Part-part Relations in an RDF/S and OWL Extension

Last modified: Mon Sep 11 18:11:59 2006 ET

Editor and Author:
Conrad Bock, U.S. National Institute of Standards and Technology, conrad.bock@nist.gov

Acknowledgments

Abstract

This describes an extension to the Resource Description Framework/Schema (RDF/S) [RDF Vocabulary] to specify relations between parts of the same whole (*composite structure*). For example, the relation between engine and wheels in a car applies in each individual car, rather than between parts of different cars or to engines in boats. The extension augments techniques for representing relations between part and whole, and proposes a small taxonomy of part-whole relations specifically for application with part-part relations. The extension can be used with others on RDF/S, such as Ontology Web Language (OWL) [OWL]. An OWL version is also provided.

Section 1 introduces composite structure, explaining why it is needed and outlining the approach taken in this document. Sections 2, 3, and 4 define the vocabulary and semantics for the three levels of the extension. Section 5 describes future work.

Contents

1. Introduction

Composite structure specifies the interconnections of parts within the same whole. It is found in practically every area of science, engineering, business, and government. For example, atoms bound to other atoms in a molecule, interlocking amino acids in DNA, connected organs in the body, interacting parts and subsystems of an automobile or aircraft, cooperating positions and departments in an organization, partners in industry collaborations and commerce, and checks and balances among bodies within a government, are all interconnected elements within an organized whole.

This section reviews the concepts in composite structure, referring to appendices and articles for more detail. Section 1.1 explains why an extension is needed, Section 1.2 introduces the overall approach and its features. Section 1.3 describes an important design decision in choosing the levels of the extension. Section 1.4 gives some conventions used in the document.

1.1 Classes and Properties Insufficient

Languages limited to classes and relations (or properties) are insufficient for representing composite structure. This is because relations and properties of classes apply across all instances of the class, rather than each instance individually. For example, suppose a powers property is specified to link engines and wheels in a car using RDF/S, as shown below (namespace prefixes are omitted for brevity):

```
<rdfs:Class rdf:about="#Car"/>
<rdfs:Class rdf:about="#Engine"/>
<rdfs:Class rdf:about="#Wheel"/>
<rdfs:Class rdf:about="#Boat"/>
<rdfs:Class rdf:about="#Propellor"/>

<rdf:Property rdf:about="#engineInCar" >
  <rdfs:domain rdf:resource="#Car"/>
  <rdfs:range rdf:resource="#Engine"/>
</rdf:Property>

<rdf:Property rdf:about="#wheelInCar">
  <rdfs:domain rdf:resource="#Car"/>
  <rdfs:range rdf:resource="#Wheel"/>
</rdf:Property>

<rdf:Property rdf:about="#engineInBoat" >
  <rdfs:domain rdf:resource="#Boat"/>
  <rdfs:range rdf:resource="#Engine"/>
</rdf:Property>

<rdf:Property rdf:about="#propellerInBoat" >
  <rdfs:domain rdf:resource="#Boat"/>
  <rdfs:range rdf:resource="#Propeller"/>
</rdf:Property>

<rdf:Property rdf:about="#powers">
  <rdfs:domain rdf:resource="#Engine"/>
</rdf:Property>
```

This allows the following:

1. The engine in one car powers the wheels in another, instead of the wheels in the same car as the engine. Similary with propellors in boats. The part-whole relations do not limit the powers relation in an individual car or boat.

2. An engine in a boat powers the wheels in a car, instead of propellers. For example, engines can be required to power wheels using OWL cardinality restrictions, but this would apply to engines in boats as well as cars, and conversely for engines and propellers. Cardinality limitations compound the problem, because they also cannot be easily limited to apply within a single individual. For example, an engine could be required to power one or more wheels, but this would apply to engines in boats as well as cars.

3. An engine powers the two right wheels of the car, or even the spare wheel, instead of the two front wheels. The class of wheels does not account for the various roles a wheel may play in a car, and the powers relation by itself does not indicate which two of the wheels should receive power.

Some attempts to workaround these difficulties might be:

- Define subclasses of engines for cars and boats, and restrict the powers property so car engines only power wheels, and boat engines only power propellers. This is cumbersome and still allows the engine in one car to power the wheels in another.

- Use a constraint language to require that an engine in a car power the wheels in the same car. There are many of these constraints in a typical composite structure, sometimes interlocking, and the user must be careful to specify the desired amount of constraint. For example, the engine should be able to power other things in the car as necessary, but not to power things outside the car. See Appendix F.

- Use individuals as "prototypes" of cars, boats, and their parts. For example, a prototype for cars might be an individual with a prototypical engine and wheels as parts, where the engine powers the wheels. New cars are created by copying the prototype, rather than instantiating a class [Prototype]. Prototype-based languages might be more intuitive for subject matter experts, especially in domains requiring physical assemblies, but they also introduce significant redundancy and coordination problems when integrated with class-based languages such as RDF/S and OWL.

Implementers and users would probably prefer a more concise representation than those above, as proposed in this document. Subclasses, restrictions, and constraints can be generated from it for use in reasoners and user interfaces. It provides an approximation of prototype-based approaches more suitable for subject matter experts to use. See Section 5.

See Appendix D for more detail and illustrations of the above example. It only covers some of the difficulties of representing composite structure with classes and properties. Additional features of composites described below, such as ports, relating parts of parts, relations with parts, and relations as parts, further strain techniques limited to classes and relations/properties.

1.2 Contextualized Properties (Connectors)

The approach to composite structure in this document distinguishes two senses of the term "part":

1. Part as an individual, for example the engine in John's car with a unique serial number.
2. Part as a role, as in "part in a play."

These are mutually defining. Parts in the first sense (individuals) play parts in the second sense (roles). For example, an engine with a serial number (individual) plays the power source (role) in John's car. Roles map an individual whole into another individual playing that role in the whole. For example, the power producer role maps John's car to an engine with a specific serial number. This is taken to be equivalent to properties in RDFS and OWL. The power producer role is a property with the class Car as domain and the class Engine as range.

The basic idea of composite structure is to specify links between parts in the first sense (individuals) by defining connections between parts in the second sense (properties). This enables composite structure to be defined at the class level, but still apply to whatever individuals play roles in each instance of the class (are values of properties of each instance). For example, the engine in a car plays the role of power producer, the wheels play the role of impeller, and the powers connection between them refers to these roles as properties on the class of cars. This helps resolve the problems described in the Section 1.1 by enabling class-level specifications to refer individuals playing roles in instances of the class.

The extension to RDF/S follows this approach by introducing a new vocabulary element to specify that resources linked to the same individual whole are themselves linked by a property, called a *connector*. For example, the engine used in a particular car is the value of a property defined on the class of cars, and the same for the wheels that are the values of another property. The connector refers to these properties in the class of cars, along with the powers property, with the semantic constraint that within each instance of the class, the engine identified for that car will have a powers property with values equal to the wheels identified for that car. Since connectors only require classes and properties, the extension is defined on RDF/S, and can be combined with other extensions to RDF/S, such as OWL. An OWL version of the extension is also provided, see Appendix C.

Some terminology will facilitate the rest of the introduction:

- *Composite class*: a class with connectors, for example, the class of cars with a connector between engine and wheels.
- *Composite individual*: an instance of a composite class, such as a car with a specific identification number.
- *Interpart property*: a property that links parts of the same composite individual, as specified by a connector, such as the powers property between engines and wheels.

With the above as a basis, there are a number of elaborations and refinements:

1. Ports, and connectors between parts of parts

   Ports are parts that can be linked outside the whole. For example, a crankshaft is part of an engine that links to entities powered by the engine, and a hub is part of the wheel that links to entities that power the wheel. In a car these ports are connected. Connectors between ports must refer to two properties on each end to identify the individuals playing the ports, one to identify the individual part the port is on, and another to identify the individual playing the port itself. This double navigation, from the whole to part, then part to port, is necessary because the same port might be on more than one part. For example, the hubs on wheels in a car appear on multiple parts, not all of which necessarily receive power. In general, connectors can link parts at any level of depth in the composite, so connectors must specify a list of properties (*property path*) that starts with the composite whole and works down to the parts to be connected. See Section 2.

2. Inheritance of composite structure

   Connectors inherit from class to subclass, just as the properties they relate (connectors are kinds of properties in the Full level, see Section 4.) For example, the class of cars might have many subclasses with additional parts and connectors, but the connector between its engine and wheels still apply. This supports abstraction of parts and their interconnections. For example, a class of vehicles can be defined over cars and boats, with a power source powering an impeller specialized to wheels and propellers in the subclasses. See Section 2.

3. Link maintenance based on connectors

   Connectors enable maintenance of links between parts according to the connector. In particular, when an individual begins to play a part in a composite, links can be established between the individual and others playing other parts, according to the connectors of the composite structure. For example, when an engine is added to a car, it is linked to the wheels already in that car. When an individual stops playing a part in an instance of a composite class, links can be destroyed if they were established only by connectors of the composite structure. For example, when an engine is removed from a car, it is unlinked from the wheels of that car. Link maintenance can apply during the creation and desruction. For example, when the car is created, the engine and wheels used in the creation are linked, and when the car is destroyed, the links between engine and wheels in the car are destroyed, even if the engine and wheels are not. See Sections 1.4 and 4.

4. Composite properties

   Composite structure applies to properties and relations as well as classes. Properties and relations can have interconnected parts, for example, the powers relation between engine and wheel breaks down into other parts and relations, such as a clutch, transmission, driveshaft, and the connections between them. Composite properties have connectors referring to the individuals playing the domain and range roles of the property (RDF subject and object), which composite classes do not have. Composite properties enable unification of classes and properties under a more general concept that optionally has connectors specifying links to things outside the whole. See Section 3.

5. Connectors as parts

   Connectors can be parts themselves (in the second sense, as properties). This simplifies the extension and provides additional expressive capabilities. Connectors as properties enable them to be inherited with standard RDF/S semantics, instead of duplicating it. Connectors (as properties) can be related by other connectors, for example, when specifying message protocols. The values of connectors (as properties) are links between resources established by the connectors. This facilitates the maintenance of these links when composites are modified or destroyed, as described in point 4 above. See Section 4.

The functionality of connectors in this extension overlaps that of composite structure in the Unified Modeling Language, version 2 (UML 2) [UML2 Composition] [UML2]. UML supports composite associations (relations) of any arity, but currently not with connectors to the end objects of links, which makes them unusable as composite relations. Its connectors are limited to parts and ports (except see the UML extension for Systems Engineering [SysML]). UML supports connectors with more than two ends and relates connectors to communication and behavior. This RDF/S extension supports composite properties (binary relations), including connectors involving the subject and object of statements, and connectors more than two levels deep. It supports binary connectors only, because RDF and OWL properties are always binary, and currently does not relate connectors to communication and behavior specifications. UML's composite structure model is based on a long history of development and implementation in a wide variety of areas, including nuclear power, truck configuration, factory floor design, telecommunication, real-time applications, software architectures, and user interfaces (the first development of composite structure the author is aware of is due to Edmund Payne) [ADL] [SDL] [ROOM] [Acme] [Expert] [Aggregation] [ValueClass]. It is an example of configurational composition [Composition Kinds].

1.3 Amount of Abstraction: Property Classes in Lite, Medium, Full Levels

An important aspect of knowledge representation design is choosing the proper level of abstraction. Too little abstraction leads to redundancy and reduced maintainability. Too much abstraction can be difficult for implementers and end users to understand how to apply. Ideally, we define high-level abstractions, and specialize them for particular applications. This has the benefits of both abstraction and concrete categories. However, implementers and users often feel they need to understand the abstractions anyway, since the abstractions appear as supertypes of the specialized concepts they are using. How much abstraction is preferable will vary by audience and application. Users of high levels of abstraction will find their applications much easier to maintain, and this trades off with the time to learn the abstractions.

The approach to abstraction in this document is to provide three levels of RDF/S composite structure extensions, beginning with very little abstraction (Lite, Section 2), a medium amount (Medium, Section 3) and full abstraction with appropriate specialization (Full, Section 4). The main difference between the levels is the degree of applying the following vocabulary element:

> *Property class*: A class of the statements having a given property as predicate, or its subproperties.

Property classes enable representation of composite properties, connectors as parts, and simpler inheritance of composite structure, see Section 1.2. They are used in the three levels as follows (also see Appendix A):

1. Lite: no property classes. This level supports composite classes only, not composite properties, or connections as parts/properties. The inheritance of composite structure is added in the extension semantics, rather than reused from RDF/S. See Section 2.

2. Medium: uses property classes for composite properties, but not connectors as parts/properties. This level reuses Lite for composite classes, which applies to property classes like any other class. It supports unification of composite classes and properties. The inheritance of composite structure is still added in the extension semantics, rather than reused from RDF/S. See Section 3.

3. Full: uses property classes for composite properties and connections as parts/properties. It reuses RDF/S property inheritance of composite structure, and supports OWL cardinalities on connectors. It reuses RDF/S Property for some of the representation of connectors, and Lite and Medium for the rest. See Section 4.

The semantics of composite structure is the same for the above levels, and likewise for graphical and textual interfaces. For example, a composite class in any of the three levels has the same semantics, even though Full represents them a bit differently, and it can be viewed with the same user interfaces, such as graphical ones that imitate blueprints. Where the levels do not overlap in scope, the semantics is augmented in each case, and additional or modified user interfaces can be employed. For example, the Medium and Full levels support composite properties, with the same additional semantics, and are viewable with the same user interfaces, such as an enhanced blueprint interface that supports decomposition of lines.

1.4 Document Conventions

(Terminology conventions) In the following sections,

- "Statements" are true statements. The term includes true triples, except where limited to true RDF statements (reified triples).

- RDF statements are assumed to have exactly one predicate, one subject, and one object. If an RDF statement has multiple predicates, subjects, and objects, this is taken as equivalent to multiple statements for all the combinations.

- The "instances" of a class refer to individuals typed by the class (RDF type) or its subclasses. The "direct instances" of a class refer to individuals typed specifically by the class.

- References to the domain and range of properties include those implied by superproperties, and any limitations to them, for example, as specified with OWL restrictions.

- References to resources, including classes and properties, include those that are the same in the OWL sense (OWL sameAs).

(Extension definition conventions) Vocabulary elements defined in the following sections are accompanied by:

- Constraints on their instances, which define valid use of the elements (*vocabulary constraints* or "modeling constraints"). Some of these could be expressed in OWL, though they are not in this document, and checked with terminology ("T-Box") reasoners. Others could be checked with extensions to reasoners for composite structure.

- Constraints on instances of instances of the elements (for those elements that are or affect subclasses of RDF Class). This is a way of defining the semantics of the elements (*semantic constraints*).

Some general notes about the constraints:

- This extension intentionally does not specify whether, when, or how constraints violations are prevented from happening, or repairs made to restore conformance. Instances of the vocabulary elements (or instances of instances of elements) might be in violation of the constraints for a long time, short time, or never. Constraint violations might be repaired by changing or retracting statements that brought the system into violation, by changing or retracting other statements, or by asserting new statements.

- The semantic constraints do not apply when they depend on vocabulary elements that violate constraints. Specifically, when an instance of a vocabulary element violates a constraint, the semantic constraints involving instances of that instance do not apply.

- The constraints are expressed in English in this document, but could be in a more formal language, such as Common Logic [CL] or the Object Constraint Language [OCL], assuming there is a way for the constraint language to refer to RDF/S elements.

2. Lite Level

This section is divided into vocabulary, Section 2.1, and semantics, Section 2.2. See Appendix E.1 for examples.

2.1 Lite Vocabulary

A connector in the Lite (and Medium) level is a kind of resource (subclassed from the connector class used in all three levels):

```
<rdfs:Class rdf:about="&complmf;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdfs:Class rdf:about="&complm;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;Connector"/>
</rdfs:Class>
```

Vocabulary constraints:

1. See constraints at hasConnector below.

Connectors carry three pieces of information (summarized in Section 1.2):

1. The predicate for statements that will link individual parts according to the connectors. This is an interpart property.

```
<rdf:Property rdf:about="&complm:propertyThatConnects">
  <rdfs:label>propertyThatConnects</rdfs:label>
  <rdfs:domain rdf:resource="&complm;Connector"/>
  <rdfs:range rdf:resource="&rdf;Property"/>
</rdf:Property>
```

Vocabulary constraints:

1. Each connector has exactly one value for the propertyThatConnects property. *(For each connector, there is exactly one statement with the connector as subject and propertyThatConnects as predicate.)*
2. See constraints at hasConnector below.

2. The path of properties from whole to the individual parts being linked at the domain end of the connector.

```
<rdf:Property rdf:about="&complmf:domainEndOfConnector">
  <rdfs:label>domainEndOfConnector</rdfs:label>
  <rdfs:domain rdf:resource="&complmf;Connector"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>
```

Vocabulary constraints:

1. Each connector has exactly one value for the domainEndOfConnector property. *(For each connector, there is exactly one statement with the connector as subject and domainEndOfConnector as predicate.)*
2. Elements of the list are properties. *(In statements with domainEndOfConnector as predicate, the object will be a list of instances of RDF Property.)*
3. See additional constraints at hasConnector and Port below.

3. The path of properties from whole to the individual parts being linked at the range end of the connector.

```
<rdf:Property rdf:about="&complmf:rangeEndOfConnector">
  <rdfs:label>rangeEndOfConnector</rdfs:label>
  <rdfs:domain rdf:resource="&complmf;Connector"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>
```

Vocabulary constraints:

1. Each connector has exactly one value for the rangeEndOfConnector property. *(For each connector, there is exactly one statement with the connector as subject and rangeEndOfConnector as predicate.)*
2. Elements of the list are properties. *(In statements with this property as predicate, the object will be a list of instances of RDF Property.)*
3. See additional constraints at hasConnector and Port below.

Classes have connectors, using the following property:

```
<rdf:Property rdf:about="&complm:hasConnector">
  <rdfs:label>hasConnector</rdfs:label>
  <rdfs:domain rdf:resource="&rdfs;Class"/>
  <rdfs:range rdf:resource="&complm;Connector"/>
</rdf:Property>
```

The constraints after the first below ensure the list of properties form a "path" that begins at instances of the composite class and ends at classes compatible with the domain and range of the propertyThatConnects. The fifth constraint interprets an empty path as beginning and ending at the same instances of the composite class. This could be replaced by user-defined *self properties* with the composite individual as a value. These properties are taken to be ports, see below, and Section 2.2.3.3. See applications of empty paths in Section 5.

Vocabulary constraints:

1. Each connector must be in exactly one class.
   (For each connector, there is exactly one statement with the connector as subject and hasConnector as predicate.)

2. The beginning of a path of properties must have a domain compatible with the class that has the connector, and the properties after that must chain range to domain.
   (For each statement with hasConnector as predicate, if the domainEndOfConnector or rangeEndOfConnector lists of the statement object (a Connector) are nonempty, the first property in each list must have as its domain the statement subject (a Class) or one of its superclasses, the second must have as its domain the range of the first property in the list or one of its superclasses, the third must have as its domain the range of second property in the list or one of its superclasses, and so on.)

3. The end of the domainEndOfConnector path of properties must have a range compatible with the domain of the propertyThatConnects.
   (For each statement with hasConnector as predicate, if the domainEndOfConnector list of the statement object (a Connector) is nonempty, the last property in it must have as its range the domain of the propertyThatConnects of the statement object (a Connector) or one of its subclasses.)

4. The end of the rangeEndOfConnector path of properties must have a range compatible with the range of the propertyThatConnects.
   (For each statement with hasConnector as predicate, if the rangeEndOfConnector list of the statement object (a Connector) is nonempty, the last property in it must have as its range the range of the propertyThatConnects of the statement object a (Connector), or one of its subclasses.)

5. If a domainEndOfConnector or rangeEndOfConnector path of properties is the empty list, constraints [3] and [4] apply, substituting the class having the connector for the range of the last property.
   (For each statement with hasConnector as predicate, if the domainEndOfConnector or rangeEndOfConnector lists of the statement object (a Connector) are empty, constraints [3] and [4] apply, respectively, substituting the statement subject (a Class) for the range of the last property in the list.)

This extenion divides properties into those that identify resources inside and outside the composite. For example, a property identifying the engine in a car is distinguished from a property identifying the driver. The driver is not in the car composite, but related to the car. The vocabulary below defines a special kind of property to identify resources the user takes to be inside the composite, the *part properties* (UML calls these composite aggregation associations, notated with a black diamond symbol, and gives them whole-to-part destruction semantics [UML2 Composition] [UML2]). Part properties are disjoint from the *nonpart properties*, which identify resources outside the composite, such as the driver, or that are not directly related to composition, such as those giving the weight or color of the car. These can be used as a workaround for composite properties and n-ary relations in the Lite level, see Section 3.2. *Port properties* are a kind of part that has values with properties linking outside the composite. For example, a property of car identifying the steering wheel is a port because its value has links to the driver. Port properties are disjoint from the part properties that do not link directly to the outside of the composite, the *internal part* properties.

A vocabulary constraint is added to connectors preventing them from specifying links between parts of parts that are not ports. For example, the composite structure for a car cannot have a connector involving the pistons, because the pistons are not a port on the engine. However, connectors can have any property on their ends, including nonparts. This means they can specify links to resources outside the composite, which is important for representing composite properties, see Section 3.2.

```
<rdfs:Class rdf:about="&complmf:PartProperty">
  <rdfs:label>PartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdfs:Class rdf:about="&complmf:NonpartProperty">
  <rdfs:label>NonpartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdfs:Class rdf:about="&complmf:PortProperty">
  <rdfs:label>PortProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;PartProperty"/>
</rdfs:Class>

<rdfs:Class rdf:about="&complmf:InternalPartProperty">
  <rdfs:label>InternalPartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;PartProperty"/>
</rdfs:Class>
```

The first four constraints below ensure the subclasses are disjoint and completely cover their superclass (OWL disjointWith and unionOf, Appendix C). The fifth is additional to the ones on domainEndOfConnector and rangeEndOfConnector properties of Connector above. The sixth is partly a semantic constraint, but could be purely a vocabulary constraint if a class of property is defined that has statements which always have the same subject and object.

Vocabulary constraints:

1. A resource cannot be an instance of PartProperty and NonpartProperty at the same time.
   (A resource cannot be typed by both PartProperty and NonpartProperty, or their subclasses, at the same time.)

2. Every property is an instance of PartProperty or NonpartProperty.
   (Any resource typed by RDF Property must be also be typed either PortProperty and NonpartProperty, or their subclasses.)

3. A resource cannot be an instance of PortProperty and InternalPartProperty at the same time.
   (A resource cannot be typed by both PortProperty and InternalPartProperty, or their subclasses, at the same time.)

4. Every instance of PartProperty must be an instance of PortProperty or InternalPartProperty.
   (Any resource typed by PartProperty must also be typed by both PortProperty or InternalPartProperty, or their subclasses.)

5. In a domainEndOfConnector or rangeEndOfConnector property path of a connector, all but the first property must be a port, or not a part.
   (For every instance of Connector, if the domainEndOfConnector property has a list of more than one element, each element after the first must either be an instance of PortProperty or NonpartProperty. Same for the rangeEndOfConnector property.)

6. Properties with values that are always the same as the subject (self properties) are ports.
   (The instances of Port include all properties for which all statements with the property as predicate have the same subject and object.)

7. See constraints in Section 3.1.

Properties are more useful with connectors when their values are "directly" linked to the whole, rather than transitively. For example, the powers connector between engine and wheels in earlier examples is between properties on cars that identify just the engines and just the wheels, rather than properties that include the parts of the engines and wheels. However, connectors will often refer to parts more than one level down from the whole, for example, connecting the crankshaft of the engine to the hubs of the wheels with property paths containing two direct part properties, see the port example in Figure E.1.4 of Appendix E.1. Direct properties can be subproperties of transitive ones [Part Whole OWL]. The above constraints do not prevent properties from being transitive, but connectors to them would establish links that are not usually desired. For example, if the engine and wheel properties were transitive, the powers connector between them would link all the parts of the engine to all the parts of the wheels. This is technically accurate in the sense that the entire engine ultimately generates power and the entire wheel receives it, but a subject matter expert concerned with parts of parts will probably focus on the parts of the engine and wheel more immediately connected.

Since this document does not specify whether constraint violations are prevented, or even enforced at all, see Section 1.4, an implementation could support properties that are not typed by any of the subclasses above, even though the subclasses completely cover the superclasses. Vocabulary and semantic constraints that do not depend on the subtypes still apply to these properties. It might be possible to deduce the classification in some cases. For example, a property that is the second element in a property path could be reclassified as a port or nonpart to satisfy the vocabulary constraint above, whereupon the semantics for these subtypes applies.

The vocabulary constraints above do not prevent an RDF datatype property from being a part, or being the end of a connector, even though it is unlikely an RDFS literal will have an interpart property value. RDF/S does not seem to prevent this, however, because the domain of a property is a class and datatypes are a kind of class (UML supports properties on datatypes). This extension is agnostic on the issue. Constraints against it can be added later if necessary. ([SysML] divides nonpart properties into those for objects and those for data values.)

Extensions can generalize the above classes as needed, for example as described in [Composition Kinds]. Within the namespaces defined in this document, the term "part" refers to configurational composition.

2.2 Lite Semantics

The semantics of connectors is described in this document as constraints on values of interpart properties, such as the powers property in the examples of previous sections. There are two kinds of semantic constraint:

1. The set of resources that must be values of an interpart property (*minimum value set*).
2. The set of resources from which values of an interpart property are drawn (*maximum value set*).

The first is a subset of the set of values for an interpart property, while the second is a superset (the terms minimum and maximum value sets for a property are adapted from [ValueClass], the notion of minimum value class is due to Anne Paulson). They are analogous to OWL restrictions hasValue and allValuesFrom, which give required values for properties and limit the range, respectively. For example, the powers connector in the car example establishes a minimum value set for the engine in a car that includes the wheels on the same car. It also establishes the same set as maximum if each wheel cannot have more than one powers connector apply to it, see Section 2.2.1. The first kind of constraint can be affected by limitations on the links otherwise dictated by the minimum value set (Section 2.2.2). The second kind of constraint must support cases where the same resource is constrained by multiple connectors for the same interpart property, and where resources are linked to others outside a composite (Section 2.2.3).

It is important to note that the second constraint only classifies interpart property values, it does not give "permission" for values of an interpart property to be any element in the maximum set. Other constraints might apply that limit the actual set of interpart property values to less than the maximum set, and this does not contradict the maximum set (though it can contradict the minimum set, see Section 2.2.2). The terms "maximum" and "minimum" are a bit misleading in this respect, but follow the OWL terminology for cardinalities.

These sets facilitate describing the semantics:

- The *whole resource set* of each connector is the set of instances of the class having the connector (hasConnector). In the car examples, the whole resources are the instances of the Car class.

- The *domain resource set* for each pair of connector and whole resource of the connector is the set of resources found by traversing from the whole resource along the domain end property path (domainEndOfConnector) of the connector. For example, the composite structure for a car might connect the crankshaft part of the engine to the hub parts of the wheels. The domain end property path contains the power source property first, then the property on engines identifying the crankshaft. The domain resource set for this connector and an individual car will be the crankshaft found by traversing these two properties beginning with the individual car. An empty path is interpreted as specifying the whole resource itself as the value to be linked at the domain end.

- A *range resource set* for each pair of connector and whole resource of the connector is the set of resources found by traversing from the whole resource along the range end property path (rangeEndOfConnector) of the connector. In the car example above connecting crankshaft to hub, the range end property path contains the impeller property first, then the property on wheels identifying the hub. The range resource set for this connector and an individual car will be the hubs found by traversing these two properties beginning with the individual car. An empty path is interpreted as specifying the whole resource itself as the value to be linked at the range end.

A domain resource set *corresponds* to a range resource set, and vice versa, if they are for the same connector and whole resource. In the example above, the domain resource set of the powers connector on a particular individual car will identify an individual engine for the domain resource set, and the corresponding range resource set will identify the individual wheels.

This section covers the simplest case first in Section 2.2.1. It places the most constraint on values of interpart properties by equating minimum and maximum value sets. Then it examines specific cases that loosen constraints, by reducing minimum sets in Section 2.2.2, and expanding them in Section 2.2.3. Section 2.2.4 combines the previous ones to give the complete semantic constraints.

2.2.1 Simple Case (Minimum and Maximum Value Sets Are the Same)

In the simple case, the minimum and maximum value sets are the same. This is when all the following are true:

- Interpart properties are not limited in a way that affects the minimum value sets.
- Properties of any resource are constrained by connectors at most once.
- Properties being connected are not known to be ports or nonparts.

More about the above in Sections 2.2.2 and 2.2.3.

The semantics of the simple case is that a connector specifies the minimum and maximum value sets for its interpart property (propertyThatConnects) on members of its domain resource sets. The minimum and maximum value sets are the same, and for each element of a domain resource set, the value sets include all and only the values of the range resource set corresponding to that domain resource set. In the car example, which has a powers connector between the properties for power source and impeller, the engine of each car (identified by the power source property) must power all and only the wheels of the same car (identified by the impeller property). See example in Figures E.1.1 through D.1.3 in Appendix E.1. Since domain and range resource sets apply to whole resources, which are instances of composite classes, connectors "inherit" to subclasses. For example, a superclass for vehicles could be introduced above car and boat that carries properties for power sources and impellers generally, along with the connector between them. This applies to cars and boats, which have specialized power sources and impellers. See example in Figure E.1.6 in Appendix E.1. This semantics assumes the authors of a composite structure will specify additional connectors if they want the engine to power more parts of the car, see Section 2.2.3.2. The individual wheels and engine in a car may change over time, and consequently the maximum and minimum value sets do also. This is analogous in OWL to changing the range of a property by changing the individuals in its range class.

2.2.2 Reducing Minimum Value Sets

Limitations on the values of interpart properties can contradict the simple minimum value sets of Section 2.2.1. In the earlier example, if maximum cardinality restrictions, as in OWL, limit an engine to power no more than two wheels, then a car with four wheels will give a minimum set for the powers property with too many wheels. Other limitations might be expressed in constraint languages that rule out some of the minimum value set. These limitations effectively reduce the minimum value set, but the semantics of the simple case does not provide for that.

One approach to reducing minimum value sets is to define additional vocabulary constraints. The particular example above can be addressed with constraints requiring the domain and range end property paths to result in cardinality limitations (by multiplication of cardinality limitations along the path) consistent with the those of interpart properties. However, vocabulary constraints will not handle maximum cardinality limitations when the same resource can be in more than one composite individual, because the set of property values is the union over all the composites in which the resource participates, see Section 2.2.3.2 (the Lite vocabulary constraints in Section 2.1 prevent contradictions by limitations on the range of a property, when they are read with the document conventions in Section 1.4 that define property range as including range limitations).

Another approach is to have the minimal value semantics "yield" to any limitations. In the previous example, if engines are limited to powering only two wheels, then under a yielding semantics powering any two of the four wheels would satisfy the connector constraint. This requires reasoning from lack of decidability, in particular that the minimum set semantics and the cardinality restrictions contradict each other regarding whether to power the excess wheels. It also introduces nondeterminism in which two wheels to power. This is desirable in some applications. For example, a car has four wheels and four axle ends, but it does not matter which wheel goes with which axle end. The properties on the car for wheels and axle ends have cardinality of exactly four, and the connector between them has an interpart property with cardinality limitation of exactly one. Minimum set semantics would link every wheel to every axle end, but yielding to the cardinality limitations links each wheel to exactly one axle end, without specifying which goes with which ("array" pattern [UML2]). Specialized reasoners for composite structure can yield to cardinality limitations efficiently if they have access to metadata.

UML takes the second approach above, at least for cardinality limitations. The number of interpart property values specified by minimum set semantics is reduced as needed by any maximum cardinality limitations. It also supports cardinalities on connectors, that can limit the numbers of values beyond the cardinalities on interpart properties. For example, an engine can power any number of wheels in general, but in cars is limited to four, or in some cars two. Cardinalities on connectors could be added to this extension if desired. They are a more expressive form of the cardinalities on connector properties in the Full level, see Section 4.

The approach taken in this document is to leave the semantics as is, which enables prevention or repair of contradictions between the minimum value set and value limitations, as desired by the application. For example, contradictions between the maximum cardinality and minimum value sets could be repaired by inferring that some resources are equivalent, or that the range end property should have fewer values. This aspect of the extension could be optional in future versions, see Section 5.

The situations above do not affect maximum value sets, see Section 2.2. The maximum value set is a superset of the interpart property values, and is still a superset, even if the values are limited below the minimum value set. In the example above, if the cardinality limitation for engines is to power only two of the four wheels in a car, the maximum value set is still the four wheels.

2.2.3 Expanding Maximum Value Sets

This section describes three ways to expand maximum values sets. Section 2.2.3.1 outlines limitations analogous to 2.2.2. Section 2.2.3.2 describes the unioning semantics needed when connectors apply multiple times to the same individual. Section 2.2.3.3 covers the maximum set semantics for ports and nonparts.

2.2.3.1 Limitations on Interpart Property Values

Limitations on the values of interpart properties can contradict the simple maximum sets of Section 2.2.1, though examples seem to indicate this is a less useful case than limitations on the minimum set. In the car example, if minimum cardinality limitations, as in OWL restrictions, are defined to require an engine to power at least four wheels, then a three-wheeled car will give a maximum set for the powers property with too few wheels. Other limitations might be expressed in constraint languages that require more values also. These limitations effectively expand the maximum value set.

The approaches to expanding maximum value sets for limitations are similar to reducing minimum value sets: vocabulary constraints, yielding, and leaving the semantics as it is. Vocabulary constraints for minimum cardinality limitations have the same problem as for maximum cardinality limitations, see Section 2.2.2. Yielding allows the maximum set to be expanded to include the minimum cardinality limitation. The approach taken in this document is to leave the semantics as is (as amended in the following sections), which enables prevention or repair of contradictions between the minimum value set and value limitations, as desired by the application. For example, contradictions between the minimum cardinality and maximum value sets could be repaired by inferring additional (existential) values for the range end property. This aspect of the extension could be optional in future versions, see Section 5.

2.2.3.2 Connectors Applying Multiple Times to the Same Individual

The simple maximum value sets of Section 2.2.1 are too narrow when the same resource is constrained multiple times by connectors for the same interpart property. For example, when the same person teaches at more than one school, the simple semantics would intersect the maximum value sets for the students in each school, which is likely to be empty. This section modifies the semantics so the maximum value set is the union of those specified by the simple maximum sets. This is analogous in OWL to enlarging a class used for allValuesFrom, either by adding other classes to it as a union or by adding individuals. The particular cases are:

1. An individual can play the same role in different instances of the same composite class. For example, the same person might be a teacher at multiple schools, where the composite class for school connects a teacher role to a student role. The maximum value set for this person's students is the union of students at the multiple schools.

2. More than one connector for the same interpart property and domain end property can apply to the same composite individual. For example, an engine in a car might power the wheels, generator, and

oil pump, each identified by different properties on the car and related to the engine by different connectors for the powers interpart property. The maximum value set for the powers property on an individual engine is the union of those specified by each connector. This case can also arise from connectors on different composite classes that apply to the same composite individual.

3. Generalizing the above, more than one connector for the same interpart property and different domain end properties, that happen to have the same values in the composite individual. In the example above this would occur if the power source property on a car were replaced with multiple properties for the power sources of the wheels, generator, and oil pump, each of which is filled by the same individual engine in a particular car. The maximum value set for the powers property on an individual engine is the union of those specified by each connector, even though the connectors have different domain ends.

4. Combining the above, more than one connector for the same interpart property and different domain end properties, that happen to have the same resources as values in different composite individuals. This can be generalized to connectors on separate composite classes. The maximum value set of the interpart property on the resources is the union of those specified by each connector, regardless of which composite individual or domain end it applies to, and which composite classes have which connectors.

The semantics above is unnecessary if the same resource is never constrained more than once by connectors with the same interpart property. For example, if engines are constrained to be in no more than one vehicle at a time, and to be the value of no more than one property on a vehicle that has a connector with the same interpart property. The first could be a maximum cardinality constraint on a "part-of" property of engine for its vehicle, while the second could be disjoint property constraints that prevent the engine from being the value of more than one property on a car (as proposed for OWL 1.1 [OWL 1.1]), or vocabulary constraints on connectors disallowing more than one connector for the same interpart property.

The situations above do not affect minimum value sets, which specify a subset of the interpart property values for each resource, connector, and whole resource. This is still a subset, even if the same resource is constrained multiple times by connectors for the same interpart property. In the teacher example above, each school at which a person teaches gives a subset of the person's students, and these are still subsets of the students they teach, no matter how many schools they teach at.

2.2.3.3 Ports and Nonparts

The semantics of maximum values sets as modified so far is too narrow, because in some situations it can prevent a resource from referring to others outside the composite, even when the resource is at the "boundary" of a composite, or not part of the composite at all, The cases are:

1. A resource might be identified by a domain end property path that has only port properties (which includes the empty path referring to the whole resource). For example, a wheel is a port on a car, because it delivers power to the road. It might have a powers connector to the engine, for situations where the car is moving faster than the engine, such as downshifting. When combined with the powers connector from engine to wheels, the semantics so far would prevent the wheel from transmitting power to anything outside the car, including the road, unless an additional connector is specified for it, see the "environment" composite below.

2. A resource might be identified by a domain end property path that has only nonpart properties. For example, the driver of a car might be identified by a nonpart property of the Car class. Connectors from this property would prevent links for the interpart property between the driver and other things outside the car, unless there are additional connectors for those also. For example, the driver operates the steering wheel, which might be specified as a connector in the Car class, but the driver might also operate a cell phone.

3. Combining the above, a resource might be identified by a domain end property path that has both port and nonpart properties (which includes the empty path referring to the whole resource), and that identifies a resource that is not an internal part, including those outside the composite. Connectors from this path would prevent links for interpart properties between the resource and others outside the composite, even when the resource is not in the composite, unless there are additional connectors for those also.

It might be a reasonable methodology to require the additional connectors mentioned above, which would essentially mandate every composite class be accompanied by another one for its environment. This would ensure the class is always used as intended. For example, a Car class would be accompanied by a composite for specifying the kinds of roads it can be driven on, the licenses required for people that drive it, minimum oxygen content of the air, and so on. In the driver example above, it might be good for the Car class to limit the other things a driver can operate while operating the steering wheel. The drawback is that all the circumstances of the car's use would need to be anticipated, or the elements of unexpected circumstances would need to be reclassified so they fit in the composite structure of the environment. This extension chooses not to enforce this methodology, but an option could be added to indicate when it must be used, whereupon the semantics proposed below would not apply, see Section 5.

This section refines the maximum value set semantics for connectors from ports and nonparts to apply only to values of an interpart property that are inside the composite. Specifically, for connectors from a port or nonpart property, or more generally from a domain end property path that has only ports and nonpart properties, the maximum value set semantics only applies to values of the interpart property that are reachable by traversing part properties from the whole resource of the connector. In the first example above, the road is not part of the car, so a powers connector in cars from the wheels to the engine does not prevent the wheel from delivering power to the road. Similarly in the driver example above, the driver is not part of the car, so the operates connector in cars from the driver to steering wheel does not prevent the driver from operating a cell phone.

The above refinement is second order, because it requires quantifying over the part properties of an individual, deducing a generalized form of transitive closure. It can be reduced to first order by assuming a finite number of part properties for the composite individual (as supported in Common Logic [CL]), or by asserting statements that enable disproving the existence of part properties not explicitly defined in the ontology (a kind of completion statement [Completion]). The refinement is also beyond first order because it requires deducing a generalized form of transitive closure, but this can be reduced to first order with completion statements that enable disproving the existence of links from resources inside the closure to those outside the closure. Specialized reasoners for composite structure can support this refinement efficiently if they have access to metadata.

Note the above refinement does not affect connectors from property paths that begin with internal parts, or more generally properties not known to be ports or nonparts. For example, a connector in cars from the crankshaft of the engine to the wheels will apply the simple maximum value set semantics of Section 2.2.1 to the values of powers property of the crankshaft, not the refined semantics above, even though the crankshaft is a port of the engine. This prevents the crankshaft from powering anything else, unless additional connectors are defined. The refinement also does not affect connectors from property paths that begin with properties typed only by RDF Property, which means it is not known if the property is a port or nonpart. The refinement only applies if the first property of the domain end path of the connector is typed by Port or NonPart or one of their subclasses.

2.2.4 Complete Lite Semantics

This section folds together the semantics of the previous ones, using the sets introduced in Section 2.2. An additional characteristic of domain resource sets is useful in describing the semantics:

> A domain resource set is *relevant* to a pair of resource and property if the set includes the resource and is for a connector of the property (propertyThatConnects), or one of its subproperties. For example, the domain resource set for the powers connector and an individual car is relevant to the engine in the car and the powers property.

The semantic constraints are:

1. *(Minimum value set semantics)* If a resource has a property with values (is the subject of a statement) and has domain resource sets relevant to the resource and the property, then the values of the property must include all values in range resource sets corresponding to the relevant domain resource sets.

2. *(Maximum value set semantics)* If a resource has a property with a value (is the subject of a statement) and domain resource sets relevant to the resource and the property, then there must be a relevant domain resource set for which one of the following is true:
    - The domain resource set has a corresponding range resource set containing the value.
    - The property path used to identify elements of the domain resource set has only port and nonpart properties, and the domain resource set is for a whole resource from which the value cannot be reached by traversing part properties from the whole resource, or from values of part properties of the whole resource, recursively.

The above may appear to quantify over properties, but is only written that way to simplify the text. The quantification is actually over the whole resources of the connectors, see Appendix F. The aspects beyond first order are in the maximum value set semantics for values of port and nonpart properties on the domain end of connectors with only port and nonpart properties in the path, second bullet above, and see Section 2.2.3.3.

As with all constraints in this document, the semantic constraints on interpart property values above intentionally do not specify whether, when, or how violations are prevented or repaired. For example:

- When an individual begins to play a part in a composite, links can be established with the individual according to the connectors of the composite structure, or these links can be established later, or the added part might be rejected because it does not have the required links.

- When an individual no longer plays a part in a composite, links can be destroyed that were due to the connectors of the composite structure, or these links can be destroyed later, or the removed part might be added back because the links established by connectors still exist, or no action might be taken because the links are also due to other composite structures or other justifications.

3. Medium Level (Composite Properties)

The Medium level includes Lite from Section 2, and introduces property classes to support composite properties. Property classes are both properties and classes, and their instances are the statements having the property (or subproperty) as predicate. Composite structure can be defined for property classes like any other class. In the earlier example of a car engine powering wheels, the engine has a powers property with the wheels as values (as specified by a connector), and the property might decompose into all the parts and interconnections involved in transmitting power from the engine to the wheels, such as the clutch, transmission, and powers relations between them. A property class for the powers property represents this in the usual way for composite classes, except it uses the RDF subject and object properties on RDF statements as ends of connectors (or paths beginning with these properties), to identify the resources linked by interpart properties. In this example the subject and object properties identify the engine and wheels.

This section is divided into vocabulary and semantics, Section 3.1, and the unification of composite classes and relations, Section 3.2. See Appendix E.2 for examples.

3.1 Medium Level Vocabulary and Semantics

The Medium level includes the Lite level, and adds vocabulary for property classes.

```
<rdfs:Class rdf:about="&compmf:PropertyClass">
  <rdfs:label>PropertyClass</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>
```

The second constraint below is additional to NonPart in Section 2.1.

Vocabulary constraints:

1. Property classes are classes of RDF statements.
(Instances of PropertyClass are subclasses of RDF Statement.)

2. The RDF subject and object properties are nonparts.
(The RDF properties subject and object are instances of Nonpart.)

3. The values of the RDF subject and object properties must be in the domain and range of the property, respectively.
(The values of the RDF properties subject and object are instances of the domain and range classes of the property, respectively.)

Property classes have one semantic constraint, as shown below. The semantics of the Medium level is the same as Lite in other respects.

1. The instances of a property class are all the RDF statements that have the property class (as a property) as predicate, or its subproperties.
(Instances of an instance of PropertyClass are all and only the RDF statements that have the instance of property class (a property) as predicate, or a subproperty of that property.)

The RDF subject and object properties of a composite property are nonparts, because a link between two resources (RDF statement or triple) is usually taken as separate from the resources. For example, destroying the link does not destroy the resources. This is important in the unification of composite classes and properties, see Section 3.2.

If the subclasses are property classes, then they are for subproperties, and their instances will be all the statements for a subproperty, or its subproperties. Otherwise, the subclasses can use some other criteria to identify subsets of the property class. In the example above, a property class for powers could have subproperty classes for hydraulic-powers, mechanical-powers, and electrical-powers, as well as subclasses for statements authored by a particular person, which are not property classes. The statements of a property class that are for that particular property, rather than its subproperties, will be direct instances of (be typed by) the property class, or its non-property subclasses. This will be true even if there are statements for subproperties that have the same subject and object. For example, statements for the powers property (powers as predicate) will not be instances of hydraulic-powers, mechanical-powers, and electrical-powers, though they may be instances of subclasses for particular authors. There may be statements for hydraulic-powers, mechanical-powers, and electrical-powers that have the same subject and object as a statement for powers.

The above does not quite reflect the semantics that properties are predicates classifying tuples, where there is one tuple for each subject and object. RDF statements are semantically the classification of a tuple, and it is a bit of a workaround to use them as instances of properties. The extension could define a vocabulary for tuples, but it would introduce a concept redundant with RDF statements and their subject/object properties. The proposal in this extension seems like an adequate compromise for now, and can be updated if other reification vocabularies become available. (UML is closer to tuple semantics with a concept that generalizes classes and associations (Classifier), and well as a concept that merges them (AssociationClass). However, it introduces bags and sequences for the ends of associations, requiring duplicate tuples that must be distinguished.)

Composite structures for property classes apply to subproperties, even when there are no property subclasses defined for them. This is because RDFS semantics requires that any two resources linked by the subproperties are also linked by the powers property (for every statement with one of the subproperties as predicate, there is another with powers property as predicate that has the same subject and object). For example, any composite structure for the powers property class applies to statements with the powers property as predicate, including those that exist because the resources are also linked by the subproperties hydraulic-powers, mechanical-powers, or electrical-powers. When there are property classes defined for subproperties, composite structure inherits as usual.

Composite properties can have ports. In the above example, the composite powers property between engine and wheels might have a port for oil to flow through to the various parts of the property. This begins to treat composite properties as first-class parts. The next step for that is to have the statements linking engine and wheels be values of properties on the composite statement. This is taken up in Full, see Section 4.

As a matter of good practice, subjects of a composite property might only be used on the domain end of connectors (domainEndOfConnector), while objects of a composite property might only be used on the range end (rangeEndOfConnector). This would avoid the unusual situation of a composite property that maps from instances of one class (the domain) to instances of another (the range), while the connectors inside the property provide a map in the other direction. Since the semantics of RDF/S treats properties as sets of tuples, rather than computational navigation, this constraint is not required in this extension. It can be added as necessary.

The representation of property classes could alternatively be a subclass of RDF Class with a property that identifies another property. This avoids the multiple classification of properties, for example, a port property that is also a property class, and mixing of metaclasses that some might find undesirable. However, the alternative introduces an element that is one-to-one with another (it would be redundant to have two property classes for the same property). It also requires an inverse property between the property classes and its property, and the overhead of traversing between them. For these reasons, this document uses the mixin approach, but the effect would be the same with the alternative.

3.2 Unification of Composite Classes and Properties

There is a strong similarity between composite properties and composite classes with nonparts for the subject and object of the property (this observation is due to Roger Burkhart and [Aggregation]). In the example of Section 3.1, the composite property between engine and wheels could be represented as a composite class with nonpart properties for the engine and wheels. In general, if a composite class with nonparts is extended to indicate which of the nonparts are to act as the subject and object (the "ends" of the relation) [Relations], it is hard to see a significant difference from a composite property. One could imagine query languages that "hopped over" the composite individual between engine and wheels, thereby providing the equivalent of property traversal (navigation in UML), and graphical interfaces that draw it as a line, as well as OWL restrictions on the end properties equivalent to domain and range. The only representational difference seems to be that a composite class with nonparts supports the equivalent of bags and sequences of property values, whereas composite properties support only sets of values. This means the equivalent of OWL cardinality on composite classes with nonparts requires a constraint to exclude the bag and sequence cases.

A composite class with more than two nonparts is essentially an n-ary relation (n>2), again assuming classes are extended to indicate which nonparts are the ends of the relation (this can be a separate RDF/S extension). For example, an n-ary relation for the act of giving could be represented as a class with nonpart properties for giver, receiver, and given thing ([Nilsson] [Nary OWL]). The technique can be applied to the RDF Statement class, using the RDF subject and object properties as two of the nonparts, and another property for the thing given. [Nary OWL] suggest this approach is inappropriate for n-aries, because it represents a statement about a statement, rather than an extension to the information in the statement itself (in the triple). For example, from their viewpoint one should not construct the ternary above based on binary statements identifying the giver and receiver, using those as the domain of a property identifying the given thing. However, it seems reasonable that the given thing is specified with a statement about the binary link between giver and receiver, because in RDF terms the subject and object of a statement are properties with statements as domain, at the same time they provide information about the content of the triple. The technique proposed here just uses additional properties of statements for the additional "ends" of the n-ary relation. The first-order translations of composite structure also assume n-aries are a special kind of statements about statements, see Appendices F.2 and F.3.

The strong similarity between nonparts and property/relation ends suggests a more abstract notion of composition that unifies composite classes and relations [Aggregation]. A composite class without nonparts cannot relate external resources to each other. It is strictly a "standalone object." A composite relation has at least one nonpart, and can act as a conventional relation if there are at least two. The unification of composite classes and relations is a composite that might or might not have nonparts, which in this document is a composite class. The unification encompasses relations not typically taken as such:

- Relations with multiple values for end properties [CDIF] [Relations]. For example, a purchase relation between people and products where a single instance of purchase has one person and multiple products.

- Relations that have exactly one nonpart [Aggregation]. These are somewhere between classes (zero nonparts) and binary relations (two nonparts).

4. Full Level (Connectors as Properties)

The Full level includes Lite from Section 2 and Medium from Section 3, but uses property classes to define connectors in a more integrated way in RDF/S. Connectors in Full are a special kind of property, rather than general resources as in Lite and Medium. The range of a connector property is a property class for its interpart property, and its values represent the links between individual parts. Connectors (as properties) inherit to subclasses with RDF/S semantics, rather than duplicating it in the semantic constraints. They can be on the domain and range ends of other connectors (connectors between connectors). The values of connector properties facilitate the maintenance of interpart links when composites are modified, or destroyed, by identifying the links due to the composite structure. Composite properties are supported in Full, because it includes property classes from Medium.

The Full level includes:

- The domainEndOfConnector and rangeEndOfConnector properties on connectors as defined in Lite, see Section 2.1.

- The Medium level's addition to Lite (property classes, see Section 3).

and changes these aspects of the Lite level:

- Redefines the Lite Connector to be a kind of property, replacing hasConnector. A connector in the Full level is a special kind of property and inherits to subclasses like any other property.

- Requires connectors (as properties) to have property classes as ranges, replacing propertyThatConnects. The range of the connector is the interpart property of the connector. The values of a connector on its whole resources are the statements that link the resources identified by the domain and range end properties of the connector (domainEndOfConnector, rangeEndOfConnector).

The vocabulary of Connector in the Full level is:

```
<rdfs:Class rdf:about="&compf;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
  <rdfs:subClassOf rdf:resource="&complmf;Connector"/>
</rdfs:Class>
```

Vocabulary constraints:

1. The range of connectors are property classes.
   (The range of an instance of Connector (a property) is an instance of PropertyClass.)

2. The rest of the constraints are the same as for Lite hasConnector, except modified to use the domain of the connector as the composite class that has the connector (hasConnector), and the range of the connector (a property class) as the interpart property (propertyThatConnects).

The semantics of connectors in the Full level is the same as in Lite level (Section 2.2), except as amended below. The second constraint reflects a portion of the minimum value set semantics from the point of view of whole resources of the connector, see Section 2.2.4.

1. The description of the semantics in Section 2.2 is modified in the same way as vocabulary constraint [2] above.

2. The values of connector properties on whole resources of the connector are the RDF statements corresponding to the interpart property values of the individuals at the domain end of the connector.
   (The values of an instance of Connector (a property) on a whole resource of the connector are all the RDF statements that have as a subject a value on the whole resource of the domain end property of the connector (domainEndOfConnector), and have as a predicate the range of the connector (a property class), and have as an object a value on the whole resource of the range end property of the connector (rangeEndOfConnector).)

If cardinality limitations, such as OWL restrictions, are applied to connector properties, they may contradict the minimum and maximum value set semantics of the connector. The discussion of cardinalities of interpart properties in Lite applies, see Sections 2.2.2 and 2.2.3.1. (Cardinalities on connector properties are less expressive than UML's multiplicities on connectors, which can be applied to each end independently.)

The values of connector properties in Full facilitate the maintenance of interpart links when composites are modified or destroyed, as described in Section 1.2. The statements to be retracted are values of connector properties. For example, if an engine is removed from a car, the links to the wheels should be destroyed, and these are identified by the connector (as a property) on the individual car. Similarly when the car is destroyed. The connector properties act as a record of the dependencies of interpart property values on the whole resource.

5. Future Work

Possibilities for future work include:

- *Extensions of description logics* (subsumption and classification algorithms, including membership sufficiency, and application of the empty connector domain and range path to reflexivity, inverses, and transitivity).

- *Interaction with other areas of ontology research* (upper ontology, ontology development and evaluation, including mappings from existing other formats such as UML, and application to semantic interoperability).

- *Semantic issues* (for example, whether to prevent links to internal parts from outside the composite, semantic options for the yielding form of value semantics, and other variations).

- *Reasoning optimizations* (efficient specialized reasoners based on common constraint repair and enforcement policies).

- *Other integration with RDF/S and OWL* (a compatible model theoretic semantics and textual syntax).

- *Bridge between prototypical and class-based approaches* (generating composite classes from composite individuals, and views emphasizing interlinked individuals).

- *Generalized connectors* (temporal constraints applicable to process ontologies, and other more general constraints between parts).

The current extension can be integrated with work from the description logic (DL) community. Some DL research on part-whole relations includes part-part [Part Whole OO] [PartOf Framework] [Part Whole Extension] ("horizontal constraints", "pp-constraints"), though they seem to be limited to direct parts (property paths with exactly one element), and do not address ports, composite properties, or connectors as properties. The closest DL constructs to connectors are complex role inclusion and role value maps, which are loosely analogous to "uncontextualized" maximum set semantics, see discussion at the first two expressions in Section F.1. DL reasoning is undecidable for these constructs [Role Inclusion][Role Value Map], but may be decidable when contextualized and combined with minimum set semantics. This is plausible since a decidable restriction of complex role inclusion has already been found [RIQ].

It would be useful to extend DL sumbsumption and classification algorithms to support composite structure [PartOf Framework]. For example, if two classes have connectors for the same end properties and interpart properties, but the interpart properties of one class are subproperties of the interpart properties of another, then the first is a subclass of the second, at least as far as composite structure is concerned. For classification, an interesting topic is sufficient conditions for class membership based on connectors. The extension as currently defined gives only necessary conditions, as RDF/S does (*partial* in OWL). If a resource is an instance of a composite class, then it must conform to the minimum and maximum value set semantics for connectors of the class and its superclasses. There are a least two kinds of sufficient condition for composites:

- If a resource conforms to the semantics for whole resources of a composite class, then it is an instance of that class, assuming it satisfies the necessary conditions of the class. In the earlier examples, a resource with properties identifying an engine and wheels that have a powers link between them is an instance of the class Car, assuming the same properties are used to identify the engine and wheels. This kind of sufficiency is only useful if the end properties of the connector can have values in instances of other classes than the composite one, in particular, if the domain end property has a wider domain than the composite class. Otherwise, the resource can be classified under the composite class without considering connectors at all. Since properties that can be used as ends of connectors usually make more sense when the connector is defined also, the composite structure sufficiency will give the same results as the end property sufficiencies, unless there is a contradiction with the composite structure semantics, whereupon the statements need to be repaired anyway.

- If a set of interlinked resources conforms to structure of a composite class, then the resources are part of some instance of the composite class. For example, an engine may power wheels, even if they are not known to be part of any whole. The second kind of sufficiency would infer existence of an instance of the Car class with properties identifying the orphaned engine and wheels (and in Full, a connector property for powers statements). This is a kind of pattern matching or analogical reasoning, a generalization of subgraph isomorphism [Analogy].

Another interaction with description logic is that the empty property path can represent contextualized versions of reflexivity, inverses, and transitivity. The empty path identifies the whole resource, so a connector with empty paths on both ends specifies that the interpart property is reflexive for instances of the class having the connector (called "local reflexivity" in OWL 1.1 [OWL 1.1], and "self properties" in Section a 2.1). The interpart property might not be reflexive in general, if its domain is wider than the class having the connector. A connector from the whole to a (nonpart) property, combined with a connector in the opposite direction, specifies that the interpart properties are inverses for instances of the class and values of the connected property ("local inverses"). The interpart properties might not be inverses in general, if the domain of the first interpart property is wider than the class having the connector, or if the domain of the second interpart property is wider than the range of the connected property. A more general form of this kind of inverse can have nonempty property paths on both ends, whereupon the conditions under which the interpart properties are inverses is more narrow. A connector from the whole to a property path with more than one element represents a limited form of complex role inclusion that specifies the whole resource has a relation to others identified by a specific chain of properties [Role Inclusion]. For example, a cousin is a child of the siblings of a parent. If all the elements of the property path are the same property, then it can be considered a limited form of transitivity.

Other areas of ontology work also apply to composite structure. For example, part-part relations is an aspect of upper ontologies and ontology evaluation, in particular, DOLCE's notion of unity [DOLCE], as reflected in OntoClean [OntoClean], underlies the part taxonomy of Section 2.1. And the notion of identity requires multiple ways of referring to the same entity, as with properties, and an equality relation between the references (see generalized connectors below). Ontology development and application can also be facilitated by mapping between this extension and other composition models, such as UML's [UML2 Composition] [UML2]. This would support semantic interoperability that depends on extracting ontologies from software specifications [AMIS], and for efficient implementation of ontologies [OntoMDA]. This could be as an addition to the mappings between RDF/S/OWL and UML being developed as part of the Ontology Definition Metamodel [ODM]. Composite structure can also be implemented in extensible ontology tools that support properties of properties [Protege].

One semantic question is whether to prevent links to internal parts from outside the composite, for example, from the driver of a car directly to the pistons. The current maximum value set semantics applied to connectors from internal parts only constrains values of interpart properties on those parts, not external entities. This is addressed by methodologies that specify a composite structure for the environment, see Section 2.2.3.3, because the fifth vocabulary constraint for the property taxonomy in Section 2.1 prevents connectors to internal parts from outside the composite (also see semantic options below). For applications not using the environment methodology, the maximum set semantics could be extended to entities unrelated to a composite, to invalidate links from them to internal parts. This begins to touch on the topic of mereotopology, which concerns spatial relationships of everyday shapes [Mereotopology 1][Mereotopology 2].

Identifying common constraint repair and enforcement policies would facilitate efficient reasoning. For example, when an engine is added to a car that has wheels, the expected repair would be to link the engine to the wheels, rather than remove the engine, or remove the wheels, or conclude that the car is not actually a car. Under this policy, the reasoner detects when a part is added to the whole resource, and reacts by establishing links as specified by the connectors under minimum value set semantics. Similarly, if a link is created between an engine in a car and a wheel another car, the expected enforcement would be to destroy the link, rather than move the wheel from one car to the other, or remove the engine and wheel from the cars, or conclude that the cars are not actually cars. Under this policy, the reasoner destroys links to resources outside the composite when they violate maximum set semantics.

Future versions of the extension can provide semantic options, to provide more flexibility in the design decisions taken in this document. For example, some applications may require the yielding form of value set semantics, see Sections 2.2.2 and 2.2.3.1. This could be indicated by a Boolean property on connectors, or their class, to specify that the connector should be interpreted with or without yielding. Another Boolean property could indicate that the port/nonpart semantics for maximum value sets should not be used, see Section 2.2.3.3, effectively mandating the use of the environment methodology.

The extension can be further integrated with RDF/S and OWL by providing a compatible model theoretic semantics and textual syntax. The semantics is currently defined as constraints on which statements are allowed and required, to validate whether to proceed with an RDF- and OWL-compliant semantics. Concise textual syntaxes for connectors compatible with N3 and OWL abstract syntaxes are important for ontology development. Connectors can be specified in N3, of course, see Appendices D and E, but a shorthand can be defined for connectors, as in [PartOf Framework].

The extension can be further integrated with RDF/S and OWL by providing:

- Model theoretic semantics and abstract syntax compatible with those of RDF and OWL [RDF Semantics] [OWL Semantics]. The current extension semantics is defined as constraints on which statements are allowed and required, to validate whether to proceed with an RDF- and OWL-compliant semantics.

- Concise textual syntaxes for connectors compatible with N3 and OWL abstract syntaxes. Connectors can be specified in N3 [N3], of course, see Appendices D and E, but a shorthand can be defined for connectors, as in [PartOf Framework].

Composite structure can approximate prototypical techniques. For example, a prototypical composite individual with interlinked individual parts can be the source for generating a composite class, in the simplest case using one part property per prototypical part. Composite structure also gives an accurate picture of how individuals are linked, rather than the aggregate information class-based models provide, as shown by the example in Section 1.1. This enables subject matter experts to treat the model as if it were actually interlinked instances, and see the class-like aspects of it as annotations on the "individuals".

This extension can be combined with possible RDF/S extensions for process specification. This relies on the fact that executing processes, must conform to their specifications, in the same way individuals conform to classes that type them. Process specifications can be represented as classes of their executions, as in [UML2 Activities] [UML2] [SysML], and as composite structures of their subprocesses. Connectors can represent temporal constraints between subprocess executions, as in [PSL][OSERA]. Combined with other extensions for messaging, and the Full level for connectors between connectors, protocols can be represented as temporal constraints on messages passing between agents.

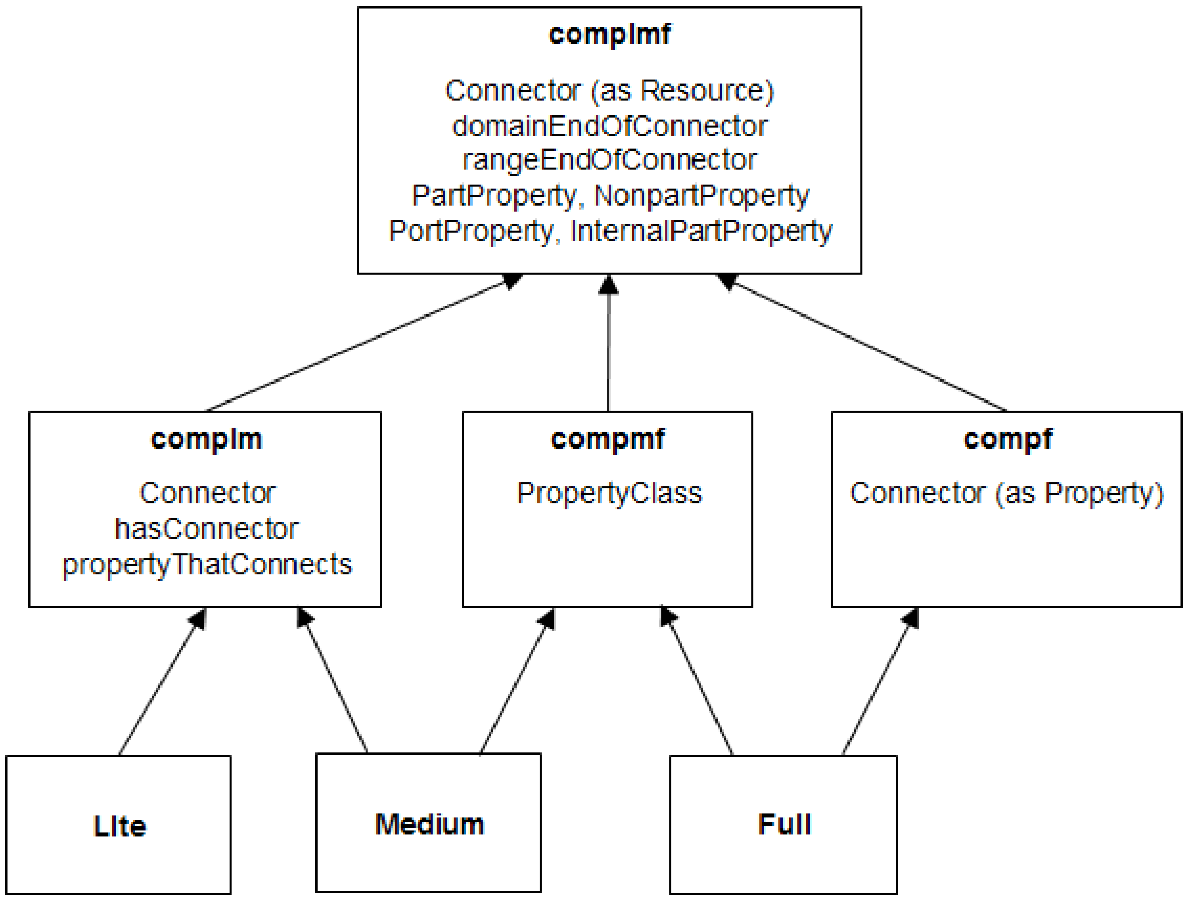
Future extensions can also generalize connectors, for example, to specify any constraint on the values of their end properties, as in [SysML], rather than just on values of properties of their domain ends. This includes the simple constraint that values of properties are the same (*binding connector* in [SysML]), which is used in representing reusable parametric equations [Parametrics]. Combined with the process specification extensions above, it can also represent constraints on message passing between the values of properties at the ends of connectors, as in [UML2 Composition] [UML2]. These connectors can address situations where one object identifies another by receiving it as an argument, rather than having it as a property value.

Acknowledgments

Thanks to Evan Wallace, Vei-Chung Liang, Xuan Zha, and Dragan Gasevic for their assistance in preparing this article, and to members of the SysML and UML 2 submission teams, and members of the IntelliCorp research and application groups.

Appendix A. RDF/S Composite Structure Quick Reference

This figure shows the dependency among the extension namespaces, and which are included in what level.



A.1 RDFS Classes

Lite, Medium, and Full:

Class name	Comment	Subclass of
complmf:Connector	Resource specifying links between values of properties of (or property paths from) the same resource.	rdf:Resource
complmf:PartProperty	Class of properties identifying parts of a composite.	rdf:Property
complmf:NonpartProperty	Class of properties for objects or values not directly related to composition.	rdf:Property
complmf:PortProperty	Class of properties identifying parts that link to outside a composite.	complmf:PartProperty
complmf:InternalPartProperty	Class of properties identifying parts that do not link to outside a composite.	complmf:PartProperty

Lite and Medium:

Class name	Comment	Subclass of
complm:Connector	Specialization of complmf:Connector to be domain of Lite and Medium properties.	complmf:Connector

Medium and Full:

Class name	Comment	Subclass of
compmf:PropertyClass	Merge of RDF property and RDFS class. Instances are all the statements having a given property as predicate.	rdfs:Class, rdf:Property

Full:

Class name	Comment	Subclass of
compf:Connector	Specialization of complmf:Connector to be property specifying links between values of properties of (or property paths from) the same resource.	rdf:Property

A.2 RDF Properties

Lite, Medium, and Full:

Property name	Comment	domain	range
complmf:domainEndOfConnector	The values of this property (or property path) are the subject ends of links specified by the connector.	complmf:Connector	rdf: List

complmf:rangeEndOfConnector	The values of this property (or property path) are the object ends of links specified by the connector.	complmf:Connector	rdf:List

Lite and Medium:

Property name	Comment	domain	range
complm:hasConnector	Property that links a class to its (complm) connectors.	rdfs:Class	complm:Connector
complm:propertyThatConnects	Property that links the values specified by the domain and range ends of a connector (predicate of statements specified by a connector).	complm:Connector	rdf:Property

Appendix B. RDF/S for Composite Structure

This appendix gives the vocabulary for extending RDF/S for composite structure, as defined in previous sections. The "nist" locations are only for illustration.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
        <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
        <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
        <!ENTITY complm 'http://www.nist.gov/comp/complm#'>
        <!ENTITY complmf 'http://www.nist.gov/comp/complmf#'>
        <!ENTITY compmf 'http://www.nist.gov/comp/compmf#'>
        <!ENTITY compf 'http://www.nist.gov/comp/compf#'>
        <!ENTITY compgen 'http://www.nist.gov/comp/compgen#'>
]>
<rdf:RDF xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:complm="&complm;"
xmlns:compf="&compf;"
xmlns:complmf="&complmf;"
xmlns:compmf="&compmf;"
xmlns:rdf="&rdf;"
xmlns:rdfs="&rdfs;"
xmlns:compgen="&compgen;">

<rdfs:Class rdf:about="&complmf;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdf:Property rdf:about="&complmf;domainEndOfConnector">
  <rdfs:label>domainEndOfConnector</rdfs:label>
  <rdfs:domain rdf:resource="&complmf;Connector"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>
```

```
<rdf:Property rdf:about="&complmf;rangeEndOfConnector">
  <rdfs:label>rangeEndOfConnector</rdfs:label>
  <rdfs:domain rdf:resource="&complmf;Connector"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

<rdfs:Class rdf:about="&complm;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;Connector"/>
</rdfs:Class>

<rdf:Property rdf:about="&complm;propertyThatConnects">
  <rdfs:label>propertyThatConnects</rdfs:label>
  <rdfs:domain rdf:resource="&complm;Connector"/>
  <rdfs:range rdf:resource="&rdf;Property"/>
</rdf:Property>

<rdf:Property rdf:about="&complm;hasConnector">
  <rdfs:label>hasConnector</rdfs:label>
  <rdfs:domain rdf:resource="&rdfs;Class"/>
  <rdfs:range rdf:resource="&complm;Connector"/>
</rdf:Property>

<rdfs:Class rdf:about="&complmf;PartProperty">
  <rdfs:label>PartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdfs:Class rdf:about="&complmf;NonpartProperty">
  <rdfs:label>NonpartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdfs:Class rdf:about="&complmf;PortProperty">
  <rdfs:label>PortProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;PartProperty"/>
</rdfs:Class>

<rdfs:Class rdf:about="&complmf;InternalPartProperty">
  <rdfs:label>InternalPartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;PartProperty"/>
</rdfs:Class>

<rdfs:Class rdf:about="&compmf;PropertyClass">
  <rdfs:label>PropertyClass</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<rdfs:Class rdf:about="&compf;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
  <rdfs:subClassOf rdf:resource="&complmf;Connector"/>
</rdfs:Class>
</rdf:RDF>
```

Appendix C. OWL for Composite Structure

This appendix gives the vocabulary for extending OWL for composite structure, similar to the RDF/S extension in Appendix B. It is modified to only specialize OWL, without significant impact on the expressiveness of the extension:

- Assumes all part properties are object properties (but not the reverse). Specializes OWL ObjectProperty into part properties and nonpart object properties.
- Datatype properties are always nonparts. No specializations of OWL DatatypeProperty is required.
- Together the above mean that nonpart object properties and datatype properties cover all nonparts. However, there is no explicit class for nonparts, to avoid introducing a superclass of OWL property concepts.

The subclasses of part properties is unchanged, consisting of internal and port properties.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
        <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
        <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
        <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema#'>
        <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
        <!ENTITY complm 'http://www.nist.gov/comp/complm#'>
        <!ENTITY complmf 'http://www.nist.gov/comp/complmf#'>
        <!ENTITY compmf 'http://www.nist.gov/comp/compmf#'>
        <!ENTITY compf 'http://www.nist.gov/comp/compf#'>
]>
<rdf:RDF xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:complm="&complm;"
xmlns:compf="&compf;"
xmlns:complmf="&complmf;"
xmlns:compmf="&compmf;"
xmlns:rdf="&rdf;"
xmlns:rdfs="&rdfs;"
xmlns:owl ="&owl;">

<owl:Class rdf:about="&complmf;Connector">
  <rdfs:label>Connector</rdfs:label>
</owl:Class>

<owl:FunctionalProperty rdf:about="&complmf;domainEndOfConnector">
  <rdfs:label>domainEndOfConnector</rdfs:label>
  <rdfs:domain rdf:resource="&complmf;Connector"/>
  <rdfs:range rdf:resource="&rdf;List"/>
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
</owl:FunctionalProperty>

<owl:FunctionalProperty rdf:about="&complmf;rangeEndOfConnector">
  <rdfs:label>rangeEndOfConnector</rdfs:label>
  <rdfs:domain rdf:resource="&complmf;Connector"/>
  <rdfs:range rdf:resource="&rdf;List"/>
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
</owl:FunctionalProperty>
```

```
<owl:Class rdf:about="&complm;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;Connector"/>
</owl:Class>

<owl:FunctionalProperty rdf:about="&complm;propertyThatConnects">
  <rdfs:label>propertyThatConnects</rdfs:label>
  <rdfs:domain rdf:resource="&complm;Connector"/>
  <rdfs:range rdf:resource="&rdf;Property"/>
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
</owl:FunctionalProperty>

<owl:InverseFunctionalProperty rdf:about="&complm;hasConnector">
  <rdfs:label>hasConnector</rdfs:label>
  <rdfs:domain rdf:resource="&owl;Class"/>
  <rdfs:range rdf:resource="&complm;Connector"/>
  <rdf:type rdf:resource="&owl;ObjectProperty"/>
</owl:InverseFunctionalProperty>

<owl:Class rdf:about="&owl;ObjectProperty">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&complmf;PartProperty"/>
        <owl:Class rdf:about="&complmf;NonpartObjectProperty"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="&complmf;PartProperty">
  <rdfs:label>PartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;ObjectProperty"/>
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="&complmf;InternalPartProperty"/>
        <owl:Class rdf:about="&complmf;PortProperty"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="&complmf;NonpartObjectProperty">
  <rdfs:label>NonpartObjectProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;ObjectProperty"/>
  <owl:disjointWith rdf:resource="&complmf;PartProperty"/>
</owl:Class>

<owl:Class rdf:about="&complmf;PortProperty">
  <rdfs:label>PortProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;PartProperty"/>
</owl:Class>
```

```
<owl:Class rdf:about="&complmf;InternalPartProperty">
  <rdfs:label>InternalPartProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&complmf;PartProperty"/>
  <owl:disjointWith rdf:resource="&complmf;PortProperty"/>
</owl:Class>

<owl:Class rdf:about="&compmf;PropertyClass">
  <rdfs:label>PropertyClass</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
  <rdfs:subClassOf rdf:resource="&owl;Class"/>
</owl:Class>

<owl:Class rdf:about="&compf;Connector">
  <rdfs:label>Connector</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
  <rdfs:subClassOf rdf:resource="&complmf;Connector"/>
</owl:Class>
</rdf:RDF>
```

Appendix D. Examples Without Composite Structure

This is RDF/S for the example in Section 1.1 in RDF/S. It uses N3 notation for readability [N3] (namespace prefixes are omitted for brevity). The original car/boat example is due to Michael Williams.

```
Car        rdf:type rdfs:Class .
Boat       rdf:type rdfs:Class .
Engine     rdf:type rdfs:Class .
Wheel      rdf:type rdfs:Class .
SpareWheel rdf:type rdfs:Class .
Propeller  rdf:type rdfs:Class .

powers rdf:type    rdf:Property ;
       rdfs:domain Engine .

engineInCar rdf:type    rdf:Property ;
            rdfs:domain Car ;
            rdfs:range  Engine .

poweredWheelInCar rdf:type    rdf:Property ;
                  rdfs:domain Car ;
                  rdfs:range  Wheel .

spareWheelInTrunk rdf:type    rdf:Property ;
                  rdfs:domain Car ;
                  rdfs:range  SpareWheel .

engineInBoat rdf:type    rdf:Property ;
             rdfs:domain Boat ;
             rdfs:range  Engine .

propellerInBoat rdf:type    rdf:Property ;
                rdfs:domain Boat ;
                rdfs:range  Propeller .
```

Figure D.1: Schema Example without Composite Structure

The following triples are valid for the above schema, showing the problems listed in Section 1.1. It assumes different resources represent different actual individuals, using, for example, OWL differentFrom.

```
AnEngineInACar  rdf:type  Engine .
AWheel          rdf:type  Wheel .
ASpareWheel     rdf:type  SpareWheel .
ACar rdf:type          Car ;
     engineInCar       AnEngineInACar ;
     poweredWheelInCar AWheel ;
     spareWheelInTrunk ASpareWheel .
AnEngineInACar powers ASpareWheelInTrunk .


AWheelInAnotherCar rdf:type Wheel .
AnotherCar rdf:type  Car ;
           wheelInACar AWheelInAnotherCar .
AnEngineInACar powers AWheelInAnotherCar .


APropeller      rdf:type Propeller .
AnEngineInABoat rdf:type Engine .
ABoat rdf:type        Boat .
      engineInBoat    AnEngineInABoat ;
      propellerInBoat APropeller .
AnEngineInACar powers APropeller .
AnEngineInABoat powers AWheel .
```

Figure D.2: Valid Triples for Schema Example Without Composite Structure

The following RDF/S modifies the schema above, to resolve the second and third problems listed in Section 1.1. It introduces classes for engines in cars, engines in boats, powered wheels, and spare wheels in trunks. It uses OWL to limit the powers property in cars and boats to wheels and propellers respectively, and to make car engines disjoint from boat engines, and spare wheels in trunks disjoint from powered wheels.

```
PoweredWheelInCar rdf:type        owl:Class ;
                  rdfs:subClassOf Wheel .
poweredWheelInCar rdf:type    rdf:Property ;
                  rdfs:domain Car ;
                  rdfs:range  PoweredWheelInCar .
PowersRestrictionToPoweredWheel rdf:type  owl:Restriction ;
                                owl:onProperty powers ;
                                owl:allValuesFrom PoweredWheelInCar .
CarEngine rdf:type        owl:Class ;
          rdfs:subClassOf Engine ;
          rdfs:subClassOf PowersRestrictionToPoweredWheel ;
          owl:disjointWith BoatEngine .
engineInCar rdf:type    rdf:Property ;
            rdfs:domain Car ;
            rdfs:range  CarEngine .
SpareWheelInTrunk rdf:type        owl:Class ;
                  rdfs:subClassOf SpareWheel;
                  owl:disjointWith PoweredWheelInCar .
```

```
    spareWheelInTrunk rdf:type    rdf:Property ;
                      rdfs:domain Car ;
                      rdfs:range  SpareWheelInTrunk .

    PowersRestrictionToPropeller rdf:type owl:Restriction ;
                                 owl:onProperty powers ;
                                 owl:allValuesFrom Propeller .
    BoatEngine rdf:type        owl:Class ;
               rdfs:subClassOf Engine ;
               rdfs:subClassOf PowersRestrictionToPropeller ;
               owl:disjointWith CarEngine .
    engineInBoat rdf:type    rdf:Property ;
                 rdfs:domain Boat ;
                 rdfs:range  BoatEngine .
```

Figure D.3: Augmented Schema Example Without Composite Structure

With the augmented schema, the individual engines and wheels are classified to satisfy the restrictions on the powers property. These classifications could be inferred from the properties they are values of (engineInCar, engineInBoat, poweredWheelInCar, spareWheelInTrunk), due to the restrictions.

```
    AnEngineInACar  rdf:type  CarEngine .
    AnEngineInABoat rdf:type  BoatEngine .
    AWheel          rdf:type  PoweredWheel .
    ASpareWheel     rdf:type  SpareWheelInTrunk .
```

Figure D.4: Changes to Valid Triples Using Augmented Schema Without Composite Structure

The augmented schema and triples above invalidate some of the undesired triples from earlier, as shown below. This resolves the last two problems listed in Section 1.1.

```
    AnEngineInACar  powers ASpareWheelInTrunk .
    AnEngineInACar  powers APropeller .
    AnEngineInABoat powers AWheel .
```

Figure D.5: Invalid Triples for Augmented Schema Example Without Composite Structure

To invalidate triples like the above in general requires subclasses for every part connected to at least one other, which usually is all of them (these subclasses are called "qua concepts" in [Qua Link], and "current types" in [Roles]). It must also include restrictions on every interproperty, such as powers. If the same part is connected to multiple others along the same interpart property, then the restriction must be to the union of the range of the multiple parts (as properties), see Section 2.2.3.2. If a part is also a port, then the solution requires limitations that are beyond OWL restrictions, see Section 2.2.3.3.

The following undesirable triple is still valid:

```
    AnEngineInACar powers AWheelInAnotherCar .
```

Figure D.6: Valid Triple for Augmented Schema Example Without Composite Structure

To invalidate the triple above in general requires subclasses for individuals, for example, the engine in John's car. This is impractical, of course, and suggests using constraint languages and rule systems instead. These must be carefully specified to provide the desired semantics, as given in Section 2.2, see

Appendix F.1. Like the subclasses above, the constraints and rules will be very repetitive and tedious for a typical industrial composite structure.

The augmented schema does not require the engine in a car to power the wheels, or to power the propeller in a boat, see Section 2.2.1. For example, the triples below are required, assuming the individual engines, wheel, and propeller are playing their respective parts. OWL support for requiring values (minCardinality and hasValue) is not expressive enough for composite structure.

```
AnEngineInACar powers AWheel .
AnEngineInABoat powers APropeller .
```

Figure D.7: Desired Triples Not Required by Augmented Schema Example Without Composite Structure

Appendix E. Examples With RDF/S Composite Structure

This appendix gives examples of the three levels of the RDF/S Composite Structure extension.

E.1 Examples With Lite RDF/S Composite Structure

The following elements are added to the schema in Figure D.1 in Appendix D. There are two connectors, one in Car, another in Boat.

```
PowersConnectorInCar rdf:type                     complm:Connector ;
                     complm:propertyThatConnects  powers ;
                     complmf:domainEndOfConnector (engineInCar) ;
                     complmf:rangeEndOfConnector  (poweredWheelInCar) .
Car complm:hasConnector PowersConnectorInCar .

PowersConnectorInBoat rdf:type                     complm:Connector ;
                      complm:propertyThatConnects  powers ;
                      complmf:domainEndOfConnector (engineInBoat) ;
                      complmf:rangeEndOfConnector  (propellerInBoat) .
Boat complm:hasConnector PowersConnectorInBoat .
```

Figure E.1.1: Schema Example with Composite Structure

The semantics of the above invalidates the undesirable triples in Appendix D, as shown below, because the powers link from the engine can only be to the wheels that are values of the poweredWheelInCar property in the same car as the engine, not the spare wheel, or impellers in other cars and boats. This assumes that different resources are different actual individuals, for example, using OWL differentFrom, and that the property values are disjoint across vehicles, for example, using cardinality restrictions from part to whole to prevent a wheel from being in more than one car.

```
AnEngineInACar  powers ASpareWheelInTrunk .
AnEngineInACar  powers APropeller .
AnEngineInABoat powers AWheel .
AnEngineInACar  powers AWheelInAnotherCar .
```

Figure E.1.2: Invalid Triples from Appendix D for Schema Example With Composite Structure

The following triples are required under the above schema by the minimum value set semantics of Section 2.2.4, assuming the engines, wheel, and propeller are playing their respective parts.

```
AnEngineInACar  powers AWheel .
AnEngineInABoat powers APropeller .
```

Figure E.1.3: Required Triples for Schema Example With Composite Structure

The following elements are added to the schema above to show an example of ports. They introduce a port on engine (a crankshaft) and on wheel (a hub), and connector between them. Connectors between ports, and parts of parts in general, use property path lists with more than one property.

```
Crankshaft rdf:type rdfs:Class .
Hub rdf:type rdfs:Class .

crankshaftInEngine rdf:type    complmf:PortProperty ;
                   rdfs:domain Engine ;
                   rdfs:range  Crankshaft .

hubInWheel rdf:type    complmf:PortProperty ;
           rdfs:domain Wheel ;
           rdfs:range  Hub .

crankshaftPowers rdf:type   rdf:Property ;
                 rdfs:domain  Crankshaft .

PortPowersConnectorInCar rdf:type                     complm:Connector ;
                         complm:propertyThatConnects  crankshaftPowers ;
                         complmf:domainEndOfConnector (engineInCar
                                                       crankshaftInEngine) ;
                         complmf:rangeEndOfConnector  (poweredWheelInCar
                                                       hubInWheel) .
Car complm:hasConnector PortPowersConnectorInCar .
```

Figure E.1.4: Schema Example of Ports

The following triples are valid for the above schema, in addition to those from earlier. The last is required by the minimum value set semantics of Section 2.2.4, given the other triples.

```
ACrankshaft rdf:type  Crankshft .
AnEngineInACar crankshaftInEngine ACrankshaft .
AHub rdf:type  Hub .
AWheel hubInWheel AHub .
ACrankshaft crankshaftPowers AHub .
```

Figure E.1.5: Valid Triples for Port Schema Example

The following elements introduce the superclass Vehicle above Car and Boat from the first schema of this section (the one without ports) to show an example of connector inheritance. The superclass consolidates the two powers connectors of the subclasses into one. Superclasses PowerSource and Impeller are introduced above engines, wheels, and propellers. A generalization of the powers property is introduced with all power sources as its domain. The properties of Vehicle are specialized and restricted in Car and Boat using OWL (assuming some representation of complete subproperties [MaxSubprop], similar to property redefinition in UML). The connector in Vehicle applies to instances of its subclasses, so the valid documents above are still valid. Connector inheritance is defined in the Lite level semantics, which is included in Medium. In Full, connectors are parts/properties and inherit by the usual RDF/S semantics.

```
# Vehicle

Vehicle     rdf:type rdfs:Class .
PowerSource rdf:type rdfs:Class .
Impeller    rdf:type rdfs:Class .
powerSourceInVehicle rdf:type    complmf:InternalPartProperty ;
                     rdfs:domain Vehicle ;
                     rdfs:range  PowerSource .
impellerInVehicle rdf:type    complmf:InternalPartProperty ;
                  rdfs:domain Vehicle ;
                  rdfs:range  Impeller .
sourcePowers rdf:type    rdf:Property ;
             rdfs:domain PowerSource ;
             rdfs:range  Impeller .
powers rdf:type           rdf:Property ;
       rdfs:domain        Engine .
       rdfs:subPropertyOf sourcePowers .
PowersConnectorInVehicle rdf:type                     complm:Connector ;
                     complm:propertyThatConnects  sourcePowers ;
                     complmf:domainEndOfConnector (powerSourceInVehicle) ;
                     complmf:rangeEndOfConnector  (impellerInVehicle) .
Vehicle complm:hasConnector PowersConnectorInVehicle .


# Car

Car    rdfs:subClassOf Vehicle .
Engine rdfs:subClassOf PowerSource .
Wheel  rdfs:subClassOf Impeller .
engineInCar rdf:type            rdf:Property ;
            rdfs:domain        Car ;
            rdfs:range         Engine ;
            rdfs:subPropertyOf powerSourceInVehicle .
poweredWheelInCar rdf:type           rdf:Property ;
                  rdfs:domain        Car ;
                  rdfs:range         Wheel ;
                  rdfs:subPropertyOf impellerInVehicle .
CarRestrictionToPowerSource rdf:type          owl:Restriction ;
                            owl:onProperty    powerSource ;
                            owl:allValuesFrom Engine .
CarRestrictionToImpeller rdf:type          owl:Restriction ;
                         owl:onProperty    impeller ;
                         owl:allValuesFrom Wheel .
Car rdfs:subClassOf CarRestrictionToPowerSource ;
    rdfs:subClassOf CarRestrictionToImpeller .


# Boat
Boat      rdfs:subClassOf Vehicle .
Propeller rdfs:subClassOf Impeller .
engineInBoat rdf:type           rdf:Property ;
             rdfs:domain        Boat ;
             rdfs:range         Engine ;
             rdfs:subPropertyOf powerSourceInVehicle .
```

```
propellerInBoat rdf:type          rdf:Property ;
                rdfs:domain       Boat ;
                rdfs:range        Propeller ;
                rdfs:subPropertyOf impellerInVehicle .
BoatRestrictionToPowerSource rdf:type          owl:Restriction ;
                             owl:onProperty    powerSource ;
                             owl:allValuesFrom Engine .
BoatRestrictionToImpeller rdf:type          owl:Restriction ;
                          owl:onProperty    impeller ;
                          owl:allValuesFrom Propeller .
Boat rdfs:subClassOf BoatRestrictionToPowerSource ;
     rdfs:subClassOf BoatRestrictionToImpeller .
```

Figure E.1.6: Schema Example for Inheritance of Composite Structure

A web page demonstrating composite structure inheritance in a pre-UML language is available [Inheriting Part Demo].

E.2. Examples With Medium RDF/S Composite Structure (Composite Properties)

The following elements modify the port schema in Appendix E.1 to show composite properties. A subproperty (powersWheel) of the powers property is defined between engines and wheels, so it can have connectors that do not affect other applications of the property, for example, in boats. It is a property class that has one part (a clutch) and two connectors, from the crankshaft port on the engine to the clutch, and from the clutch to the hub port on the wheel. The connectors form a "sequence" from a port on the subject of the property, to a part, to a port on the object of the property. Note this is not equivalent to mathematical function composition ("o"), because the connectors specify links between individuals that are not the engine or wheels. The clutch is identified by a property of the property class. The ports are identified by traversing along the subject and object properties of the property class, and then the port properties for engines and wheels, respectively (crankshaftInEngine, hubInWheel). The connectors are for different properties (crankshaftPowers and clutchPowers), to prevent an infinite recursion, and because crankshafts and hubs are not engines and wheels (crankshaftPowers and clutchPowers are not subproperties of powers). A more general powers property could be defined that has the others as subproperties.

```
powersWheel rdf:type           compmf:PropertyClass ;
            rdfs:subclassOf    rdf:Statement ;
            rdfs:range         Wheel ;
            rdfs:subPropertyOf powers .
Clutch rdf:type rdfs:Class
clutchInPowersWheel rdf:type    rdf:Property ;
                    rdfs:domain powersWheel ;
                    rdfs:range  Clutch .
crankshaftPowers rdf:type    rdf:Property ;
                 rdfs:domain Crankshaft .
clutchPowers rdf:type    rdf:Property ;
             rdfs:domain Clutch .

CrankshaftPowersConnectorInPowersWheel
    rdf:type                     complm:Connector ;
    complm:propertyThatConnects  crankshaftPowers ;
    complmf:domainEndOfConnector (subject crankshaftInEngine) ;
    complmf:rangeEndOfConnector  (object  clutchInPowersWheel) .
```

```
ClutchPowersConnectorInPowersWheel
    rdf:type                     complm:Connector ;
    complm:propertyThatConnects  clutchPowers ;
    complmf:domainEndOfConnector (subject clutchInPowersWheel) ;
    complmf:rangeEndOfConnector  (object  hubInWheel) .

powersWheel complm:hasConnector CrankshaftPowersConnectorInPowersWheel ,
                                ClutchPowersConnectorInPowersWheel .
```

Figure E.2.1: Schema Example for Composite Property With One Part and Two Connectors

The following triples are valid for the above schema. The first triple modifies the corresponding powers triple in Appendix E.1 to use the powersWheel property class. The third reifies the powersWheel triple and identifies its clutch. The last two are required by the composite structure of powersWheel defined above.

```
AnEngine powersWheel AWheel .
AClutch rdf:type Clutch .
AnEngineInACarPowersAWheel rdf:type  powersWheel ;
                           rdf:predicate powersWheel ;
                           rdf:subject AnEngineInACar ;
                           rdf:object AWheel ;
                           clutchInPowersWheel AClutch .
ACrankshaft crankshaftPowers AClutch .
AClutch clutchPowers AHub .
```

Figure E.2.2: Valid Triples for Schema Example for Composite Property With One Part and Two Connectors

The following schema shows the benefit of using composite properties to represent interconnections in a reusable way. The property class captures how to connect device ports to a cable, which is used twice in the network (adapted from an example by Robert Thompson). This is not possible with a composite class alone, which requires the connections between port and cable to be restated each time the cable is used in a network. The devices have one port with two pins. The cable has two ports with two pins each. The composite property specifies how to connect the cable pins to the corresponding ones on device ports, using four connectors, two on each end. The property paths have three elements when identifying the cable pins in the composite property, from property class to cable, from cable to cable port, from cable port to cable port pin.

```
  # Device, DevicePort, DevicePortPin

Device        rdf:type rdfs:Class .
DevicePort    rdf:type rdfs:Class .
DevicePortPin rdf:type rdfs:Class .
portOnDevice rdf:type    complmf:PortProperty ;
             rdfs:domain Device ;
             rdfs:range  DevicePort.
pin1InDevicePort rdf:type    complmf:PortProperty ;
                 rdfs:domain DevicePort ;
                 rdfs:range  DevicePortPin .
pin2InDevicePort rdf:type    complmf:PortProperty ;
                 rdfs:domain DevicePort ;
                 rdfs:range  DevicePortPin .

       # Cable, CablePort, CablePortPin
```

```
Cable        rdf:type rdfs:Class .
CablePort    rdf:type rdfs:Class .
CablePortPin rdf:type rdfs:Class .
port1InCable rdf:type    complmf:PortProperty ;
             rdfs:domain Cable ;
             rdfs:range  CablePort .
port2InCable rdf:type    complmf:PortProperty ;
             rdfs:domain Cable ;
             rdfs:range  CablePort .
pin1InCablePort rdf:type    complmf:PortProperty ;
                rdfs:domain CablePort ;
                rdfs:range  CablePortPin .
pin2InCablePort rdf:type     complmf:PortProperty ;
                rdfs:domain CablePort ;
                rdfs:range  CablePortPin .

  # Property between cable port pins and device port pins

pinAttached rdf:type    rdf:Property ;
            rdfs:domain CablePortPin ;
            rdfs:range  DevicePortPin .

  # Composite Property between device ports

devicePortAttached rdf:type        compmf:PropertyClass ;
                   rdfs:subclassOf rdf:Statement ;
                   rdfs:domain     DevicePort ;
                   rdfs:range      DevicePort .
cableInDevicePortAttached rdf:type    rdf:Property ;
                          rdfs:domain devicePortAttached ;
                          rdfs:range  Cable .
Connector1InDevicePortAttached
    rdf:type                     complm:Connector ;
    complm:propertyThatConnects  pinAttached ;
    complmf:domainEndOfConnector (subject pin1InDevicePort) ;
    complmf:rangeEndOfConnector  (cableInDevicePortAttached
                                  port1InCable pin1InCablePort) .
Connector2InDevicePortAttached
    rdf:type                     complm:Connector ;
    complm:propertyThatConnects  pinAttached ;
    complmf:domainEndOfConnector (subject pin2InDevicePort) ;
    complmf:rangeEndOfConnector  (cableInDevicePortAttached
                                  port1InCable pin2InCablePort) .
Connector3InDevicePortAttached
    rdf:type                     complm:Connector ;
    complm:propertyThatConnects  pinAttached ;
    complmf:domainEndOfConnector (cableInDevicePortAttached
                                  port2InCable pin1InCablePort) ;
    complmf:rangeEndOfConnector  (object pin1InDevicePort) .
Connector4InDevicePortAttached
    rdf:type                    complm:Connector ;
    complm:propertyThatConnects pinAttached ;
    complmf:domainEndOfConnector (cableInDevicePortAttached
                                  port2InCable pin2InCablePort) ;
    complmf:rangeEndOfConnector  (object pin2InDevicePort) .
```

```
devicePortAttached complm:hasConnector Connector1InDevicePortAttached ,
                                       Connector2InDevicePortAttached ,
                                       Connector3InDevicePortAttached ,
                                       Connector4InDevicePortAttached .
  # Network
Network rdf:type rdfs:Class .
device1InNetwork rdf:type    rdf:Property ;
                 rdfs:domain Network ;
                 rdfs:range  Device .
device2InNetwork rdf:type    rdf:Property ;
                 rdfs:domain Network ;
                 rdfs:range  Device .
device3InNetwork rdf:type    rdf:Property ;
                 rdfs:domain Network ;
                 rdfs:range  Device .

Connector1InNetwork
    rdf:type                     complm:Connector ;
    complm:propertyThatConnects  devicePortAttached ;
    complmf:domainEndOfConnector (device1InNetwork portOnDevice) ;
    complmf:rangeEndOfConnector  (device2InNetwork portOnDevice) .

Connector2InNetwork
    rdf:type                     complm:Connector ;
    complm:propertyThatConnects  devicePortAttached ;
    complmf:domainEndOfConnector (device2InNetwork portOnDevice) ;
    complmf:rangeEndOfConnector  (device3InNetwork portOnDevice) .

Network complm:hasConnector Connector1InNetwork ,
                            Connector2InNetwork .
```

Figure E.2.3: Schema Example for Composite Property Capturing Cable Connection

E.3 Example of Full RDF/S Composite Structure (Connectors as Properties)

The following schema modifies the one at the beginning of Appendix E.1 to use connectors as properties. The powers property is a property class, and is the range for the connector properties.

```
powers rdf:type        compmf:PropertyClass ;
       rdfs:subclassOf rdf:Statement ;
       rdfs:domain     Engine .

powersConnectorInCar rdf:type                     compf:Connector ;
                     rdfs:domain                  Car ;
                     rdfs:range                   powers ;
                     complmf:domainEndOfConnector (engineInCar) ;
                     complmf:rangeEndOfConnector  (poweredWheelInCar) .

powersConnectorInBoat rdf:type                     compf:Connector ;
                      rdfs:domain                  Boat ;
                      rdfs:range                   powers ;
                      complmf:domainEndOfConnector (engineInBoat) ;
                      complmf:rangeEndOfConnector  (propellerInBoat) .
```

Figure E.3.1: Schema Example with Connectors as Properties

The following triples are valid and required under the above schema, assuming the engines, wheel, and propeller are playing their respective parts. The statements corresponding to the interpart triples are values of the connector properties.

```
AnEngineInACar powers AWheel .
AnEngineInACarPowersAWheel rdf:type      powers ;
                           rdf:predicate powers ;
                           rdf:subject   AnEngineInACar ;
                           rdf:object    AWheel .
ACar powersConnectorInBoat AnEngineInACarPowersAWheel .

AnEngineInABoat powers APropeller .
AnEngineInABoatPowersAPropeller rdf:type      powers ;
                                rdf:predicate powers ;
                                rdf:subject   AnEngineInABoat ;
                                rdf:object    APropeller .
ABoat powersConnectorInBoat AnEngineInABoatPowersAPropeller .
```

Figure E.3.2: Valid Triples for the Schema Example with Connectors as Properties

Appendix F. Examples in First-Order Logic

This appendix gives first-order expressions for the some of the examples in Appendix E. They are written in Common Logic Interchange Format [CL], using the question mark convention for variables. The properties from earlier examples are translated to predicates that have the subject as the first argument and object as second, but no implied navigation direction is implied in first-order logic. Definitions of classes and properties, such as Car and engineInCar, are omitted for brevity. The expressions could probably be written in a rule language, but care must be taken to support contrapositives, see comments in the examples.

The extension described in this document has a number of benefits in conjunction with first-order expressions. It

- Provides a source for generating expressions in first order languages, which are fairly complicated in the general case.
- Is more easily mapped to and from user interfaces, both textual and graphical. It is an intermediary representation between subject matter experts' views and logical expressions.
- Is more incremental to modify, facilitating multi-author development. The representation for connectors can be added, changed, or removed without affecting other connectors. The equivalent logical expressions are not incremental in the general case, because the same expression covers multiple connectors.

F.1 First Order Examples of Lite RDF/S Composite Structure (Simple Case, Expanding Maximum value Sets)

The following are expressions for the examples of Appendix E.1 reflecting the simple case semantics of Section 2.2.1. The first expression gives the minimum value set semantics, and the second is for the maximum set (the latter is a limited form of complex role inclusion in DL [Role Inclusion][RIQ], see discussion of Expression E.1.2.). The first requires the engine in a car to power the wheels in the same car, while the second requires anything powered by an engine (that is in a car) to be a powered wheel of the same car as the engine. These are logical implications, so the contrapositive applies to both. For the first expression, if an engine does not power a particular wheel, then they cannot both be in the same car, or the car they are both in is not actually a car. For the second, if something is not the powered wheel of a car, then the engine in the car cannot power it, or the car has no engine, or the car is not actually a car. If the constraint is written in a rule language, a semantics should be chosen that supports contrapositives, or additional rules written for them.

```
    (forall (?car ?engine ?wheel)
       (if (and (Car ?car)
                (engineInCar ?car ?engine)
                (poweredWheelInCar ?car ?wheel))
           (powers ?engine ?wheel)))

    (forall (?car ?engine ?poweredThing)
       (if (and (Car ?car)
                (engineInCar ?car ?engine)
                (powers ?engine ?poweredThing))
           (poweredWheelInCar ?car ?poweredThing)))
```

Expression E.1.1: First Order Example of Simple Case

The following expressions are for the first case of expanding maximum value sets due to multiple connectors, Section 2.2.3.2, where the same individual plays the same role in different instances of the same composite class. The first expression is for the minimum value set, the second for the maximum. The first requires anyone teaching at school to teach all the students at the school (the example assumes a very small school). The second requires every teacher at a school who is teaching someone to be teaching at a school where that person is a student (this is a limited form of complex role inclusion in DL, which would require all the students of an individual teacher to attend every school the teacher works at, because it does not contextualize the interpart property to each school [Role Inclusion][RIQ]). The contrapositives of these are if a teacher at a school does not teach a particular student, then they cannot both be at the same school, or the school they are at is not actually a school; and if a teacher and a person are never at the same school, then either the teacher does not teach the person, or the teacher is not at any school, or the school they are at is not actually a school.

```
    (forall (?school ?teacher ?student)
       (if (and (School ?school)
                (teacherAtSchool ?school ?teacher)
                (studentAtSchool ?school ?student))
           (teaches ?teacher ?student)))

    (forall (?school ?teacher ?personTaught)
       (if (and (School ?school)
                (teacherAtSchool ?school ?teacher)
                (teaches ?teacher ?personTaught))
           (exists (?someschool)
              (and (teacherAtSchool ?someschool ?teacher)
                   (studentAtSchool ?someschool ?personTaught)))))
```

Expression E.1.2: First Order Example with Expanding Maximum Values Sets, Case 1

The following expressions are for the second case of expanding maximum value sets due connectors applying multiple times to the same individual, Section 2.2.3.2, where more than one connector for the same interpart property and domain end property applies to the same composite individual. The first three are for the minimum value set, the last for the maximum set. The first three repeat the simple case for multiple connectors. The last uses a disjunction to restrict what an engine in a car can power.

Contrapositives apply as usual, but there are more to enumerate, so for brevity are not described here.

```
(forall (?car ?engine ?wheel)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (poweredWheelInCar ?car ?wheel))
       (powers ?engine ?wheel)))

(forall (?car ?engine ?generator)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (generatorInCar ?car ?generator))
       (powers ?engine ?generator)))

(forall (?car ?engine ?oilpump)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (oilpumpInCar ?car ?oilpump))
       (powers ?engine ?oilpump)))

(forall (?car ?engine ?poweredThing)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (powers ?engine ?poweredThing))
       (or (poweredWheelInCar ?car ?poweredThing)
           (generatorInCar ?car ?poweredThing)
           (oilpumpInCar ?car ?poweredThing))))
```

Expression E.1.3: First Order Example with Expanding Maximum Values Sets, Case 2

The following expressions are for the third case of expanding maximum value sets due connectors applying multiple times to the same individual, Section 2.2.3.2, where more than one connector with the same interpart property applies to the same composite individual, but with different domain end properties that happen to have the same values in the composite individual. The first three are for the minimum value set, the last for the maximum set. The first three repeat the simple case for multiple connectors. The last uses a disjunction to restrict what the engine can power when it happens to play all the power source roles. Contrapositives apply as usual.

```
(forall (?car ?engine ?wheel)
   (if (and (Car ?car)
            (powerSourceForWheelInCar ?engine)
            (poweredWheelInCar ?car ?wheel))
       (powers ?engine ?wheel)))

(forall (?car ?engine ?generator)
   (if (and (Car ?car)
            (powerSourceForGeneratorInCar ?car ?engine)
            (generatorInCar ?car ?generator))
       (powers ?engine ?generator)))

(forall (?car ?engine ?oilpump)
   (if (and (Car ?car)
            (powerSourceForOilPumpInCar ?car ?engine)
            (oilpumpInCar ?car ?oilpump))
       (powers ?engine ?oilpump)))
```

```
(forall (?car ?engine ?poweredThing)
   (if (and (Car ?car)
            (and (powerSourceForWheelInCar ?car ?engine)
                 (powerSourceForGeneratorInCar ?car ?engine)
                 (powerSourceForOilPumpInCar ?car ?engine))
            (powers ?engine ?poweredThing))
       (or (poweredWheelInCar ?car ?poweredThing)
           (generatorInCar ?car ?poweredThing)
           (oilpumpInCar ?car ?poweredThing))))
```

Expression E.1.4: First Order Example with Expanding Maximum Values Sets, Case 3

The fourth case of expanding maximum value sets connectors applying multiple times to the same individual, Section 2.2.3.2, is the most general, where more than one connector, with the same interpart property, on separate composite classes, can apply to different composite individuals, with different domain end properties, that happen to have the same resources as values in the composite individuals. The maximum set semantics requires disjunctions across the composite classes having the connectors, and across the domain end properties.

The first order expressions so far can apply to values of port and nonpart properties, as long as connectors from them have a domain end path starting with an internal part, or at least a property not known to be a port or nonpart, see Section 2.2.3.3. The following expressions are for the port example in Appendix E.1, which is a Car class with a connector between ports on the engine and wheels. The engine is not known to be a port or nonpart on the car, so connectors in the car from its ports are not subject to the maximum value set expansion for ports and nonparts.

```
(forall (?car ?engine ?wheel ?crankshaft ?hub)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (crankshaftInEngine ?engine ?crankshaft)
            (poweredWheelInCar ?car ?wheel)
            (hubInWheel ?wheel ?hub))
       (crankshaftPowers ?crankshaft ?hub)))

(forall (?car ?engine ?crankshaft ?poweredThing)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (crankshaftInEngine ?engine ?crankshaft)
            (crankshaftPowers ?crankshaft ?poweredThing))
       (exists (?wheel)
          (and (poweredWheelInCar ?car ?wheel)
               (hubInWheel ?wheel ?poweredThing)))))
```

Expression E.1.5: First Order Example of Ports

The maximum value set semantics for ports and nonparts is beyond first order when any connectors from them have a domain end path consisting only of ports and nonparts, though it can be reduced to first order in common applications. The first example in Section 2.2.3.3 has a connector in a car from the wheels to the engine, for power delivery back into the engine from the wheels when the car is moving faster than the engine. Under the simple case semantics, the expression for this connector would prevent the wheel from delivering power to the road, assuming there is no connector defined for that in a larger composite structure of the environment, see Section 2.2.3.3. The following expression puts all the semantics beyond first order in an isPartOf predicate, and uses it to apply the maximum set semantics only to things outside the car. It is beyond first order because it is true only if there part properties can be traversed from the whole to the part, which quantifies over part properties determining membership in a generalized transitive closure. It can be reduced to first order in various ways, as sketched in Section 2.2.3.3.

```
(forall (?car ?engine ?wheel)
   (if (and (Car ?car)
            (poweredWheelInCar ?car ?wheel)
            (engineInCar ?car ?engine))
       (wheelPowering ?wheel ?engine)))

(forall (?car ?engine ?poweredThing)
   (if (and (Car ?car)
            (poweredWheelInCar ?car ?wheel)
            (wheelPowering ?wheel ?poweredThing))
       (or (engineInCar ?car ?poweredThing)
           (not (isPartOf ?poweredThing ?car)))))
```

Expression E.1.6: Example of Maximum Value Set Semantics for Ports

F.2 First Order Example of Medium RDF/S Composite Structure (Composite Properties)

The following expressions are for the first example in Appendix E.2. The first two are for the connector between crankshaft and clutch, and the second two are for the connector between clutch and hub. The parts of the link between engine and wheel are represented as ternaries, identifying the link by its subject and object in the first two arguments and its part with a third. This is analogous to representing n-aries as special kinds of statements about statements, see Section 3.2. The expressions for the two connectors are separated so the constraints apply independently. For example, the crankshaft will still be linked to the clutch, even when the hubs happen to not be in the wheels, and similarly for the clutch and the hubs whenever the crankshaft is not in the engine. Contrapositives apply as usual.

```
(forall (?engine ?wheel ?clutch ?crankshaft)
   (if (and (powersWheel ?engine ?wheel)
            (crankshaftInEngine ?engine ?crankshaft)
            (clutchInPowersWheel ?engine ?wheel ?clutch))
       (crankshaftPowers ?crankshaft ?clutch)))

(forall (?engine ?wheel ?poweredThing ?crankshaft)
   (if (and (powersWheel ?engine ?wheel)
            (crankshaftInEngine ?crankshaft ?engine)
            (crankshaftPowers ?crankshaft ?poweredThing))
       (clutchInPowersWheel ?engine ?wheel ?poweredThing )))

(forall (?engine ?wheel ?clutch ?hub)
   (if (and (powersWheel ?engine ?wheel)
            (clutchInPowersWheel ?engine ?wheel ?clutch)
            (hubInWheel ?wheel ?hub))
       (clutchPowers ?clutch ?hub)))

(forall (?engine ?wheel ?clutch ?poweredThing)
   (if (and (powersWheel ?engine ?wheel)
            (clutchInPowersWheel ?engine ?wheel ?clutch)
            (clutchPowers ?clutch ?poweredThing))
       (hubInWheel ?wheel ?poweredThing)))
```

Expression E.2.1: First Order Example of Medium RDF/S Composite Structure

F.3 First Order Example of Full RDF/S Composite Structure (Connectors as Properties)

The following expressions are for the example in Appendix E.3, which is the simple case of Appendix E.1 using connectors as properties. The links between engine and wheels are parts of the car, which is represented as a ternary, identifying the car by the first argument, and the link by its subject and object in the second two arguments. This is analogous to the representation of n-aries as special kinds of statements about statements, see Section 3.2, except the base statement is the object instead of the subject. The third expression requires the values of the connector properties to be statements for the interpart property and values of the domain and range end of the connector. It ensures there are powers links corresponding to the connector property values, and the contrapositive, that there are no connector property values without a corresponding powers link.

```
(forall (?car ?engine ?wheel)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (poweredWheelInCar ?car ?wheel))
       (and (powersConnectorInCar ?car ?engine ?wheel)
            (powers ?engine ?wheel))))

(forall (?car ?engine ?poweredThing)
   (if (and (Car ?car)
            (engineInCar ?car ?engine)
            (powersConnectorInCar ?car ?engine ?poweredThing)
            (powers ?engine ?poweredThing))
       (poweredWheelInCar ?car ?poweredThing))


(forall (?car ?engine ?wheel)
   (if (powersConnectorInCar ?car ?engine ?wheel)
       (powers ?engine ?wheel)))
```

Expression E.3.1: First Order Example of Full RDF/S Composite Structure

References

[AMIS]
"*The AMIS Approach to Systems Integration: An Overview*," Libes, D., Barkmeyer, E., Denno, P., Flater, D., Potts Steves, M., Wallace, E., Barnard Feeney, A., U.S. National Institute of Standards Internal Report 7101, http://www.nist.gov/msidlibrary/doc/nistir7101.pdf, May 2004.

[Analogy]
"*Analogical reasoning*," Sowa, J.,Majumdars, A., in Conceptual Structures for Knowledge Creation and Communication, de Moor, A., Lex, W., Ganter, B. (eds.), Lecture Notes in Artificial Intelligence, 2746, Springer, http://www.jfsowa.com/pubs/analog.htm, 2003.

[ADL]
"*A Framework for Classifying and Comparing Architecture Description Languages*," Nenad Medvidovic and Richard N. Taylor, Proceedings of the 6th European Foundations of Software Engineering Conference, 60-76, http://portal.acm.org/citation.cfm?id=267903, 1997.

[Acme]
"*Acme: Architectural Description of Component-Based Systems*," Garlan, D., Monroe, R., Wile, D., in Foundations of Component-Based Systems, Leavens, G., Sitaraman, M. (eds.), pp. 47-68,

Cambridge University Press, http://www.cs.cmu.edu/afs/cs/project/able/ftp/acme-fcbs/acme-fcbs.pdf, 2000.

[Aggregation]

"*A More Complete Model of Relations and Their Implementation, Part IV: Aggregation*," Bock, C., Odell, J., Journal Of Object-Oriented Programming 11:5, pp. 68-70, http://www.conradbock.org/relation4.html, September 1998.

[CDIF]

"*Metamodeling in EIA/CDIF---meta-metamodel and metamodels*," Flatscher, R., in Association of Computing Machinery Transactions on Modeling and Computer Simulation (TOMACS), 12:4, pp. 322-342, 12:4, pp. 322-342 http://portal.acm.org/citation.cfm?id=643120.643124, October 2002.

[CL]

Common Logic (CL) - A framework for a family of logic-based languages, International Standards Organization, http://philebus.tamu.edu/cl/docs/cl/32N1377T-FCD24707.pdf, 2005.

[Completion]

"*Negation as Failure*," Clark, K., in Logic and Databases, Gallaire, H., Minker, J. (eds.), pp. 293-322, Plenum Press, 1978.

[Composition Kinds]

"*Six Different Kinds of Composition*," James Odell, Journal of Object Oriented Programming, 5:8, pp. 10-15, http://www.conradbock.org/compkind.html, January 1994.

[DOLCE]

"*WonderWeb Deliverable D17, The WonderWeb Library of Foundational Ontologies* ," Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L., http://wonderweb.semanticweb.org/deliverables/documents/D17.pdf, August 2002.

[Expert]

Developing Expert Systems, Payne, E., McArthur, R., Wiley, 1990.

[Inheriting Part Demo]

Inheriting Part Structure, Bock, C., http://www.conradbock.org/compinheritance.html, 1995.

[Maximal Subproperties]

"*Extending OWL with Maximal Subproperties: An Approach to Define Qualified Cardinality Restrictions and Reflexive Properties*," Pokraev, S., Brussee, R., in Proceedings of Workshop on OWL: Experiences and Directions (OWL 2005), Freeband/A-MUSE, http://www.mindswap.org/2005/OWLWorkshop/sub33.pdf, November 2005.

[Mereotopology 1]

"*Toward a Geometry of Common Sense: A Semantics and a Complete Axiomatization of Mereotopology*," Asher, N., Vieu, L., in Proceedings of Fourteenth International Joint Conference on Artificial Intelligence (IJCAI), Mellish, C. (ed.), pp. 846-852, http://citeseer.ist.psu.edu/asher95toward.html, 1966.

[Mereotopology 2]

"*Mereotopology: A Theory of Parts and Boundaries*," Smith, B., Data & Knowledge Engineering, 20, pp. 287-303, http://ontology.buffalo.edu/smith/articles/mereotopology.htm, 1996.

[N3]

Notation 3, An RDF language for the Semantic Web, Berners-Lee, T., http://www.w3.org/DesignIssues/Notation3, November 2001.

[Nary OWL]

Defining N-ary Relations on the Semantic Web, Noy, N., Rector, A. (eds.), http://smi-web.stanford.edu/people/noy/nAryRelations/n-aryRelations-2nd-WD.html, September 2005.

[Nilsson]

Principles of Artificial Intelligence, Nilsson, N., Tioga Publishing, 1980.

[OCL]

Object Constraint Language, version 2.0, Object Management Group, http://doc.omg.org/formal/06-05-01, May 2006.

[ODM]

Ontology Definition Metamodel, Object Management Group, http://doc.omg.org/ad/06-05-01, May

2006.

[OntoClean]

"*Supporting ontological analysis of taxonomic relationships*," Welty, C., Guarino, N., Data and Knowledge Engineering, 39, pp. 51-74, http://citeseer.ist.psu.edu/452329.html, 2001.

[OntoMDA]

Ontology Modeling and MDA, Djuric, D., Gasevic, D., Devedzic, V., Journal of Object Technology, 4:1, pp. 109-128, http://www.jot.fm/issues/issue_2005_01/article3, January/February 2005

[OSERA]

Open Source eGovernment Reference Architecture, U.S. General Services Administration, http://osera.modeldriven.org, 2006.

[OWL]

OWL Web Ontology Language Overview, McGuinness, D., van Harmelen, F. (eds.), World Wide Web Consortium Recommendation, http://www.w3.org/TR/2004/REC-owl-features-20040210, February 2004.

[OWL 1.1]

The OWL 1.1 Extension to the W3C OWL Web Ontology Language, Patel-Schneider, P., http://www-db.research.bell-labs.com/user/pfps/owl/overview.html, December 2005.

[OWL Semantics]

OWL Web Ontology Language Semantics and Abstract Syntax, Patel-Schneider, P., Hayes, P., Horrocks, I. (eds.), World Wide Web Consortium Recommendation, http://www.w3.org/TR/2004/REC-owl-semantics-20040210, February 2004.

[Parametrics]

"*Achieving Fine-grained CAE-CAE Associativity via Analyzable Product Model (APM)-based Idealizations*," Peak, R., invited presentation at Workshop on Developing a Design/Simulation Framework,http://eislab.gatech.edu/pubs/seminars-etc/2005-cpda-dsfw-peak, 2005.

[PartOf Framework]

"*A framework for part-of hierarchies in terminological logics*," Padgham, L., Lambrix, P., in Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR-94), Doyle, J., Sandewall, E., Torasso, P. (eds.), pp. 485-496, http://citeseer.ist.psu.edu/padgham94framework.html, 1994.

[Part-Whole Extension]

"*A whole-part extension for description logics*," Speel, P., Patel-Schneider, P., in Proceedings of the European Conference on Artificial Intelligence (ECAI'94) Workshop on Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology, pp. 111-121, 1994.

[Part Whole OO]

"*Part-Whole relations in Object-centered systems: an overview*," Artale, A., Franconi, E., Guarino, N., Pazzi, C., in Data and Knowledge Engineering (DKE), 20, pp. 347-383, http://citeseer.ist.psu.edu/artale96partwhole.html, 1996.

[Part Whole OWL]

Simple part-whole relations in OWL Ontologies, Rector, A., Welty, C. (eds.), http://www.w3.org/2001/sw/BestPractices/OEP/SimplePartWhole, August 2005.

[Protege]

The Protégé Ontology Editor and Knowledge Acquisition System, Stanford Medical Informatics, http://protege.stanford.edu, 2006.

[Qua Link]

"*The QUA link*," Freeman, M., in Schmolze, J., and Brachman, R. (eds.), Proceedings of the KL-ONE Workshop, Artificial Intelligence Technical Report No. 4, Schlumberger, pp. 54-64, 1981.

[Prototype]

Prototype-based programming: concepts, languages, and applications, Noble, J., Taivalsaari, A., Moore I. (eds.), Springer-Verlag, 1999.

[PSL]

(ISO 18269) Process Specification Language, International Standards Organization, http://www.tc184-sc4.org/SC4_Open/SC4%20Legacy%20Products%20(2001-08)/PSL_(18629)/, 2006.

[RDF Vocabulary]
RDF Vocabulary Description Language 1.0: RDF Schema, Brickley, D., Guha, R. (eds.), World Wide Web Consortium Recommendation, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210, February 2004.

[RDF Semantics]
RDF Semantics, Hayes, P. (ed.), World Wide Web Consortium Recommendation, http://www.w3.org/TR/2004/REC-rdf-mt-20040210, February 2004.

[RIQ]
Decidability of SHIQ with Complex Role Inclusion Axioms, Horrocks, I., Sattler, U, Artificial Intelligence, 160:1-2, pp. 79-104, http://citeseer.csail.mit.edu/585553.html, December 2004.

[Role Inclusion]
Obstacles on the Way to Qualitative Spatial Reasoning with Description Logics: Some Undecidability Results, Wessel, M., Proceedings of the International Workshop in Description Logics 2001 (DL2001), http://citeseer.ist.psu.edu/666529.html, 2001.

[Role Value Map]
Subsumption in KL-ONE is Undecidable, Schmidt-Schauß, M., Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR'89), Brachman, R., Levesque, H.. Reiter, R. (eds.), pp. 421-431, http://www.ki.informatik.uni-frankfurt.de/papers/schauss/KLONE-UNDEC2004.ps, May 1989.

[ROOM]
Real-Time Object-Oriented Modeling, Selic. B., Gullekson, G., Ward, P., Wiley, April 1994.

[Relations]
"*A More Complete Model of Relations and Their Implementation, Part I: Relations as Object Types*," Bock, C., Odell, J., Journal Of Object-Oriented Programming, 10:3, pp. 38-40, http://www.conradbock.org/relation1.html, June 1997.

[Roles]
"*A More Complete Model of Relations and Their Implementation, Part III: Roles*," Bock, C., Odell, J., Journal Of Object-Oriented Programming 11:2, pp. 51-54, http://www.conradbock.org/relation3.html, May 1998.

[SDL]
Specification and description language (SDL), International Telecommunication Union Recommendation Z.100, http://www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf, 1999.

[SysML]
Systems Modeling Language (SysML) Specification, Object Management Group, http://doc.omg.org/ptc/06-05-04, May 2006.

[UML2]
Unified Modeling Language: Superstructure, version 2.0, Object Management Group, http://doc.omg.org/formal/05-07-04, August 2005.

[UML2 Activities]
"*UML 2 Activity and Action Models*," Bock, C., Journal of Object Technology," 2:4, pp. 43-53, http://www.jot.fm/issues/issue_2003_07/column3, November-December, 2003.

[UML2 Composition]
"*UML 2 Composition Model*," Bock, C., Journal of Object Technology," 3:10, pp. 47-73, http://www.jot.fm/issues/issue_2004_11/column5, July-August, 2004.

[ValueClass]
"*Representing Composites with Valueclass Enhancements and the Relation Form of Recursive Units*," Mishelevich, D., Filman, R., Bock, C., Morris, P., Paulson, A., Treitel, R., in New Generation Knowledge System Development Tools, Phase 2 Interim Report, DARPA Contract F30602-85-C-0065, May 1988.

Contents