DDI Cross Domain Integration: Architecture and Alignment with Other Standards

Contents

I. Overview

This document covers those aspects of DDI Cross Domain Integration (DDI – CDI) which deal with the internal design of the model for different purposes, and with the way in which it is expected to be used in specific implementations which are platform- and syntax-specific.

As a platform-independent model (PIM), DDI - CDI does not contain all of the information which is needed in an implementation model. It is, however, understood than implementers will wish to add additional information either to the model itself or one or more of its representations, and a framework for doing so is provided. (See the Design Notes and Modelling Approach section below for more information on what is in this review package.)

The DDI - CDI Model uses some patterns to assist in assuring that it retains internal consistency, and to make its structure consistent from the perspective of users. This is done by employing design patterns for some key features of the model.

DDI - CDI is both designed to be used with other standards and specifications – notably other DDI specifications, but also others – and is itself a user of classes from other standard models.

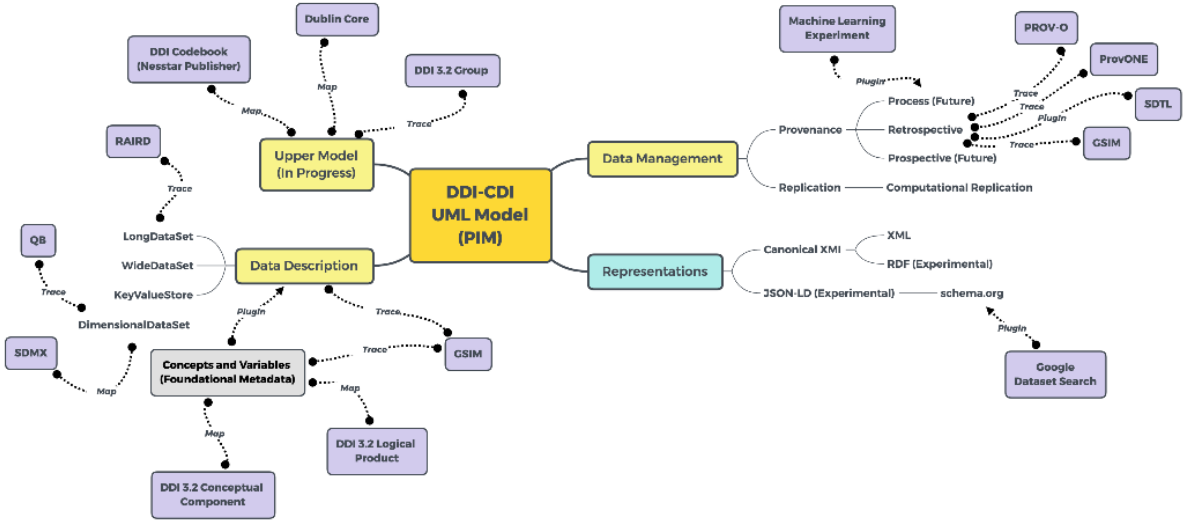
This document describes all of these features of DDI - CDI.

II. Alignment and Use of External and the DDI Family of Standards in DDI - CDI

A. Introduction

DDI – CDI is designed to support the integration of data within systems, and as such it is expected that the alignment and hand-off to other standards will be significant. In order to understand what DDI – CDI does in relation to other standards and the information they provide, we use an “Upper Model” which at this point is purely conceptual. (It is likely that in future this will become a more formal part of the DDI – CDI model, but at this point is provided for informational purposes only.) The use of some form of Upper Model may be helpful in implementing DDI – CDI, and the “Detailed Examples and Use Cases” document in this package provides an example of how this can be done.

Here is a figure that provides an overview of DDI - CDI and its relationship with other standards – both external standards like GSIM, PROV-O and schema.org together with standards in the DDI family of standards including DDI Codebook, DDI Lifecycle and C2Metadata:



Note the different types of relationships various standards have with DDI - CDI: *(UML) Trace*, *Map*, *PlugIn* and *Uses*.

B. Relationship Types between Standards

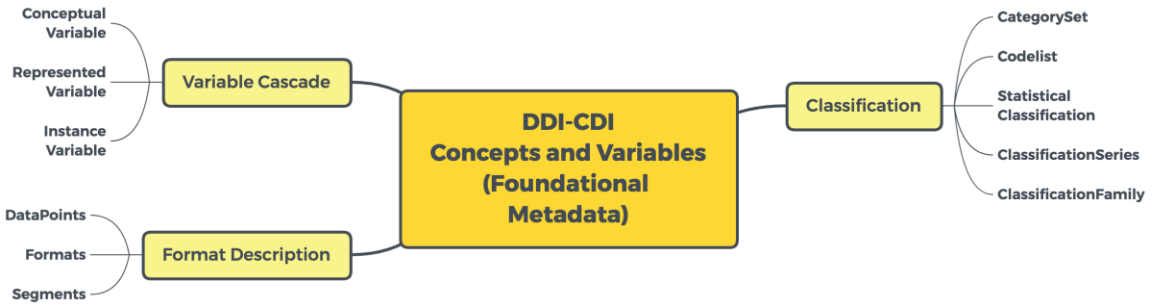
1. UML Trace

The UML Specification 2.5.1 describes the *(UML) Trace* relationship as follows:

> *"Specifies a trace relationship between model elements or sets of model elements that represent the same concept in different models. Traces are mainly used for tracking requirements and changes across models. As model changes can occur in both directions, the directionality of the dependency can often be ignored. The mapping specifies the relationship between the two, but it is rarely computable and is usually informal."*

Because trace relationships are informal, they contribute to the understanding of what is intended in the DDI - CDI model. This may not be sufficient to inform a mapping on the technical level. The trace relationship appears only in the documentary diagrams of the DDI - CDI specification, and not in the canonical model expressed in UML (including the XMI expression).

In a second figure DDI - CDI Foundational Metadata which "traces" to the DDI 3.2 (Lifecycle) Conceptual Component and the DDI 3.2 (Lifecycle) Logical Product is presented in more detail:



2. Map

In addition to (UML) Trace, there is also a *Map* relationship between a DDI - CDI component and another standard. In a Map relationship corresponding properties and/or classes are noted together with information about quality of the match using SKOS terminology: *exactMatch*, *closeMatch*, *broadMatch* and *narrowMatch*. Across a Map relationship it becomes possible to compute elements of one standard from the other, depending on the quality of the match and the direction of the relationship.

3. PlugIn

PlugIn is another type of relationship. In a PlugIn relationship between UML classes and/or their properties it is possible to plug in components from one UML model into another. PlugIn facilitates the constructions of unique profiles that are more or less in line with ISO 10000.

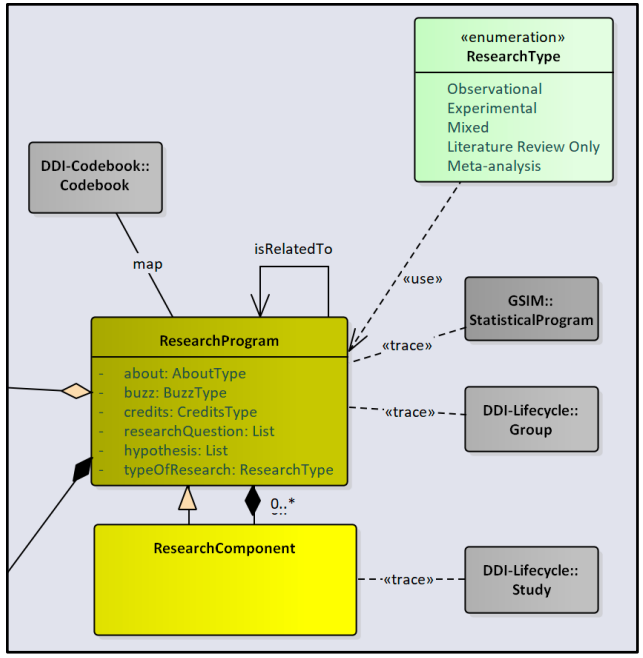
C. Alignment Specifics between DDI - CDI and other Standards in the DDI Family of Standards

1. DDI Codebook 2.5 (Nesstar Publisher)

a. Relationship with DDI - CDI: Map relationship with Upper Model

b. Description:

The DDI - CDI Upper Model includes a ResearchProgram class. It is composed of zero or more ResearchComponents that inherit from ResearchProgram. ResearchProgram includes three property groups – *about*, *buzz* and *credit*. *About* refers to the function and/or coverage of a ResearchProgram or ResearchComponent. *Buzz* refers to its social media profile – the audience, their comments, the reviews an audience might publish and so forth. Finally, *credit* follows CRediT and the Contribution Role Hierarchy and specifies the roles humans and/or programs play in the creation of a ResearchProgram and/or its ResearchComponents.



It is important to note that one or more of these property groups may be of use in other classes too. Consider a research center with a ResearchProgram that builds many research products (ResearchComponents) where some of the ResearchComponents relate to other ResearchComponent to form one or more time series and/or one or more sets of cross-sectional research products grouped by theme. The

ResearchProgram and/or its ResearchComponents may be associated with a ResearchDataStore which contains InformationObjects we might also want to *credit* and/or *buzz* and/or *about*. Indeed, this might as well be the case with data and metadata elements that compose these InformationObjects too.

Currently, in the Core Foundational Metadata there is a citation class called Annotation from which most of the foundational classes inherit. This complicates the foundational classes, one would like to say, immeasurably. As such, Annotation *does not* represent the future approach of Core with citation. Instead the Upper Model is a laboratory DDI - CDI is using to hatch the future approach. In one approach the various property groups might be implemented as a set of complex (structured) data types that other classes like ResearchProgram and ResearchComponent might specify or not as properties. In another implementation citations might become plugins to the model from other models. And so forth.

A table of these properties and their sources by property group is included in Appendix (A).

Additionally, Appendix (B) is a map between Codebook and the classes and property groups from the Upper Model.

And Appendix (C) is a poster from a map between Codebook and the Core foundational classes. This map is noteworthy in several respects. First, the map includes the *paths* one would follow through the foundational classes to get to the DDI - CDI foundational class property corresponding to a DDI 2.5 "leaf" in Codebook. Secondly, these paths are model dependent so, for example, a change in the relationship between classes in the model changes the map. And, finally, this map, using the *paths*, is *machine actionable*. This is in contrast to the map in Appendix (B) which is more conceptual.

2. DDI Lifecycle 3.2 Group Component

a. *Relationship with DDI - CDI:* Trace relationship with Upper Model

b. *Description:*

The DDI - CDI Upper Model includes a ResearchProgram class. It is composed of zero or more ResearchComponents that inherit from ResearchProgram. The DDI 3.2 Group component has a trace relationship with these two Core classes.

3. DDI Lifecycle 3.2 Conceptual Component and Logical Product Component

a. *Relationship with DDI - CDI:* Map relationship with Foundational Metadata (Concepts and Variables)

b. *Description:*

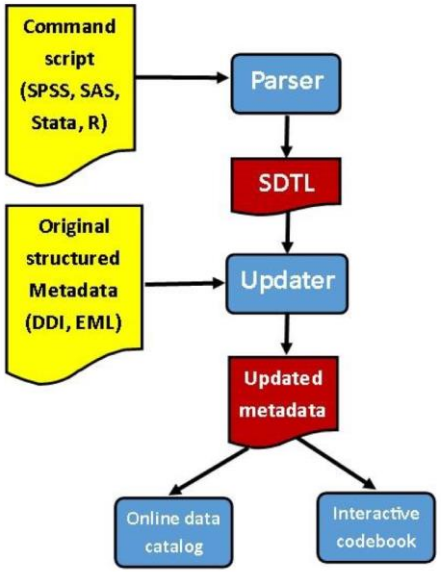
The map is a future because DDI Lifecycle 3.2 doesn't support Statistical Classification the way it is supported in DDI Lifecycle 3.3 (in evaluation) and the DDI - CDI Foundational Metadata

4. Structured Data Transformation Language

a. Relationship with DDI - CDI: Plugin relationship with Data Management Model

b. Description:

C2Metadata's Structured Data Transformation Language (SDTL) is an independent intermediate language for representing data transformation commands. Commands in four software packages (SPSS, Stata, SAS, and R) are translated into JSON schemas, which are machine actionable.



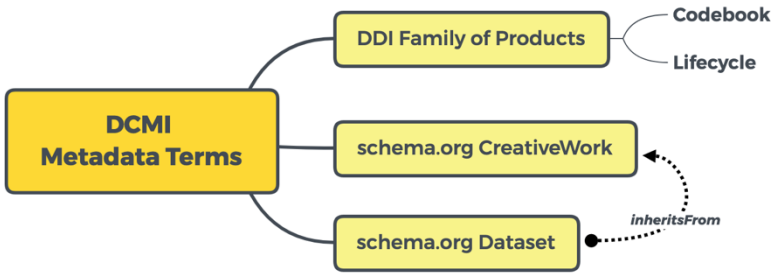
D. Alignment Specifics between DDI - CDI and External Standards

1. Dublin Core Metadata Initiative Metadata Terms

a. Relationship with DDI - CDI: Map relationship with Upper Model

b. Description:

The Dublin Core Metadata Initiative (DCMI) Metadata Terms is a superset of the Dublin Core Metadata Element Set (DCMES). Historically speaking, these vocabulary terms – the core together with its extension -- are the basis for all digital resource description. Indeed, DCMES and its DCMI Metadata Terms extension are core vocabulary terms in both DDI Codebook and DDI Lifecycle in the DDI family of products. DCMES and its DCMI Metadata Terms extension are vocabulary terms that are also at the core of schema.org CreativeWork which, in turn, provides the context in schema.org for the schema.org Dataset.



Previously, it was noted that the DDI - CDI Upper Model includes a ResearchProgram class which is composed of zero or more ResearchComponents that inherit from ResearchProgram and that ResearchProgram includes three property groups – *about*, *buzz* and *credit* – that are aligned with DDI Codebook. These same property groups are also aligned with DCMI Metadata Terms and a map between DCMI Metadata Terms and the DDI - CDI Upper Model has been included in the Appendix (C).

2. schema.org Dataset

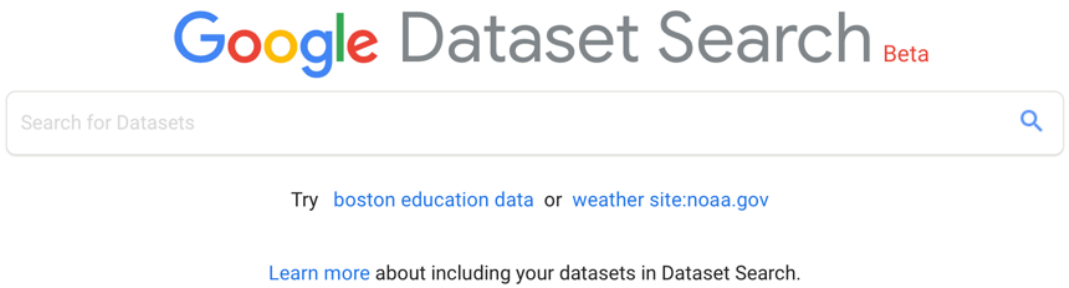
a. *Relationship with DDI - CDI:* Map relationship with Upper Model

b. *Description:*

schema.org is "a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet, on web pages, in email messages, and beyond. In addition to people from the founding companies (Google, Microsoft, Yahoo and Yandex), there is substantial participation by the larger Web community, through public mailing lists such as public-vocabs@w3.org and through GitHub. See the releases page for more details" (from About schema.org).

The schema.org Dataset is "a body of structured information describing some topic(s) of interest" consisting of the Dataset context (CreativeWork), the Dataset variables measured each of which has a PropertyValue that includes the value, its type, any semantics associated with the type like codes and their concepts, a unit identifier, the measurementTechnique together with an extension mechanism whose use can be determined by the community of practice.

A schema.org Dataset has several representations including JSON-LD. Using JSON-LD or another representation, Google Dataset Search "lets you find datasets wherever they're hosted, whether it's a publisher's site, a digital library, or an author's personal web page."



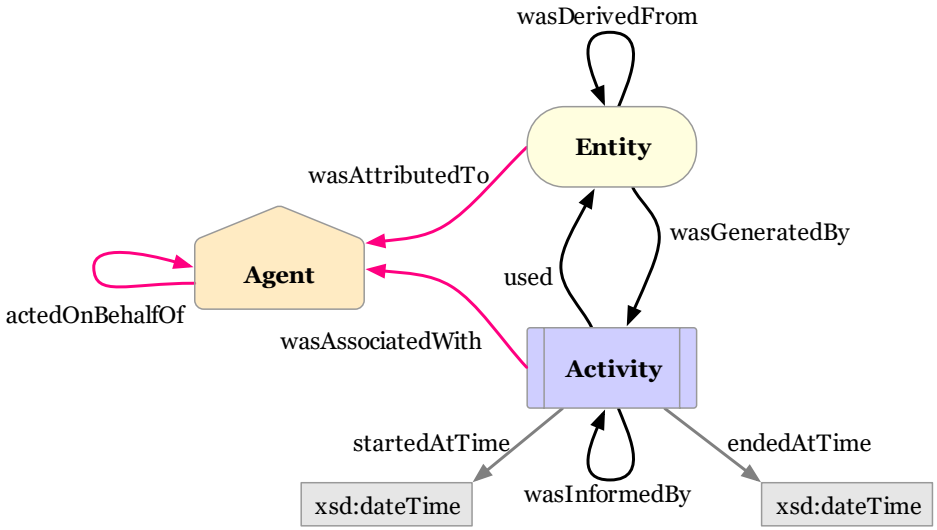
Note that a schema.org Dataset example markup represented in JSON-LD is included in the DDI - CDI Examples Document. This example is based on the Health and Demographic Surveillance Systems (HDSS) event dataset schema that is widely used across much of Sub-Saharan Africa for purposes of demographic surveillance.

3. PROV-O

a. *Relationship with DDI - CDI:* Trace relationship with Data Management

b. *Description*

In this specification, the standard states: “It provides a set of classes, properties, and restrictions that can be used to represent and interchange provenance information generated in different systems and under different contexts. It can also be specialized to create new classes and properties to model provenance information for different applications and domains.”



The intention of the recommendation is to provide an extensible model, which can be applied to different domains and systems. DDI - CDI uses PROV-O in this way, by specializing specific classes to reflect those found in the DDI - CDI model.

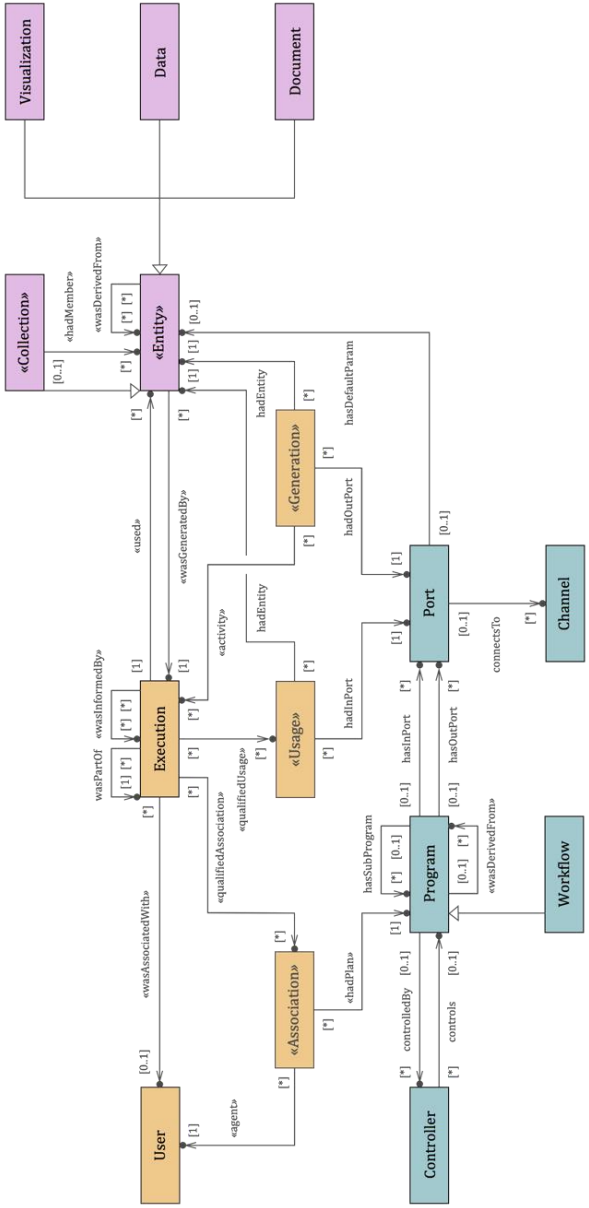
PROV-O is expressed as an OWL2 ontology, rather than as a UML model. Classes depicted in the DDI - CDI diagrams represent the corresponding classes as described in the PROV-O OWL2 definition, but the UML classes are created by the DDI - CDI to represent those classes – they are not a formalism which can be taken directly from the specification in UML form. In DDI - CDI, the trace relationships typically represent specializations of the more generic PROV-O classes, but would indicate that the properties and relationships of these classes can also be applied to the appropriate DDI - CDI objects.

4. ProvONE

a. *Relationship with DDI - CDI:* Trace relationship with Data Management

b. *Description*

Recently, PROV-O has been extended by ProvONE. ProvONE makes PROV-O data and computer program specific. In PROV-O entities didn't distinguish data at different level of specificity. And the PROV-O Plan entity lacked the specificity to describe the structure of computer programs and the specific successions of activities (workflows) that programs create. Here is the ProvONE Conceptual Model:


Visualization
Data
Document
«Collection»
«Entity»
«Generation»
«Usage»
«Association»
Execution
Port
Channel
Program
Workflow
User
Controller
«hadMember»
«wasDerivedFrom»
«used»
«wasGeneratedBy»
hadEntity
hasDefaultParam
hadOutPort
«activity»
hadInPort
connectsTo
«wasInformedBy»
wasPartOf
«qualifiedAssociation»
«qualifiedUsage»
hasInPort
hasOutPort
hasSubProgram
«wasAssociatedWith»
«hadPlan»
controlledBy
controls
«agent»

5. GSIM Concept Group (1.2)

a. *Relationship with DDI - CDI:* Trace relationship with Concepts and Variables (Foundational Metadata)

b. *Description*

GSIM provides a reference model of all the information – data, metadata, metrics, etc. – used in statistical production by national statistical agencies, and supra- and international statistical institutions. It is a product of the High-Level Group for the Modernization of Official Statistics (HLG-MOS), coordinated by the UN Economic Committee for Europe's Statistical Programme.

As a reference model, GSIM is primarily intended to facilitate more precise communication between implementers but does not serve directly as an implementation standard. It is possible that DDI - CDI can be used as a model for the implementation of some GSIM constructs. GSIM is modeled in UML, which makes the trace relationships function exactly as described in the UML specification quoted above. Correspondences between DDI - CDI classes and those in GSIM are often very direct, as they often model identical phenomenon with different levels of focus. Nuances of these relationships will be described on a class-by-class basis as appropriate in DDI - CDI.

GSIM consists of several groups including the Structure Group, the Business Group and the Concept Group. The GSIM Concept Group traces to Concepts and Variables (Foundational Metadata) in DDI - CDI. Both the DDI - CDI Variable Cascade and the Datum constructs in DDI - CDI extend the comparable classes from GSIM.

6. GSIM Structure Group (1.2)

a. *Relationship with DDI - CDI:* Trace relationship with Data Description

b. *Description*

See above for general information about the GSIM reference model. As described above, GSIM consists of several groups including the Structure Group, the Business Group and the Concept Group. The GSIM Structure Group traces to Data Description in DDI - CDI. Data Description adds new structure components to the group of structure components defined in the GSIM Structure Group enabling DDI - CDI to describe types of data that GSIM does not describe.

7. GSIM Business Group (1.2)

a. *Relationship with DDI - CDI:* Trace relationship with Data Management

b. *Description*

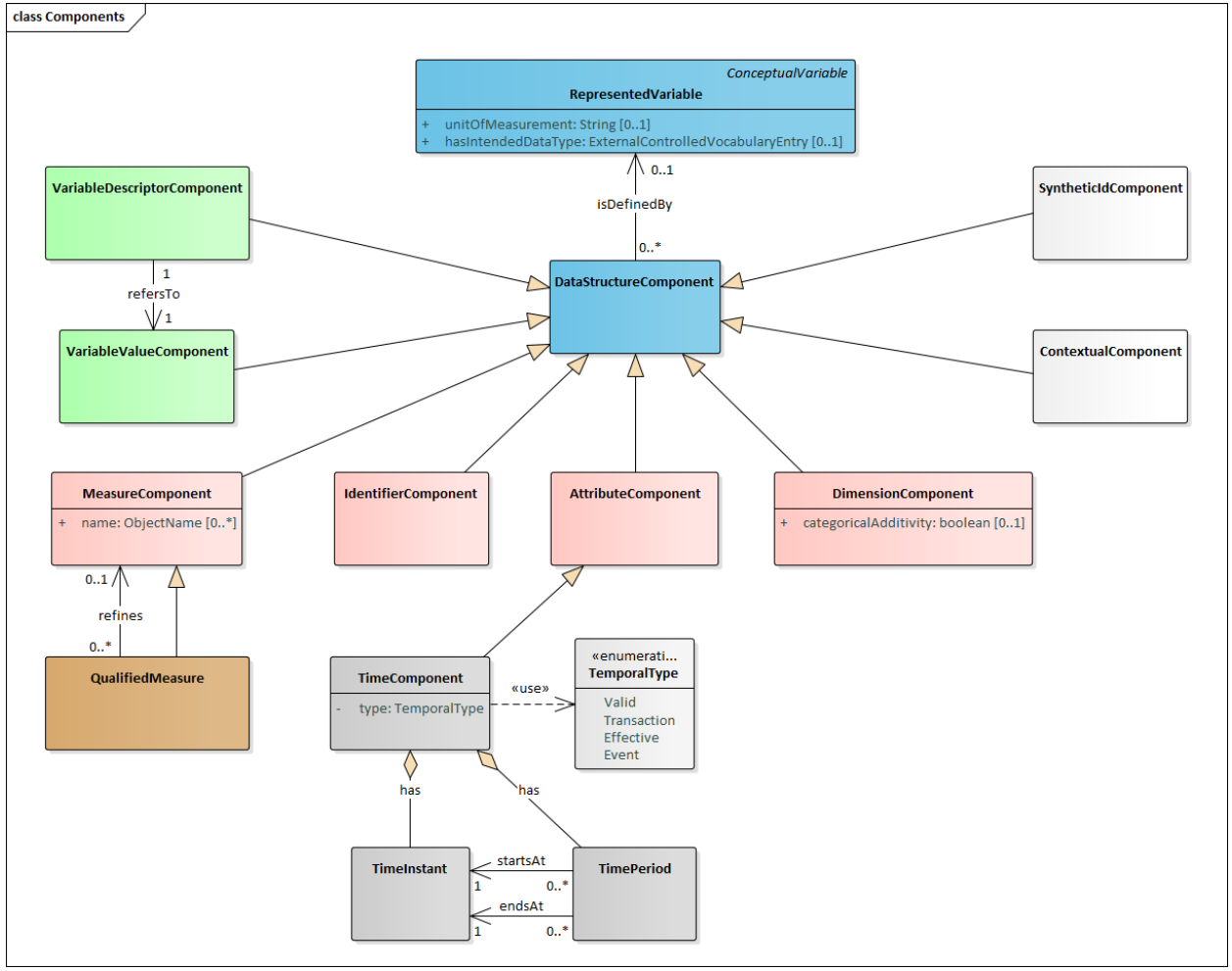
See above for general information about the GSIM reference model. As described above, GSIM consists of several groups including the Structure Group, the Business Group and the Concept Group. The GSIM Business Group traces mostly to Data Management in DDI - CDI. Data Management traces to PROV-O and ProvONE more than it does to the GSIM Business Group.

8. ISO 17369 Statistical Data and Metadata Exchange (SDMX)

a. *Relationship with DDI - CDI:* Map relationship with Data Description DimensionalDataSet

b. *Description:*

The current version of SDMX as of this writing is the 2.1 Consolidated version 2013. The SDMX Information Model provides a UML formalization against which the DDI - CDI model can be mapped. Recall from Document Two (the Detailed Model) that DDI - CDI has a rich set of data structure components that can be combined to define many dataset types including the dataset definitions supported by the SDMX Information Model.



A map between the DDI - CDI DimensionalDataSet and components of the SDMX Information Model is under development.

9. The RDF Data Cube Vocabulary (QB)

a. *Relationship with DDI - CDI:* Trace relationship with Data Description DimensionalDataSet

b. *Description:*

The RDF Data Cube Vocabulary is based on the SDMX 2.0 standard and covers the description of multi-dimensional data sets. It provides a set of classes and properties but does so without using any standard formalization. Consequently, the classes represented in the DDI - CDI model are created by DDI - CDI to reflect Data Cube classes, but do not come from any specific published UML model. They will carry with them the properties of the Data Cube classes when they appear in trace relationships with the DDI - CDI model.

10. RAIRD Information Model

a. *Relationship with DDI - CDI: :* Trace relationship with Data Description LongDataSet

b. *Description*

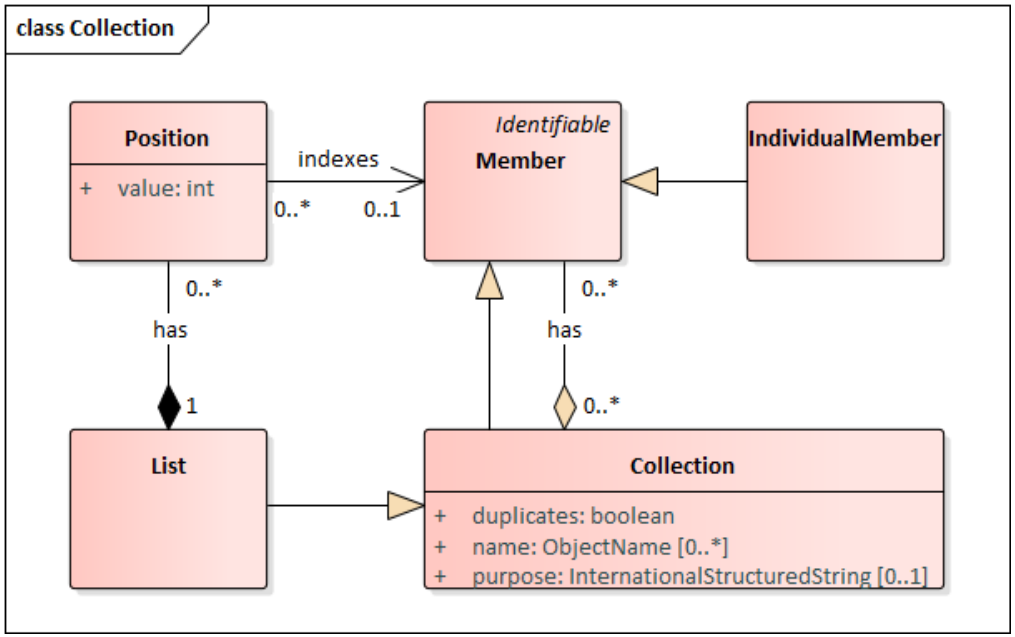
RAIRD supports the description of an event history dataset. In an event history dataset RAIRD builds on the GSIM datum-based data structure and adds an event period such that all observations, regardless of type, may be represented as follows:

Unit Identifier	Variable Reference	Value	Start Date	End Date
0937	DOB	1971-05-03	1971-05-03	-
0937	GENDER	1	1971-05-03	-
0937	MAR_STAT	M	2003-08-04	2010-02-02
0937	CIVIL_UNION_ID	4765	2003-08-04	2010-02-02
0937	MAR_STAT	D	2010-02-02	2012-02-02
0937	MAR_STAT	M	2012-02-14	-
0937	CIVIL_UNION_ID	5678	2012-02-14	-
2100	DOB	1972-03-05	1972-03-05	-
2100	GENDER	2	1972-03-05	-
2100	MAR_STAT	M	2012-02-14	-
2100	CIVIL_UNION_ID	5678	2012-02-14	-

III. Design Patterns

A. Using the Collections pattern

DDI-CDI introduces a generic Collections pattern that can be used to model different types of groupings and aggregations of objects, from simple unordered sets to all sorts of hierarchies, nesting and sequences.



A collection is basically a container, which could be either a set, i.e. unique elements or a bag, i.e. repeated elements. Collections can also be extended with richer semantic, e.g. generic, partitive, and instance, among others, to support a variety of DDI 3.x and GSIM structures, such as Node Sets, Schemes, Groups, sequences of Process Steps, etc. This pattern provides an abstraction to capture commonalities among a variety of seemingly disparate structures.

A Collection consists of zero, one or more Members. A Member could potentially belong to multiple Collections. Sets are defined by setting the *duplicates* property to *false,* bags by setting it to *true*. Membership in a Collection is maintained by a *has* aggregation.

List is an extension of Collection for sequentially ordered collections. It uses Position and its *value* property to indicate the location of a Member in the sequence. Note that Position does not extend from Identifiable because it's never managed independently from the List it belongs to.

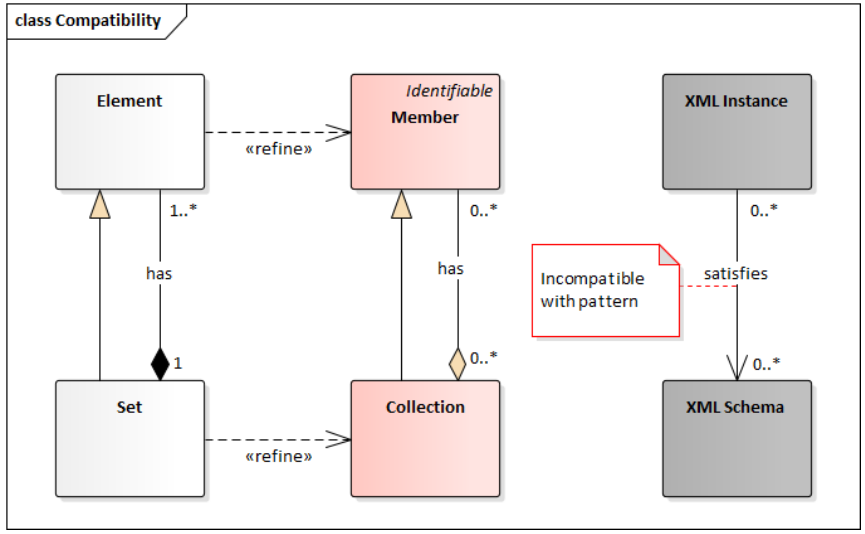
A Collection is also a Member, which allows for nesting of collections in complex structures. Members have to belong to some Collection, except in the case of nested collections where the top level is a member that doesn't belong to any collection. In addition, IndividualMember can be used to indicate that the member is not itself a Collection.

This pattern can be used via a special type of association called *refine*. DDI-Core uses *refine* to say that a group of class "behave" like the Collections pattern.

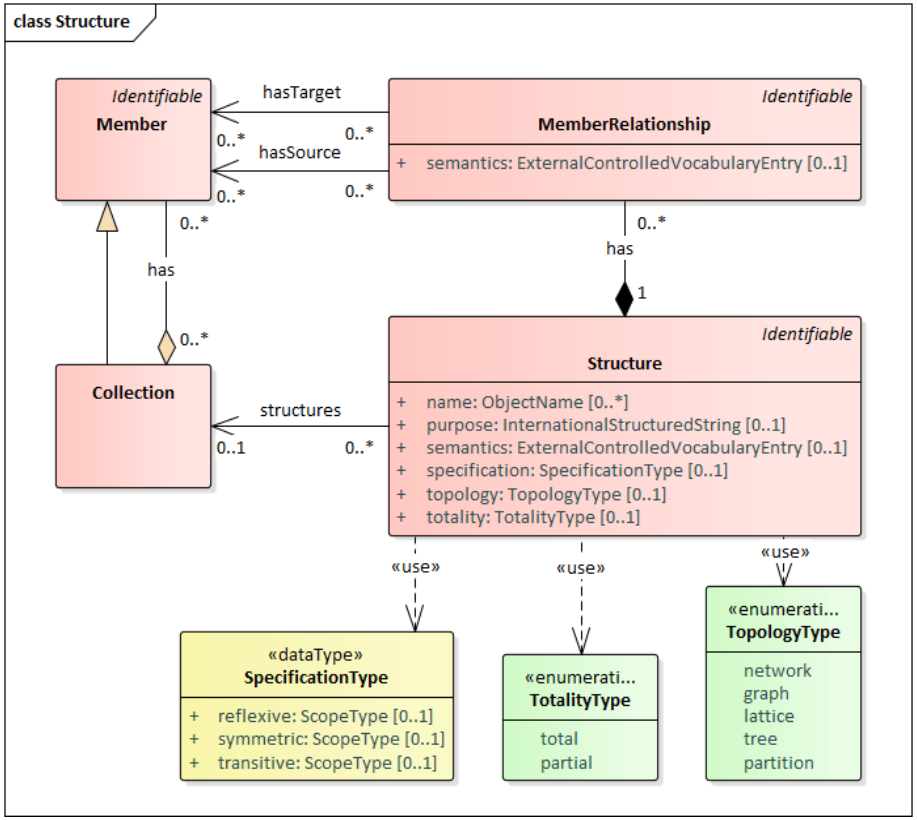
Refine is a sort of weak realization. To refine this pattern, all classes involved must be associated in a way that is compatible with the pattern. As a rule of thumb, a more restrictive type of association than the one that appears in the pattern is compatible, a looser one is not. For instance, since a Collection uses a *has* aggregation to identify its member, classes realizing the pattern need to be related by either an aggregation (same type) or a composition (more restrictive). In addition, the association has to be in the right direction, so that the class refining Collection is the "whole" (the diamond end) and the class refining

Member is the "part". Similar compatibility rules apply to cardinality. Furthermore, all associations must be refined, with the exception of *IsA* associations, which are usually part of the pattern definition and do not apply to individual refinement in the same way. Renaming properties and associations does not affect compatibility as long as the documentation clearly explains how they map to the association in the pattern.

For instance, consider the model diagram below. A Set is defined in this example as being composed of at least one Element, i.e. no empty Sets are allowed, and an Element always belong to one and only one Set. This is indicated by the cardinalities on the *has* association. Consequently,



deleting the Set will also delete its Elements. Such definition is compatible with the Collections pattern and thus Set and Element can refine Collection and Member, respectively. In contrast, Schema and XML Instance cannot refine the pattern because the Schema is not a grouping of XML Instances so the notion of a Collection being a container of Members doesn't hold.



Beyond the sequential ordering provided by List, the Collections pattern includes a Structure class that supports more complex structures and orderings of members via MemberRelationship.

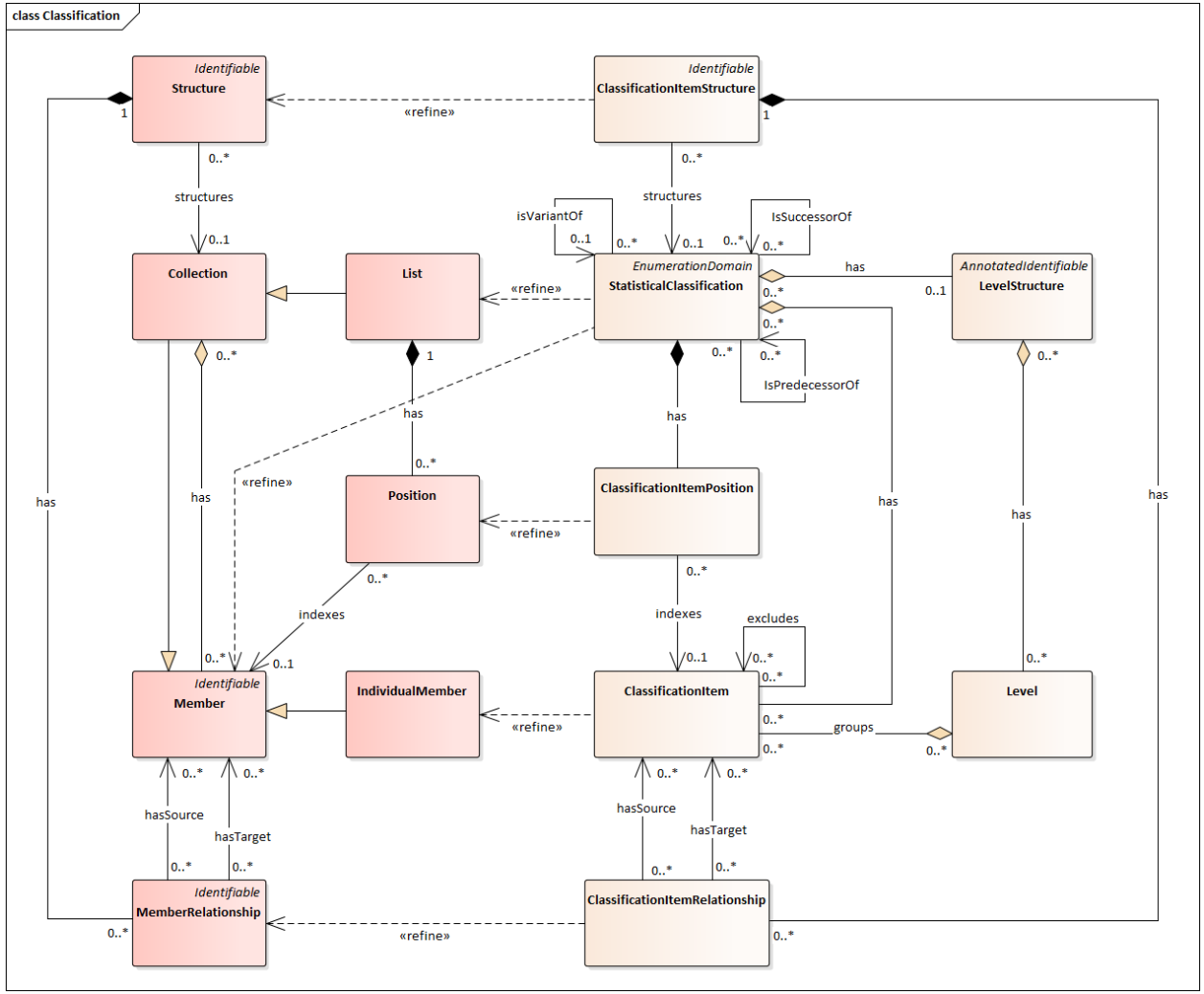
A Structure consists of one or more MemberRelationships, which are tuples linking Members at the end of the *hasSource* and *hasTarget* associations.

A Structure can have a *specification* property, e.g. *reflexive*, *symmetric*, and *transitive*, a *totality* property, e.g. *total* or *partial*, and a *topology* property, e.g. network, graph, lattice, tree, partition. These properties are defined by SpecificationType, TotalityType and TopologyType, respectively. A Structure can also have *semantics* defined by an ExternalControlledVocabularyEntry.

A Structure is said to be *total,* if all members of the associated collection are related to each other (otherwise, it is said to be *partial*); s*ymmetric*, if for any pair of members, *a*, *b* in the associated collection, whenever *a* is related to *b* then also *b* is related to *a* (otherwise, it is said to be *anti-symmetric*); r*eflexive,* if all members of the associated collection are related to themselves (otherwise, it is said to be *anti-reflexive*); and *transitive,* if for any members *a*, *b*, *c* in the associated collection, whenever *a* is related to *b* and *b* is related to *c* then *a* is also related to *c* (otherwise, it is said to be *anti-transitive*).

These characteristics can be combined to define different types of Structures, e.g. equivalence relations and partial order relations, among others. Equivalence relations are useful to define partitions and equivalence classes (e.g. Levels in a StatisticalClassification). Partial order relations can be used to represent lattices (e.g. class hierarchies, partitive relationships), parent-child relations can define trees and acyclic precedence relations can represent directed acyclic graphs (e.g. molecular interactions, geospatial relationships between regions).

Let us illustrate how this model works with a simple instance. Consider a North American Industry Classification System (NAICS) StatisticalClassification with ClassificationItems representing type of economic activity, such as *Mining*, *Manufacturing*, *Finance*, etc. The following diagram shows how the statistical classification classes refine the Collections pattern.



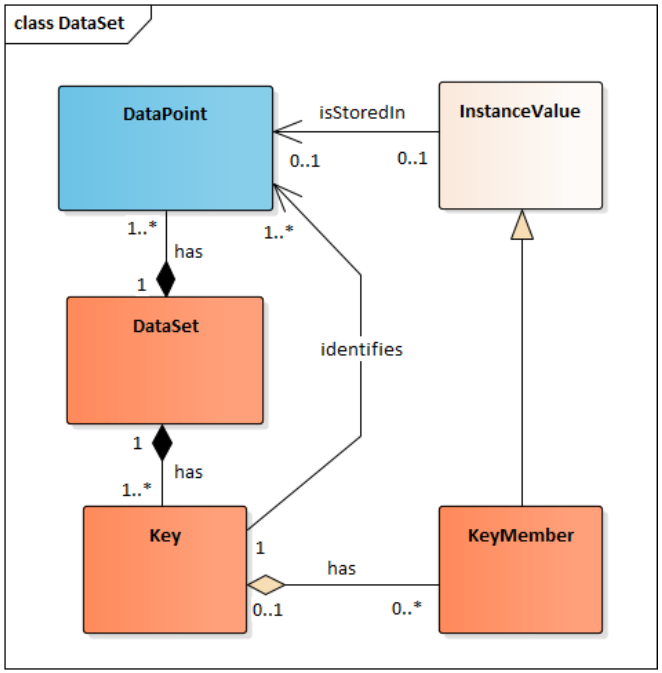
Note that StatisticalClassification refines Collection and ClassificationItem refines IndividualMember. This means we can view ClassificationItems such as *Manufacturing*, *Machinery manufacturing*, and *Educational services* in NAICS as Members organized in a hierarchy by a ClassificationItemStructure which consists of a set of ClassificationItemRelationships representing the parent-child relationships between

items. For instance, <*Manufacturing*, *Machinery manufacturing*> is a ClassificationItemRelationship in which *Manufacturing* is the source and *Machine manufacturing* is the target.

Note that by maintaining the hierarchy in a separate structure, i.e. ClassificationItemStructure, items can be reused in multiple classifications. For instance, a NAICS variant groups economic activities into two main industry groupings: the *goods-producing industries* and the *services-producing industries*. Because of the separation of hierarchy and categories, adding that high-level grouping doesn't require a change in the structure and definition of the underlying industry ClassificationItems.

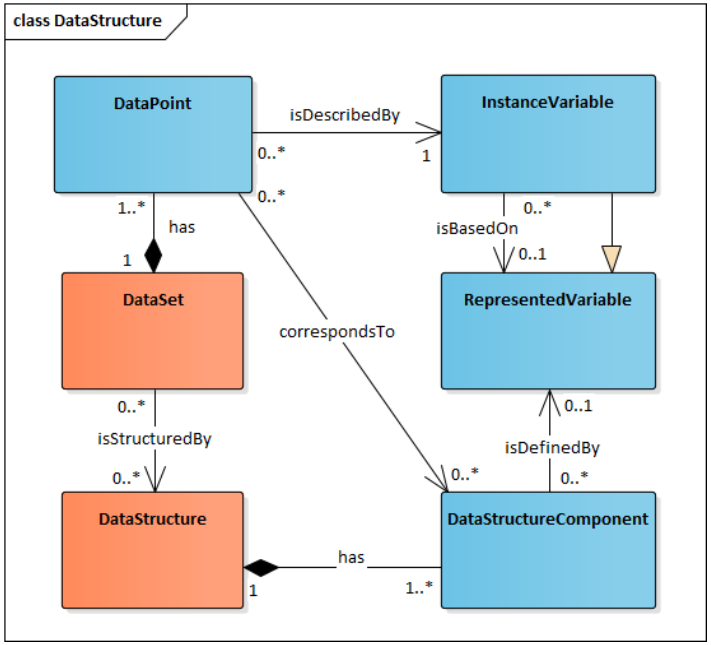
B. Using the Data Description pattern

Another pattern in DDI-CDI is data description. This pattern organizes data into datasets and their data structures. DDI-CDI includes four basic types of data sets: Key-Value, Wide, Long and Dimensional. All these types of data sets, and more, can be described with the same pattern.



A DataSet is an organized collection of data that consists of DataPoints and Keys. A DataPoint stores an InstanceValue, which is essentially a single data instance. Within DataSets, DataPoints are uniquely identified by Keys, which are collections of InstanceValues and as such they are also stored in other DataPoints. Each InstanceValue that forms part of a Key is called a KeyMember. For instance, a social insurance number and a date can be two key members that together form a key which identifies data points containing blood test results of a patient. The set of data points identified by a key constitutes a *record* or a *row* in rectangular data files and other traditional data structures. We don't have an explicit notion of a record in our model to support other types of flexible data organizations.

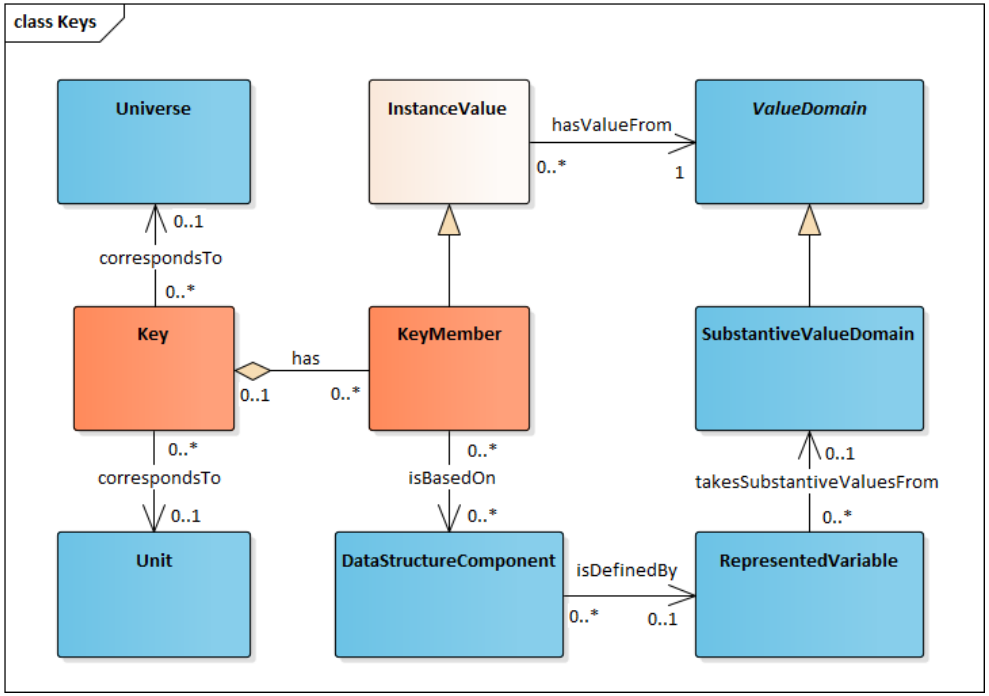
DataSets are further described by DataStructures. This model also supports schema on-read type of data description, in which data can be stored with basic or no descriptive information and then descriptions can be added as necessary at the time of use.



A DataStructure consists of multiple DataStructureComponents defined by RepresentedVariables. Essentially, a data structure component is the use of a represented variable in the context of a given data structure. These components could be of different types, the most common ones being *identifier*, *dimension*, *measure* and *attribute*. For instance, the same marital status variable can play the role of a *measure* in one data structure and a *dimension* in another. In such a case there are two DataStructure components, i.e. a MeasureComponent and a DimensionComponent, using the same marital status variable.

Each DataPoint in a DataSet corresponds to some DataStructureComponent in its associated DataStructure. Note that the same DataStructure can apply to multiple DataSets. Therefore, the InstanceVariable associated to each DataPoint needs to be related to the RepresentedVariable used by the DataStructureComponent that corresponds to that DataPoint.

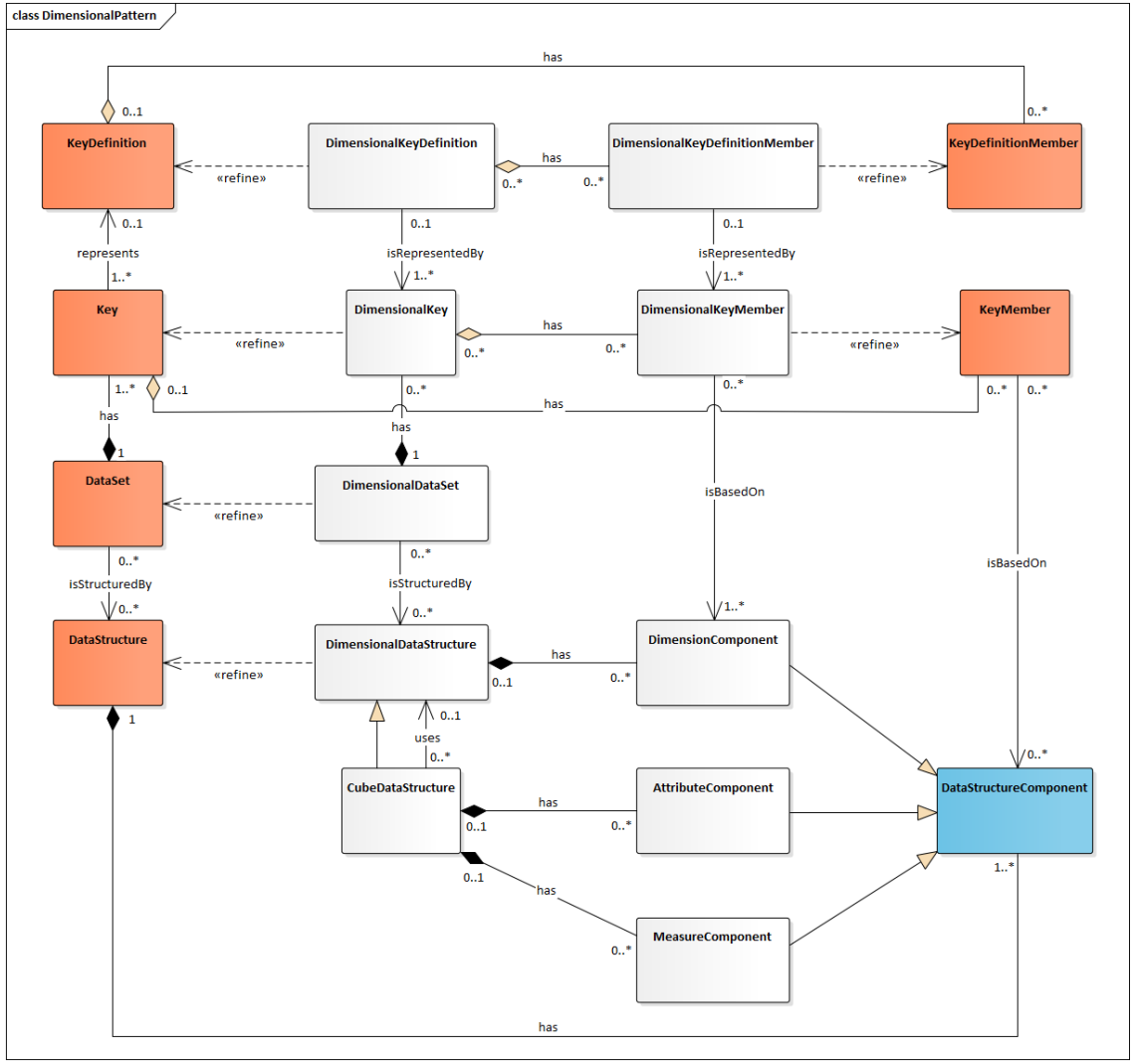
There is a similar relationship between KeyMembers and DataStructureComponents. Remember that KeyMembers are also InstanceValues and they are based on some DataStructureCompoment. As such, a KeyMember is a value from some ValueDomain, which is the same ValueDomain of the RepresentedVariable associated with the DataStructureComponent the KeyMember is based on. The diagram below shows these relationships.



In addition to identifying data points and relating to data structure components, a key is also related to either a Unit or a Universe. A Key corresponds to a Unit when the DataPoints they identify contain microdata whereas it corresponds to a Universe when they contain aggregate/macro data. Note that corresponding to a Unit (or a Universe) doesn't mean uniquely identifying it, since different Keys can correspond to the same Unit, like in a long DataSet where multiple rows correspond to measures related to the same Unit. The job of uniquely identifying Units (and Universes) rests on the IdentifierComponents (and the DimensionComponents, respectively). Keys uniquely identify only sets of DataPoints, which form records or rows.

As mentioned, Keys are groups of KeyMembers, which are InstanceValues coming from ValueDomains. As such, they usually have conceptual counterparts called KeyDefinition and KeyDefinitionMember, respectively. A KeyDefinition conceptually define the DataPoints a Key identify. They do that by grouping ConceptualValues (the KeyDefinitionMembers), which are concepts represented by the KeyMembers. In other words, Keys are representations of concepts in KeyDefinitions. These concepts come from the ConceptualDomain of the ConceptualVariable associated to the RepresentedVariable from whose ValueDomain KeyMembers take values from.

Let us see how this pattern works with an example. Consider a data cube that contains the average income and the total population by city. City is a DimensionComponent and average income and total population are MeasureComponents. They are grouped into a CubeDataStructure which structures a DimensionalDataSet. (CubeDataStructure is an extension of DimensionalDataStructure, which is a collection of dimensions that can be reused and maintained separately.) DimensionalDataStructure refines DataStructure and DimensionalDataSet refines DataSet.

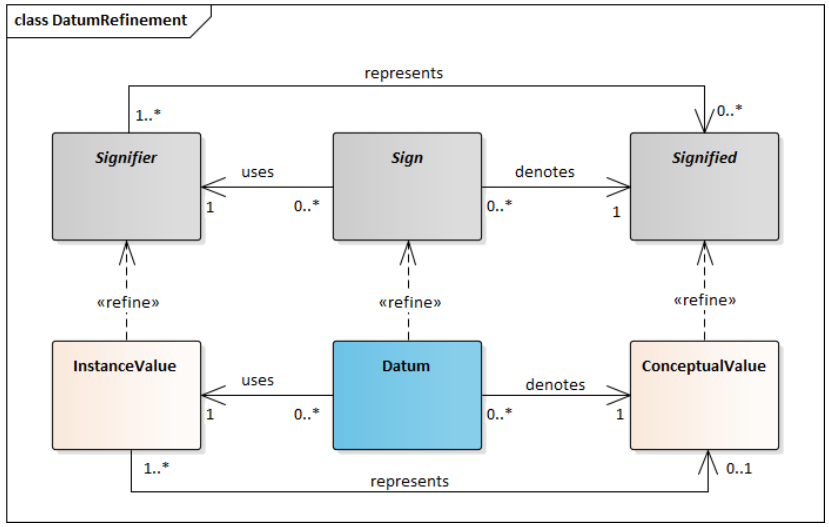


The rest of the pattern needs to be refined by the appropriate concrete classes. Key, KeyMember, KeyDefinition and KeyDefinitionMember are refined by DimensionalKey, DimensionalKeyMember, DimensionalKeyDefinition and DimensionalKeyDefinitionMember, respectively. Note that rather than being based on DataStructureComponent, a DimensionalKeyMember is based on DimensionComponent instead. That's a valid refinement of the pattern since DimensionComponent is an extension of DataStructureComponent. In this way, the refinement is more precise than if it were based on DataStructureComponent.

C. Using the Signification pattern

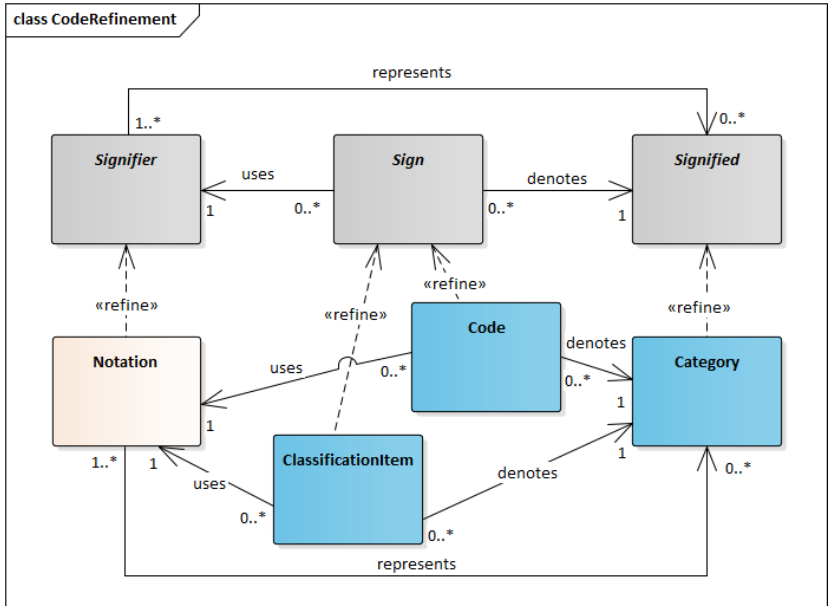
A Sign links a Signified with a Signifier that denotes it. A Signifier is a concept whose extension consists of tokens (perceivable objects). The Signified is the concept being represented by a Signifier. For instance, the concept of *integer five* is the Signified and all its different representations are tokens of its Signifier.

Signifier, Sign and Signified become part of the DDI-CDI signification pattern.



A Datum is an example of a Sign that links an InstanceValue with a ConceptualValue. An InstanceValue is a single data instance as it appears in a DataPoint. It's the representation of a concept, more precisely of a ConceptualValue, which is a concept with the notion of equality defined. This

notion of equality is important when representing data since data needs to be copied and compared, which is only possible with some equality operation. InstanceValue, ConceptualValue and Datum therefore refine Signifier, Signified and Sign, respectively.



The reason for making Signifier, Sign and Signified into a pattern to be refined as opposed to classes to be extended is that Concepts are not always Signifieds, which is what a specialization would imply. In fact, *a Concept is a Signified only if there is a Signifier that represents it*. The refinement means that the Concept is going to behave like a Signified only in the context of the pattern.

Another example of the use of the pattern is Code, which enters into the picture as a refinement of Sign. A Code then is a Sign that has Non-Linguistic Signifiers and where the Signified is a Category (Concept). ClassificationItem is a similar refinement of Sign linking a Notation to a Category. The Signifier in this case is Notation, which is just the representation of the Category within the context of a Code or a Classification Item.

IV. UML Subset

Please see the document "UML Class Diagram - Practioner's Subset for Data Modeling - Detailed List" in this package at `\DDI-CDI Public Review 1\1 Specification Documents\Supporting Documents`.

V. Design Notes and Modelling Approach

A. Introduction

DDI-CDI follows a model-driven approach. Model Driven Architecture® (MDA®) is an approach to software design, development and implementation spearheaded by the Object Management Group (OMG), a computer industry standards consortium. In the here used meaning, it starts with a model of data and the description of it (the application's business functionality and behavior) using Unified Modeling Language (UML). This model remains stable as technology evolves, extending and thereby maximizing software ROI (return on investment). Portability and interoperability are built into the architecture.

DDI-CDI uses a subset of the Class Diagrams of UML version 2. This subset of UML class diagram elements (see preceding section) is intended for data modeling. It focuses on core elements which are well known in object-oriented programming. The subset focuses on elements which describe classes, their interrelationships, and their attributes. This subset enables simple modeling, easy understanding, portability to many UML tools, and good mapping options to target representations. It is described in the document on "UML Class Diagram - Practioner's Subset for Data Modeling".

The specification UML Version 2.5.1[1] is used as canonical UML specification. The namespaces of UML 2.4.1 and XMI 2.4.1 are used in the representation as Canonical XMI. The version 2.4.1 is currently implemented in a larger number of UML tools. The namespaces of UML and XMI can be changed to the ones of 2.5.1. The used subset of UML is defined identically in the UML versions 2.4.1 and 2.5.1.

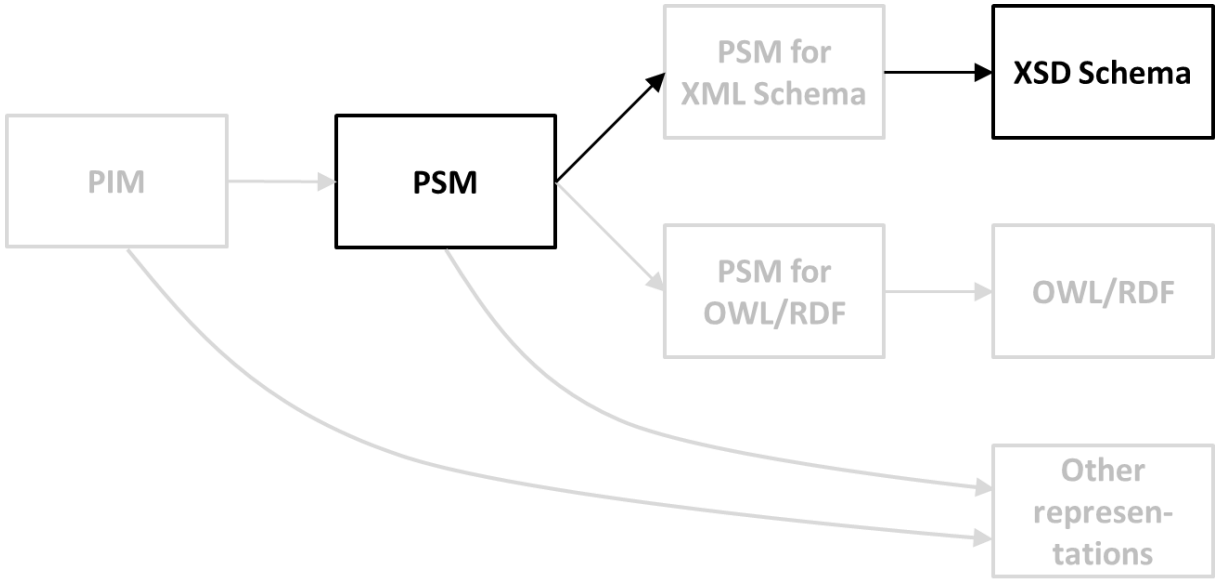
B. Type of Model

In model-driven architecture, a distinction is made between the platform independent model (PIM) and the platform-specific models (PSM). The PIM is translated to one or more PSMs.

[1] UML 2.5.1, https://www.omg.org/spec/UML/2.5.1/PDF

DDI-CDI uses this approach to make a distinction between the PIM (for conceptual purposes), the generic PSM (for multiple syntax representations with an object-oriented approach), and dedicated PSMs for specific syntax representations (encodings). All models are realized in UML. The PSMs are using the UML subset mentioned above.

PIM, PSM, and syntax representations in DDI-CDI (grey parts are not active yet):



Only the generic PSM exists currently and the derived XML Schema representation. The specific PSM for XML Schema is identical to the generic PSM.

The OWL/RDF representation is in the works.

The current model development was done in the generic PSM. It is planned for the future that the model development is done in the PIM which is then translated according to a set of business rules to the generic PSM. The differences between the future PIM and the generic PSM might comprehend for example navigability and multiplicity in associations.

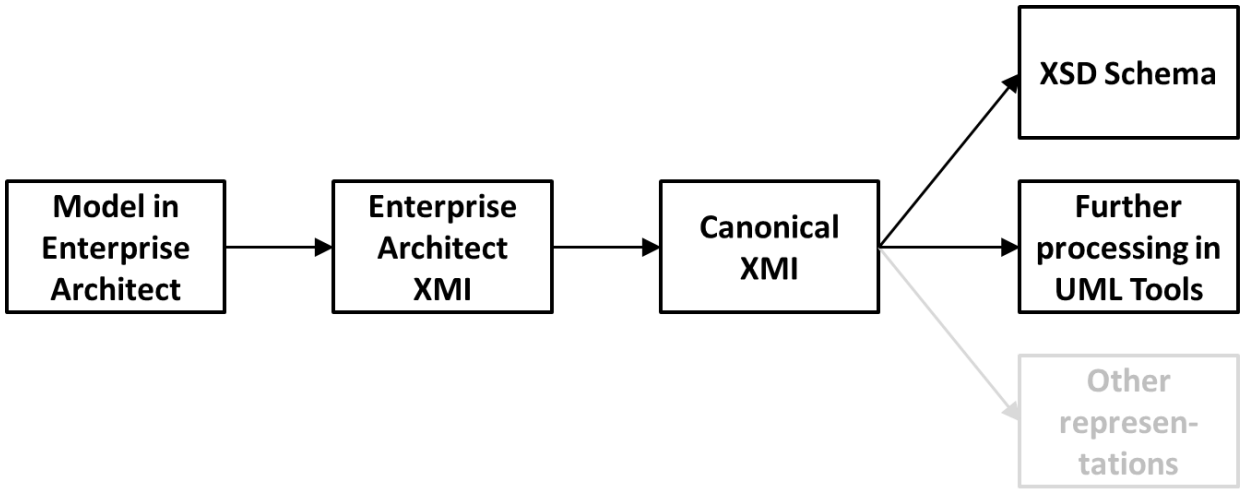
The UML tool Sparx Enterprise Architect[2] is used for the development of the generic PSM. The generic PSM is available as Enterprise Architect file. It can be viewed with the freely available viewer Enterprise Architect Lite[3].

C. Model Transformations

The transformation from the generic PSM to the XML Schema is done with XSLT programs. The Canonical XMI representation of the PSM is the input to these programs.

Enterprise Architect can export an own flavor of XMI. This is transformed by a XSLT to Canonical XMI.

Transformation chain:



D. Canonical XMI

DDI-CDI is available as Canonical XMI. This enables the import of the model into common UML tools. These tools can be used to analyze the model, to relate it to other UML models, and to generate syntax representations which are provided by these tools.

[2] Sparx Enterprise Architect, https://sparxsystems.com/products/ea/
[3] Enterprise Architect Lite, https://sparxsystems.com/bin/EALite.msi

The XML Metadata Interchange (XMI) is an Object Management Group (OMG) standard for exchanging metadata information via Extensible Markup Language (XML). The different vendors of UML tools have often XMI flavors which are specific to their tools. OMG proposed Canonical XMI to improve interoperability.

> "*Canonical XMI: A specific constrained format of XMI that minimizes variability and provides predictable identification and ordering. The constraints are detailed in Annex B.*"[4]

The Canonical XMI is available in two kinds one has the same association names as in the Enterprise Architect file, one unique association names across the model. The latter is intended for the transformation to XML Schema and for the import in some UML tools which complain about non-unique association names. The background is that strict UML requires unique names per item type in one package. The unique association names are constructed on the basis of the original association name plus the names of the connected classes.

E. Notes on Modeling

1. Structural Items

Package

- A package expresses a region of the interrelated content
- Packages (and classes) are named and organized in a way that they can be easily moved to another location of the model
    - The name of each package is a unique name (in the scope of all items) in the whole model

Class

A class can be understood as a blueprint for an object. It describes the type of objects for which DDI-CDI is a model for.

- Classes (and packages) are named and organized in a way that they can be easily moved to another location of the model
    - The name of each class must be a unique name (in the scope of all items) in the whole model
- Attributes are used
- Many classes have the attributes agency, id, and version. These items build a composite identifier aligned with the international registration data identifier (IRDI)[5]. Instantiated objects of these classes can be globally uniquely identified by these identifiers. This approach enables reuse of these objects on a granular level.

[4] XML Metadata Interchange (XMI) Specification, Version 2.5.1, https://www.omg.org/spec/XMI/2.5.1/PDF
[5] ISO/IEC 11179-6:2015, Information technology - Metadata registries (MDR) - Part 6: Registration, Annex A, Identifiers based on ISO/IEC 6523, http://standards.iso.org/ittf/PubliclyAvailableStandards/c060342_ISO_IEC_11179-6_2015.zip

Attribute

A class attribute is typed by a data type.

2. Relationships

Association

Only binary associations are used, i.e. two classes are related.

A notion of direction is used in DDI-CDI to be able to properly name associations so that they read as semantic triple (subject-predicate-object). Additionally, the direction is defined by a navigable association end at the "object" class. The association has unspecified navigability at the end of the "subject" class.

Associations are rendered in diagrams by connecting classes with a line. The navigable end is indicated by an open arrowhead (→) on one end of an association and owned by the class on the opposite end.

The definition of direction/navigation can be understood just as a recommendation, see also this citation from the official UML specification in the section on the semantics of associations.

> *„Navigability means that instances participating in links at runtime (instances of an Association) can be accessed efficiently from instances at the other ends of the Association. The precise mechanism by which such efficient access is achieved is implementation specific. If an end is not navigable, access from the other ends may or may not be possible, and if it is, it might not be efficient.*
>
> *NOTE. Tools operating on UML models are not prevented from navigating Associations from non-navigable ends."*[6]
>
> "*Specifying a direction of traversal does not necessarily mean that you can't ever get from objects at one end of an association to objects at the other end. Rather, navigation is a statement of efficiency of traversal.*"[7]

For specific uses, the direction of an association might not make sense. In this case, the navigation definition can be just ignored.

Association names should be unique in one package if possible. This is not always suitable in terms of achieving short names. Some UML tools comply in a strict sense to the UML rule that elements of related or the same type should have unique names within the enclosing package. For this purpose, a second representation of the model in Canonical XMI is provided which has unique association names per package.

[6] UML 2.5.1, Semantics of associations, page 200, https://www.omg.org/spec/UML/2.5.1/PDF

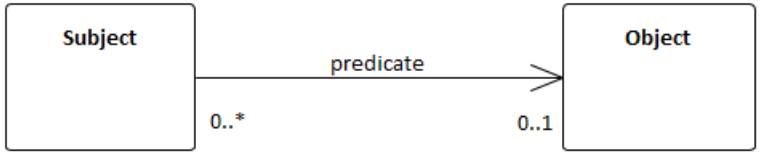
[7] The Unified Modeling Language User Guide, Booch, Grady; Rumbaugh, James; Jacobson, Ivar; Reading, Mass., 1999, page 144

Multiplicity

Multiplicity is formally defined as a lower and upper bound. Simply put: a multiplicity is made up of a lower and an upper cardinality. Cardinality is how many elements are in a set.

The default multiplicity of the "subject" class is 0..n. The multiplicity of the "object" class is usually 0..1 or 0..n. Zero for the lower cardinality allows flexibility in the process of producing metadata.

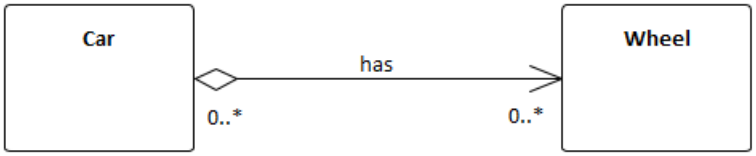
Association rendering:



Aggregation

Any semantics in aggregation are not seen which are not already covered by a common association with appropriate directed names, but it could provide a way of easily visualizing a whole/part relationship.

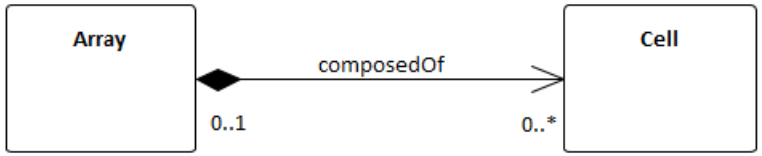
Aggregation rendering:



Composition

Composition is used for cases in which there is a strong lifecycle dependency, e.g. a cell in an array cannot exist without the array. However, it could provide a way of easily visualizing strong lifecycle dependency.

Composition rendering:

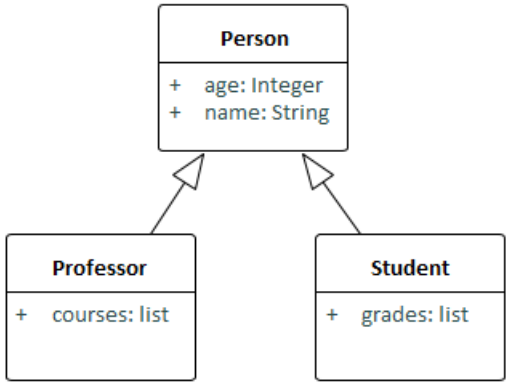


Generalization

A class can be an extension of another class (the general class). Attributes and associations of the general class are inherited. Only single inheritance is used, i.e. a class can only extend one other.

A data type can be an extension of another data type (the general data type). Attributes of the general data type are inherited. Only single inheritance is used, i.e. a data type can only extend one other. This applies also to primitive data types and enumerations.

Class generalization rendering:



3. Data Type Definition

Data Type

A data type can be a UML primitive data type, a structured data type, or an enumeration.

The UML primitive data types[8] are used: Boolean, Integer, Real, String, and UnlimitedNatural (the latter is only in XMI for the unlimited value of an upper cardinality).

[8] XMI representation of UML primitives, https://www.omg.org/spec/UML/20100901/PrimitiveTypes.xmi

A structured data type can have multiple attributes which are defined by other data types.

Some XML Schema primitive data types[9] are used. They are defined as UML primitive data types and defined semantically by the related XML Schema data type definition. Following XML Schema primitive data types are used: anyURI, date, and language.

All structured data types make finally use of the mentioned four UML primitive data types and three XML Schema primitive data types.

The UML primitive data types can be mapped to XML Schema data types in representations where they exist like in XML Schema and OWL/RDF.

Mapping of primitive data types[10]:

UML	XML Schema
PrimitiveTypes::Boolean	http://www.w3.org/2001/XMLSchema#boolean
PrimitiveTypes::Integer	http://www.w3.org/2001/XMLSchema#integer
PrimitiveTypes::Real	http://www.w3.org/2001/XMLSchema#double
PrimitiveTypes::String	http://www.w3.org/2001/XMLSchema#string
PrimitiveTypes::UnlimitedNatural	http://www.w3.org/2001/XMLSchema#string

Comment

Each item can have a definition which is expressed as UML comment.

4. Naming Convention

All items are named according to rules aligned with ISO/IEC 11179-5[11]. The names are possible compounds of multiple nouns and adjectives. Instead of a separator, the first letter of each name part within a single name is capitalized (sometimes called CamelCase).

Names of classes, data types, and enumeration literals start with an uppercase letter. Names of associations and attributes start with a lowercase letter.

[9] XML Schema primitive data types: https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes
[10] UML 2.5.1, XMI Serialization of the PrimitiveTypes model library, page 754, https://www.omg.org/spec/UML/2.5.1/PDF
[11] ISO/IEC 11179-5, Information technology - Metadata registries (MDR) — Part 5: Naming principles, http://standards.iso.org/ittf/PubliclyAvailableStandards/c060341_ISO_IEC_11179-5_2015.zip

F. Model Outline

The model outline of the Canonical XMI representation is listed below. These packages are intended for the users of the model. The main package “Classes” contains sub-packages with all classes for the three main areas foundational (Conceptual), data description (DataDescription), and process/provenance (Process) and supporting areas. The main package “DataTypes” contains sub-packages with all supporting data types.

- **DDICDILibrary**
  - Classes
    - Agents
    - Conceptual
    - DataDescription
      - Components
      - Dimensional
      - KeyValue
      - Long
      - Wide
    - FormatDescription
    - Identification
    - Miscellaneous
    - Process
    - Representations
  - DataTypes
    - Enumerations
    - StructuredDataTypes
    - XMLSchemaDataTypes

The model as Sparx Enterprise Architect file has two additional packages: “Diagrams” which holds all diagrams used in the specification document, and “DesignPatterns”, which supports mainly the specification developers.

- DDICDI
  - DDICDIModels
    - **DDICDILibrary (see above)**

- DesignPatterns
  - CollectionsPattern
  - DataDescriptionPattern
  - SignificationPattern
- Diagrams

VI. Appendixes

A. The DDI - CDI Upper Model Properties and their Sources by Property Group

Upper Model Property[12]	Property Group	Source(s)
about	About	schema.org CreativeWork,
abstract	About	schema.org CreativeWork, DC
accessMode	About	schema.org CreativeWork
accessModeSufficient	About	schema.org CreativeWork
accessRights	About	DC, DDI Codebook
accessibilityAPI	About	schema.org CreativeWork
accessibilityControl	About	schema.org CreativeWork
accessibilityFeature	About	schema.org CreativeWork
accessibilityHazard	About	schema.org CreativeWork
accessibilitySummary	About	schema.org CreativeWork

[12] Generally speaking, the names of the properties have been adopted from schema.org's CreativeWork. In the case that there is a single source other than CreativeWork, the name of the property comes from the single source. One exception are properties whose names begin with the prefix *sd*. *sd* is an abbreviation for structured data in CreativeWork. Structured data refers not to what a CreativeWork is about – the Dataset context -- but the CreativeWork object itself. Codebook also makes this distinction. Some Codebook only properties have been added to the *sd* series. Definitions for each of the properties are linked. The link goes to DCMI Metadata Terms when it is the origin or it has greater specificity. Otherwise the link goes to CreativeWork in schema.org. When a property is sourced both to CreativeWork and DC (Dublin Core), review the Dublin Core / DDI Core Upper Model Map in this Appendix (C) to determine the nuances of these relationships.

Upper Model Property[12]	Property Group	Source(s)
accountablePerson	Credits	schema.org CreativeWork, DC
accrualMethod	About	DC
accrualPeriodicity	About	DC. DDI Codebook
accrualPolicy	About	DC
aggregateRating	Buzz	schema.org CreativeWork
alternativeHeadline	About	schema.org CreativeWork, DC
associatedMedia	About	schema.org CreativeWork, DC
audience	About	schema.org CreativeWork, DC
audio	About	schema.org CreativeWork, DC
author	Credits	schema.org CreativeWork, DC
award	Buzz	schema.org CreativeWork
character	About	schema.org CreativeWork
citation	About	schema.org CreativeWork, DC
citationRequirement	About	DDI Codebook
comment	About	schema.org CreativeWork
commentCount	Buzz	schema.org CreativeWork
concept	About	DDI Codebook
conditionsOfAccess	About	schema.org CreativeWork, DC
confidentiality	About	DDI Codebook
contentLocation	About	schema.org CreativeWork, DC
contentRating	Buzz	schema.org CreativeWork
contentReferenceTime	About	schema.org CreativeWork, DC
contributor	Credits	schema.org CreativeWork, DC
copyrightHolder	Credits	schema.org CreativeWork, DC
copyrightYear	Credits	schema.org CreativeWork, DC
correction	About	schema.org CreativeWork, DC
creativeWorkStatus	About	schema.org CreativeWork
creator	Credits	schema.org CreativeWork, DC
dataQualityMetrics	About	DDI Codebook
date	About	schema.org CreativeWork, DC
dateAccepted	About	schema.org CreativeWork, DC

Upper Model Property[12]	Property Group	Source(s)
dateCopyrighted	About	schema.org CreativeWork, DC
dateCreated	Credits	schema.org CreativeWork, DC
dateModified	About	schema.org CreativeWork, DC
datePublished	About	schema.org CreativeWork, DC
dateSubmitted	About	schema.org CreativeWork, DC
disclaimer	About	DDI Codebook
discussionURL	Buzz	schema.org CreativeWork, DC
editor	Credits	schema.org CreativeWork, DC
educationalAlignment	Buzz	schema.org CreativeWork, DC
educationalUse	Buzz	schema.org CreativeWork
encoding	About	schema.org CreativeWork, DC
encodingFormat	About	schema.org CreativeWork, DC
estimateOfSamplingError	About	DDI Codebook
exampleOfWork	About	schema.org CreativeWork, DC
expires	About	schema.org CreativeWork, DC
funder	Credits	schema.org CreativeWork, DC
genre	About	schema.org CreativeWork, DC
hasPart	About	schema.org CreativeWork, DC
headline	About	schema.org CreativeWork, DC
identifier	About	schema.org CreativeWork, DC
inLanguage	About	schema.org CreativeWork, DC
interactionStatistic	Buzz	schema.org CreativeWork, DC
interactivityType	Buzz	schema.org CreativeWork, DC
isAccessibleForFree	Buzz	schema.org CreativeWork, DC
isBasedOn	About	schema.org CreativeWork, DC
isFamilyFriendly	Buzz	schema.org CreativeWork
isPartOf	About	schema.org CreativeWork, DC
keyWords	About	schema.org CreativeWork, DC
learningResourceType	About	schema.org CreativeWork
license	Buzz	schema.org CreativeWork, DC
locationCreated	About	schema.org CreativeWork, DC

Upper Model Property[12]	Property Group	Source(s)
mainEntity	About	schema.org CreativeWork, DC
material	About	schema.org CreativeWork, DC
materialExtent	About	schema.org CreativeWork, DC
mentions	About	schema.org CreativeWork, DC
notesOnDataCollection	About	DDI Codebook
offers	Buzz	schema.org CreativeWork
position	About	schema.org CreativeWork
producer	Credits	schema.org CreativeWork, DC
provider	Credits	schema.org CreativeWork, DC
publication	Buzz	schema.org CreativeWork, DC
publisher	Credits	schema.org CreativeWork, DC
publisherImprint	Credits	schema.org CreativeWork
publishingPrinciples	Credits	schema.org CreativeWork
recodingAndDerivation	About	DDI Codebook
recordedAt	About	schema.org CreativeWork, DC
releasedEvent	Buzz	schema.org CreativeWork, DC
researchQuestion	About	DDI - CDI ResearchProgram and ResearchComponent
researchHypothesis	About	DDI - CDI ResearchProgram and ResearchComponent
responseRate	About	DDI Codebook
review	Buzz	schema.org CreativeWork
schemaVersion	About	schema.org CreativeWork
sdDatePublished	About	schema.org CreativeWork
sdIdentifier	About	DDI Codebook
sdLicense	About	schema.org CreativeWork
sdPublisher	About	schema.org CreativeWork
sdVersion	About	DDI Codebook
sdVersionNotes	About	DDI Codebook
security	About	DDI Codebook
sourceOrganization	Credits	schema.org CreativeWork

Upper Model Property[12]	Property Group	Source(s)
spatial	About	schema.org CreativeWork, DC
spatialCoverage	About	schema.org CreativeWork
sponsor	Credits	schema.org CreativeWork, DC
studyType	About	DDI - CDI ResearchProgram and ResearchComponent
temporal	About	schema.org CreativeWork, DC
temporalCoverage	About	schema.org CreativeWork
text	About	schema.org CreativeWork
thumbnailURL	About	schema.org CreativeWork, DC
timeRequired	About	schema.org CreativeWork
translationOfWork	About	schema.org CreativeWork
translator	Credits	schema.org CreativeWork, DC
typicalAgeRange	Buzz	schema.org CreativeWork
unitOfAnalysis	About	DDI Codebook
universe	About	DDI Codebook
version	About	schema.org CreativeWork, DC
video	About	schema.org CreativeWork, DC
weighting	About	DDI Codebook
workExample	About	schema.org CreativeWork, DC
workTranslation	About	schema.org CreativeWork, DC

B. DDI Codebook / DDI - CDI Upper Model Map

Section	Subsection	Nesstar Publisher	Upper Model schema.org Dataset	Comparison
1. Metadata Production				
		Producer	sdPublisher	Codebook is broader
		Production Date	sdDatePublished	Codebook is broader
		DDI Document Version	sdVersion	Exact match
		Version Notes	sdVersionNotes	Exact match
		DDI Document ID Number	sdIdentifier	Exact match
2. Study Description				
	Identification			
	2.1.1.1	Title	Headline	Exact match
		Alternative Title	alternativeHeadline	Exact match
	2.1.1.4	Translated Title	translationOfWork	Close match
	2.1.1.5	ID Number	identifier	Exact match
	2.1.5.1	Study Type	typeOfResearch (ResearchProgram, ResearchComponent)	Codebook is broader
	2.1.5.2	Series Information	isPartOf, disambiguatingDescription	Codebook is broader
	Version		version	Exact match
	2.1.6.1	Production Date	datePublished	Close match
	2.1.6.3	Notes		not found in schema.org

Section	Subsection	Nesstar Publisher	Upper Model schema.org Dataset	Comparison
	Overview			
	2.3.2	Abstract	abstract	Exact match
	2.2.3.8	Kind of Data	genre	Codebook is narrower
	2.2.3.6	Unit of Analysis	unitOfAnalysis	Exact match
	Scope			
	2.2.4	Description of Scope	identifier extension (PropertyValue)	Codebook is narrower
	2.2.1.2	Topics Classifications	identifier extension (PropertyValue)	Codebook is narrower
	2.2.1.1	Keywords	keywords	Exact match
	Coverage			
	2.3.1.3	Country	spatialCoverage, contentLocation	Codebook is broader
	2.3.1.4	Geographical Coverage	spatialCoverage, contentLocation, locationCreated	Codebook is broader
	2.2.3.7	Universe	identifier extension (PropertyValue)	Codebook is narrower
	Producers and Sponsors			
	2.1.2.1	Investigators	author, creator, accountablePerson	Codebook is broader
	2.1.3.1	Other Producers	Contributor, editor	Codebook is broader
	2.1.3.6	Funding	Funder	Exact match
	2.1.2.2	Other Acknowledgements	Contributor, sponsor	Codebook is broader

Section	Subsection	Nesstar Publisher	Upper Model schema.org Dataset	Comparison
	2.4.1.2	Study Site	contentLocation	Codebook is narrower
	Sampling			
	2.3.1.4	Sampling Procedure	potentialAction	Codebook is narrower
	2.3.1.5	Deviations from Sample Design	correction	Codebook is narrower
	2.3.3.1	Response Rates	responseRate	Exact match
	2.3.1.12	Weighting	weighting	Exact match
	Data Collection			
	2.2.3.2	Dates of Collection	accrualMethodology, accrualPeriodicity, accrualPolicy	Codebook is broader
	2.3.1.6	Mode of Data Collection	potentialAction	Codebook is narrower
	2.3.1.3	Frequency of Data Collection	accrualMethodology, accrualPeriodicity, accrualPolicy	Codebook is broader
	2.3.1.9	Notes on Data Collection	notesOnDataCollection	Exact match
		Questionnaires	measurementTechnique	Codebook is narrower
	2.3.1.2	Data Collectors	contributor	Codebook is narrower
	Data Processing			
	2.3.1.13	Data Editing	correction	Codebook is narrower
	2.3.2	Other Processing	potentialAction	Codebook is narrower

Section	Subsection	Nesstar Publisher	Upper Model schema.org Dataset	Comparison
	Data Appraisal			
	2.3.3.2	Estimate of Sampling Error	estimateOfSamplingError	Exact match
	2.3.3.3	MEIRU Data Quality Metrics	dataQualityMetrics	Exact match
	Data Access			
	2.4.2.4	Access Authority	provider	Codebook is narrower
	2.4.2.1	Confidentiality	confidentiality	Exact match
	2.4.2.7	Access Conditions	conditionsOfAccess, license	Codebook is broader
	2.4.2.5	Citation Requirement	citationRequirement	Exact match
	Disclaimer and Copyright			
	2.4.2.8	Disclaimer	disclaimer	Exact match
	2.1.3.2	Copyright	copyrightYear, copyrightHolder	Codebook is broader
	2.2.12	Contacts	provider, accountablePerson	Codebook is narrower
	2.1.4.2	Contact Persons	provider, accountablePerson	Codebook is narrower
3. File Description				
	Data Files			
	3.1.2	Contents	description	captured under study description
	3.1.7	Producer	author, creator	captured under study description

Section	Subsection	Nesstar Publisher	Upper Model schema.org Dataset	Comparison
	3.1.10	Missing Data	notesOnDataCollection	Codebook is narrower
	3.2	Notes	notesOnDataCollection	Codebook is narrower
4. Variables Description				
			variableMeasured	Exact match
	4.2.15	Definition	description	Codebook is narrower
	4.2.12	Universe	identifier extension (PropertyValue)	Codebook is narrower
	4.2.21	Concepts	identifier extension (PropertyValue)	Codebook is narrower
	Question			
	4.2.8.1	Pre-Question Text	measurementTechnique	Codebook is narrower
	4.2.8.2	Literal Question	measurementTechnique	Codebook is narrower
	4.2.8.3	Post-Question Text	measurementTechnique	Codebook is narrower
	4.2.8.6	Interviewer Instructions	measurementTechnique	Codebook is narrower
	Derivation			
	4.2.19	Recoding and Derivation	recodingAndDerivation	Exact match
	Security			
		Security	conditionsOfAccess, confidentiality	Codebook is broader

Section	Subsection	Nesstar Publisher	Upper Model schema.org Dataset	Comparison
5. External Resources				
	Resource Description		citation	citation is its own ResearchComponent in a ResearchProgram with the same properties as a Codebook
		Type	about	Codebook is narrower
		Title	headline	Exact match
		Subtitle	alternativeHeadline	no equivalent in Schema.org
		Author(s)	author, creator	Codebook is broader
		Date	dateCreated	Codebook is broader
		Country	contentLocation	Codebook is broader
		Language	inLanguage	Exact match
		Format	encodingFormat	Codebook is broader
		ID Number	identifier	Exact match
	Contributor(s) and Rights			
		Contributor(s)	contributor	Exact match
		Publisher(s)	publisher	Exact match
		Rights	accessRights	Exact match

C. Another Codebook / Core Map

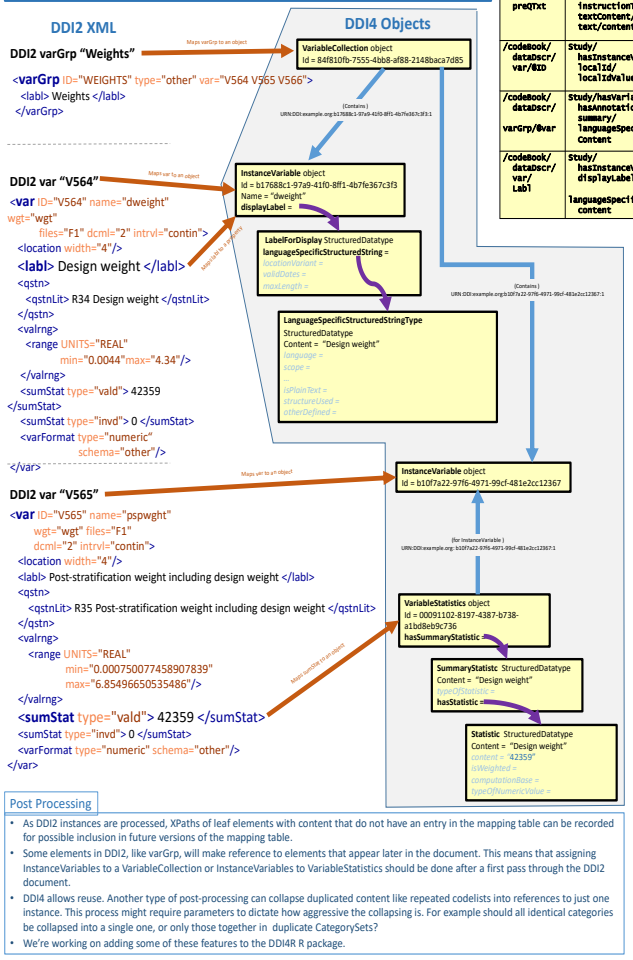
Please note that the name for DDI – CDI while under development was “DDI 4”.

The poster which appears here is difficult to read in this format, but is available online at Mapping DDI2 to DDI4.

Mapping DDI2 to DDI4, Larry Hoyle and Joachim Wackerow

- This poster describes a mapping of **leaf** DDI2 elements into corresponding DDI4 **properties**. The mapping can be described as a multi-column, **machine actionable, table**.
- DDI2 and the later versions differ in the underlying reliance on reusable objects. This makes mapping from DDI2 to DDI4 complicated. Software using this table can also collapse content that is repeated in DDI2 (like CodeLists) into single **reused DDI4 objects**.
- The table is derived from a spreadsheet listing DDI2 elements important to CESSDA. That table was further refined at the DDI4 Norway Sprint. The table below right explains the role of each column in the mapping table. Those columns contain the information needed to move information from one model to the other. Examples in the table below are color coded to corresponding cells in the table above.

The example below is taken from the European Social Survey file ESS1e06.6.xml



Post Processing

- As DDI2 instances are processed, XPaths of leaf elements with content that do not have an entry in the mapping table can be recorded for possible inclusion in future versions of the mapping table.
- Some elements in DDI2, like varGrp, will make reference to elements that appear later in the document. This means that assigning InstanceVariables to a VariableCollection or InstanceVariables to VariableStatistics should be done after a first pass through the DDI2 document.
- DDI4 allows reuse. Another type of post-processing can collapse duplicated content like repeated codelists into references to just one instance. This process might require parameters to dictate how aggressive the collapsing is. For example should all identical categories be collapsed into a single one, or only those together in duplicate CategorySets?
- We're working on adding some of these features to the DDI4R R package.

Sample rows from the mapping table. The full table currently lists 181 DDI25 XPaths.
This table can grow as new element types are found in DDI2 instances.

DDI25	DDI4PropertyName	IdentifiablesMapping	AbstractSubstitution	ParameterValues	IsIdRef	DDI4IdRefPath	DDI4IdRefAppend	Notes
/codeBook/ dataDscr/ var/ anlysunit	Study/ hasInstanceVariable/ usesUnitType/ Definition	/codeBook:Study, /codeBook/ dataDscr/ var:InstanceVariable, /codeBook/dataDscr/ var/anlysunit:UnitType						
/codeBook/ dataDscr/ var/ qstn/ preQTxt	Study/ hasInstanceVariable/ sourceCapture/ hasInstruction/ instructionText/ textContent/ text/content	/codeBook:Study, /codeBook/dataDscr/ var:InstanceVariable, /codeBook/dataDscr/var/ qstn:RepresentedQuestion, /codeBook/dataDscr/var/ qstn:Instruction	Capture: RepresentedQuestion, DynamicTextContent: LiteralText	RepresentedQuestion/ hasInstruction/ InstructionText/ textContent/ purpose/ languageSpecificStructuredString/ content='PreQuestionText'				
/codeBook/ dataDscr/ var/@ID	Study/ hasInstanceVariable/ localId/ localIdValue	/codeBook:Study, /codeBook/ dataDscr/ var:InstanceVariable			FALSE			Program will need to match DDI2 variable ID to DDI4 DdiUrn for references to the variable like varGrp
/codeBook/ dataDscr/ varGrp/@var	Study/hasVariableCollection/ hasAnnotation/ summary/ languageSpecificString/ Content	/codeBook:Study, /codeBook/ dataDscr/ varGrp:VariableCollection		Study/ hasVariableCollection/ hasAnnotation/ summary/ languageSpecificString/ scope='DDI2 variable IDs'	TRUE	contains/ member	TRUE, FALSE	requires matching DDI2 ID with DDI4 DdiUrn and inserting DDI4URN references into the VariableCollection
/codeBook/ dataDscr/ var/ Labl	Study/ hasInstanceVariable/ displayLabel/ languageSpecificStructuredString/ content	/codeBook:Study, /codeBook/dataDscr/ var:InstanceVariable						

Column Descriptions for the Table Above

Column	Function	Example	Details
DDI25	The unpredicated XPath of a DDI2 text or attribute node	/codeBook/dataDscr/var/anlysunit	Each piece of information to be imported from DDI2 should have a corresponding XPath listed.
DDI4PropertyName	The corresponding path to a leaf in DDI4	Study/hasInstanceVariable/ usesUnitType/definition	The first node in this path is a DDI4 class. The remaining nodes are properties in a chain down to a leaf value. The value of some properties are references to other objects, that object may need to be created. Other values are "structured datatypes".
IdentifiablesMapping	This maps a DDI2 sub-path to a DDI4 Identifiable class. An object of that class will need to be created for each unique instance of that DDI2 sub-path.	/codeBook:Study, /codeBook/dataDscr/var:InstanceVariable , /codeBook/dataDscr/var/anlysunit:UnitType	In the example to the left, for each unique var element in a DDI2 instance, the same DDI4 InstanceVariable needs to be used. A predicated XPath identifies a specific DDI4 object, e.g. /codeBook[1]/dataDscr[2]/var[7] indicates a sepcific InstanceVariable. In this example the 7th variable in the 2nd dataDscr element of the codebook always maps to the same reusable InstanceVariable.
AbstractSubstitution	Some references in the DDI4 model are to abstract classes. In these cases it is necessary to specify which extension of the abstract class should be used in the mapping.	Capture:RepresentedQuestion, DynamicTextContent:LiteralText	In this example a sourceCapture associates with the abstract class Capture. The mapping will use the RepresentedQuestion extension of the Capture.
ParameterValues	Some values in DDI4 can use additional explanatory metadata. This column lists the path and the value for that information.	RepresentedQuestion/hasInstruction/ InstructionText/textContent/purpose/ languageSpecificStructuredString/ content='PreQuestionText'	The LiteralText above is further described as "PreQuestionText"
IsIdRef	Is this value a reference to an ID in the DDI2 XML (an xs:IDREF)?, if so this will ultimately need to be transformed into a proper reference in DDI4 through a DDI URN.	TRUE	An example is the @var attribute of the DDI2 varGrp. This will need to be implemented as a reference to an InstanceVariable in a VariableCollection in DDI4.
DDI4IdRefPath	This is the sub-path within the last DDI4 identifiable object for the DDI URN of the referenced object.	contains/member	In the case of a varGrp, the VariableCollection has a contains/member value that is the URN of the DDI4 InstanceVariable created to match the DDI2 var referenced by the @var IDREF.
DDI4IdRefAppend	This describes whether to append or replace values in the DDI4 path.	TRUE, FALSE	In the case above, there may be more than one member property under contains, but there can only be one value for member.
Notes	Used to describe any notes for the mapping	requires matching DDI2 ID with DDI4 DdiUrn and inserting DDI4URN references into the VariableCollection	In this example, it describes what is needed in the matching process of DDI2 IDs and DDI4 URNS.

DDI25, DDI4PropertyName, and IdentifiablesMapping columns from DDI2 anlysUnit
AbstractSubstitution and ParameterValues from DDI2 preQTxt
IsIdRef, DDI4IdRefPath, DDI4IdRefAppend, and Notes from DDI2 varGrp/@var

D. DDI - CDI Upper Model / Dublin Core Map

Upper Model	Dublin Core	Notes
about	subject	dc is skos:broader
abstract	abstract	dc is skos:exactMatch
accessMode	format, type, medium	DC has a controlled vocabulary for type which includes the set of media types specified by the Internet Assigned Numbers Authority dc is skos:closeMatch
accessModeSufficient		
accessibilityAPI		
accessibilityControl		
accessibilityFeature		
accessibilityHazard		
accessbilitySummary		
accessRights	rights	dc is skos:exactMatch
accountablePerson	publisher	dc is skos:closeMatch
accrualMethod	accrualMethod	dc is skos:exactMatch
accrualPeriodicity	accrualPeriodicity	dc is skos:exactMatch
accrualPolicy	accrualPolicy	dc is skos:exactMatch
aggregateRating		
alternativeHeadline	alternative	dc is skos:exactMatch
associatedMedia	type	dc is skos:closeMatch
audience	audience	dc is skos:exactMatch
audio	type	dc is skos:broader
author	author	dc is skos:exactMatch
award		
character		
citation	bibliographicCitation	dc is skos:closeMatch
citationRequirement		
comment		
commentCount		
concept		

Upper Model	Dublin Core	Notes
conditionsOfAccess	available, accessRights, isRequiredBy, requires	dc is skos:narrower
confidentiality		
contentLocation	spatial	dc is skos:broader
contentRating		
contentReferenceTime	temporal, valid	dc is skos:narrower
contributor	contributor	dc is skos:exactMatch
copyrightHolder	rightsHolder	dc is skos:exactMatch
copyrightYear	dateCopyrighted	dc is skos:exactMatch
correction	provenance	dc is skos:broader
creativeWorkStatus	valid	dc is skos:narrower
creator	creator	dc is skos:exactMatch
dataQualityMetrics	provenance	dc is skos:broader
date	date	dc is skos:exactMatch
dateAccepted	date	dc is skos:broader
dateCreated	date, dateCopyrighted, dateSubmitted	dc is skos:narrower
dateModified	modified	dc is skos:exactMatch
datePublished	issued	dc is skos:closeMatch
dateSubmitted	dateSubmitted	dc is skos:exactMatch
disclaimer		
discussionURL	hasPart	dc is skos:broader
editor	contributor	dc is skos:broader
educationalAlignment	educationLevel	dc is skos:broader
educationalUse		
encoding	medium, type	dc is skos:closeMatch
encodingFormat	medium, type	dc is skos:broader
estimateOfSamplingError		
exampleOfWork	references, isReferencedBy, provenance	dc is skos:narrower

Upper Model	Dublin Core	Notes
expires	valid	dc is skos:narrower
funder	contributor	dc is skos:broader
genre	type	dc is skos:closeMatch
hasPart	hasPart	dc is skos:exactMatch
headline	title	dc is skos:closeMatch
inLanguage	language	dc is skos:closeMatch
interactionStatistic	hasPart	dc is skos:broader
interactivityType	instructionalMethod	dc is skos:broader
isAccessibleForFree	license	dc is skos:broader
isBasedOn	provenance	dc is skos:closeMatch
isFamilyFriendly		
isPartOf	isPartOf	dc is skos:exactMatch
keyWords	subject	dc is skos:closeMatch
learningResourceType		
license	license	dc is skos:exactMatch
locationCreated	provenance	dc is skos:broader
mainEntity	subject	dc is skos:broader
material	medium	dc is skos:closeMatch
materialExtent	extent	dc is skos:broader
mentions	references	dc is skos:closeMatch
notesOnDataCollection	provenance	dc is skos:broader
offers		
position		
producer	publisher	dc is skos:closeMatch
provider	contributor	dc is skos:broader
publication	provenance	dc is skos:broader
publisher	publisher	dc is skos:exactMatch
publisherImprint		
publishingPrinciples		
recodingAndDerivation		
recordedAt	provenance	dc is skos:broader

Upper Model	Dublin Core	Notes
releasedEvent	provenance	dc is skos:broader
researchQuestion		
researchHypothesis		
responseRate		
review		
schemaVersion		
sdDatePublished		
sdIdentifier		
sdLicense		
sdPublisher		
sdVersion		
sdVersionNotes		
security		
sourceOrganization		
spatial	spatial	dc is skos:exactMatch
spatialCoverage	spatial	dc is skos:broader
sponsor	contributor	dc is skos:broader
studyType	type	dc is skos:broader
temporal	temporal	dc is skos:exactMatch
temporalCoverage	temporal	dc is skos:broader
text		
thumbnailURL	hasPart	dc is skos:broader
timeRequired		
translationOfWork		
translator	contributor	dc is skos:broader
typicalAgeRange		
unitOfAnalysis	type	dc is skos:broader
universe	type	dc is skos:broader
version	hasVersion, isVersionOf	dc is skos:closeMatch
video	hasPart	dc is skos:broader
weighting		

Upper Model	Dublin Core	Notes
workExample	isFormatOf	dc is skos:broader
workTranslation	isFormatOf	dc is skos:broader