QuickRDA

Underlying Metamodel

Erik Eidt

Version 3.3

August, 2011

***Copyright (c) 2011, 2012 Hewlett-Packard Development Company, L.P.***

[Introduction 5](#_Toc336067119)

[Formal Description of Metamodel 6](#_Toc336067120)

[Background 6](#_Toc336067121)

[DMI Metamodel Diagram 7](#_Toc336067122)

[Values & Serialization 10](#_Toc336067123)

[Capabilities of Concepts 11](#_Toc336067124)

[Basic Entities 12](#_Toc336067125)

[Basic Properties 12](#_Toc336067126)

[Appendix A: Differences with DMI For Those Familiar with OWL 13](#_Toc336067127)

[Background 13](#_Toc336067128)

[References & Other Readings 15](#_Toc336067129)

*Table of Figures*

[Figure 1. The Underlying Metamodel 7](#_Toc336067130)

*Table of Tables*

[Table 1. Basic Entities of the Underlying Metamodel 12](#_Toc336067131)

[Table 2. Basic Properties of the Underlying Metamodel 12](#_Toc336067132)

# Introduction

QuickRDA is a tooling system that allows domain models to be defined — domain models are defined in terms of domain language; the domain languages are, in turn, defined in terms of an underlying metamodel. This paper describes the QuickRDA underlying metamodel.

Whereas the domain models are defined in terms of domain language, and domain language in terms of the underlying metamodel: the underlying metamodel is self-grounding, meaning that it defines its concepts in terms of each other. Thus, the underlying metamodel should be taken as a set of intimately related concepts and relationships whose definitions are intertwined.

First, some conventions: We’ll introduce semi-formal English terms using ***lower case***and***this bold italics style***; we’ll introduce terms of QuickRDA’s underlying metamodel using Uppercase Titling and This Font Style.

The terms: set, type, and class are easily mixed up — and who’s to say what’s right. For our purposes, rather than debate, we’ll just say that a set is a concept that groups members. A type is a set whose members have qualities in common (i.e. going to common relationships). A class is a family of related types (i.e. going to sub-classing).

On that foundation, we’ll define the term Class (as a class as described above), and Concept, which is, well, a member — a member may or may not be a Class. A Class, of course, is also a Concept — and Concept is a Class!

Relationships between members are captured as Statements, which have a subject, a verb, and an object. A verb can be spoken of independent of any Statements it’s used in, and when so spoken of called Properties. Properties are associated with Classes by a Domain and Range: this is the mechanism for establishing the common qualities of the members of a type (and a Class).

Some readers will notice a similarity with RDF + OWL. The most significant addition is that which provides explicit support for the information domain.

Statements *are* Concepts, and thus can be referred to by other Statements; among other things, this is the mechanism used to put relationships themselves into collections along with their simple concepts counterparts. Since Statements can be referred to, they have labels. Thus, the full form of a Statement is as a quad having a label, a subject, a verb, and object.

The following section formally depicts the underlying metamodel in terms of its concepts and relationships, and then discusses each of these in turn.

## Formal Description of Metamodel

Though the metamodel is relatively small, it can be very confusing. Because of this, we’re going to use two parallel approaches, each in turn, for describing the metamodel.

The first approach is to start with a background on the notion of class and instance using the Object Management Group’s Unified Modeling Language, UML, with diagramming to provide a formalism of the metamodel along with a discussion of the metamodel using UML terminology.

The second way is to define the metamodel using its own terms, which results in a foundational set of Classes & Properties, similar to OWL, and axiomatic statements (quads similar to RDF triples).

### Background

In order to discuss the metamodel diagram in UML terms, first let’s start with some terminology regarding the notion of class and instance and what we mean by using those terms.

Instances can represent individuals, anything really. Classes are used to identify a particular set of members, formally known as called instance members, or just instances, having commonality. This commonality among the instances can range from mere membership in the same class, to qualities that we expect all members to exhibit. Classes are used to describe this commonality once, instead of repeating it for every instance. We expect each instance of a class to exhibit its personalization of commonality of qualities that involve its class.

In UML, commonality of qualities is expressed with an Association between one class and (the same or) another class. An Association between two classes directly relates to a Link between two instances of those classes. End-point cardinality (and other restrictions) may be part of the Association: these apply to the expected and possible number of Links related to the Association.

In RDF, such commonality of qualities is expressed using Properties, whose Domain is one class and Range is (the same or) another class. When an instance is a member of a class then the properties of the class manifest themselves as relations between instances, which are statements or Triples of the form Subject-Predicate-Object. For a given triple statement, we expect the Subject to be an instance of the Domain of the Property and the Object to be an instance of the Range of the Property.

RDF does not directly give statements a type (there is the type rdf:Statement, but that is a parallel construct to the triples themselves); DMI, however, formalizes the notion of a relation, integrating it into the metamodel, and giving it its own class and instances.

Classes can have super and sub relationships with other classes. Instances of a sub class are also instance members of the super class. Thus, a subclass satisfies or matches a domain or range specification of its parent. Conversely, subclasses of instances of some class, X, are also considered instance members of X.

### DMI Metamodel Diagram

The following diagram captures the DMI metamodel using UML notation. It makes heavy use of the notion of class and instance, as you’ll see. One significant contribution of DMI over RDF is the integration of the statement itself directly into the metamodel as a first-class construct.



Figure 1. The Underlying Metamodel

A discussion of what is shown in Figure 1 follows entirely in terms of UML. (A description of the DMI metamodel defined it is own terms can be found in subsequent sections.) Words or phrases in bold refer directly to elements of the diagram.

1. There is a fundamental notion of instances, which are represented by the class Concept; a Concept is an instance of one or more classes. Every instance is an instance of Concept.
2. There is a fundamental notion of classes, which are represented by the class Class; a class is also an instance, thus a class has a type, which is **Class**, and, instances can also be classes. The foundation class for all instances is **Concept**; even Class is a subclass of Concept. Concept, which is a class, has as its type Class. Subclasses of Concept are also instances of Class. This relationship between the fundamental class, Concept, and the Class itself is the standard one found in object-oriented programming languages, for example , java.lang.Object and java.lang.Class. *OWL, even OWL 2, shies away from defining the fundamental class and instance this way, I believe due to the historical pressure to support known computable and efficient subsets (which manifest in the separate the levels of OWL: Lite, DL, full).*
3. Classes have a notion of sub class, and inversely — though not diagrammed — super class, expressed as the **Subclass Of** association. In this diagram, UML generalization notion is used to express Links of the Subclass Of association.
4. Concepts have zero or more names. A **Name** is first-class, so they are also Concepts. Many naming schemes can be created and used in parallel, including URIs and IRIs, namespace qualified names, product taxonomies, etc… The diagram shows namespace qualified names using **Simple Name** and **Qualified Name**.
5. **Property** is the class for defining properties. Domain Properties are instances of this class. The Property class is involved in two associations: **has Domain** and **has Range**, meaning that we expect instances of Property (which are Domain Properties) to manifest their personalization of these Associations as Links.
6. **Relation** is the class that begins to formalize the notion of quads — the DMI equivalent to RDF triples. The Relation class is involved in two associations: **has Subject** and **has Object**; instances of Relation (which are Statements) will have their personalization of these associations.
7. Each **Domain Property** is an instance of Property, as well as subclasses of Relation. From being an instance of Property, Domain Properties personalize the Domain and Range associations by specifying links: **has Domain** and **has Range**. From being a subclass of Relation, Domain Properties are classes having instance members with a Subject and Object. Domain Properties apply of the type in their links **has Domain** and **has Range** in refining the class or type of Subject and Object provided by Relation.
8. A **Statement** is an instance of Domain Properties. Since Domain Properties are subclasses of Relation, which is a subclass of Concept, thus, Statements are instances of Concepts. We note that this formalization of Statements as Concepts, which is a unique contribution of DMI over RDF, allows Statements to be referred to. Thus, the four elements forming a quad are: the subject Link, and the object Link exhibited from being instances of Relation, their type specification as an instance of a Domain Property (the predicate), and the ability to refer to them by being instances of Concept.
9. The notion of sub properties is obtained through the use of the subclass association applied between user-defined properties. There is no separate or other notion of sub or super property is necessary.
10. It may be appreciated that additional classes similar to Relation could be introduced to allow for Statements of other, more complex forms beyond subject-predicate-object.
11. Note: Metamodel Associations describing min and max cardinality for Property need to be added.

#### Values & Serialization

The notions of Values & Value Types are not fully described as yet. The intent of DMI is to clearly separate the concerns of conceptual value definition from their serialization.

XML Schema, for example, conflates the notion of values in a value space with the serialization of such, making it difficult to both properly and usefully define the notion of equality among values. RDF builds on this conflation by limiting serialization to a single string, while adding additional characters to that string for the XSD type and internationalization specification (then requiring specialized query extensions to access these second-class embedded fields). As DMI is specifically designed as an interchange format, it addresses this conflation by separating the notion of the definition values and their value spaces from the serialization format used.

#### Capabilities of Concepts

##### As Instance

All entities are an instance of the class Concept. As an instance of Concept, an item has one or more types, which are Classes.

##### As Class

Any entity can operate as a class, having Class as its type. As an instance of Class, an item can be involved in sub and super class relationships, and can have members, instances, and can commonality among those instances in the class can be described via properties.

A Domain Property is also a class representing the statements using it as their Class.

Even Values that are instances of Value Types can themselves be classes; when they are they refer to they are singletons that refer to the value. Thus, for example, one property can have a Range of Boolean, and, another can have the Range of True.

##### As Property

A Domain Property is an instance of Property and a subclass of Relation. As a class, their members are instances, which are statements that use the property as a predicate.

## Basic Entities

The following concepts for entities are defined in the metamodel.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Qualifier*** | ***Name*** | ***Instance Of*** | ***Subclass Of*** | ***Description*** |
| DMI | Concept | Class | Concept | The base type for everything |
| DMI | Class | Class | Concept | The type for classes |
| DMI | Relation | Class | Concept | The type for relations, which are instances of properties |
| DMI | Property | Class | Concept | The type for properties |
|  |  |  |  |  |
| DMI | Nothing | Concept | Concept | The non-value |
| DMI | Name | Class | Concept | The type for externally visible names |
| DMI | Literal | Class | Concept | The base type for defining value spaces |

Table 1. Basic Entities of the Underlying Metamodel

The concept Property itself has type Class; instances of Property are properties; properties themselves have instances, which are relations. Each of the properties is itself also a class that represents its instances (which are relations). Sub-properties are specified as subclasses.

Axiomatic statements are formed from the Instance Of and Subclass Of column, using the definitons of Is An Instance Of and Is A Subclass Of from the next section.

## Basic Properties

These properties are defined in the metamodel — they are qualified by com, hp, DMI (e.g. in the com.hp.DMI namespace).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Qualifier*** | ***Name*** | ***Domain*** | ***Range*** | ***Description*** |
| DMI | Has Simple Name | Name | Literal | Associates Subject with an externally visible simple name |
| DMI | Has Qualifier | Name | Name | Associates Subject with an externally visible qualifier name |
| DMI | Has Name | Concept | Name | Associates Subject with Name |
| DMI | Description | Concept | Literal | A textual description of the subject |
| DMI | Is An Instance Of | Concept | Class | Subject is an instance of Object class |
| DMI | Is A Subclass Of | Class | Class | Subject is subclass of Object |
| DMI | Has Domain | Property | Class | Subject is a property whose domain is Domain |
| DMI | Has Range | Property | Class | Subject is a property whose range is Range |

Table 2. Basic Properties of the Underlying Metamodel

Axiomatic statements are form formed around the properties of the underlying metamodel as follows: each property is an instance of Property and a subclass of Relation; further, each property Has Domain as listed and Has Range as listed.

# Appendix A: Differences with DMI For Those Familiar with OWL

## Background

The underlying metamodel is based on logic similar to OWL with the addition of the “THAT” operator from IKL, and a formalization of the relationship between relations (statements as quads), properties, and classes. Concepts, given token as identifiers (internal to the unit of interchange), are described as having properties through the use of quads; one or more quads may be used to describe a concept.

Each concept in DMI can have qualities of classes, of instances, of relations and of properties. As classes we expect concepts to have super and sub classes, and instances. As instances we expect concepts to have names and types (classes). As relations we expect concepts to relate subjects via verbs to objects. As properties we expect concepts to have domains and ranges (linking properties to classes), and super and sub properties, and relations that use the property.

The logical consequences of applying a “THAT” operator to RDF-style triples are mostly that:

1. Because statements can be referenced (by other statements), Statements have an identity (a label), and thus, triples become quads.
2. The quads (Statements) are given an official type, which is called Relation. Relation formally introduces the notion of subject, verb, and object.
3. User-defined Properties become Classes — whose instances are Statements, specifically the ones that use the property as a verb to connect subject and object, and that as a result,
4. We don’t need the notion of sub-properties separately from subclass since properties are classes, and further that,
5. The notion that a properties provides for domains and ranges is formally linked to notion that statements have subjects and objects.
6. Also, we have an explicit (or in some sense don’t need a separate out of band) mechanism to declare the “class for the statements using given property as their verb” (or predicate in RDF-speak), which is missing in RDF/OWL.

In RDF, statements are triples. Statements express relationships; however, statements themselves are not a first-class construct: relationships cannot be involved in relationships. In RDF, putting a relationship into a relationship requires introduction of a parallel construct (usually using classes and instances) mirroring the statement mechanism. Even then, there is then no specific way to relate the asserted statement to these parallel construct. Class and instance (individually first-class notions) form a valuable pair; however, no similar pairing notion exists between properties (a first-class notion) and triples. Further, omniscient authors can pre-create this mirror structures, or avoid the need for them by reifying relationships directly; however, this cannot be done in all cases, and, it violates a separation of concerns between various authors and users of information: either the authors must consider the potential user who wants to put quote or put their information in a collection, or the users must convert the original authors statements into another form.

RDA DMI uses quads, which are relations, a first-class construct. Relationships can be the source or target of other relationships. In the elevation of relationships to first-class, we add an identifier to relations. To complete the first-class integration of relations, we formalize the property-relationship pair, by tying them to the class-instance constructs. A relationship is a usage of a property. There is a class that represents the set of relations that are usages of a given property. (This class expands to include sub-properties, and usual class-instance reasoning applies.)

As a result of promoting relations to a first-class concept, properties now have instances, which are relations that use the property. Relations have a type, which is that of the property; properties are classes that represent collections of relations — in the same way that any class represents the collection of its instances.

Given that the notion of first-class relations turns properties into classes, the notion of sub-property is subsumed by the notion of subclass. A subclass of a property is a property.

However, a parent of a property is not necessarily also a property; the parent is not automatically a property merely due to being the parent of some property — to be a known as a property, the parent (or any property) should exhibit qualities of properties, such as having a domain and range or being a subclass of a known property.

# References & Other Readings

BLOGIC, Pat Hayes (Talk on what can be improved in RDF)

<http://videolectures.net/iswc09_hayes_blogic/>

Axiomatic Triples in RDF

<http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/>

IKL

<http://www.ihmc.us/users/phayes/IKL/GUIDE/GUIDE.html>

Common Logic

<http://common-logic.org/>

RDF

<http://www.w3.org/standards/techs/rdf#w3c_all>

OWL

<http://www.w3.org/standards/techs/owl#w3c_all>

Lazysoft Associative Model, Sentences Database

<http://www.lazysoft.com/docs/other_docs/amd_whitepaper.pdf>