OMG Document Number: ad/2010-05-01) as a representation for views and viewpoints. The architecture defined in diagram definition could be retargeted to be integrated with this SIMF model.



Figure 1 Diagram Definition Architecture

### Template

Issue Scope: All major modeling languages

#### Summary of issue

#### Example

#### Other considerations

#### Requirement

#### Tests

#### Possible representation

#### Example

## System Stories

### The pump

The following is a user story with respect to composition and the parts of a composition.

Last Thursday I complained that most ontologies do not give adequate treatment to what I call system components, and if ontology is going to gain traction within the systems world, it needs to get a better understanding of this central idea in systems engineering.

I illustrated the issue by telling the (simplified) life story of a system

component: the pump, P101, at the bottom of a distillation column. Here is its story.

The designer creates a drawing of the distillation column including at the bottom of the column a pump to pump away the column bottoms. He labels it P101, decides that one pump will be sufficient, and gives the specification for the pump in terms of Net Positive Suction Head, differential head, flow rate, materials of construction, and many other things.

The construction engineer picks up the drawing and specification and notices he has to install a pump as P101. Fortunately, he has a pump in stock from a previous project, that has been in stores unused for 5 years which exactly meets the specification. On it is stamped Serial No S3556.

The designer and the Operator comes to see the pump be installed, and once the connections are made, he gives the pump a friendly kick and says to the construction engineer "It's good to see P101 realized at last". The construction engineer says in return "Yes, and it's good to get S3556 off my hands at last." He turns to the operator and says "Why don't we change your drawings to show S3556 instead of P101?" The operator says "No, don't do that, it's a replaceable part, and one day another pump will be put there, and I don't want to have to change all the drawings and other documentation that refers to P101 each time it is replaced, as far as I am concerned it's the same pump whatever is installed there."

Some time later the pump breaks down and needs to be taken back to the workshop. The maintenance engineer says to the operator "Hi, can I take

S3556 installed as P101 back to the workshop?" The operator replies "Sure, but what am I supposed to do without my P101? If it does not exist I cannot operate my distillation column." The maintenance engineer responds, "I understand. We have another pump S4567, that meets the same specification as P101. We'll replace S3556 with it and you will only be without P101 for a few hours. I don't understand how you can continue to call it P101 though when all the parts have changed at once." The operator replies "I don't care about that. What I care about is what is connected in my system to pump the liquid from the bottom of the column. As long as it does that, it is P101 to me."

Later the distillation column is demolished. The operator says, "A sad end, I was very fond of P101, but it is no more." The demolition engineer says, "Yes indeed. Fortunately, we can take S4567 and use it on another plant."

It's probably worth summarising the key characteristics of a system

component:

- It comes into existence the first time it is installed.

- It is identical to the equipment items installed, whilst they are installed (but not before or after).

- It can survive complete replacement of all its parts at once.

- It can survive periods of non-existence.

- It ceases to exist when the system it is a component of ceases to exist.

This is clearly rather different from the life of ordinary physical objects.

However, relatively few ontologies recognise that such things exist. Many try to fob system components off as being classes, or abstract individuals, though these clearly do not have the required characteristics.

Ontologists need to step up to the mark here and provide proper recognition for system components.

Regards

Matthew West

Information Junction

Tel: +44 1489 880185

Mobile: +44 750 3385279

Skype: dr.matthew.west

matthew.west@informationjunction.co.uk

http://www.informationjunction.co.uk/

http://www.matthew-west.org.uk/

In response to the above another layer was added.

Consider that there is a generic model for a distillation column which may be specialized and is then used for a specific distillation column as is the story above. P101 may be in both models or “P101” may be renamed in the specific model. What is the relationship between these models and “P101” in each?

## Satisfaction of requirements

The following table is intended as a tool to evaluate various approaches to SIMF, related to the above requirements. Each requirement is listed with a “score” (1 being best, 5 being worst) and an explanation.

|  |  |  |
| --- | --- | --- |
| Requirement | Score | Comment |
| Associations, Association Classes & Properties |  |  |
| Closed Vs. Open Descriptions |  |  |
| Composite Concepts |  |  |
| Composition |  |  |
| Constraints |  |  |
| Dominant Structure & Context |  |  |
| Expressions |  |  |
| Expressions about individuals, types and sets using the same predicate |  |  |
| Federation Semantics |  |  |
| Model Semantics |  |  |
| Multiple Names for Anything |  |  |
| Nested Models & Graphs |  |  |
| Parameterization |  |  |
| Perspective agnostic assertions |  |  |
| Reification Choices |  |  |
| Roles |  |  |
| Scope and composition of concept definitions |  |  |
| Subject of a Model |  |  |
| Type/Instance |  |  |
| Units & Quantities |  |  |
| Views, Viewpoints & Notations |  |  |

1. Common logic: <http://en.wikipedia.org/wiki/Common_logic> [↑](#endnote-ref-1)
2. Synonym set (or synset) is a term used in [Wordnet](http://wordnet.princeton.edu/) [↑](#footnote-ref-1)