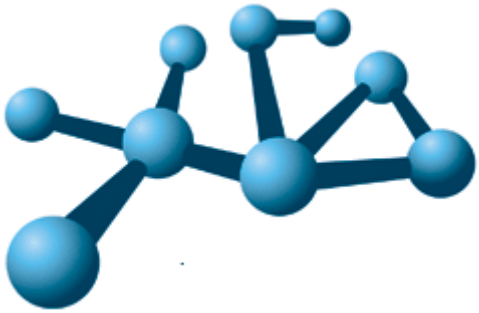


Semantic Modeling Profile

Contents

1 Semantic Modeling UML Profile 3
1.1 Semantic Modeling Introduction 3
1.1.1 What is a semantic model? 3
1.1.2 Semantic Domain Models as Reference Models 4
1.1.3 Advantages of semantic domain models 6
1.1.4 Thinking Semantically 8
1.1.5 Expressing Semantic Models in UML 9
1.2 Essential Semantic Model 10
1.2.2 Packages Representing Subject Areas 11
1.2.3 Classes representing Domain Entities 12
1.2.4 Class Generalization 12
1.2.5 UML Associations Representing Domain Relationships 13
1.2.6 Association Classes as “First Class” Relationships 14
1.2.7 Attribute Properties 14
1.2.8 Roles 15
1.2.9 Phases 16
1.2.10 Example Essential Model 18
1.2.11 What semantic models don’t assume 18
1.3 Extended Semantic Precision of an Essential Model 20
1.3.1 Class Semantics 20
1.3.2 Kinds of classes 23
1.3.3 Equivalent Class 25
1.3.4 Property Hierarchies 25
1.3.5 Annotation 26
1.3.6 Assertions about concepts 27
1.3.7 Constraining properties and associations 28

1.3.8 Tightening a property's type 29
1.3.9 Inferring a type from its properties 30
1.3.10 Property Chain 31
1.3.11 Equivalent Property 32
1.4 Semantic Modeling Profile (SMP) Reference 34
1.4.1 Diagram: Semantic Modeling Profile (SMP) 34
1.4.2 Stereotype Annotation 34
1.4.3 1.2.3 Stereotype Annotation Property 35
1.4.4 1.2.4 Stereotype Anything 35
1.4.5 Stereotype Base Unit Value 35
1.4.6 Stereotype Category 35
1.4.7 UML Attribute 35
1.4.8 Stereotype Semantic model 36
1.4.9 Stereotype Disjoint With 36
1.4.10 Stereotype Enumerates 36
1.4.11 Stereotype Equivalent Class 36
1.4.12 Stereotype Equivalent Property 37
1.4.13 1.2.13 Stereotype External Reference 37
1.4.14 Stereotype Has Value 38
1.4.15 Stereotype Information Model 38
1.4.16 Stereotype Intersection 38
1.4.17 Stereotype In Context Of 38
1.4.18 Stereotype Model 38
1.4.19 Stereotype Phase 39
1.4.20 Stereotype Quantity Kind 39
1.4.21 Stereotype Relationship 39
1.4.22 Stereotype Represents 40
1.4.23 Stereotype Resource 40
1.4.24 Stereotype Sufficient 41
1.4.25 Stereotype Synonym 41
1.4.26 Stereotype Union 41
1.4.27 Stereotype Unit Value 41
1.4.28 Stereotype Value 42

1 Semantic Modeling UML Profile

This section defines the UML profile for Semantic Modeling (SMP). In order to improve UML's suitability for modeling real-world concepts, this profile interprets standard UML with semantic extensions, as detailed below. The SMP is part of an in-progress standards effort within the OMG, it is provided here as an open sourced resource under the "LGPL" License[1].

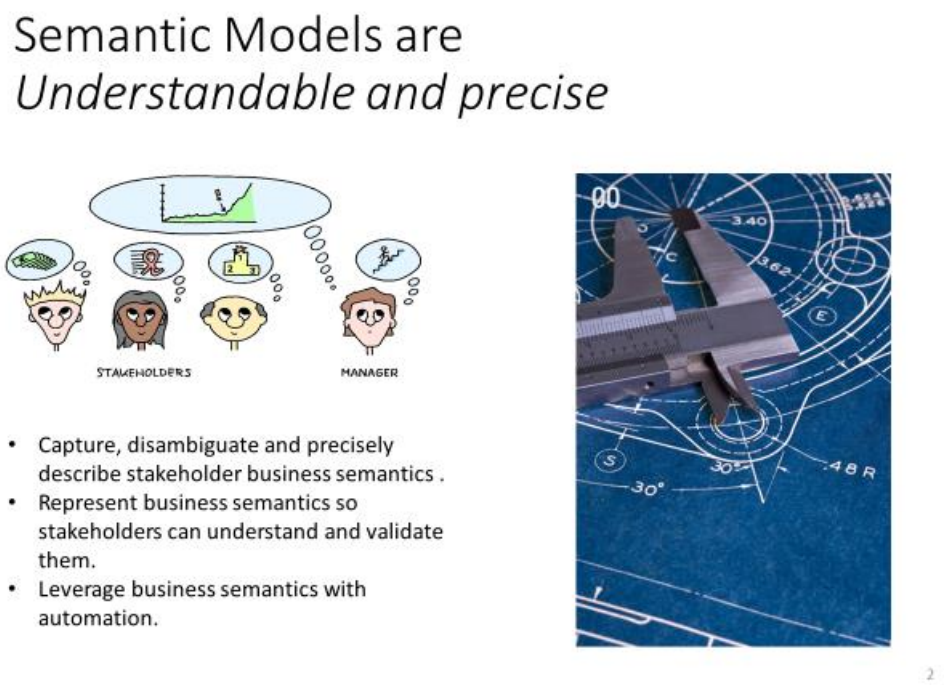
1.1 Semantic Modeling Introduction

1.1.1 What is a semantic model?

A semantic model is a model of concepts about a subject area, as understood by a community of stakeholders, defined in such a way that the meaning of those concepts can be precise, understood by the stakeholders and processed by computer software.

Semantic models are *structured information* about the subject area, as opposed to unstructured information, such as natural language. The concepts, terms and structure used to define a semantic model is a language for Semantic Modeling. Expressing a model in a Semantic Modeling language helps to define the meaning of the concepts in that model, the model semantics. Each concept in a model becomes the definition of a term in the domains vocabulary, linking how stakeholders speak about their domain and how it may be expressed in a systems.

While a Semantic Modeling language provides a foundation for defining the semantics of concepts, it is not enough. The language must be used with care and precision to capture, disambiguate and precisely define the concepts.



[1] LGPL: https://www.gnu.org/licenses/lgpl-3.0.en.html

There are multiple terms that, more or less, capture some of the intent of semantic models. Some of these terms include:

- Semantic model
- Ontology
- Conceptual Domain Model
- Computation Independent Model (CIM)
- Conceptual Information Model
- Concept Model
- Structured Taxonomy

1.1.2 Semantic Domain Models as Reference Models

Anything can be modeled semantically. The focus of the Semantic Modeling UML profile is to capture the *semantics of domain concepts*. Domain concepts are the actual things we deal with in our business, mission or life such as: people, places, things, agreements, and situations. Semantic domain models may look somewhat like data models or object-oriented programming models, but these are models of technology solutions – not the domain. When we attempt to mix the model of a domain with a solution architecture both tend to get distorted. We create the semantic domain model as a reference for what design models mean and the real-world things they represent and impact. From this point forward we will consider "Semantic Model" and "Semantic Domain Reference Model" to be synonymous.

For this reason, the semantic domain model is considered a reference model – not a solution design. It is referenced by multiple designs to help define and federate them. This is particularly important for data models; while a semantic domain model may look somewhat like a data model, the concerns should not be mixed – the domain model is a model of domain concepts, not how those concepts are represented in data schema or object-oriented programs.

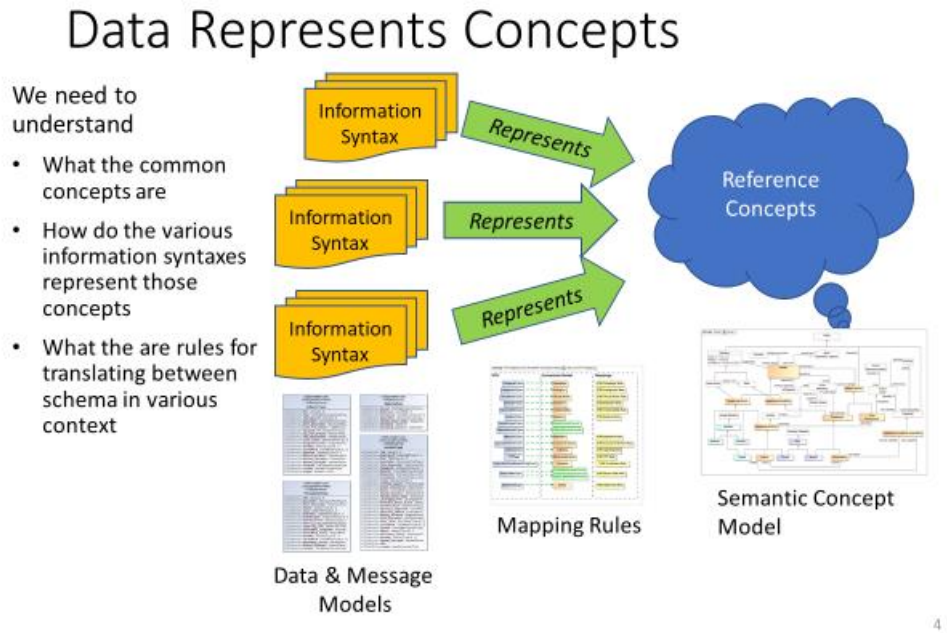
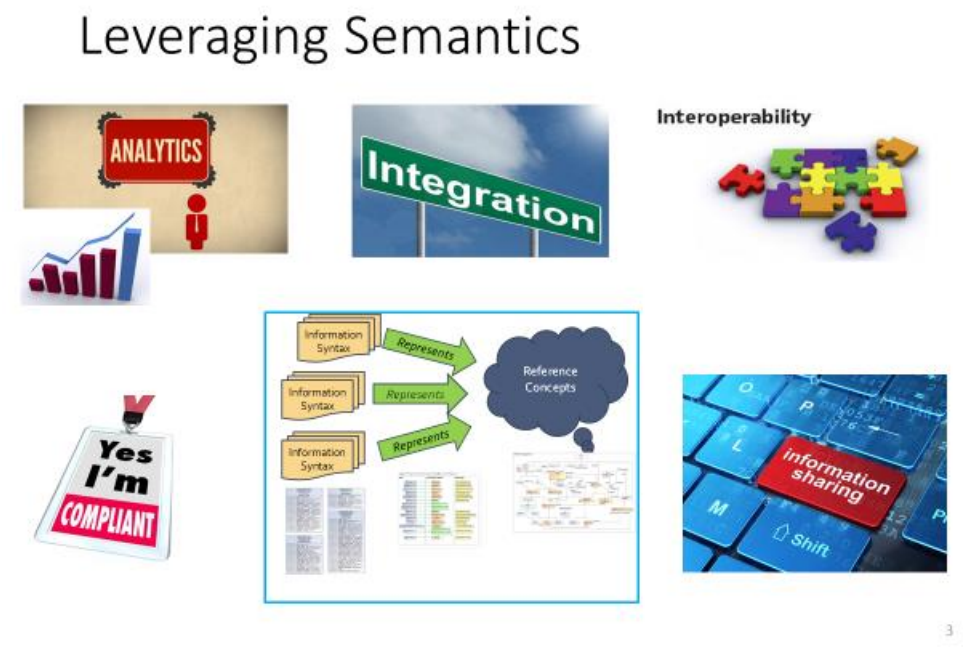


Figure 1 Data Represents Concepts

The pattern of "data represents concept" is essential to leveraging semantic models. This pattern frees implementations from slavish compliance with a structured "enterprise data model" while connecting data to its meaning in the semantic model.

1.1.3 Advantages of semantic domain models

A semantic model defined by subject matter experts is more durable than a data model or logical information model designed with a particular system in mind. Thus, one definition of concepts and properties can be represented by multiple logical models, each optimizing for different technical goals. Note that there are multiple interpretations of "logical model" that span from almost conceptual to near a data schema. Semantic models can be used to help federate any of these levels of abstraction.



The key use cases for semantic models fall into the following categories:

1.1.3.1 Federation

Systems have been designed to solve specific problems for specific organizations or stakeholders. Requirements for these systems and the technologies they are implemented in drive data models, services models, component boundaries, algorithms, security concerns and other design related viewpoints. The process of collecting requirements and producing bespoke designs has been proven to create operational systems.

As computing has evolved we have moved from the era of independent systems to *systems of systems*. Systems no longer stand-alone but are part of a vast network of interconnected systems that power our enterprises and society. These systems of systems most operate in a coordinated way and, within an enterprise, be managed to reliably enable our enterprise processes. But, since each such system is independently designed they don't naturally work together – new "integration" systems are designed to "connect the dots", the network of systems and integrations becomes complex and very difficult to manage or change. There is no visibility to how the system of system fits together or works as a whole.

The Semantic Domain Model becomes a reference point for the system of systems, by capturing the domain (business) concepts that all these systems operation on, it becomes a "pivot point" between the different ways different data structures represent the same concepts about the same domain. This pivot point can be used by people or systems for federation, federation provides:

- **Federated Analytics** – capturing data from within and outside the enterprise to derive new knowledge, including data science and machine learning.
- **Collaboration and Information Sharing** – working between different business units, enterprises and systems in an efficiently and collaboratively.
- **Data & Systems Integration** – "Connecting the dots" between all the systems and data that must be integrated.

1.1.3.2 Data & Application Governance

Due to the natural proliferation of overlapping and related systems, the same business concept can be represented in multiple systems, databases, messages and applications. Most of these systems probably have a different representation, term or schema for these same concepts as discussed above as the "system of systems". Where metrics are produced from source facts, they are often derived from these system and technology artifacts in ways that are inscrutable for business stakeholders (and sometimes even for the technologist!). Policies about data, such as what is sensitive or personally identifiable is difficult and expensive since there is no one place to make a policy about what the data represents (such as a SSN), but only how it appears in dozens of system records.

A semantic model defines what data means, not how it is represented. Implementation schema and/or data models can reference semantic models for their meaning. This provides advantages for both sides; an element in a data schema can be traced back to its semantics to find the policies and business stewards applicable to that concept. A business concept can be traced back to the data elements that represent it along with identification of "systems of record", "systems of reference" and the lineage of how business facts travel though the system. The use cases for leveraging semantics for governance include:

- **Grounding the meaning** of data in a simpler, more business focused semantic model.
- **Applying business rules to business concepts** which can then "flow through" to data assets.
- **Separation** of the business governance of information from the technical governance of data assets and applications
- **Understanding where business "facts" are stored** or transmitted and which are authoritative
- **Understanding the lineage of business information** through the system of systems

1.1.3.3 Inference, Machine Learning and Rule Processing

Once information is semantically well defined and federated it can be processed algorithmically to derive new knowledge and insight. There are multiple approaches and technologies to leveraging information, but they all depend on the semantics of the information. Some of these approaches and technologies include:

- **Application code –** application code can be written, and sometimes generated, to leverage the semantic model for more "intelligent data".
- **Rule based systems** – business and operational rules operate on domain concepts to enforce policy, validate compliance and deduce new information.
- **Inference engines** – semantic models mapped to logic based languages, such as the "Ontology Definition Language: (OWL), ERGO, and Common Logic (CL) can infer new information from existing data.
- **Machine Learning** – Machine learning depends on well defined (semantic) data to define and discover patterns in data.

1.1.3.4 Forward Engineering

Forward engineering is the process of taking a more general model, such as a semantic reference model, adding design choices, and then producing a design that is consistent with both the reference semantics and the business or technology choices. Forward engineering can be manual or automated. When designs are forward engineered from semantic models they are more grounded in business concerns, easier to maintain and more naturally able to integrate and federate. Forward engineering is particularly effective in producing:

- Data models
- Message models for services and API architectures
- Implementation of business rules

1.1.4 Thinking Semantically

A semantic model is not an information or data model. When someone think about concepts, they think about real-world things, not data structures or even natural language text about those things. These real-world concepts become the pivot points around which we define and relate the many terms, languages and data structures that describe those things. For example, every Person *has biological mother* one Person, which is essential to being a Person. Such concepts provide criteria that narrow the definition of *what* a Person concept *is*, it does not specify that a system should store every person's mother. For another example, it would be reasonable for a semantic model to assert that an eye has a measurable visual acuity, but not to define how visual acuity will be represented within a computer as bits and bytes, or how often visual acuity will be stored within a database. Such technical concerns should be elaborated in a data model, which has elements that can be well defined by a semantic model. Note, however, that things such as tables and columns *are* valid concepts in their own right – as "data things", but they are different from the real-world concepts they might represent.

Semantic models can be modular. A semantic model may refer to things in a number of other semantic models. This is useful for refining another organization's semantic model, separately maintaining overlapping concepts between organizations or disciplines, or more easily managing smaller subject areas.

A semantic model consists of a network of concepts with a simple essential structure. That structure is the definition of classes, relations between them and their characteristics. Classes represent abstractions of "things" in our world – including physical things like trees or people and "made up" things like agreements.

Other concepts connect those things - the relations between things are UML associations that have properties. Things have characteristics such as weight or color. Things can also have properties that are attributes of a class. This basic network of classes, associations, and properties forms the foundation of the semantic model and defines the conceptual framework and vocabulary of a domain. Each of these concepts may have names, which form the vocabulary of a domain of interest. Various assertions are then made about these concepts and their connections that further refine the semantics of those concepts – multiplicities of relationships, specializations between concepts, essential properties of things, etc.

One of the fundamental ways we understand and organize concepts is their arrangement into hierarchies, where general concepts are specialized to form more specific concepts within a specific context or with more specific characteristics. A semantic model can arrange all the fundamental elements into conceptual hierarchies using generalization relationships. In contrast, another kind of hierarchy is a structural data hierarchy – where data elements contain other data elements. As the purpose of a semantic model is not representing data, data hierarchies are not part of a semantic model, they are typically part of logical information models that represent concepts. To allow for the many viewpoints that can exist for any concept, a concept can be in many generalization hierarchies at the same time.

1.1.5 Expressing Semantic Models in UML

A semantic model can be expressed in many ways, this profile uses UML with the Semantic Modeling profile. UML, with a few enhancements, makes an effective Semantic Modeling environment supported by standards and multiple tools. UML is understood by most architects and is already a part of the "toolbox" in most enterprises. While UML has typically been used for system design, it has proven very effective for Semantic Modeling. Also, by having semantic models in UML they can be more easily connected with design models for federation, inference and forward engineering.

The capability of UML to have multiple diagrams, dictionaries or other ways to express the same model for different stakeholder viewpoints is particularly effective. While we typically start with class diagrams, such diagrams are only one way to view a model.

The Semantic Modeling profile defines the interpretation of UML concepts used, extends UML concepts with "stereotypes" and makes some UML semantics more specific to Semantic Modeling. The formal semantics of the foundation of UML may be found in the Foundational UML specification. While there are some extensions, every effort is made to use "generic UML" class diagrams, as they are well understood and supported. SMP only provides stereotypes to extend UML to make semantic models more expressive. For example *Roles* and *Phases* are defined to describe how entities may be classified in different context or over time. These extended notions are introduced here, in subsequent sections. Readers are referred to the UML specification and the many books and courses on UML for an in-depth treatment of generic UML.

The subset of UML used for Semantic Modeling is primarily that known as "Class models", the most commonly used part of UML. Our scope further narrows what we utilize to exclude behaviors and methods – elements used for object-oriented design. Those elements may be present, but they are ignored for the purposes of Semantic Modeling.

Logical design models expressed in UML can be related to their meaning in a semantic model. SMP provides specific capabilities to enable these connections.

Expressing semantic models in UML is not a dead-end or "paper architecture", the semantic model expressed in the SMP can be combined with business or technical choices and transformed to a variety of technologies, these include:

- Web Ontology Language (OWL)
- ERGO Logic
- Entity/Relational models
- Schema for SQL, XML, Jason, and other data technologies

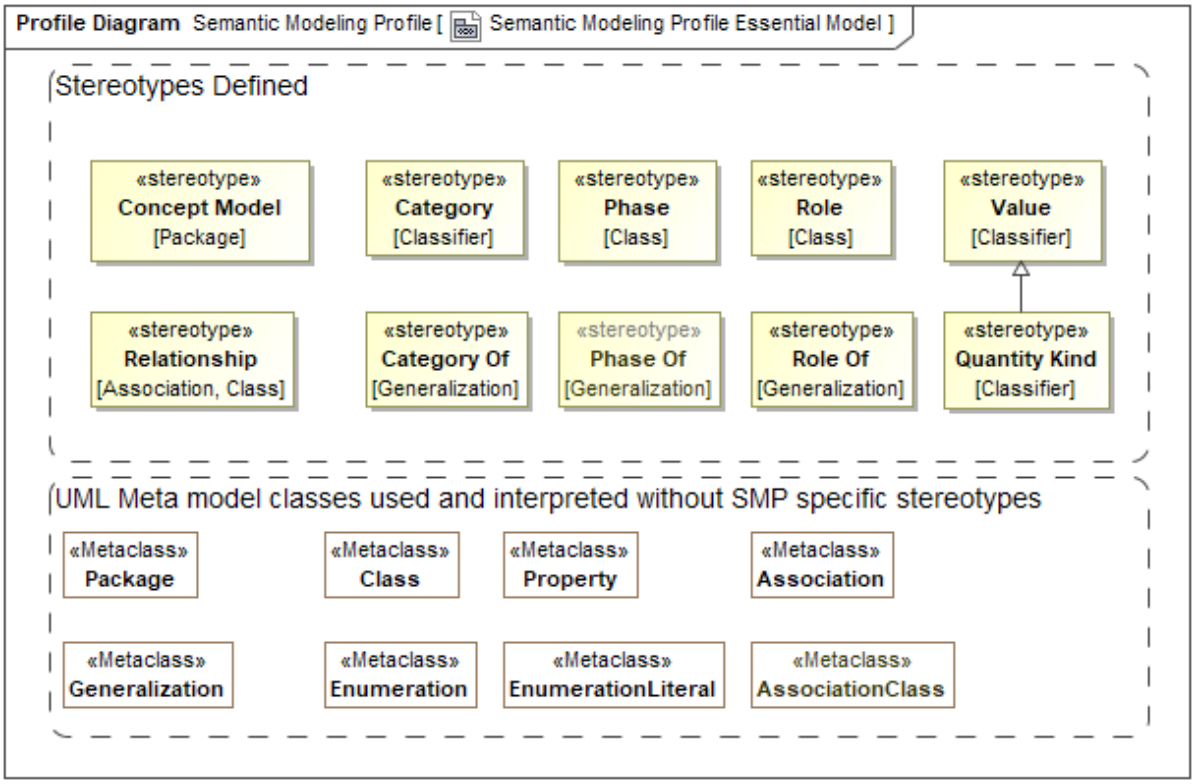
Some tools also provide "reverse engineering" and/or "round trip" capabilities to import existing ontologies into UML based semantic models.

The SMP specification does not include the specification for mapping to other technologies, such mappings build on the SMP. Such mappings may be done manually by a programmer or automated using "Model Driven Architecture" (MDA ®)[2]. A formal specification for mapping to description logic and OWL is in process of being standardized and published at: https://spec.edmcouncil.org/UMLCMP/index.html

[2] MDA: https://www.omg.org/mda/

1.2 Essential Semantic Model

While a semantic model may contain detail and precision to any level required, experience has shown that a very few kinds of concepts are used to define the essential[3] (or foundational) business meaning and structure of a semantic model.



The following are the fundamental elements of any essential model:

- Models and sub-models as the definitional scope for concepts, each model representing a subject area of interest to stakeholders. The set of such models, connected and related, define a "lattice" of models that, together, model a domain. Models are defined as UML packages.
- Precise natural language definitions for concepts that capture stakeholder understanding but are carefully constructed to resolve ambiguity or conflict. Every concept has a definition.
- Hierarchies of the kinds of entities found in a domain, entities can include physical things like people or places as well as "social constructs" like a sale or ownership. Entities are represented as UML classes; no special stereotype is required.
- Hierarchies of roles entities play in various context, roles become the connective tissue between entities and relationships. Roles are defined using a SMP stereotype of a class.
- Hierarchies of "phases" (or states) entities can be in. Phases help capture how things can change over time, such as person that could be a teenager or adult. Another example is an invoice that is paid or unpaid.

[3] Term "Essential Model" suggested by David Hay - https://www.linkedin.com/in/davehaytx

- Hierarchies of Relationships as the way entities and roles are connected. Relationships are defined using UML association and association classes.

The following are help define the vocabulary of the domain so are included in the essential model but may not be used for more abstract essential models.

- Hierarchies of the attributes provide a next-level of detail but may be critically important to a domain. Attributes describe some characteristic or attribute of an entity, role or relationship. Attributes may or may not be considered part of the essential model.
- Hierarchies of categories define other ways stakeholders think about and classify entities.
- Enumerations, also called “code lists” define specific instances that may be important in a domain.
- Values and quantity kinds define the kinds of data that attributes may hold, such as “Weight” , “Length” or “Currency”. Most common quantity kinds and units can be found in pre-defined libraries.

Experience has shown that the fundamental elements: models, definitions, entities, roles, phases, relationships and attributes form the framework on which the additional detail and precision of the semantic model are based. These are the concepts that define the vocabulary of the domain and how the concepts of that vocabulary are connected. With the essential model in place, additional semantic assertions can be made that enhance the precision of that model, help to validate it and provide additional benefits for leveraging automation and inference.

The following sections defines how UML with conceptual extensions is used to represent the foundational network of concepts using classes, associations, and properties. Additional constraints, expressed as rules, are then attached to this basic framework to enhance semantic expression and the ability of automation to federate and analyze information about those concepts.

1.2.1.1 Extending the essential model

The essential model defines the vocabulary of the domain and how the concepts of that vocabulary are connected. With the essential model in place, additional semantic assertions can be made that enhance the precision of that model, help to validate it and provide additional benefits for leveraging automation and inference. The modeling language for expressing these more precise semantics are defined in section 1.3.

1.2.2 Packages Representing Subject Areas

Any realistic enterprise level semantic model will be substantial and can’t be “monolithic”. Models will typically will reuse generic concepts that cross domains. The full semantic model of a domain can be thought of as a “lattice” of interconnected subject areas, each subject area is modeled as a UML package. The package contains a set of definitions for concepts that can be used, related to, or refined in other packages.

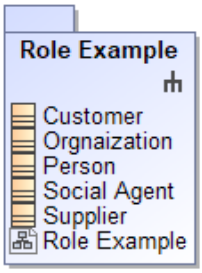


Figure 2 Subject Area Package Example

The above example package “Role Example” contains definitions for concepts such as person and customer. A package is authored under the control of some authority, but may be used or extended by others.

Subject Area packages may also contain other packages.

1.2.3 Classes representing Domain Entities

Classes specify, or classify, a set of things, according to some set of rules or understanding. Classification is the essential mechanism of conceptualization we use. Classes specify a set of things belonging to that class – this is called the class's *extent*. Each element of the class is an *instance* of that class –it is something the class classifies. Classifications may be arranged in hierarchies from general to more specific.

In the UML semantic model, a class is diagrammed as a box with a name at the top. In some cases, a definition is also shown inside or next to the box in a "note" form.

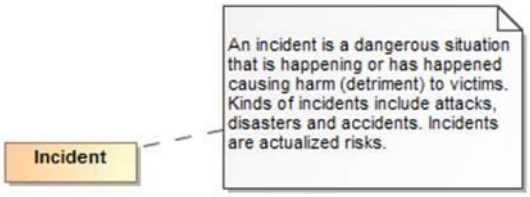


Figure 5 Example of a Class

The above example shows the class "Incident" and its definition. It should be noted that a class is *a way* to classify something. It is natural to classify something multiple ways. For example, we may classify a situation as also being a danger or, to someone else, an opportunity to do harm. This is different from many technology models (e.g. Java) that only allow something to be classified in one way and the classification is fixed. *The basic assumption of a semantic model is that unless specified otherwise, something may be classified in any number of ways and those classifications may change over time*.

1.2.3.1 Instances

While not usually used in the definition of the semantic model, instances can also be shown in UML and are utilized to illustrate examples or to define well known instances, like the "United States of America".

Instances are important in understanding what classes classify. Since the model is conceptual, instances of classes are proxies, or "signs", for the "real thing" in the world – not data about them or other technology artifacts.

Instances are also shown as a box, but have a ":" separating the name of the instance from its classes.

Figure 6 Instance Example

The above example shows a information about an instance named "Joe Smith" that is classified as a "Person" and a "Victim".

1.2.4 Class Generalization

Since Aristotle, classes have been arranged in hierarchies – from most general concepts to more specific ones. In UML this is shown as a Generalization – an arrow with a solid line from the more specific concept to the more general. All semantic elements that are kinds of classes can be generalized, this includes entities, roles, phases and relationships. Attributes can also be generalized, but use a different syntax. The more general class is known as the *Superclass* (or *Supertype*) and the more specific the *Subclass* (or *Subtype*). Generalization has some specific semantic rules:

- Everything that is true about the superclass must be true about all its subclasses
- The extent of the subclass is a subset of the extent of the superclass
- All properties and associations that apply to instances of a class also apply to instances of all its subtypes

In a semantic model, a class may have any number of superclasses or subclasses. In contrast, some technologies (Like XML Schema) limit the number of superclasses to one.

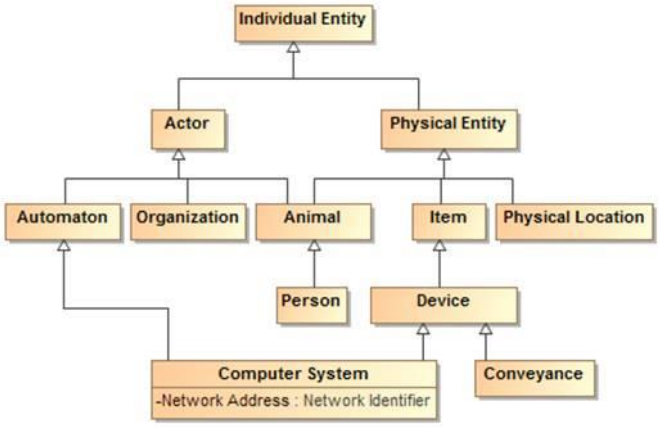


Figure 7 Class Hierarchy Example

The above example shows a class hierarchy with multiple levels.

Note that all properties and associations defined for all superclasses of a class apply to that class. For that reason a complete understanding of a class and its potential properties must include such superclasses.

A generalization is a subsumption relationship between a more general class and a more specific class. Subsumption means that every instance of the specific class is also an instance of the subsuming general class. Because of this subsumption relationship, the specific class inherits all of the conditions and capabilities of the more general class.

For a simple example, if we define “Futsal Team” as a subclass of “Soccer Team”, then the set of individuals in “Futsal Team” must be a subset of the set of individuals in “Soccer Team”.

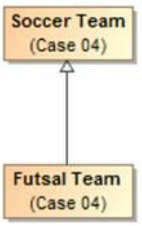


Figure 8 Simple Generalization Example

There are four variations on generalization described in the following subsections. The first variation corresponds to the example above: overlapping and incomplete subclasses. That variation is the default in both UML and Semantic Modeling.

1.2.5 UML Associations Representing Domain Relationships

Relationships (modeled in UML using “associations”) describe facts about how entities are related. Relationships are shown as lines between the classes that have related instances. At each end of the line is an

"association end" property – the association end describes how the instances of the class on the far end relate to those of the near end. If there are constraints to how many instances may be related, these are also shown. Since an association has at least two ends, the association may be read in any direction, but is the same "fact". The association end properties are typically verbs or verb phrases, but in some cases noun phrases may be more appropriate based on the modeling style being uses In either case the name denotes the intent of the class *at the other end of the line*.

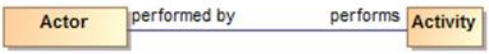


Figure 19 Association Example

The above example says that there are relations between actors and activities such that the *actor performs the activity* and the activity is performed by the actor. These are considered two ways to "read" the same fact. Like any fact, relations may be true for some period of time or in some specific situation.

This combination of classes and associations with ends forms the basis for nouns and verbs common to human language. The terms used for the nouns and verbs should be both consistent with their semantics and resonate with stakeholders – sometimes this is a bit of a challenge.

In some cases the ends of the relation are sufficient to define it, in other cases it makes more sense to give the association a name and its own definition. Associations and association ends, like classes, can be part of a hierarchy.

1.2.6 Association Classes as "First Class" Relationships

In a semantic model any "fact" may have properties and may participate in other relationships. Of particular importance is the "provenance" of the fact – where the fact came from and thus how much it can be trusted. Facts can also be time-bound, true for some period or only valid within some context. Where an association may have additional specific properties or may participate in other relationships, an "association class" is used. As implied by its name, an association class has both the properties of an association and the properties of a class. More complex associations between things use association classes. An association class is diagrammed as an association line and a class box with a dashed line between the association line and its class. While the association line and box may seem somewhat visually distinct – they are the "same concept".

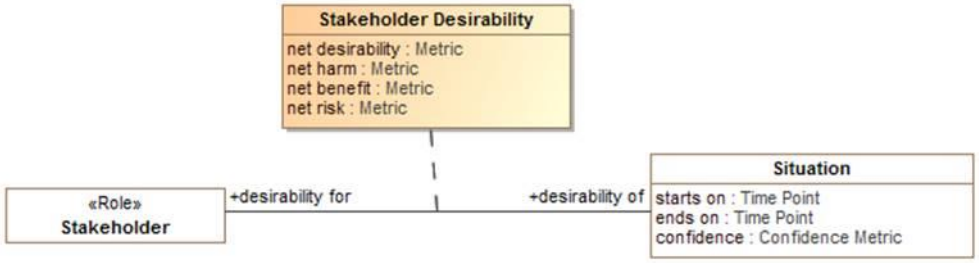


Figure 21 Association Class Example

The above example shows the "Stakeholder Desirability" relation. Between any situation and any stakeholder there can be some metrics as to how much that stakeholder desires or wants to avoid that situation. The Stakeholder Desirability association class represents these as properties of the association: net desirability, net harm, net benefit and net risk – which can all be positive or negative reflecting a benefit or harm, respectively.

1.2.7 Attribute Properties

Properties represent attributes or qualities inherent in something, such as size, weight or a time. Each property has a "subject", the kind of thing that property can be a property of and a "type" for the kind of value that

represents that quality. A property is a characteristic that an individual can have, or, as explained in a subsequent section, an individual *must* have to qualify as a particular kind of thing.

Most properties are attributes of entities, roles or relationships and have some kind of value. The term for a property is usually expressed as a verb phrase, such as "Vehicle *has weight* Mass" or "Geographic Region *identified by* Address". Some methodologies use noun phrases for attribute property terms, such as “birth date”.

Fundamental to the semantics of any attribute is the kind of thing that can have to characteristic defined by the attribute, this is sometimes called the attributes “domain”. Also fundamental is the type of value the attribute can have, sometimes called its “type” or “range”. A very few attributes may not have any specific domain or range – these attributes are defined using an <<Anything>> type (see below). Semantic models do not assume that all attributes must be defined as part of the initial definition of a class or that all defined attributes are required for all application purposes or schema. Note that some methodologies initially define attributes without domains and ranges and then restrict them for a particular use – we allow but do not recommend this pattern as it fails to capture fundamental semantics of the attributes.

Most properties are relations with value types, such as a point in time or a mass (the use of primitive data types like int and string is discouraged in semantic models), usually expressed as a prepositional phrase, such as "Person *born on* Date" or a noun phrase, such as “Person *birth date* Time Point”.

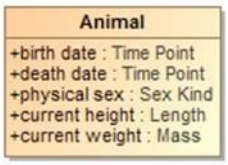


Figure 18 Example of Properties

The above example shows that an animal has the qualities of birthdate, death date, physical sex, height and weight. Note that these is no assumption that these qualities may be known, required or that different data sources may or may not agree on them – just that an animal has these qualities. Instances of properties are facts about the entity they describe. In semantic models, attributes are only used for qualities, never to relate different entities.

The type of an attribute is normally a data value of some kind, such as length or weight, attributes do not normally have entities as their types, but this is not prevented to allow for variation in modeling styles.

Attribute properties may be diagrammed as a property within a class or at the end of an association that is only navigable on one way (using UML “Navigable”). Where an attribute property is defined in the same scope as the class it may be defined within the box. Where an attribute is being added in another model it must be shown as an association.

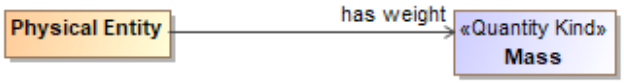


Figure 3 Example of property defined using an association

A much smaller number of properties represent metadata, usually expressed as a noun phrase, such as "Anything *description* String" or "Anything *see also* URI". To represent metadata, this profile provides a stereotype called «Annotation Property» that can be applied to a standard UML property in a semantic model.

1.2.8 Roles

Roles are facet classes that are expected to be dynamic and contextual, such as teacher, victim or president. A role is defined using a class with the <<Role>> stereotype and, optionally, a <<Role Of>> generalization.

Implementation technologies should interpret roles as classifications that may be added to or removed from an instance over time and may be defined in a particular context. A role is usually required to be a role of some particular other class, for example a teacher is expected to be a role of a person (at least until a computer takes her job). The constraint of what a role must be a role of is defined using a <<Role Of>> stereotype of a generalization. A role will also frequently have a relationship with something where the multiplicity is at least one. For example, a person is a parent if they have at least one child.

Many implementation languages don't have the capacity to represent roles, so roles are sometimes implemented is the single and unchangeable "type" of a class or DBMS table. The problem with this is that the same individual may not be connected across all their roles. Specifically representing roles allows the same individual to play multiple roles and for these roles to change – this better reflects the reality of the world and the way we think about it.

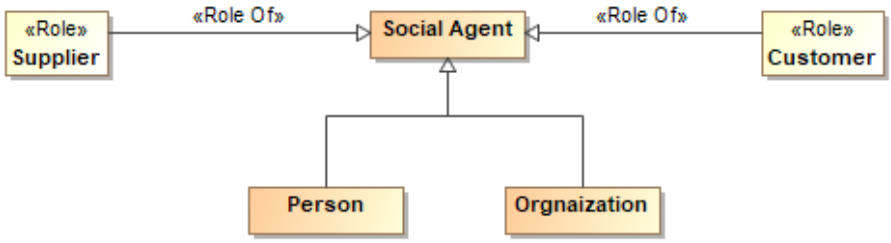


Figure 22 Role Example

The above example shows that a "social agent" can be a person or organization and that either could be classified as being able to play the Supplier or Customer roles.

Roles help to decouple concepts in models and specifically allow an instance to "play" multiple roles at the same time or over time. Roles, when combined with quantification constraints, clearly define the semantics of roles. For example, we could say that a victim must be a victim of some incident and an owner must own something.

There are various implementation patterns for roles; SMP does not define a specific implementation pattern, such choices will be based on target technology and application requirements. Examples of such implementation patterns include defining a separate technology object for each instance of an entity playing a role, defining roles in separate graphs or using multiple classification.

Roles may also be differentiated as being a role inherent in an entity (E.g. "Joe is a pilot") and roles with respect to some relationship (E.g. "Joe is a pilot of flight UA940"). "with respect to" will be explained in the next section.

1.2.9 Phases

Phases are classes that are expected to classify an instance over a specific span of time or temporal condition, such as a teenager, "Adult" or "Paid Invoice". A teenager is a person between the ages of 13 and 19 (inclusive) – perhaps "Adult" is of age 20 or older. Phase may be considered a synonym for the "State" of something.

A phase is defined as a class with the <<Phase>> stereotype. Phases use the <<Phase Of>> stereotype of a generalization to define what a phase must be a phase of.

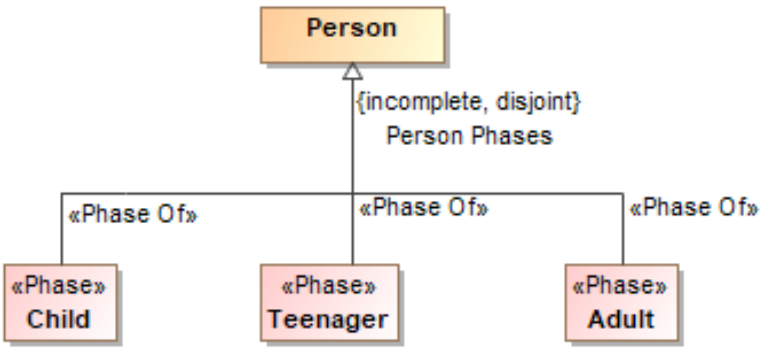


Figure 23 Phases of a person

Like roles, phases help to decouple concepts in models and specifically allow an instance to "be in" multiple phases (or multiple roles) at the same time or over time. If an instance cannot be in two phases at the same time or be in a role and a phase a "disjoint with" constraint can be used to state that restriction. In the example above, the phases are disjoint.

1.2.10 Example Essential Model

The following is a small example of a model utilizing all of the above concepts.

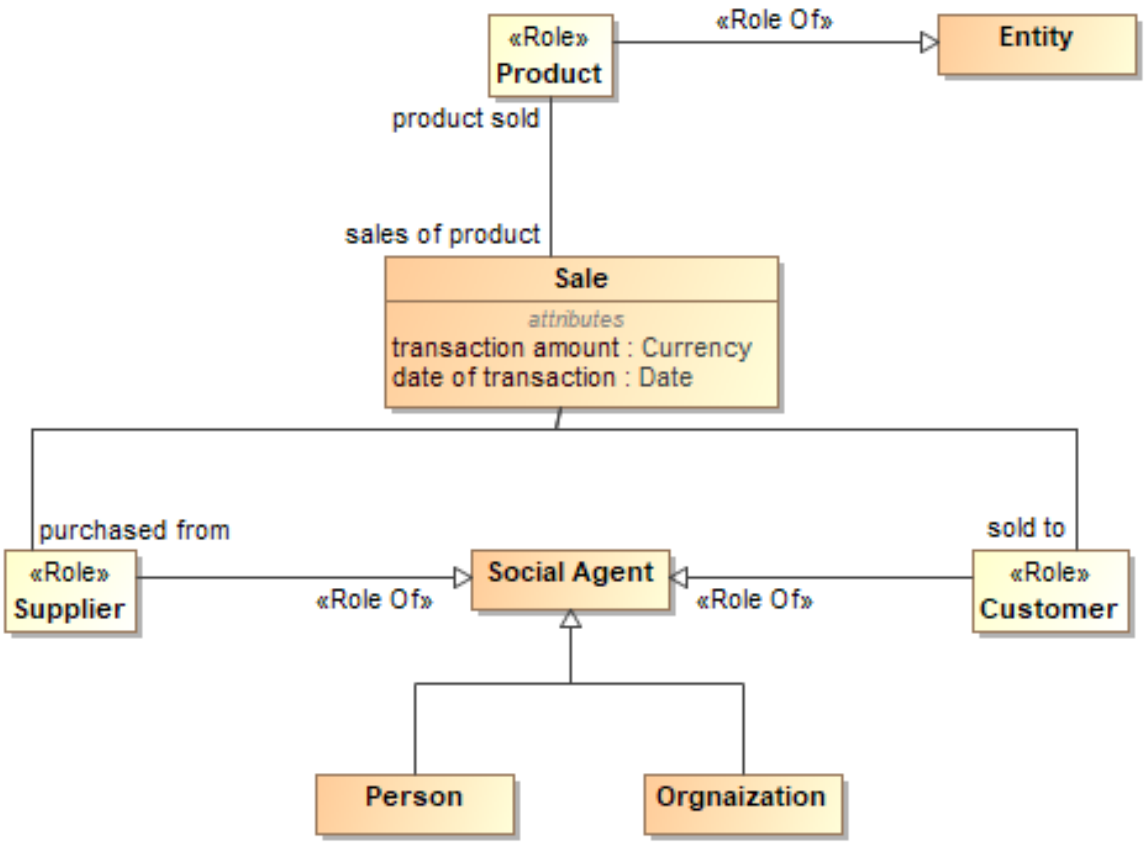


Figure 4 Example Essential Model

The above is an example essential model of a simple sale. A sale is made between a "Supplier" and a "Customer", roles that can be played by people or organizations. Each sale has a transaction amount, date and a set of products sole, where "Product" is a role of any entity.

Such an essential model should define the vocabulary of the domain and each concept should have a vetted business definition.

The essential model should be used as the foundation for all documentation and discussion about the domain and any systems in the domain. If it doesn't work for those purposes, the model should be revised.

1.2.11 What semantic models don't assume

Any application, database or inference engine makes some assumptions, or "commitments" about data when applied to the model. These assumptions may differ for various use cases or technologies. For this reason a semantic model is silent on these assumptions and leaves such choices up to the implementing systems.

- **Open world or closed world** – the "open world assumption" (e.g. OWL) is that a model or dataset is incomplete, this impacts what can and can not assumed, for example in an open world assumption the lack of a customer in a data set is insufficient to assume the entity is not a customer. It is therefore o.k. to assume a customer exists if it makes the data consistent. A closed worlds assumption assumes data

sets are complete for the purposes of any query or computation. The SMP does not commit to either assumption.

- **Constraint enforcement or proof theory** – Where a constraint is stated in a model, for example that only people have social security numbers, a constraint system (e.g. a SQL edit) will not allow incorrect data to be entered. A proof theory logic (e.g. OWL) will infer that anything (e.g. a Dog) must also be a person if assigned a SSN unless Dog and Person have explicitly been made disjoint. The SMP does not commit to either assumption.
- **Unique Name Assumption –** The unique name assumption is that things with different identifiers are different things (e.g. a primary key in SQL), the opposite (e.g. OWL) is that things with different identifiers may be assumed to be the same thing as a result of proof theory. The SMP does not commit to either assumption.
- **Single Classification –** Most implementation technologies that support inheritance (e.g. Java) only allow a class to have one supertype (single inheritance) and only allow an individual instance to have one type (single classification). Semantic models allow both multiple inheritance and multiple classification. Note that most ontology languages do support multiple inheritance and multiple classification.
- **First order –** A "first order" logic (e.g. OWL) does not allow statements to be made about other statements, rules or context. First order restrictions are introduced to make automation of inference engines more practical. Semantic models and "higher order logics" do not introduce first order constraints so that "real world" concepts like context, time, rules, obligations, permissions and policies can be represented.

The above "lack of commitment" allows SMP to be used to more directly model the domain, without being distorted by implementation concerns. When a semantic model is mapped to a specific technology or logic some or all of the above assumptions may be made and appropriate patterns applied to implement the commitment.

1.3 Extended Semantic Precision of an Essential Model

The essential model defines the concepts and vocabulary of a domain that has direct value. However, the meaning of concepts in natural language definitions is not fully captured in model semantics. Improving the semantic precision has a number of benefits:

- Models and data can be validation for consistency
- Business friendly interpretations of the model semantics can aid in stakeholder understanding
- Logical "inference engines" and rule based systems can leverage semantics to infer new knowledge
- Automation can more accurately "forward engineer" implementation from the semantic models

The following sections define the modeling language for semantic precision.

1.3.1 Class Semantics

Generalization semantics applies to all concepts based on classes and associations, this includes: Entities, Roles, Phases, and Relationships.

1.3.1.1 Overlapping and Incomplete Subclasses

This variation is the default in Semantic Modeling. An instance can be a member of the superclass and / or any number of subclasses. In this sense, the classification of instances is "incomplete"—sometimes an instance is classified by one or more specific subclasses, and sometimes it is not.

For example, the diagram below shows four instances (represented by white diamonds). One is an instance of "Manufacturer", one is an instance of "Windshield Manufacturer", one is an instance of "Car Manufacturer", and one is an instance of both "Windshield Manufacturer" and "Car Manufacturer".

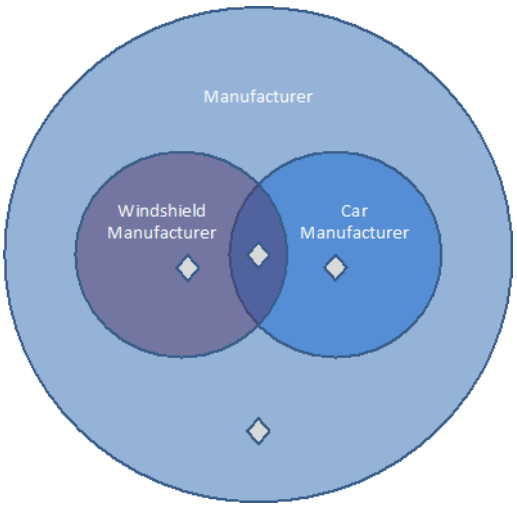


Figure 9 An example of incomplete subclasses

In both standard UML and in Semantic Modeling, incomplete and overlapping subclasses are shown with either no notation, or with the equivalent notation {incomplete, overlapping} near the generalization arrow.

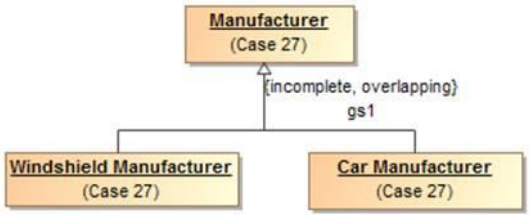


Figure 10 Incomplete and overlapping subclasses in standard UML notation

1.3.1.2 Disjoint and Incomplete Subclasses

This variation means that an instance can only be classified by at most one of the disjoint classes. Disjoint classes cannot have any overlap in their instances.

The diagram below shows three instances. One is an instance of "Cat", one is an instance of "Dog", and one is an instance of "Animal". An instance classified as both "Cat" and "Dog" is impossible because there is no overlap between the two classes. In the most basic terms, an instance of a "Cat" cannot be an instance of a "Dog", and vice versa.

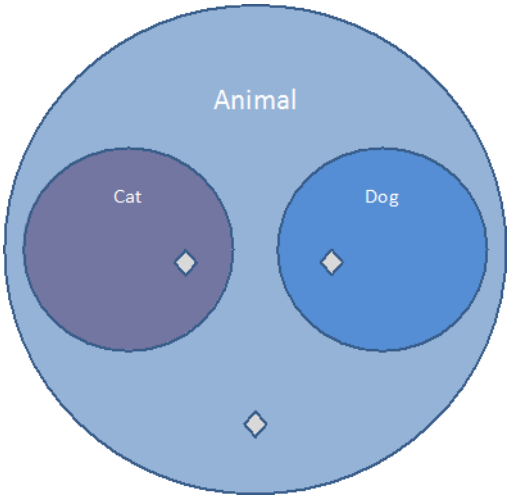


Figure 11 Disjoint Subclasses

The following diagram shows an example of disjoint subclasses in standard UML notation. It shows that "Dog", "Cat", and "Mouse" are all subclasses of "Animal". In addition, the standard UML {incomplete, disjoint} notation declares all of the subclasses to be incomplete and disjoint. Intuitively, an instance of the subclass "Dog" is an instance of the superclass "Animal", but it cannot also be an instance of the "Cat" or "Mouse" subclasses. It is incomplete because there can be many more kinds of animals.

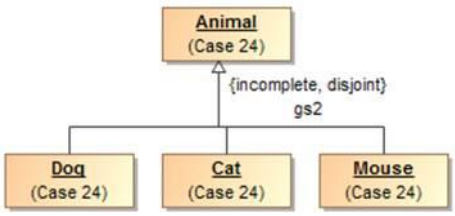
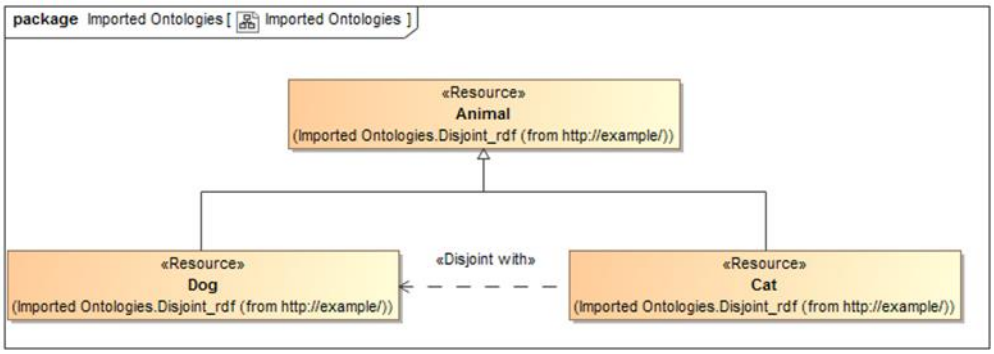


Figure 12 Incomplete and disjoint subclasses in standard UML notation

The profile also supports a dependency stereotyped as «Disjoint With» to specify that anything can be disjoint, even if they are not subclasses of a common super type. disjoint subclasses. For example, the class Animal has three disjoint subclasses, Cat and Dog.



[CC7]

Figure 13 Alternative «Disjoint With» Stereotype

1.3.1.3 Complete and Overlapping Subclasses

This variation means that an instance can only be classified by at least one of the subclasses; it cannot be classified by only the superclass. Keep in mind that an instance of a subclass is indirectly an instance of a superclass at the same time.

For example, the following diagram shows 7 instances. As various combinations of land, sea, air and space vehicles (and there can be more combinations) – but every vehicle must be at least one of these classes.

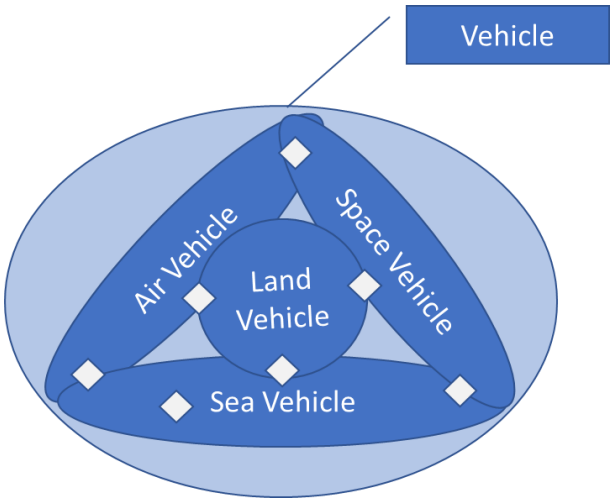


Figure 14 An example of complete overlapping subclasses

The diagram below shows an example of complete and overlapping subclasses in standard UML notation.

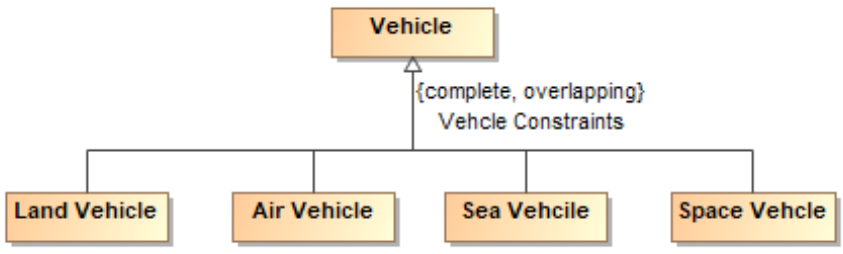


Figure 15 Complete & overlapping subclasses in standard UML notation

1.3.1.4 Disjoint and Complete Subclasses

This variation means that an instance can only be classified by one of the subclasses. The instance cannot be classified as only the superclass, and it cannot be classified by two subclasses at the same time.

For example, in the subsequent diagram, two instances are shown. One is an instance of “Person”, and one is an instance of “Organization”. There can be no instance of “Social Agent” that is not also an instance of one of the subclasses, and there can be no instance that is classified as both a “Person” and a “Organization” at the same time.

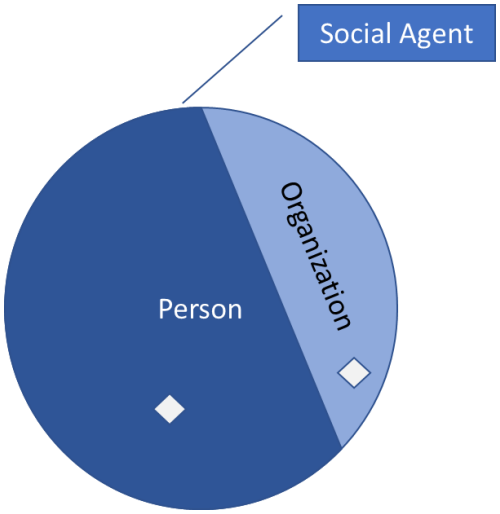


Figure 16 Disjoint and complete instances

The diagram below shows an example of disjoint and complete subclasses in standard UML notation. The diagram shows that “Person” and “Organization” are subclasses of “Social Agent”. In addition, the standard UML {complete, disjoint} notation declares that the subclasses are complete and disjoint.

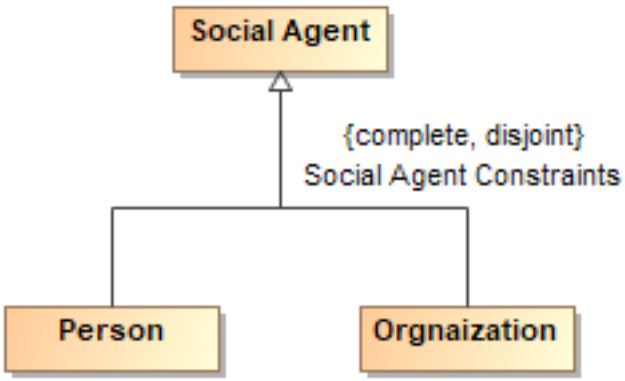


Figure 17 Disjoint and complete subclasses in standard UML notation

1.3.2 Kinds of classes

There are additional Semantic Modeling specific stereotypes documented in the reference section that further define the semantics of a class. Some of these stereotypes are very important for understanding the semantic model and are further explained here. These are roles, phases and quantity kinds.

1.3.2.1 Anything

The stereotype «Anything» can be applied to any class to make it special[CC16] . Every such special class is equivalent to one topmost class (⊤) of which all other classes are subclasses. Thus, a property of a class marked as «Anything» is inherited by all subclasses. In addition, while the name of a such a marked class is irrelevant, consistently naming such classes "Anything" in all semantic models avoids any confusion with normal classes.

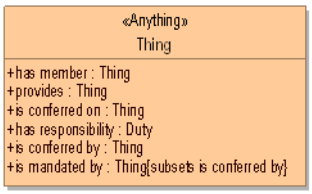


Figure 22 «Anything» Example

The Semantic Modeling foundation model provides a default "Thing" that can be used or extended in any model (note: to add properties to the default "Thing", one-way associations must be used).

1.3.2.2 Union

A «Union» is a class that has an extent (set of instances) which is equivalent to the union of the extents of all types that specialize the Union (Subclasses). Specializing types shall include subtypes and types that realize the union. The union can be either named or unnamed. When it is unnamed, it can only be used at the domain or range of a property.[CC17]

Note: UML realizations are included to support unions across external models because UML generalization cannot be used across external models due to the ownership of generalization.

Unions may be named, but the name is optional. Unnamed unions are considered "anonymous". Some methodologies do not permit anonymous classes but the profile does not prevent it.

The following diagram states that an instance of a Person may have a value of type Cat or Dog for the *cares for* property. The diagram also states that an instance of a Cat or a Dog may have a value of type Person for the *cared for by* property.

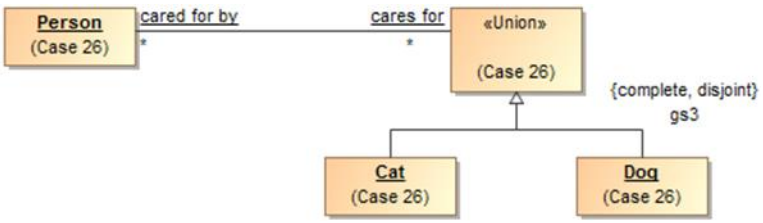


Figure 23 An anonymous union class

1.3.2.3 Intersection

An «Intersection» is a class that has an extent (set of instances) equivalent to the intersection of the extents of all supertypes (remember that the extend of a class is all of its instances). Intersection is a stronger statement than a subtype, as a subtype may be a subset of the intersection. An instance of all the supertypes implies an instance is also an instance of the intersection type.

For intersection, The Semantic Modeling profile considers UML generalization and UML realization equivalent. This is due to ownership and legacy considerations in UML. Generalization is the preferred representation.

Fundamental to understanding and describing something is physical and other qualities such as temperature, length and color. Many data models fail to capture units of measure explicitly which can and has[1] resulted in

dramatic systems failures. A concept for somethings weight should properly be typed by a measure of weight, not an “int” or “real” – which are just ways to represent numbers without knowing what they mean. Of course there needs to be numbers, but in relation to their units.

In that there are different units that can represent the same kind of measure, such as degrees Celsius and degrees Fahrenheit can represent the same temperature – an abstraction is used above like units. The abstraction for a measurable unit is called a <<Quantity Kind>>. Examples of quantity kinds include Length, mass, temperature, frequency, etc.

As any element of measurement data must be specific to a specific unit in a specific data exchange, the <<UnitType>> stereotype is used to define a unit for a quantity kind. A <<Represents>> stereotype of generalization (Diagrammed as a green arrow) is used to say that the unit represents the quantity kind.

In the example above, the “Area” quantity kind (indicated by a black shaded class) can be represented by (the green lines) “Square Meter”, “Square Feet” or an “Acre”. One unit may be nominated as the “Base Unit” and will be used to express conversion factors between the units. As per SI specifications, the Square Meter is the base unit.

By convention quantity kinds are used in fully conceptual models whereas units are used in data models. The “Animal” example shows quantity kinds being used to define properties of animals.

1.3.3 Equivalent Class

An «Equivalent Class» stereotype applied to a generalization can specify equivalence between two classes. Class equivalence expresses a generalization relationship stereotyped as **«Equivalent Class»**. Tools may optionally draw this with a double-headed arrow.

The following figure shows two equivalent classes in a diagram.

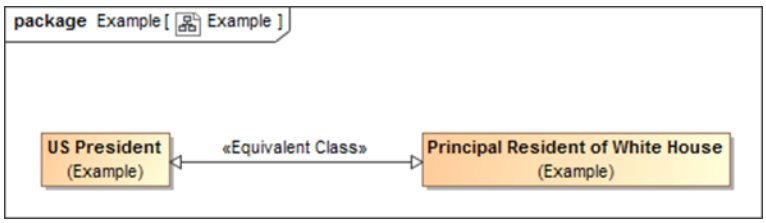


Figure 36 Two Equivalent Classes in the Semantic modeler

In the example, the equivalence class arrow defines that the two classes are semantically equivalent to each other. Equivalent class is particularly useful for “joining” independently conceived semantic models.

1.3.4 Property Hierarchies

Like class hierarchies, attributes and association ends (we will just call both properties from now on) can also be arranged in hierarchies of more or less specific properties. In UML, property hierarchies are represented using either UML “Subsets” or “Redefines” constraints. A property subsets or redefines is shown next to its name in in the diagram (Note that by convention this is not shown on essential diagrams, only the primary definition of the property). If a property completely subsumes the other in a particular context it uses a “Redefines” – that is the redefining and redefined properties have the same set of values. If the more general concept can also be used in the context a “Subsets” is used.

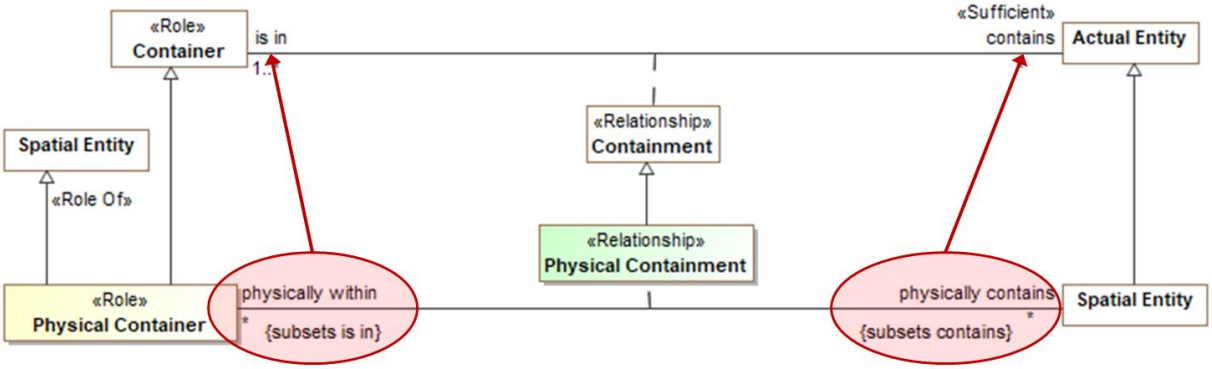
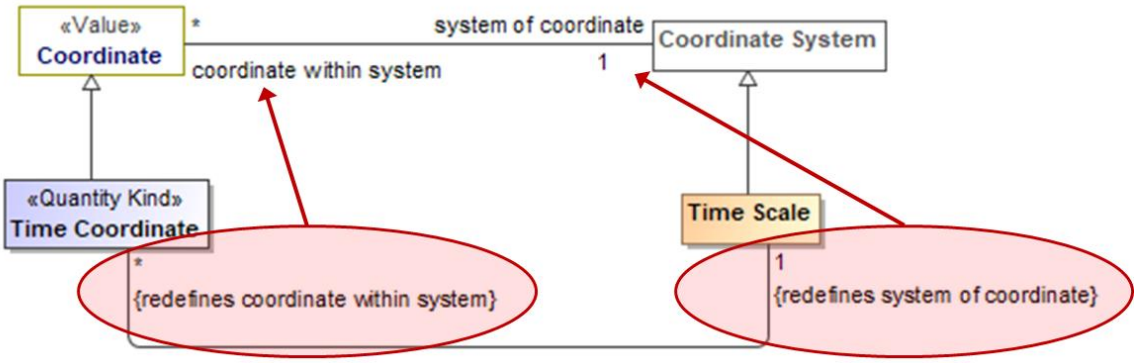


Figure 20 Example of Association End Hierarchy using Subsets

The above example shows that the "physically within" and "physically contains" properties are specializations of the "is in" and "contains" concepts, respectively. Since "Subsets" is used, "is in" and "contains" may also be used, perhaps for other perspectives.



The above example uses "redefines" to redefine "coordinate within system" to only apply to a "Time Coordinate" with respect to a "Time Scale" and, likewise, "system of coordinate" may only apply to a Time Scale" with respect to a "Time Coordinate".

If meaningful for the domain, new terms may have been defined for the redefined properties. Where a redefined or subset property has no name, it is an indication that the property type and/or multiplicity is merely constrained in some way. No new properties or associations are actually defined for a restriction (more on this below).

1.3.5 Annotation

This profile provides a way to comment on any element using *annotations*. One can annotate classes, properties, and models using an open-ended system of *annotation properties*. An annotation property defines information

about the model (metadata), not about the subject domain. A property can be made an *annotation property* using the «Annotation Property» stereotype on a UML property[CC14] .

Every «Annotation» is a textual value for an «Annotation Property». An annotation describes some subject using an annotation property and a (usually textual) value. An annotation should specify a tagged value called "value for" that refers to an «Annotation Property».

For example, the following diagram illustrates several UML comments stereotyped with «Annotation»

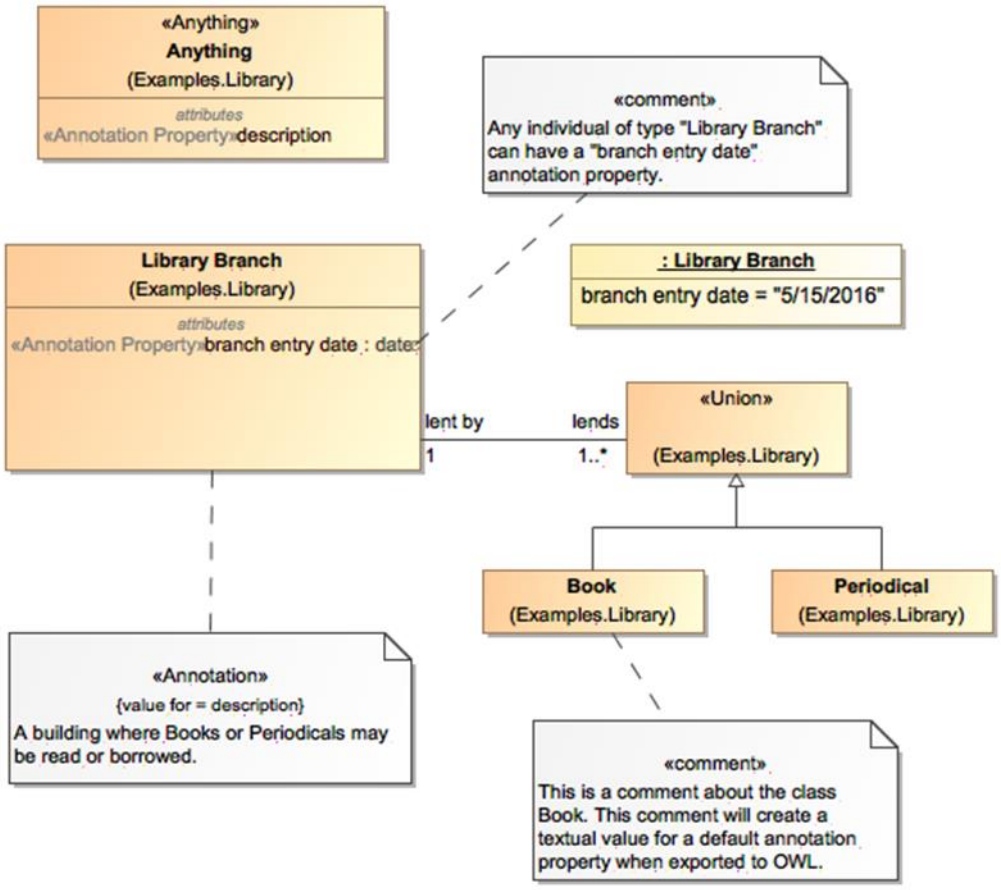


Figure 22 Annotation Examples[CC15]

1.3.6 Assertions about concepts

Above we defined the network of essential concepts as classes, relationships and properties. Additional assertions may be made about those concepts using both UML foundational and extended profile capabilities. The following define the kinds of assertions that can be made. Note that the term "property" applies to both simple properties and the ends of associations.

1.3.6.1 Property Ownership

The Semantic Modeling profile of UML interprets the owner (defining class) of a property *definition* as the subject of that property (its domain) and the context in which that property must conform to certain constraints.

Constraints may be placed on a property. These constraints can include multiplicity, which includes a minimum cardinality and a maximum cardinality, a type for the property, existential quantification, and universal quantification. [CC22] When an instance is a member of a class, all of that class' constraints must be met.

Property ownership is not interpreted as “slots” in an object. Property values may or may not be independent of the instance that defined them, thus supporting an OWL/RDF, or “open world”, interpretation of properties and associations.

1.3.6.2 Cardinality

Cardinality defines how many value of a property may exist for a particular subject instance. For example, how many ages can a person have? The obvious answer is that a person can have at most one age at any one point in time. Thus cardinalities represent the number of instances at any one time – regardless of how it is represented.

UML allows the cardinality of a property to be left unspecified, in which case it defaults to 1..1. The Semantic Modeling profile interprets unspecified cardinalities as 1 (one) based on UML defaults. Note that conceptual models do define what you may or must know or what the requirements of a data model are – they define what must be true about the world as it is conceived.

1.3.7 Constraining properties and associations

A cardinality of one or more defined for a property requires that an instance of the related element must exist for an instance of the domain (owning class) of that property or association end to be valid. For example, a living person must have exactly one living brain. This is known as an *existential quantification* (∃) or qualified constraint in first order logic. Existential quantification is defined using UML cardinality and, potentially, *subsets*.

An existential quantification can be stated for a newly defined property or an existing one. For a newly defined property this is done by simply stating cardinality greater than one. For example, a phone must have at least one button with a “has buttons” association end and a cardinality of “1..*”. When a new property is being defined it is given a name. If an existing property is being constrained (without a new property being defined) it subsets or redefines [CC23] the existing property and does not need a name. In the Semantic Modeling profile of UML, any cardinality requiring one or more creates an existential quantification constraint.

A property is not limited to a minimum and a maximum cardinality (known as multiplicity) for just one type. A property can have a multiplicity for a superclass, while at the same time having a more specific multiplicity for one or more subclasses of that superclass. This type of constraint is an assertion that, among other possible values, the number of values of one of these subclasses is between some minimum and maximum cardinality.

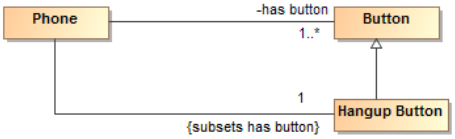


Figure 27 Phone constraint: A phone must have a hangup button

For example, we may say a phone must have one or more buttons with a “has button” property but exactly one of those buttons must be the “hang up button”. We would then define an unnamed property with the type “hang up button” that subsets the “has button” property with a cardinality of 1. If we wanted the Hangup Button to also define a new property, we would give that property a name.

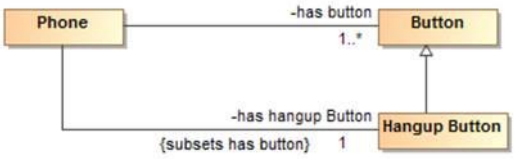


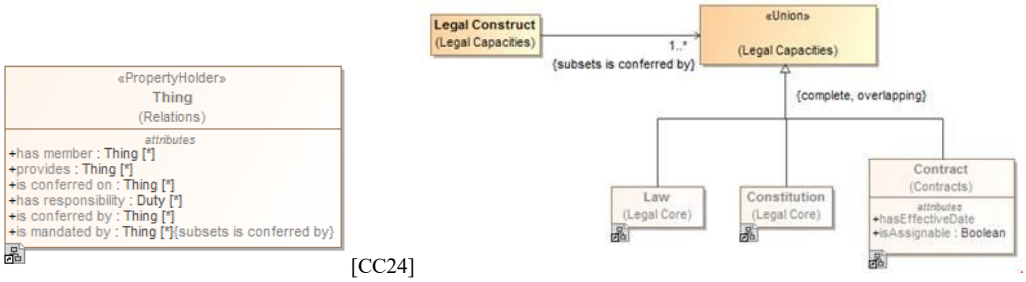
Figure 28 Hangup button with new property

In the Semantic Modeling interpretation of UML, subsetting or redefining a property without giving the new property a different name (or leaving off the new property name altogether) creates a constraint without defining a new property.

As {subsets} or {redefines} with an omitted name is not well defined in UML, in the Semantic Modeling profile it is used to state that a subset of values must meet the stated cardinality and type constraints of the subsetting property. It does not define an instantiable property of the domain, although it does indicate a context in which this constraint holds: the owning class and its subclasses.

The diagram below shows an existential quantification constraint on the global property "is conferred by" (from the Anything "Thing). The multiplicity is such that at least one of the instances of the property constraint must be one of the types in the union.

Note that the property adding the constraint is unnamed. This is equivalent, in this case, to naming this property the same as the property being constrained ("is conferred by" from the Anything "Thing").



[CC24]

Figure 29 Constraining a global property

1.3.8 Tightening a property's type

Sometimes it is necessary, in the context of some class, to constrain *all* the values of a property to a particular type. When defining a new property, the type of that property asserts that all values of that property must be of the given type. This is known as a *universal* quantification or *for-all* constraint (∀) in first order logic. This kind of constraint is an assertion that only values of the specified type are valid, and the number of values must be between some minimum and maximum multiplicity.

Where all values of a property must be of a given type in a specialized property, UML *{redefines}* is used as part of the definition of the property. [CC25] If the redefined property is given a name, a new property with the quantification is defined. If the redefined property does not have a name, the existing property is constrained in the more specialized context (usually a subclass).

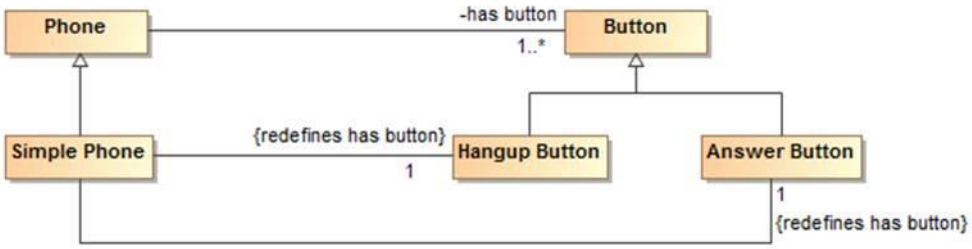


Figure 30 Example of redefines

The example above shows a "simple phone" that has exactly two buttons and they must be an answer button and a hangup button. Since redefines is used, no other buttons are allowed.

The diagram below shows the introduction of a new property "consists of", defining a universal quantification constraint on the property. The constraint states that, in the context of Soccer Team and any of its subclasses, all values of this property must be of the type "Soccer Player" and that there must be between 5 and 11 values of this property.

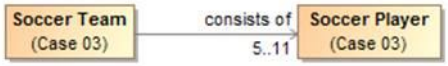


Figure 31 Example of Cardinality range

1.3.9 Inferring a type from its properties

There are also conditions that allow types to be inferred, these are <<Sufficient>> and <<Necessary and Sufficient>>. The semantics of what qualifies a type is defined by SMP but the choice to perform inference is an implementation option outside of the scope of SMP.

The <<Sufficient>> stereotype is used to classify instances based on existing property values and types. A class with at least one <<Sufficient>> condition allows an instance to be classified by the type if (a) the all sufficient properties have at least one value and (b) the instance is classified by all the <<Sufficient>> types and (c) does violate any cardinality constraints. The <<Necessary and Sufficient>> stereotype asserts that the minimum cardinality must be one or more. <<Sufficient>> and <<Necessary and Sufficient>> when applied to a generalization or property with a minimum cardinality of one are equivalent.

Figure 33 Phone example for sufficient

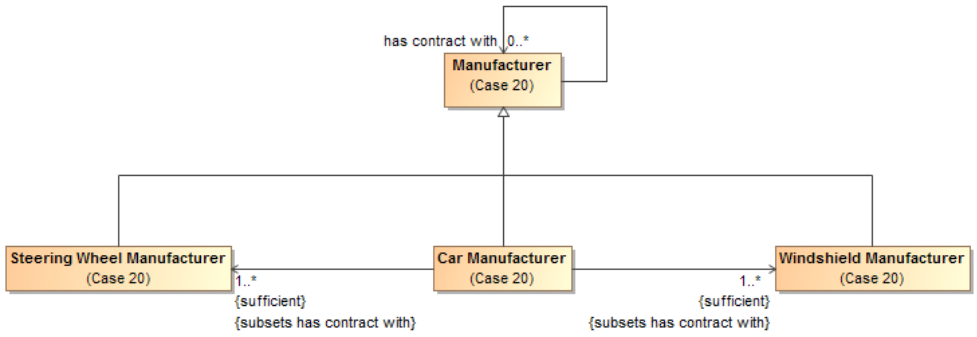
The diagram above defines a phone as *any "electronic giz" that has a hangup button*. The existence of a hangup button is sufficient to know an electronic giz is a phone. Something with a hangup button that was not an electronic giz would not be considered a phone.

In the Semantic Modeling interpretation of UML, a property that has the «Sufficient» stereotype applied to it indicates that when an instance satisfies the multiplicity and type constraints for all the sufficient property's'

values, and has at least one such value, not only is a *necessary* condition for being an instance of the class met, it is a *sufficient* condition.. This necessary and sufficient condition could allow an inferencing engine to classify that instance as a member of the class that owns the property. All necessary and sufficient constrains of the class and all superclasses must be met for an instance's type to be inferred.

In the Semantic Modeling interpretation of UML, a property that has the «Sufficient» stereotype applied to it indicates that when an instance satisfies the multiplicity and type constraints for all the sufficient property's' values, not only is a *necessary* condition for being an instance of the class met, a *sufficient* condition is also met. This necessary and sufficient condition allows an inferencing engine [CC32] to classify that instance as a member of the class with that condition. Once an instance is classified automatically, the conditions on any other properties that have the «Sufficient» stereotype, including those inherited from superclasses, merely become *necessary* conditions the instance must meet to be a *valid* member of the owning class. An instance satisfying the constraints of all the «Sufficient» properties is enough for an inferencing engine to automatically classify an instance.

The diagram below shows that when an instance with the property "has contract with" satisfies specific multiplicity ("1..*") and type constraints (of type 'Steering Wheel Manufacturer" and "Windshield Manufacturer") for the property's values, the instance meets necessary and sufficient conditions to be a member of the class "Car Manufacturer". Therefore, an inferencing engine [CC33] would classify this as an instance of the class "Car Manufacturer". As discussed above, an instance meeting all of these necessary and sufficient conditions is enough to classify the instance. The conditions on the values of these properties become necessary conditions on an instance for it to be a valid member of class "Car Manufacturer."[CC34] Also, an instance meeting all of the necessary and sufficient conditions is enough to distinguish instances of the class "Car Manufacturer' from its parent class "Manufacturer."



[CC35]

Figure 35 An example of necessary and sufficient condition

1.3.10 Property Chain

A property chain is useful for composing a property from two or more other properties that are put together in a chain[CC36] . It defines the property with reference to the other properties. The property chain allows you to navigate from a starting property (the one with the stereotype «Property Chain») through a chain of properties that take a path through the same or multiple other classes.

A property chain is an ordered list of linked properties that determine the value of the stereotyped property.

The following example describes a Person class that has two subclasses "Female Person" and "Male Person", and four properties "has parent", "has father", "has uncle", and "has brother". The stereotype of the property "has uncle" will be **«Property Chain»,** and the tagged value is **chain = has father, has brother**. (Note that the «Property Chain» stereotype is suppressed in this diagram, but the tagged values are not.)

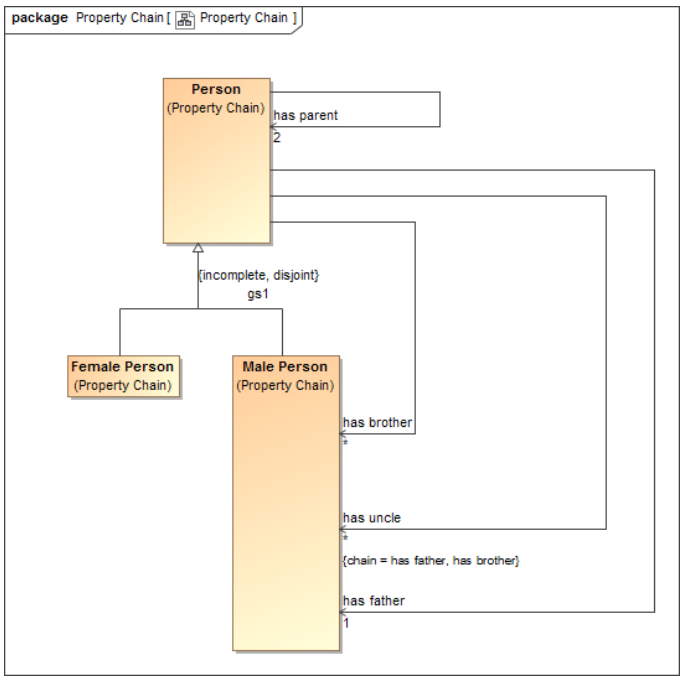


Figure 34 Property Chain Example

1.3.11 Equivalent Property

An «Equivalent Property» allows you to represent properties that have the same meaning. You can make a property equivalent to two or more other properties by applying the stereotype **«Equivalent Property»** to the referenced properties and the tagged value "**equivalent to**" the equivalent properties.

The following figure shows the equivalent properties in a diagram Bad Example .

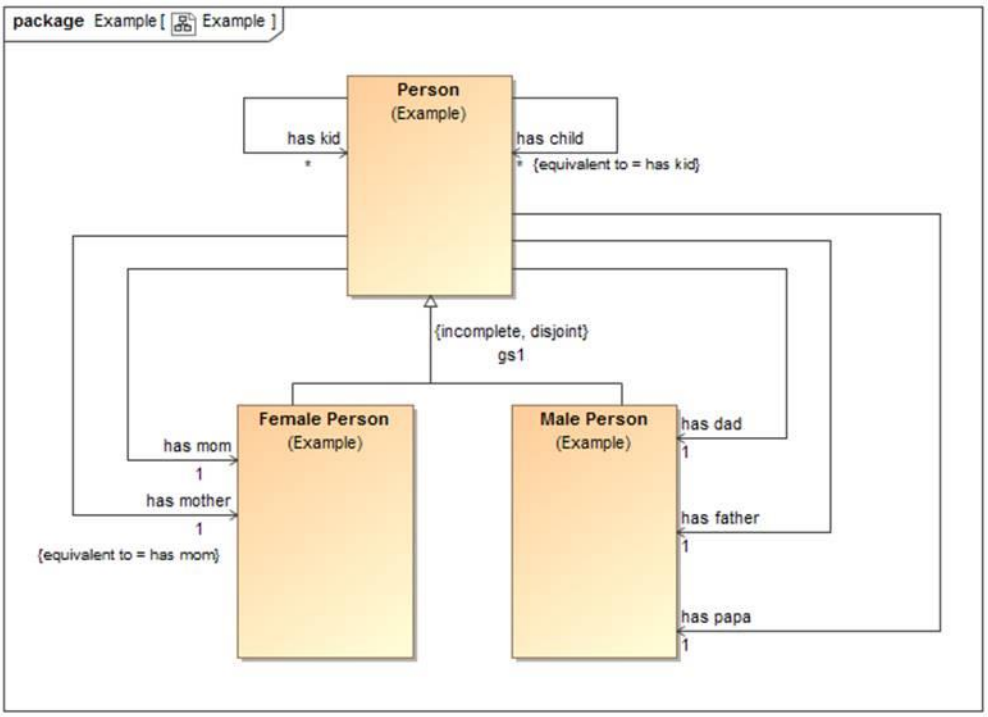


Figure 35 Equivalent properties Example

In the example, the property “has mother” is equivalent to the property “has mom”.

1.4 Semantic Modeling Profile (SMP) Reference

The Semantic Modeling profile defines the Semantic Modeling capabilities of Semantic Modeling in UML.

1.4.1 Diagram: Semantic Modeling Profile (SMP)

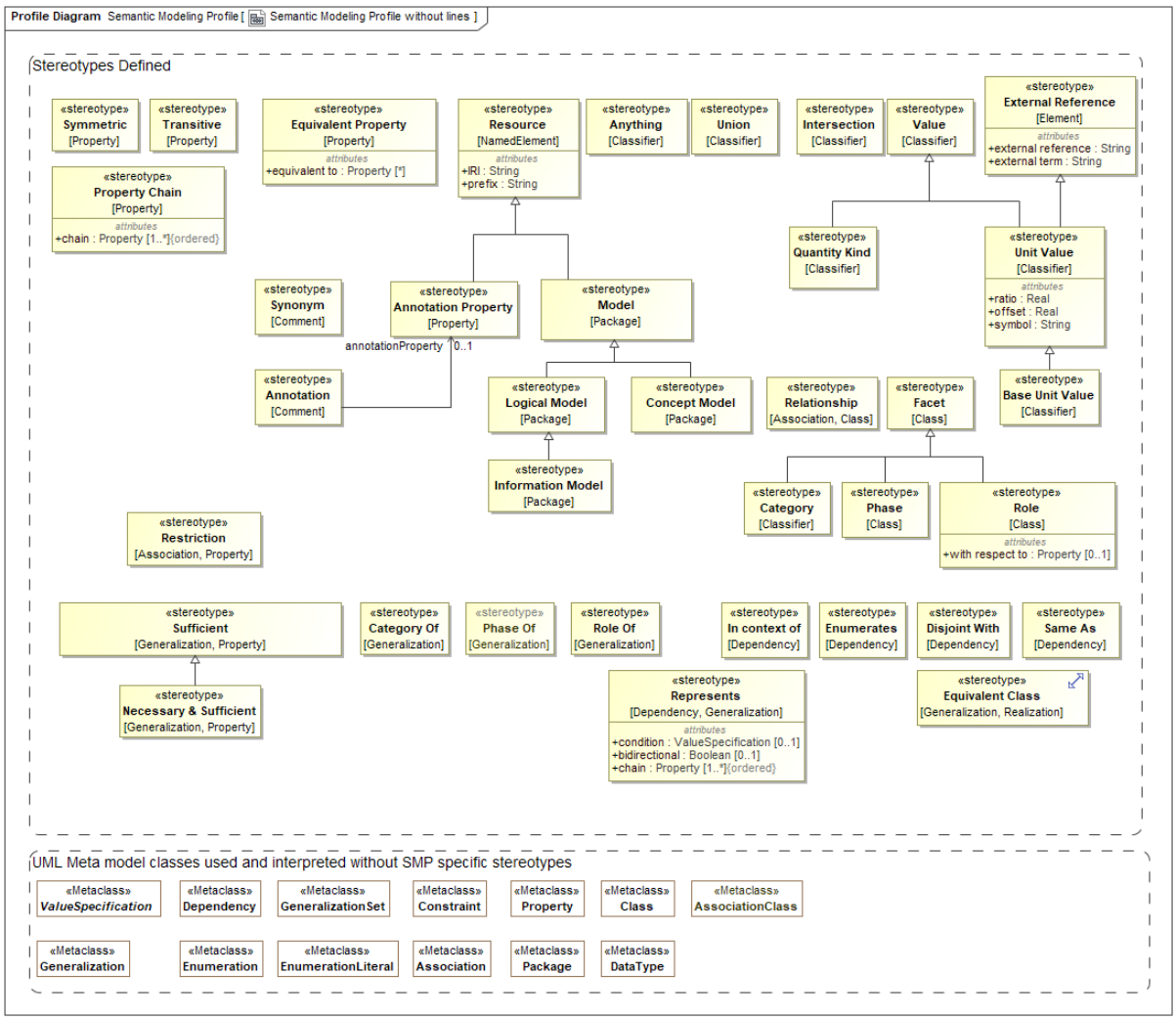


Figure 1 SMP Semantic Modeling Profile

1.4.2 Stereotype Annotation

An <<Annotation>> comment provides a textual "body" as a "value for" one <<Annotation Property>> describing the annotatedProperty(s).

Base Classes

- Comment

Tag Definitions

- value for : Annotation Property [1]

  <value for> is the property for which the <<Annotation>> is providing a value.

1.4.3 1.2.3 Stereotype Annotation Property

An <<Annotation Property>> is a kind of <<Resource>> that asserts a property represents metadata rather than assertions about the subject domain.

Base Classes

- Property

Direct Supertypes

- Resource

1.4.4 1.2.4 Stereotype Anything

<<Anything>> is a class that represents anything and is equivalent to all other classes of anything in any other model or logic. The defined class is equivalent to SM:Anything, OWL:Thing and other "top level" classes.

Because of this equivalence, every class in every model virtually inherits from Anything, just as all OWL classes virtually inherit from owl:Thing.

<<Anything>> classes may be used to define "global properties".[CC46]

Base Classes

- Classifier

1.4.5 Stereotype Base Unit Value

<<Base Unit Value>> is a kind of <<Unit Value>> that marks one Unit Value of a quantity kind as the base Unit Value within a model. The base Unit Value provides the basis for conversions between units of the same quantity kind. The base unit always has a ratio of one and an offset of zero.

Base Classes

- Classifier

Direct Supertypes

- Unit Value

1.4.6 Stereotype Category

A category is a facet that is a classification or division of people, events or things regarded as having particular shared characteristics. Categorization is typically contextual, potentially transient and may or may not be formally defined.

As with all facets, categories are non-rigid. Something classified by a category must also be classified by an entity type. An entity may be classified by any number of categories and those categories may change over time.

Base Classes

- Class

Direct Supertypes

- Facet

1.4.7 UML Attribute

An attribute represents kind of characteristic a type of thing may have, e.g. paint may have a color.

[IDEAS] Property: An IndividualType whose members all exhibit a common trait or feature. Often the Individuals are states having a property (the state of being 18 degrees centigrade), where this property can be a CategoricalProperty (qv.) or a DispositionalProperty (qv.).

[ISO 1087] type of characteristics: category of characteristics (3.2.4) which serves as the criterion of subdivision when establishing concept systems. NOTE The type of characteristics colour embraces characteristics (3.2.4) being red, blue, green, etc. The type of characteristics material embraces characteristics made of wood, metal, etc.

[UML] Property

Base Classes

- Property

1.4.8 Stereotype Semantic model

A <<Semantic model>> is a kind of <<Model>> that represents concepts in a real or possible world. Instances of elements in a semantic model are "real world" things, not data about those things.

Base Classes

- Package

Direct Supertypes

- Model

1.4.9 Stereotype Disjoint With

A <<Disjoint With>> dependency is an assertion that two model elements do not and may not denote any of the same set of entities.

When applied to a classifier, every element of the classifier's extent (set of instances) is included in the set of disjoint things.

Base Classes

- Dependency

1.4.10 Stereotype Enumerates

An <<Enumerates>> dependency asserts that the supplier of the dependency is a type and the client of the dependency is a package containing a complete set of possible instance specifications. In this way, <<Enumerates>> is more general than a UML Enumeration because it can enumerate more than just UML data types.

Base Classes

- Dependency

1.4.11 Stereotype Equivalent Class

A <<Equivalent Class>> generalization is an assertion that two classes have the same extents (set of instances). Unlike ontological languages it is not assumed that the two elements are consistent, as statements from different context may or may not agree.

Base Classes

- Generalization

1.4.12 Stereotype Equivalent Property

<<Equivalent Property>> is a declaration that a property is equivalent to one or more other properties (using "equivalent to") or is equivalent to a chain of other properties (using "chain"). <<Equivalent Property>> with at least one value for the "equivalent to" property is an alternative way of expressing <<Equivalent To>>, without introducing additional lines on a diagram.

Either "equivalent to" or "chain" must have a value.

Base Classes

- Property

Tag Definitions

chain : Property [*]

An ordered list of properties forming a "property composition" expressing a traversal path that is equivalent to the stereotyped property. This is similar to a "property chain".

Due to potential "missing information" in creating a chain, a chain may or may not be able to be determined from asserting the chained property. Such a determination is defined in the mapping rules for that property in a particular context.

Note that a chain may also be defined with mapping rules.

equivalent to : Property [*]

A set of properties that the <<Equivalent Property>> is equivalent to. Note that equivalence can also be declared with a <<Equivalent To>> dependency.

1.2.12 Stereotype Equivalent To

An <<Equivalent To>> dependency is an assertion that two model elements represent the same thing or the same set of things. Unlike ontological languages it is not assumed that the two elements are consistent, as statements from different contexts may or may not agree.

Base Classes

- Dependency

1.4.13 1.2.13 Stereotype External Reference

<<External Reference>> provides traceability to the source of a "fact" in a model based on some external information resource. This references helps to facilitate provenance. Reference is a statement about the model data and has no semantic implication. Source reference may impact the trust in a statement but the evaluation of trust is outside of this specification.

External reference is combined with the owned comment(s) to create Semantic Modeling descriptions.

Base Classes

- Element

Tag Definitions

external reference : String

Specifies the location URL of the external resource. The format must comply with [RFC3987].

external term : String

The external term or location of the information in the source. The form of expression of the term or term path is dependent on the referenced technology.

1.4.14 Stereotype Has Value

A <<Has Value>> dependency asserts that the client of the dependency is a type and the supplier of the dependency is an instance specification that defines acceptable values for one or more properties of that type. Each slot of the instance specification is a possible value for a corresponding property in the type.

<<Has Value>> corresponds to one or more OWL property restrictions containing a "hasValue" constraint.

Base Classes

- Dependency

1.4.15 Stereotype Information Model

An <<Information Model>> is a kind of <<Model>> that represents

a model for some purpose, independent of technical implementation. An information model may contain logical models or data models, as well as other logical viewpoints.

Base Classes

- Package

Direct Supertypes

- Model

1.4.16 Stereotype Intersection

An <<Intersection>> is a class that has an extent (set of instances) equivalent to the intersection of the extents of all supertypes. Intersection is a stronger statement than a subtype, as a subtype may be a subset of the intersection. An instance of all the supertypes implies an instance is also an instance of the intersection type.

For intersection, The SMP considers UML generalization and UML realization equivalent. This is due to ownership and legacy considerations in UML. Generalization is the preferred representation.

Note: Realizations are included to support unions across external models. UML generalization cannot be used across external models due to the ownership of generalization.

Base Classes

- Classifier

1.4.17 Stereotype In Context Of

<< In context of>> is an assertion that the client of the dependency is in the context of the supplier of the dependency. All assertions and rules defined in the supplier context apply to the client and everything in the context of the client (i.e., it is transitive). Packages, classes, situations and instances are typical contexts. Note that <<Is In Context>> is the default interpretation of a dependency, if no stereotype is specified it will be interpreted as <<Is In Context>>.

Base Classes

- Dependency

1.4.18 Stereotype Model

<<Model>> is stereotype of package that may have an id (see <<Resource>>) and/or a namespace prefix (like the "dc" in "dc:title").

Base Classes

• Package

Tag Definitions

namespace prefix : String

A hint as to an appropriate abbreviation for a model that may be used in some technology mappings, such as XML. The prefix should be short and contain only letters and numbers and must start with a letter. e.g., "dc" in "dc:title".

Direct Supertypes

• Resource

1.4.19 Stereotype Phase

A <<Phase>> (a.k.a. "State") is a classification of an entity based on change of that entity over time. A <<Phase>> is a <<Phase Of>> the types that may have that phase (e.g., "Teenager").

A phase is a [DOLCE] "non rigid sortal", a type that may change over the lifetime of an entity.

Base Classes

• Class

1.4.20 Stereotype Quantity Kind

<<Quantity Kind>> is an aspect common to mutually comparable quantities represented by one or more units. Units with a common quantity kind may be algorithmically converted to any other unit of that quantity kind. e.g. temperature.[JCGM 200:2008].

Units with a common quantity kind may be algorithmically converted to any other unit of that quantity kind. e.g. temperature. SMP takes a wider view of quantity kinds to include conversions that may be contextual and time dependent, such as currencies.

Base Classes

• Classifier

Direct Supertypes

• Value

1.4.21 Stereotype Relationship

A relationship defines a condition involving related things. A relationship may be asserted within a context as true or false within that context. Each instance of a relationship has a number of bindings to the "ends" of the relationship which do not change for the life of the relationship..

A relationship may be true or false within its context (including a time frame) but is atomic in its truth value.

Relationships may participate in (be bound to) other relationships and as such bindings involving a relationship may change over time. That is, relationships are "first class" objects.

The relationship stereotype may be used with association classes or classes. All associations are implicitly relationships. Classes stereotyped as relationships should stereotype the relationship ends as <<Involves>>.

Base Classes

• Class (including association class)

1.4.22 Stereotype Represents

<<Represents>> is an assertion that the source *concrete* type or feature provides a more concrete way to represent the target reference type. Represents may be used within conceptual models or from a physical model to a conceptual model.

- A representation that is a dependency or realization makes no assumption that the types are substitutable.
- A representation that is a generalization is substitutable for what it represents.

Base Classes

- Dependency
- Generalization

Tag Definitions

condition : ValueSpecification

<condition> is an expression that must be true for the source to represent the target.

bidirectional : Boolean

<bidirectional> implies a direct mapping between instances of the types in both directions.

<bidirectional> is equivalent to a mapping with a rule mapping properties of each type but is lower precedence than other mappings - if types have a more specific map it will apply first.

1.4.23 Stereotype Resource

A <<Resource>> is anything that can be referenced by an identifier in a model, ontology or vocabulary. The resource identifier is often an IRI.[CC54]

Base Classes

- NamedElement

Tag Definitions

id : String

A unique identifier for any resource.

When defined for a Package, id has the format defined in [RFC3987]. In this case, it is equivalent to UML:URI, and setting one will set the other.

Stereotype Restriction

A restriction is a property or association that constrains an existing property or association rather than defining a new concept in a domain.

Properties with no name or the same name as a property they subset or redefine are implicitly restrictions. Associations with a restriction as an end are implicitly restrictions.

Base Classes

- Association
- Property

Stereotype Role

A <<Role>> is a classification of an entity based on that entity's behavior, participation in a situation, or capabilities. A <<Role>> <<Facet Of>> the types that may play that role. e.g., "Teacher" <<Facet Of>> “Person”.

A role is a [DOLCE] "non rigid sortal", [CC55] a type that may change over the lifetime of an entity.

Base Classes

- Class

1.4.24 Stereotype Sufficient

Specifying <<Sufficient>> for one or more of a classes or association’s properties means that an instance having an acceptable cardinality of values for all of those properties implies that the instance is an instance of that type.[CC56]

Base Classes

- Property

1.4.25 Stereotype Synonym

<<Synonym>> defines an alternate name for the annotated elements of the comment. The alternate name is the body of the comment.

The alternate name will not be the "preferred name" of the element.

Base Classes

- Comment

1.4.26 Stereotype Union

A <<Union>> is a class that has an extent (set of instances) which is equivalent to the union of the extents of all types that specialize the Union (Subclasses). Specializing types shall include subtypes and types that realize the union.

Note: UML realizations are included to support unions across external models because UML generalization can not be used across external models due to the ownership of generalization.

[MathWorld] Given two sets A and B, the union is the set that contains elements or objects that belong to either A or to B or to both.

Base Classes

- Classifier

1.4.27 Stereotype Unit Value

A <<Unit Value>> is a <<Value>> with an <<External Reference>> that represents a type of a quantity value referencing a specific unit. A Unit Value is a required type of a property representing a quantity.

[JCGM 200:2008] A Unit is a real scalar quantity, defined and adopted by convention, with which any other quantity of the same quantity kind can be compared to express the ratio of the two quantities as a number. e.g. Degrees Centigrade, Miles.

Each Unit Value represents refinement of a quantity kind using generalization and is thus substitutable for that quantity kind. Typically, quantity kinds are used in conceptual models and Unit Values in physical or logical models.

Unit Values may only subtype quantity kinds and numbers.

Note that Unit Values are not units, but the type of quantity values expressed with reference to a common unit as defined in [JCGM 200:2008].

Each instance of a Unit Value shares a common unit (as defined by standards) with a reference defined by "external reference" and "external term".

Classifiers defined as <<Unit Value>> shall semantically subclass the SMP model “Unit Value” class.

Base Classes

- Classifier

Tag Definitions

offset : Real

The difference between zero in the unit and zero in the base unit after the ratio is applied to the base unit as defined within the same model.

ratio : Real

The multiplier by which to multiply the unit to convert to the base unit as defined within the same model.

symbol : String

The accepted symbol for a unit. e.g. "g" for "Gram".

Direct Supertypes

- External Reference
- Value

1.4.28 Stereotype Value

A <<Value>> is a type representing an atomic unit of information without independent identity. Values include numbers, strings and enumerations. In some cases values may have internal structure. Values do not change over time.

Quantity kinds and units are also values. Values may stereotype any classifier. UML data types, including primitives and enumerations, are implicitly values.

Classifiers defined as <<Value>> shall semantically subclass the SMP model “Value” class.

Base Classes

- Classifier