Concept Spaces:  
Towards a Unified Semantic Model

Paul C. Brown

Version 3.0

Revised 12 February 2014

Table of Contents

[1 Overview 4](#_Toc380500232)

[1.1 Motivation 4](#_Toc380500233)

[1.1.1 SIMF Complexity 4](#_Toc380500234)

[1.1.2 Simplification Rationale 5](#_Toc380500235)

[2 Design Principles 5](#_Toc380500236)

[2.1 Relationships and Entities Are Just Concepts 5](#_Toc380500237)

[2.2 Instances and Types 6](#_Toc380500238)

[2.3 Meta-Levels 6](#_Toc380500239)

[3 Concept Representation Language (CRL) 7](#_Toc380500240)

[3.1 Element 8](#_Toc380500241)

[3.2 Context Relation 8](#_Toc380500242)

[3.3 Reference 9](#_Toc380500243)

[3.4 Constraint 10](#_Toc380500244)

[3.4.1 Constraint Language 11](#_Toc380500245)

[3.5 Refinement Relation 11](#_Toc380500246)

[4 Stored Representation 12](#_Toc380500247)

[4.1 Reified Refinement Relation 12](#_Toc380500248)

[4.2 Stored Representation 13](#_Toc380500249)

[5 The Foundational Concept Space 15](#_Toc380500250)

[5.1 Foundation Concept Space 16](#_Toc380500251)

[5.2 Concept 16](#_Toc380500252)

[5.3 Reference 16](#_Toc380500253)

[5.4 Constraint 17](#_Toc380500254)

[5.5 Refinement Relation 17](#_Toc380500255)

[5.6 Binding 20](#_Toc380500256)

[6 Some Example Concepts 21](#_Toc380500257)

[6.1 Simple Types 21](#_Toc380500258)

[6.2 Binary Associations 21](#_Toc380500259)

[6.3 Collection 22](#_Toc380500260)

[6.4 Set 23](#_Toc380500261)

[6.5 List 24](#_Toc380500262)

[6.6 Ordered Set 25](#_Toc380500263)

[7 A Simple Mapping Example 25](#_Toc380500264)

[8 Mapping Patterns 30](#_Toc380500265)

# Overview

This document presents a proposal called Concept Spaces. Its foundation is a singular Concept Representation Language (CRL) for representing all models and the mappings between them. Each model created is called a concept space, and a mapping between two models is, itself, another concept space.

A concept space can be used to represent a particular level of abstraction, such as those required by SIMF: the conceptual data model (CDM), language-independent model (LIM), and physical data schema (PDS). A concept space can also be used to represent relations between other concept spaces, such as the model-bridging relations (MBR) required by SIMF.

The representational elements of the Concept Representation Language are not constrained to belong to a particular level of abstraction (e.g. M0, M1, or M2 from fact-based reasoning or CDM, LIM, and PDS from SIMF). Instead, each modeled level of abstraction is a representational element as is the membership of some other element in that level of abstraction. Concept Spaces accomplishes this with a minimalist representation scheme comprising just a few concepts and relations.

## Motivation

### SIMF Complexity

The SIMF RFP requires models that address different levels of abstraction:

* Common Domain Model (CDM)
* Logical Information Model (LIM)
* Physical Data Schema (PDS)

SIMF also requires model-bridging relations (MBR). These define connections between different sets of elements in the same or different models.

If each of the models (CDM, LIM, PDS) is expressed in a different language, then each combination of models will require a different MBR, each of which will essentially be its own language. Assuming that bridging directly from CDM to PDS is not required, this means that five different MBRs will be required:

* CDM to CDM
* CDM to LIM
* LIM to LIM
* LIM to PDS
* PDS to PDS

From an adoption perspective, requiring users of the standard to learn eight different languages, even if they are similar to each other, seems prohibitively complex.

### Simplification Rationale

Each model and each model-bridging relation is a representation of something. Each representation is expressed in a representation language. The representation language defines the kinds of things that can be represented and the kinds of relationships between those things that can be expressed.

Different languages emerge when different kinds of things are represented in different ways. This also results in different representations for mappings between different kinds of things.

The root cause of the complexity is thus the variation in the ways in which different kinds of things are represented. But what if we had a uniform way of defining those representations – and the relationships between them? This is the idea behind the Concept Representation Language (CRL): a minimal set of building blocks from which all representations and relationships can be constructed. Representations built with this language are known as Concept Spaces.

# Design Principles

There are a significant number of semantic models around, and the SIMF effort seeks to provide an approach towards at least bridging between them. It is the conjecture of this paper that there are at least three complicating factors of models that present barriers to the SIMF goals and could be removed. These complicating factors are embedded in virtually all existing semantic models (if there are exceptions, I am not specifically aware of them). These factors are:

* Relationships are modeled differently from the entities they relate
* Instances are modeled differently from their types
* Models at one meta-level are treated as being distinct from models at other meta-levels

It is the position of this paper that these three distinctions are not fundamental, and that a representational scheme that is not burdened by these distinctions will greatly simplify the task of satisfying the SIMF requirements.

## Relationships and Entities Are Just Concepts

The distinction between entities and relationships is not fundamental. Both are concepts, and the distinction relates to the relative roles that one plays with respect to the other. There is nothing that prevents the same concept from playing the role of an entity and also playing the role of a relation. For example, the concept of Customer can be an entity in one context and a relationship between a person and a company in another. If relationships and entities are represented differently, mapping their semantics between these contexts, e.g. mapping entities to relationships, gets complicated.

More fundamentally, there are concepts that just cannot be expressed if this distinction is enforced. Consider the notion of analogy: A is to B as C is to D. Expressing this idea involves three relationships: one relating A to C, the other relating C to D, and the third relating the “A is to B” relation to the “C is to D” relation. This is a relation between relations.

Figure 2‑1 illustrates this difficulty in the UML Class notation, which makes this distinction between entities (classes) and relationships (associations). This notation allows the expression of the original ABRelation and CDRealtion as well as the mapping of A onto C and B onto D. What it cannot express is the mapping of the ABRelation onto the CDRelation. Of course, if you add association classes to these two relations, you can now show an association between these classes, but that’s a bit of a hack.



Figure 2‑1: Representing a Relation in UML Class notation

These considerations lead us to the following principle:

Unification Principle 1: A relationship is a concept

## Instances and Types

A type is a specification for a set or collection of entities. The idea of an instance is that it is an entity that conforms to the specification of the type. This does not necessarily imply that there is anything inherent about any given entity that makes it either a type or an instance. The notion of type refers to the *relative* role of one entity vs. another.

Consider the notion of a set of types. The members of this set are, on the one hand, instances of the abstract concept of type (hence their membership in the set), while on the other hand they are, themselves, types.

This consideration leads to the following principle:

Unification Principle 2: A concept can be, simultaneously, both a type and an instance

## Meta-Levels

It is common to treat the concepts and relationships associated with one level of abstraction as being somehow fundamentally different from those at another level of abstraction. However, maintaining such a distinction precludes a uniform representation of the elements of the different levels and mappings between them. Such representations are a requirement of SIMF. This leads to the following principle:

Unification Principle 3: Meta-levels are just concepts, on a par with other concepts

# Concept Representation Language (CRL)

The motivation for Concept Spaces is that all of the various models required by SIMF are just representations of concepts (as opposed to the concepts themselves). What each element actually represents varies – the fact that the element is a representation does not. SIMF needs to not only model different levels of abstraction (CDM, LIM, PDS), but also the mappings between the levels. After all, the goal of SIMF is to be able to map the lower level representations into higher-level representations and then back down to a different set of lower level representations as a means of bridging between the different lower-level representations.

The foundation of Concept Spaces is the Concept Representation Language (CRL). The language provides a standardized way of representing concepts at any level of abstraction, including relationships of arbitrary arity. In this language we don’t care what is being represented – it’s all just elements and relationships. The goal is to have a uniform way of representing anything – including the mappings.

We now define the concept representation language used to model concepts (Figure 3‑1). It consists of:

* A set of Elements. An element is a representation of a concept. The element has a definition that defines the concept and a unique identifier (UUID). A commonName provides as a human-readable form of identification. A version is included for experimental purposes, but it is not yet a fundamental part of the language.
* A special type of Element, a Reference, whose referencedElement is a pointer to an Element. The Reference itself, being an Element, has an identity of its own independent of its owner.
* A special type of Element, a Constraint, that specifies a constraint that must be satisfied. The References of the constraint identify the Elements to which the constraint applies. A Constraint applied to an Element also applies to every refinement of that Element. The constraint being applied is a logical expression whose variables are the References to Elements. Constraint expressions may utilize path expressions that walk the Context relations. In these expressions, “.” Is the referenced Element (or its refinement), “../” moves up one step to the owner and “/<commonName> to walk down one step in the ownership hierarchy.
* A Context relation, which indicates the ownership of Elements by another Element. Owned elements have exactly the same lifetime as their owners. Note that this is a lifetime of the representation, not the concept being represented. Consider this to be a definitional hierarchy: the definition of the owner includes the definition of all of its owned elements.
* A Refinement relation, which subsumes generalization and instantiation for representations. This relation has an associated constraint: every constraint on the abstraction must also be satisfied by its refinement. One common constraint is the multiplicity constraints on specific bindings.



Figure 3‑1: Concept Space Overview

## Element

Definition: an Element is a representation of a concept. Note that there is no assumption of uniqueness with respect to this representation: there can be multiple representations of the same concept. In fact, independently developed models of the same concepts will, by definition, have different representations for the same concepts.

Element serves as the building block from which all concept representations are constructed. Each Element contains a definition, a human-readable description of the concept represented by the Element. The identifier of the Element, specified as a UUID, ensures a unique identity for the representation.

An Element has a version (to be detailed later).

The graphic notation for an element is a rectangle with a circled “E” to indicate it’s an element. The commonName of the element is displayed.



Figure 3‑2: Graphic Representation of an Element

## Context Relation

Definition: the Context relation identifies the owner of the Element. If an Element represents a dictionary definition of a concept, its owner is the Element that represents the dictionary itself. Other contexts may be less formal, representing a conversational situation (“Paul’s discussion with Sjir and Cory on Friday”) or a point in time (“In the last century…”). The ownedElements of a Context relation have exactly the same lifetime as the owner.

An Element (the whole) can own an arbitrary number of Elements, as specified by the Context relation. Each Element can only be owned by one. The Context relation forms a strict forest structure.

The Context relation subsumes two UML 2 ideas, the composition model relating a classifier to its parts[[1]](#footnote-1) and the inner elements structure relating packages to their contained definitions

Principle 1 implies that it must be possible to have some structure to the Element. This is satisfied by the ownedConcepts of the Context relation. This structure allows an Element to be defined as an aggregate of other Elements. Later we will see how notions like relations can be constructed using this relation.

One possible graphical notation for context is shown in Figure 3‑3. Here an element X is shown to own three references that are labeled A, B, and C.

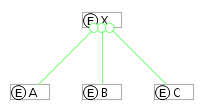


Figure 3‑3: Graphical Notation for Context using UML-style Composition Relations

One point to note here is that the context relationship itself is not an Element (i.e. is not reified). {As of yet, no motivation for reifying this relation has been identified.}

## Reference

Definition: A Reference is a mechanism by which an Element (the owner of the reference) can point too) another Element.

The full graphical notation (Figure 3‑4) shows the reference as a rectangle with the owner being designated by an arc whose owner end has a ⊕ symbol. In cases in which the reference itself is not the target of another reference, its presence clutters up the diagram. In such cases the shorthand graphical notation of Figure 3‑4b will be used. Here the arrow itself represents the reference, and the label on the arrow identifies the reference. In both versions, a solid arrow points to the referencedElement.

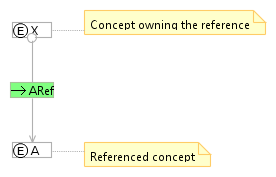


Figure 3‑4: Full Graphical Notation for a Reference

Showing the actual reference concept all the time can clutter up a diagram. Accordingly we define a shorthand notation as shown in Figure 3‑5. Note that the underlying model still has the explicit reference concept.



Figure 3‑5: Shorthand Reference Notation

## Constraint

Definition: A Constraint is a logical condition that must be satisfied. A constraint contains one or more references to the concepts that are being constrained. The constraint expression defines the constraint being imposed.

A constraint applies to the constrained concept(s) and all refinements of the constrained concepts.

Figure 3‑6 shows the graphic representation of a constraint. Here the Constraint, C1, has a reference to the concept X, which is the concept being constrained. The actual constraint expression is shown in the property view of the constraint.

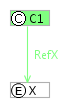


Figure 3‑6: Graphic Representation of a Constraint

### Constraint Language

The constraint language is still to be determined at this time. A tentative proposal is that Constraints are defined using set-theoretic expressions. The sets being referenced are those defined by the referenced Element’s relationships.

We have the following sets that can be referenced:

<ElementName>.ownedConcepts(): the set of owned concepts

We have the following predicates defined:

<ElementName>.<identifier>(): the set of owned elements with the indicated identifier.

A count(<set>) function is defined that returns the count of the indicated set membership.

For a Reference, <Reference>.referencedElement() returns the referencedElement.

## Refinement Relation

Definition: Refinement is a relation between two concepts. Beyond a simple declaration of the relationship, the refinement imposes a constraint that the refined element must satisfy all the constraints of the abstraction. While refinement is a binary relation, there is no restriction on their use other than circular relationships are not permitted. Thus it is permissible to create many-to-many relationships between elements: an Element may be a refinement of multiple abstractions, and an Element can serve as an abstraction for multiple refinements.

The Refinement relation provides the foundation for representing both generalization and instantiation.

The full graphical representation of the Refinement relation is shown in Figure 3‑7. The Refinement, itself a refinement of an Element, has a reference to the abstracted element and another reference to the refined element. The common name given to the refinement is just a convention to clarify the meaning of the refinement.

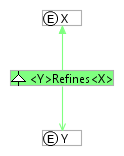


Figure 3‑7: Refinement Relation Notation Using UML Generalization Notation

The explicit representation of the refinement is generally not required. Figure 3‑8 shows a shortand notation for refinement.

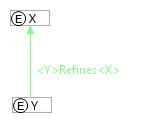


Figure 3‑8: Shorthand Notation for Refinement Relation

# Stored Representation

Before showing the stored representation, an exploration of the reified refinement relation is in order.

## Reified Refinement Relation

In the conceptual models represented in Concept Spaces it is likely that these models will have concepts (e.g. generalization, instantiation) that are, themselves, refinements of the Refinement relation. Thus the Refinement relation, itself, needs to be an Element that can participate in other relationships. To support this, the Refinement relation is reified as shown in Figure 4‑1.



Figure 4‑1: Reified Refinement Relationship

## Stored Representation

From the reified version of the model a straightforward stored representation can be defined. The model consists of four data types and one XML element as shown in Figure 4‑2.



Figure 4‑2: Model of Stored Representation

Here’s the XSD that defines the stored representation:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<schema targetNamespace=*"http://simf.omg.org/ConceptSpace"* elementFormDefault=*"unqualified"* xmlns=*"http://www.w3.org/2001/XMLSchema"* xmlns:tns=*"http://simf.omg.org/ConceptSpace"*>

<complexType name=*"Element"*>

<sequence>

<element name=*"identifier"* type=*"string"* maxOccurs=*"1"*

minOccurs=*"1"*>

</element>

<element name=*"commonName"* type=*"string"* maxOccurs=*"1"*

minOccurs=*"1"*>

</element>

<element name=*"definition"* type=*"string"* maxOccurs=*"1"*

minOccurs=*"1"*>

</element>

<element name=*"version"* type=*"string"* maxOccurs=*"1"* minOccurs=*"1"*></element>

<element name=*"ownedConcepts"* type=*"tns:Element"*

minOccurs=*"0"* maxOccurs=*"unbounded"*>

</element>

</sequence>

</complexType>

<complexType name=*"Document"*>

<sequence>

<element name=*"conceptSpace"* type=*"tns:Element"* maxOccurs=*"1"* minOccurs=*"1"*></element>

</sequence>

</complexType>

<element name=*"document"* type=*"tns:Document"*></element>

<complexType name=*"Reference"*>

<complexContent>

<extension base=*"tns:Element"*>

<sequence>

<element name=*"referencedElement"* type=*"string"* maxOccurs=*"1"* minOccurs=*"1"*></element>

<element name=*"referencedElementVersion"* type=*"string"* maxOccurs=*"1"* minOccurs=*"1"*></element>

</sequence>

</extension>

</complexContent>

</complexType>

<complexType name=*"Constraint"*>

<complexContent>

<extension base=*"tns:Element"*>

<sequence>

<element name=*"constraintExpression"* type=*"string"* maxOccurs=*"1"* minOccurs=*"1"*></element>

</sequence>

</extension>

</complexContent>

</complexType>

<complexType name=*"Refinement"*>

<complexContent>

<extension base=*"tns:Element"*>

<sequence>

<element name=*"abstractElement"* type=*"string"*   
 maxOccurs=*"1"* minOccurs=*"1"*></element>

<element name=*"refinedElement"* type=*"string"*   
 maxOccurs=*"1"* minOccurs=*"1"*></element>

</sequence>

</extension>

</complexContent>

</complexType>

</schema>

# The Foundational Concept Space

The foundational concept space defines the foundational concepts from which all other concepts derive. These are the concepts of the concept space itself:

* The Foundation concept space
* Concept
* Reference
* Constraint
* Refinement

Additionally we define one more concept that seems to be fundamental:

* Binding

## Foundation Concept Space

The Foundation concept space is represented by a rectangle with an  icon (Figure 5‑1). The icon indicates that this is an instance of the Element class shown in Figure 4‑2. The commonName of the Element class is displayed in the rectangle. What makes this a concept space is the fact that the element has no owner.



Figure 5‑1: Graphical Representation of the Foundation Concept Space

## Concept

A Concept is an Element that is owned by the Foundation concept space (Figure 5‑2). The owner relation, shown as a link with a circle at the owner end, is a graphical representation of the Element’s owner relation from Figure 4‑2.

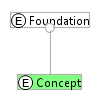


Figure 5‑2: Graphical Representation of Concept in the Foundation Concept Space

## Reference

A Reference is denoted by a rectangle with an  icon (Figure 5‑3). It represents an instance of the Reference class from Figure 4‑2. The referencedElement of this class is denoted by a link with an arrow at the referencedElement end. The Reference is owned by the Foundation concept space. The Reference is a refinement of Concept (refinement is described in a subsequent section).

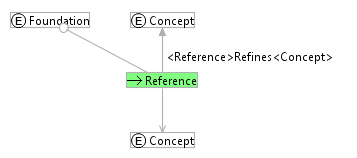


Figure 5‑3: Full Graphical Representation of the Foundational Reference

In creating many diagram, the full details of the reference are not of interest. To make these diagrams easy to draw, we define a shorthand notation for a reference (Figure 5‑4). The shorthand notation shows the reference as a link between its owner (in this case, Foundation) and the referencedElement (in this case, Concept). Note that the shorthand notation does not show the refinement relation: this must be defined elsewhere.

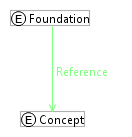


Figure 5‑4: Shorthand Representation of the Foundational Reference

## Constraint

A Constraint is denoted by a rectangle with a  icon (Figure 5‑5). It represents an instance of the Constraint class from Figure 4‑2. The expression attribute of the Constraint class is not shown graphically: it is displayed as a property of the Constraint representation in the editor. Constraint is a refinement of Concept.

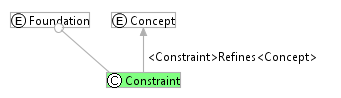


Figure 5‑5: Graphical Representation of Constraint in the Foundation Concept Space

## Refinement Relation

The Refinement relation in the concept space is represented by a rectangle with a  icon (Figure 5‑6). This represents an instance of the Refinement class from Figure 4‑2. This element has two references: the abstractConcept reference, which is denoted by a link with an “A” at the abstractConcept end; and the refinedConcept, which is denoted by a link with a dash across it at the refinedConcept end.

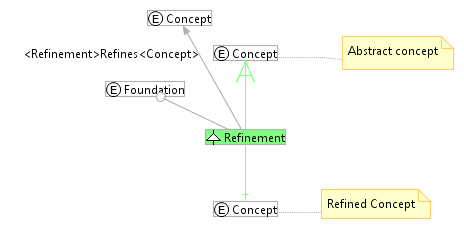


Figure 5‑6: Refinement Full Graphical Representation

As with the references, many usages of the Refinement concept do not require the level of detail shown in the full representation. Accordingly, we define a shorthand notation ().

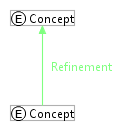


Figure 5‑7: Refinement Shorthand Notation

The Refinement representation is a simplified version of what it really represents. The full reified version of the relation is shown in Figure 5‑8. Here the references are explicitly shown. There are also two constraints. The AbstractConceptConstraint, defined by count(./AbstractConcept())=1, requires that there be exactly one AbstractConcept reference. The RefinedConceptConstratint, count(./RefinedConcept())=1 , requires that there be exacely one RefinedConcept reference.

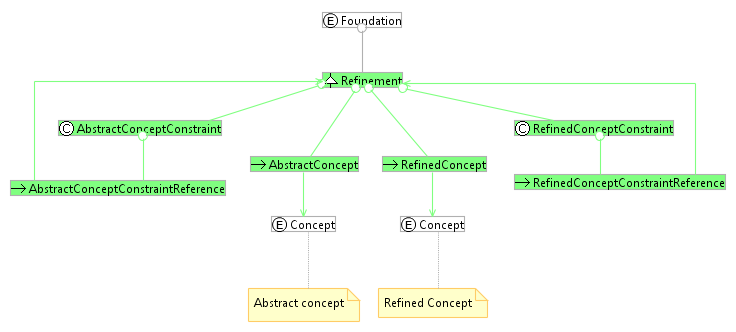


Figure 5‑8: Reified Refinement Relation

## Binding

<THIS SECTION REQUIRES REVISION>

A Binding is a derived concept that is somewhat similar to a reference with one difference. A Reference defines a role that a concept (the referencedElement) plays with respect to the reference’s owner, with the existence of the role being part of the definition of the owner. A Binding defines a role that one concept (the Object) plays with respect to another concept (the Subject), but the ownership of the Binding is independent of the subject.

Graphically we have:



Figure 5‑12: Reference in the Foundational Concept Space

# Some Example Concepts

## Simple Types

SimpleTypes (Figure 6‑1) is a concept space used here to illustrate how simple types could be defined. Values of simple types are refinements of the type (this is true for any type). For simple types that are enumerations, it is convenient to model the values as being owned by the type, but that is not a requirement – it is a modeling choice. Values for the type Boolean are shown as examples.

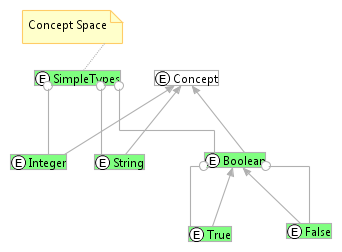


Figure 6‑1: Simple Types

## Binary Associations

A binary association is a concept with a contained concept for each of the association roles (Figure 6‑2) and for each of the bindings. An association can have an arbitrary number of roles, and therefore can represent an n-ary association. Constraints can be added to express limitations on the number of occurrences allowed for each role.

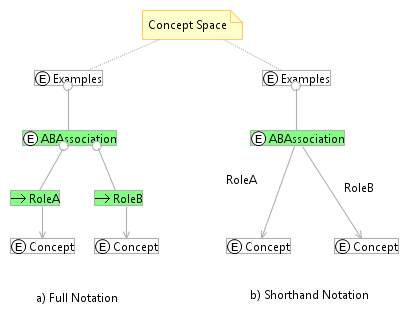


Figure 6‑2: Representation of a Binary Association

## Collection

A Collection is a concept with a reference defining the Member role Figure 6‑3. This particular definition allows any concept to be a member of the collection.

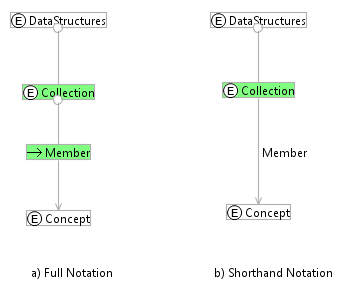


Figure 6‑3: Collection of Concepts

Figure 6‑4 shows an instance of a collection, CollectionInstance, with three members, P, Q, and R. Elements P, Q, and R are arbitrary refinements of Concept – their Refinement relations have been omitted for clarity. The CollectionInstance has three references, each of which is a subclass of Collection.Member.

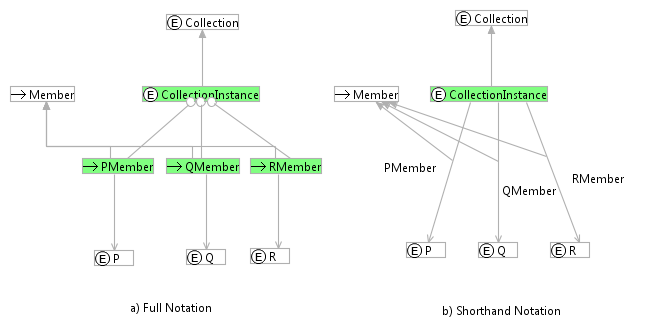


Figure 6‑4: Example Instance of a Collection

## Set

A set is just a subclass of Collection as shown in Figure 6‑5. The MemberUniquenessConstraint requires that each element can only appear as a member once.

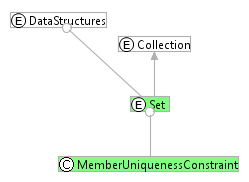


Figure 6‑5: Set Definition

The uniqueness constraint can be specified as:

∀m1,m2∈./Members(), m1≠m2⇒ m1.referencedConcept≠ m2.referencedConcept

A PeopleSet (set of people) is shown in Figure 6‑6. The PeopleSet is a Set that has a refined definition of Member that references Person. Person is a refinement of Concept (this relation is not shown for simplicity).

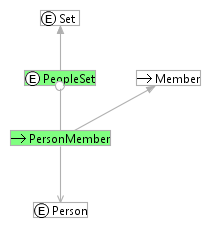


Figure 6‑6: PeopleSet

## List

A list is an ordered collection of concepts (Figure 6‑7). The List itself has an OrderingConstraint that specifies the total ordering constraint on the list members. The ListMember reference owns a Predecessor reference to the previous member in the list.

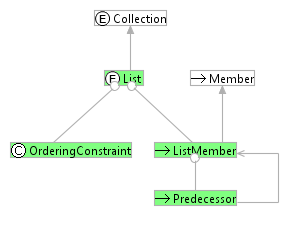


Figure 6‑7: List

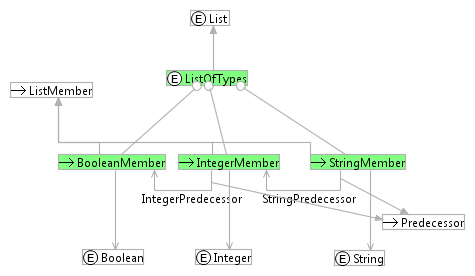


Figure 6‑8: List of Types Example

## Ordered Set

Finally, we define the OrderedSet as a subclass of both Set and List (Figure 6‑9). Note that the constraints of both Set and List are inherited.

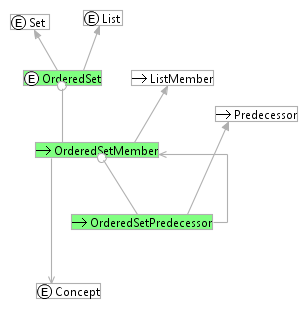


Figure 6‑9: Ordered Set

# A Simple Mapping Example

Since the primary purpose of SIMF is to facilitate mapping between different representations of related information, it makes sense to try out the proposed representation with an example.

The use of XML or other inherently hierarchical models faces challenges when the inherent information structure is not tree-oriented. A common example is the Order-Shipments model shown in Figure 7‑1. In some businesses, Orders can have multiple shipments associated with them, i.e. split shipments are allowed. These same businesses may also allow consolidated shipments – more than one order can be included in one shipment.



Figure 7‑1: Order-Shipments Model

The CDM representation of this situation is shown in Figure 7‑2. Note that the multiplicity constraints are not represented in this simplified model.



Figure 7‑2: Concept Space Representation of Orders-Shipments Common Data Model (CDM)

A common situation is that individual systems take a particular perspective on this information. An order management system, for example, might use the representation shown in Figure 7‑3.



Figure 7‑3: UML Representation of an Order-centric Logical Information Model (OLIM)

The Concept Space representation of this data structure is shown in Figure 7‑4. Each Shipment is bound to the Order by a binding with role <shipments>. The implementation detail of the binding is shown for reference, as this detail will be necessary when showing mapping.



Figure 7‑4: Concept Space Representation of the Order-centric Logical Information Model (OLIM)

The mapping between the OLIM and the CDM can be represented as follows:



Figure 7‑5: OLIM to CDM Mapping With Full Representation of CDM and OLIM

Figure 7‑6 shows the same mappings, but using the shorthand notation for the OLIM and CDM models. Since this notation is more compact, it will be used in subsequent diagrams.



Figure 7‑6: OLIM-CDM Mapping with OLIM and CDM Shorthand

Another system might have a shipment-centric perspective (Figure 7‑7).



Figure 7‑7: Concept Space Representation of a Shipment-centric Logical Information Model (SLIM)

The mapping from the CDM to the SLIM can be represented as follows:



Figure 7‑8: CDM to SLIM Mapping

Putting the two mappings together we have:



Figure 7‑9: Concept Space Representation of OLIM-CDM-SLIM Mappings.

This set of mappings can then be used as the basis for mapping an order-centric data structure into a shipment-centric data structure as follows:

1. For each Shipment(OLIM), generate a Shipment(SLIM). The generated SLIM structure includes the concept in the dashed box: the OrderSet(SLIM).
2. For each Order(OLIM), generate a correspongind Order(SLIM).
3. For each ShipmentSetMember(SLIM), generate a corresponding OrderSetMember using the catenated OrderShipmentRelationMapping(OLIMtoCDM) and OrderShipmentRelationMapping(SLIMtoCDM) and their subordinate mappings to identify the appropriate shipments and orders for each instance.

# Mapping Patterns

Mappings can be abstracted as patterns. Figure 8‑1 shows a BindingToAssociationMapping that has four owned concepts. Two of these, the BindingPattern and AssociationPattern, identify the parts of the binding and association involved in the mapping. These pattern bindings indicate the actual elements to which the pattern is being applied. The remaining parts of the mapping reference the bindings of the SetPattern and AssociationPattern, making the overall mapping generic.



Figure 8‑1: OLIM to CDM Mapping Using Patterns



Figure 8‑2: Pattern Mapping Using Shorthand Notation

1. Conrad Bock: “UML 2 Composition Model”, in *Journal of Object Technology*, vol. 3, no. 10, November-December 2004, pp. 47-73. [↑](#footnote-ref-1)