DDI - Cross Domain Integration:

Specification Overview

DDI - Cross Domain Integration: Specification Overview

Version 1.0 Release

Web page: https://ddialliance.org/Specification/DDI-CDI/

Git repository: https://github.com/ddi-cdi/ddi-cdi

DDI Alliance 2025

Ann Arbor, Michigan, USA

Copyright & License

https://ddialliance.org/

License

Attribution

Notices

Credits

Members of the Cross Domain Integration (CDI) Working Group shepherded the standard into its final form and produced the final documentation. Listed in alphabetical order they are:

Arofan Gregory (chair)
Dan Gillman
Flavio Rizzolo
Hilde Orten
Jay Greenfield
Joachim Wackerow
Larry Hoyle
Oliver Hopt
Wendy Lee Thomas (Technical Committee contact)

Over 100 people have contributed to the development of the Data Documentation Initiative Cross Domain Integration (DDI-CDI) specification. A more complete description of their contribution to the work can be found at https://github.com/ddi-cdi/ddi-cdi/blob/main/CREDITS.md.

Contents

I. Overview

The DDI - Cross Domain Integration (DDI - CDI) specification provides a model for working with a wide variety of research data across many scientific and policy domains. It provides a level of detail which supports machine-actionable processing of data, both within and between systems, and is designed to be easily aligned with other standards.

It focuses on the key elements of the data management challenges facing research today: an exact understanding of data in a wide variety of formats, coming from many different sources. Two elements are critical for dealing with these challenges: a flexible means of describing data that can reveal the connections between the same data existing in different formats, and a means of describing the provenance of the data at a detailed (but comprehensible) level: the processes which produced it must be transparent.

DDI - CDI covers these areas in a fashion intended to make it optimally useful to modern systems, which often employ a variety of models, and comply with a range of related specifications for both functions related to data description and process/provenance. The model is designed to be easy to fit into such systems, by aligning with relevant external standards, and to be align-able with them into the future.

II. Purpose

The DDI - CDI specification describes a model and supporting elements for implementing it in the areas of data description and process/provenance. It is not intended to supplant existing specifications for these purposes, but to fill in the information which such specifications often do not capture. For data, this is the description of a single piece of information – a datum – which can be used to play different roles in different data structures and formats. For provenance and process, this is the packaging of specific machine-level processes, which may be described in many different ways, into a structure which relates them to the business processes described at a level understandable to human users.

In order to serve this purpose, the DDI - CDI specification uses a Unified Modeling Language (UML) formalization so that it can be mapped against other models within systems more easily. Several different syntax expressions of the model are made available to support implementation. (For a description of how UML is used in the model, see Section VIII. A.)

Several important features of the specification can be highlighted, to show how it serves this purpose:

- Domain-independence
- Datum-Oriented Data Description
- Provenance and Process Description
- Foundational Metadata
- Interoperability, Sustainability, and Alignment with Other Standards

Each of these will be addressed in more detail, and an outline of the specification documents is presented.

III. Key Features of the Specification

A. Domain Independence

DDI - CDI is designed to be used with research data from any domain. In order to do this, it is fundamentally based on the structure and other generic aspects of the things it describes. It does not attempt to be a domain model of semantics, nor a model specific to the life-cycle of a particular domain of science or research. (Historically, DDI has focused on the Social, Behavioral, and Economic [SBE] sciences and some types of health research – to see how DDI - CDI relates to other DDI specifications, see Sections IV. and V., below.)

DDI - CDI is intended to be complimentary to (and used in combination with) other standards and models which focus more on domain-specific aspects (such as semantics and life-cycle models). Such generic elements such as classifications and variables are given a detailed formal treatment but are agnostic as to the domain. It is left to the user to employ whatever domain semantics are demanded by the data with which they are working.

This feature of the specification makes it well-suited to combining data coming from more than one domain or system, to allow a description that supports systems which perform data integration, harmonization, and similar functions. Cross-domain data sharing is becoming increasingly common, and DDI - CDI is intended to provide support for this type of application.

B. Datum-Oriented Data Description

DDI - CDI embraces a form of data description which is based on its atomic components: individual datums. Any given datum can play different roles in different formatting of the same data set, depending on how it is processed and transformed. In order to retain the continuity of a given datum across different formats and throughout a series of processes, DDI - CDI allows it to be described playing different roles in different structures.

DDI - CDI provides four basic types of structural description for data sets: wide data, long data, dimensional data, and key-value data. These four types (and their sub-types) provide coverage for many common data formats today. While not comprehensive, they cover the majority of cases that the developers of this specification have seen. These include many of the newer forms of data such as streaming data, “big” data, registers, and instrument data. The underlying approach is one which could – and may be – expanded in future. By assigning appropriate roles to the variables which contain the datums across each of these different formats, however, it is possible to understand how data passes from one form to another.

C. Provenance and Process Description

If we are to fully understand data, we also need to know how it has been processed and transformed. Given our ability to describe how a different datum can be used in different data sets, it becomes desirable to understand also how those data sets relate to one another in terms of the processes which use them. This can be understood as an important aspect of data provenance.

There are many different ways of describing process and provenance. Popular models include the Business Process Modelling and Notation (BPMN) standard and the PROV Ontology (from W3C). There

are a multitude of syntaxes for driving data transformation, cleaning, and analysis in packages such as R, SAS, Stata, MATLab, SPSS, Python, and so on. There are also some emerging standard models for specifically describing such processes (e.g., Structured Data Transformation Language [SDTL], Validation and Transformation Language [VTL]).

DDI - CDI attempts to do something which complements the use of such models, by connecting specific processes interpretable by machines at the lowest level (described in a package-specific syntax or language) with the higher-level flows which combine these into human-readable documentation of business processes. Both traditional linear (deterministic sequencing) processing and the newer declarative (non-deterministic sequencing) processing approaches are supported.

D. Foundational Metadata

In order to formally describe data at a detailed level, there are many component elements which themselves must be modelled. Concepts used for statistical data but also widely applicable – including categories and variables – are a core part of this, but the range is broad. These components are included in DDI - CDI as "foundational metadata."

Terminology for such constructs varies widely across domains. DDI - CDI has attempted to provide common terms for these components, and to adopt common models from other standards where it seemed useful.

One area which deserves particular attention is the "variable cascade" – a model for how data are described at different points in their creation, processing, and use, which is designed to optimize reuse. While many different models have a "variable" of some form, the one presented in DDI - CDI reflects the experience of working with this important construct in many of the specifications and standards which have preceded it. It is a nuanced view of how variables relate and are understood across different systems, and – although not simple – it is a powerful model which helps solve some of the commonly encountered problems in data description and management.

E. Interoperability, Sustainability, and Alignment with Other Standards

DDI - CDI is fundamentally a model which is intended to be implemented across a wide variety of technology platforms, and in combination with many other standards, models, and specifications. To support this use, it is formalized using a limited subset of the Unified Modelling Language (UML) class diagram part (UCMIS - UML Class Model Interoperable Subset). The model is provided in the form of Canonical XMI (restricted XML Metadata Interchange) – an interchange format for UML models supporting the import into many different modelling and development tools. Further, syntax representations are provided in XML Schema and RDF/OWL (with serializations in Turtle and JSON-LD), so that direct implementations of the model are possible if needed.

The platform-independence of the model makes it more easily applicable across a broad range of applications and helps ensure that it will be sustainable even as the technology landscape evolves.

DDI - CDI builds on many other standard models and is aligned with them where appropriate. This is shown in the model itself, where formalizations from other models and specifications are refined,

extended, or directly used. The specification includes a description of what these other standards and models are, and how they are used in DDI - CDI.

IV. DDI - CDI and the Suite of DDI Specifications

DDI - CDI is a different type of specification than its predecessors. It is not a continuation of or replacement for earlier DDI specifications such as DDI Codebook or DDI Lifecycle. It is intended to be complementary to these specifications for those applications – which were mostly in the SBE sciences – where DDI is used.

DDI - CDI builds on the work of many years in the DDI 4 "Moving Forward" project (not to be confused with the DDI Lifecycle version 4.0) and brings some of the strengths of that effort to light. In this, it shares many features with later versions of DDI Lifecycle, which has also incorporated some of that work. Notably, the "variable cascade" comes from earlier DDI 4 "Moving Forward" models (and its antecedents, like the Generic Statistical Information Model [GSIM]), as does the overall approach to describing non-rectangular data.

The DDI - CDI Model is the first specification produced by the DDI Alliance which uses a conceptual model expressed in UML as its basis. It is intended to describe many of the types of data which earlier DDI specifications describe. Due to the way in which data today is increasingly used across traditional domain boundaries, however, DDI - CDI is also (and of necessity) capable of describing data from many related domains.

The purpose of the specification differs somewhat from the earlier DDI Codebook and DDI Lifecycle specifications. Due to changes in the way in which information technology is applied to research and statistics, some new features are emphasized. Notably, the diversity of data types analyzed in a given project has increased, and the range of sources for that data has grown, with corresponding changes in the technology used to manage it.

The functional goal of the specification is also different: where DDI Codebook was an XML representation of a data dictionary, and DDI Lifecycle a more complex model designed to support metadata from data conception and capture through publication and reuse, DDI - CDI is an attempt to describe data and its provenance independent of these contexts.

Both DDI Codebook and DDI Lifecycle combine the description of structure (e.g., a table of records) and the description of meaning. In both, the primary structural form is a table or a cube. A variable and a column (for tabular data) are basically synonymous. DDI - CDI disentangles structural description from description of meaning. This allows description of structural forms like long (also called "tall") tables or key-value stores.

The growing demand for data from different sources, and from external domains, requires that some different types of data be described. The provenance of this data – that is, the processes by which it has been assembled for use – are of increasing importance in understanding what it is and how it can be used. While traditional SBE data was often collected using questionnaires, alternate sources of data such as registers (such as administrative records for marriages, employment, etc.) and sensors are becoming

increasingly common and have in some cases always been typical. New sources of data from social media and other sites are also increasingly used.

The DDI - CDI model applies the important features of the pioneering (but unreleased) DDI 4 "Moving Forward" work to these functions: describing various types of data in a way which makes them subject to integration and transformation into usable forms, and providing the information needed to understand their origins and provenance.

Because the way in which such a model can be implemented is more variable than it is for traditional SBE data management systems, the emphasis in DDI - CDI is on a model, formalized in UML, and made available using the Canonical XMI format. This expression of the model is the canonical form. It supports the exchange of UML models between various tools, including both modelling and development environments. While the XML syntax representation for metadata instances is still supported, it is no longer the canonical (official) format for the specification, as it is for DDI Codebook and DDI Lifecycle.

DDI - CDI is aligned with earlier DDI specifications, most notably DDI Lifecycle, as we anticipate that DDI - CDI might be used as an integration model for systems based on these earlier specifications. The intention is that DDI - CDI be a tool which can supplement systems using earlier versions of DDI, enabling them to better handle new types of data.

V. The Context of DDI - CDI

The purpose of this section is to show how DDI – Cross Domain Integration (DDI - CDI) fits into the overall context of research and the metadata which is used to describe research activities and resources. It is expected that metadata not described in DDI - CDI will exist in most (if not all) implementations, but that such metadata will often be specific to the domain, infrastructure, or system in which the implementation is built. DDI - CDI has several basic techniques for tying the metadata it describes into the larger set of information used. These include references to non-DDI - CDI metadata descriptions such as catalog descriptions (presumably structured in a fashion meaningful to the system according to a standard), a mechanism for making more granular "external" references to concepts, vocabularies, and other metadata resources (again, presumably structured in a standard fashion), and the availability of granular identifying, cataloguing, and citational information on a large number of DDI - CDI objects, allowing them to be referenced in turn.

The illustration below shows an example of how DDI - CDI metadata might fit into a larger information set employed by a domain-specific implementation. Only the areas labelled "DDI - CDI" are covered by the present model – all other boxes are examples of the type of non-DDI - CDI metadata standards with which it is intended to be integrated. (They are only illustrative examples, and do not convey any preference for nor comprehensive listing of such standards.)

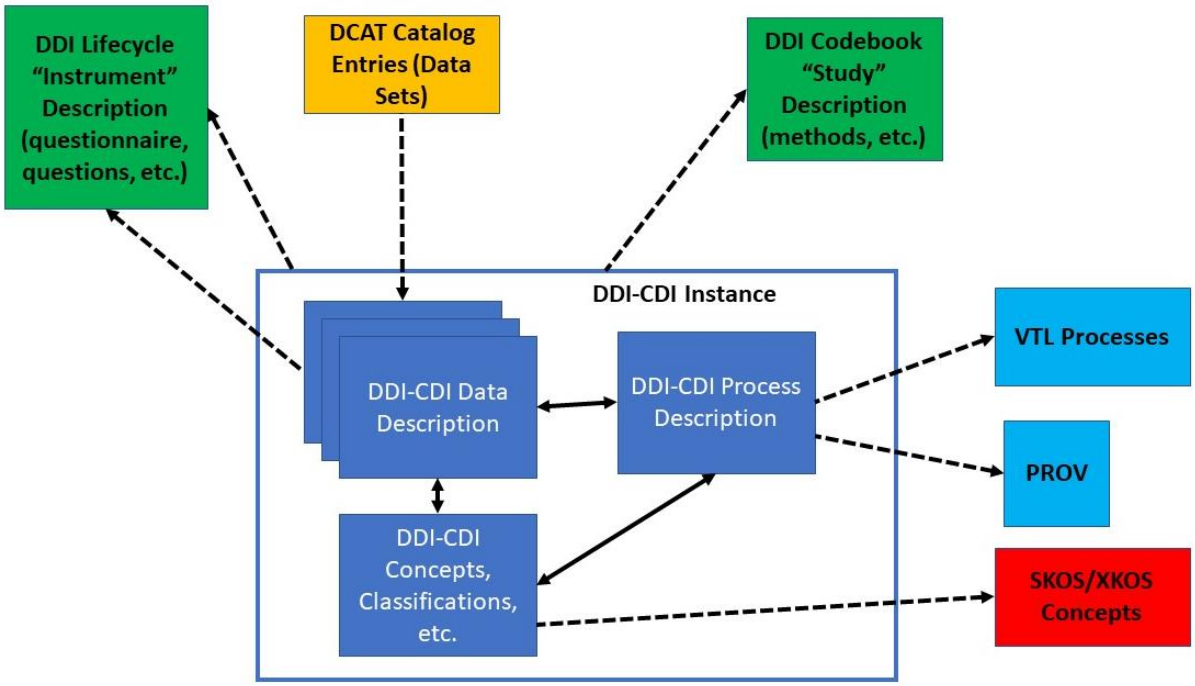
The dark blue box contains several different types of metadata modelled according to DDI - CDI – in this example, we have three different data sets which represent different manifestations of the data as it passes through a process of some kind.

The orange box represents Data Catalog Vocabulary (DCAT) catalog entries describing these three data sets, and making reference to the variable descriptions provided by the more granular DDI - CDI metadata.

The green boxes represent other DDI metadata according to different specifications. The DDI Codebook instance contains a description of the overall research effort which produced the data, giving information on the methods used in collecting and producing the data. The DDI Lifecycle instance contains a granular description of the questionnaire which was used during data collection. The light blue boxes represent additional information described according to PROV (for a step-by-step description of the process) and VTL (for descriptions of specific functions which were applied to the data) as it was transformed from one data set to another at specific points in the processing. These are referenced from the overall process description provided by the DDI - CDI process description. (They might also reference each other, although that is not shown here.)

The red box represents a set of Simple Knowledge Organization System (SKOS) concepts used as DDI - CDI concepts, code lists, and categories in the representation of variables and their definition.

Dotted lines represent references made external to the DDI - CDI instance metadata, which will vary depending on what syntax representation is used, but will typically consist of URLs.



To facilitate this type of referencing and linking, three basic features of DDI - CDI will be introduced, so that the use of these features will be clear for understanding the more detailed description of the model, below. These are CDI Content, (external) References, and Catalog Details.

A. Sets of DDI - CDI Metadata: The Wrapper Element

There is a notion of a set of CDI metadata, collected for a specific purpose, such as describing a collection of data and its related metadata, providing metadata for reuse, describing a process, etc. This type of package will be implemented in different ways according to the syntax employed. In the XML Schema syntax representation, it is termed “Wrapper”. It is an XML element which describes a single object acting as the container for any metadata appearing within the DDI - CDI model organized for a particular purpose – the metadata of interest as determined by the user.

The notion of “internal” versus “external” in DDI - CDI is always made in reference to this idea of a discrete set of metadata, pulled together for a purpose. Because this type of a packaging mechanism often has corresponding constructs in other standards and models, there is an external reference (see Section V. C.) to some other metadata which is assembled at the same level as the Wrapper for the purposes of a particular application, but which is not modelled according to DDI - CDI.

For example, in the DDI Codebook metadata standard there is a notion of a “Study”, which brings together a package of metadata concerning a set of data files and related metadata from a data production or use activity, at a specific time, to support a distinct research effort. An example would be the collection of files for one wave of a large social survey, or some census within a country.

While the data and processes of such a “Study” could be described in DDI - CDI (and indeed could largely be generated programmatically from it in some cases), some of the other metadata contained in the DDI Codebook instance for those data files could not. The idea here is that a metadata set can reference a complimentary set (or sets) of metadata which are needed to support application functions, exchange, etc. alongside the DDI - CDI metadata.

In this case, the connection to the source or supplemental metadata is not lost when DDI - CDI is employed to describe specific parts of the overall information set. Different syntax representations of the model will have different mechanisms for making these references. In the XML representation, the packaging element called “Wrapper” is used.

In the XML representation of the DDI - CDI model, the attribute “supportingInformation” is used to make an external reference to one or more complementary metadata sets. For this purpose, it would be used to make an “external” reference using the standard CDI structure (see Section V. C.).

Note that in the XML the Wrapper element may also be annotated with CatalogDetails (see Section V. D.).

For bindings in RDF other properties could be used, according to the RDF vocabulary employed to package the metadata. (For example, a DCAT *Dataset* might have a *Relationship* property, qualified appropriately with a *hadRole* property, to a *CatalogRecord*. This will be dependent on the syntax representation of the DDI-CDI model.)

B. Syntax Representation

DDI - CDI is intended to be implemented with a variety of syntaxes. While some reference syntax implementations will be provided (the XML and RDF syntaxes are part of this release) it is understood that many others will also be generated. This section briefly describes the approach to syntax

implementation taken by DDI - CDI. This can have a significant effect on how DDI - CDI is combined with other standards as described above.

The DDI - CDI UML model is the central component in the specification: all syntax representations are assumed to align with specific elements of the model. Different communities of use are expected to identify which parts of the model they are using, and to further indicate how those parts of the model will be expressed in their target syntax. The reference syntaxes serve a dual purpose: both as direct implementations of the model, and as reference points for communities of users to describe their own implementations. The selection of constructs from the model, and their syntactic expressions should be documented by user communities, including whatever controlled vocabularies are assumed in processing their DDI - CDI instances.

Because the set of syntaxes is not specified by DDI - CDI, it has been necessary to provide a more complete set of classes, relationships, and attributes than may be required for some syntaxes. A primary example of this can be seen in RDF: by its nature, the RDF families of technology assume that linking on the Web is the basis for both identification of resources and their retrieval, and that many resources will be made available in a decentralized fashion. While these assumptions are good ones for the applications RDF is intended to support, they may or may not align with the needs of specific user communities in their production, management, and sharing of data and metadata.

From an RDF perspective, DDI - CDI is probably too baroque when it comes to matters of identification and linking, as a result of the broader requirements it is intended to fulfill. When implementing DDI - CDI in RDF syntaxes, it makes sense to “flatten” some of the more baroque constructs so that they fit more naturally.

Implementers should be aware of this aspect of the DDI - CDI design, but should also be aware that users of their resources may not share the same assumptions about syntax implementation. In cases where the model has been “flattened” to align with a particular syntax, care should be taken to do this in a consistent way that can be “un-flattened” when transformation into another syntax is wanted.

As an example, in an implementation where all identifiers are referenceable URLs, it may be desirable to flatten the Reference type in DDI - CDI into a single one of its fields (the URI) and to only permit resolvable URLs to populate attributes using that type. So long as this is done consistently, a community whose technology does not align with such an approach should be able to programmatically restore the “flattened” missing structure if needed. In most cases, taking a consistent approach to the model for such syntax representations is sufficient to guarantee the intended level of interoperability across communities of use.

C. External References and Identification

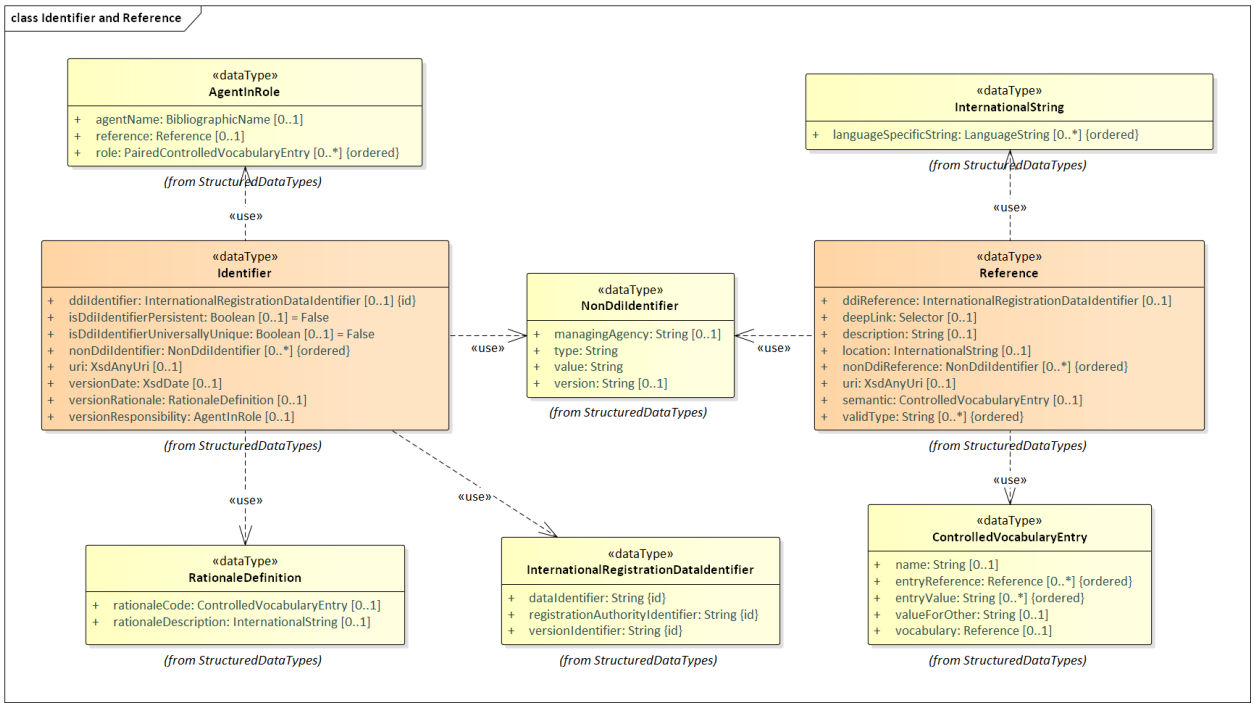


Figure 1: Identifier and Reference Classes

DDI - CDI employs a generally consistent mechanism for making references to information objects outside the scope of an instance of DDI-CDI metadata. Such references are termed “external” references.

It should be noted that this mechanism may also be used for some types of “internal” references, notably in cases where the target for such a reference is very broad, or to avoid some types of cross-package dependencies in the UML model.

The Reference type allows for several types of links to other classes. It will generally have a semantic associated with it by the attribute using it, but this can be further qualified as needed through the use of a controlled vocabulary. There is also a field for providing a description of the reference in plain text.

The mechanisms for linking include several fields. Among these are fields for providing a DDI-conformant identifier, a non-DDI-conformant identifier which is not expressed as a URI, or a URI of some type (including URLs). The attribute Location is provided to allow for linking in other cases (such as links to non-digital or local resources). Deep linking – the addressing of internal segments of the referenced, identified entity – is also supported, in line with the W3C specification for this function.

In the example pictured in the section introduction above, the links between and among DDI - CDI objects might or might not use this mechanism, but the references to SKOS Concepts, SDTL scripts, and

the DDI Codebook “Study” metadata (etc.) would be external references using the mechanisms supplied here.

It is incumbent on implementers to provide guidance when defining the subset of DDI - CDI constructs to be used, including their syntax implementation (if different from those provided with the specification). References such as those described here could easily be “flattened” into simple URLs for implementation in RDF, for example. This is an intended feature of the model’s design: the richness of the model need not be “completely” expressed in cases where the needed function is better handled using an equivalent syntactic mechanism.

In the example above, it might be the case that the only allowed content within a community implementation for an attribute of the Reference type pointing to a Concept would be a URL to a target conforming to the SKOS vocabulary.

D. Catalog Details

DDI - CDI metadata is not only a platform for referencing other metadata, but may itself be referenced. In the example above, a catalog entry expressed in DCAT would be making a reference to a DDI - CDI information set for the purposes of exposing variable descriptions and other details of the catalogued data sets.

In order to be referenced from across a wide variety of potential sources, DDI - CDI provides the CatalogDetails type. This supports the provision of public (non-CDI) identifiers and other citational information, access information, summary and provenance information, links to related resources, and the ability to assign a type to the object from a controlled vocabulary with values meaningful to potential processors.

While it may be typical to simply reference metadata objects with their assigned CDI identifiers, the ability to capture other identifiers may be very useful. In those cases where immediate access to the described object is not desirable (due to constraints of size or access) the summary information may also provide an easy way of capturing brief descriptions of the metadata object for presentation to human users. Provenance, access, and citation information is also important in its own right.

For these reasons, CatalogDetails is made available as an attribute on many of the publishable and reusable metadata objects in CDI, including Data Sets and Data Stores, Classifications, Concepts, Individuals and Organizations, and elsewhere.

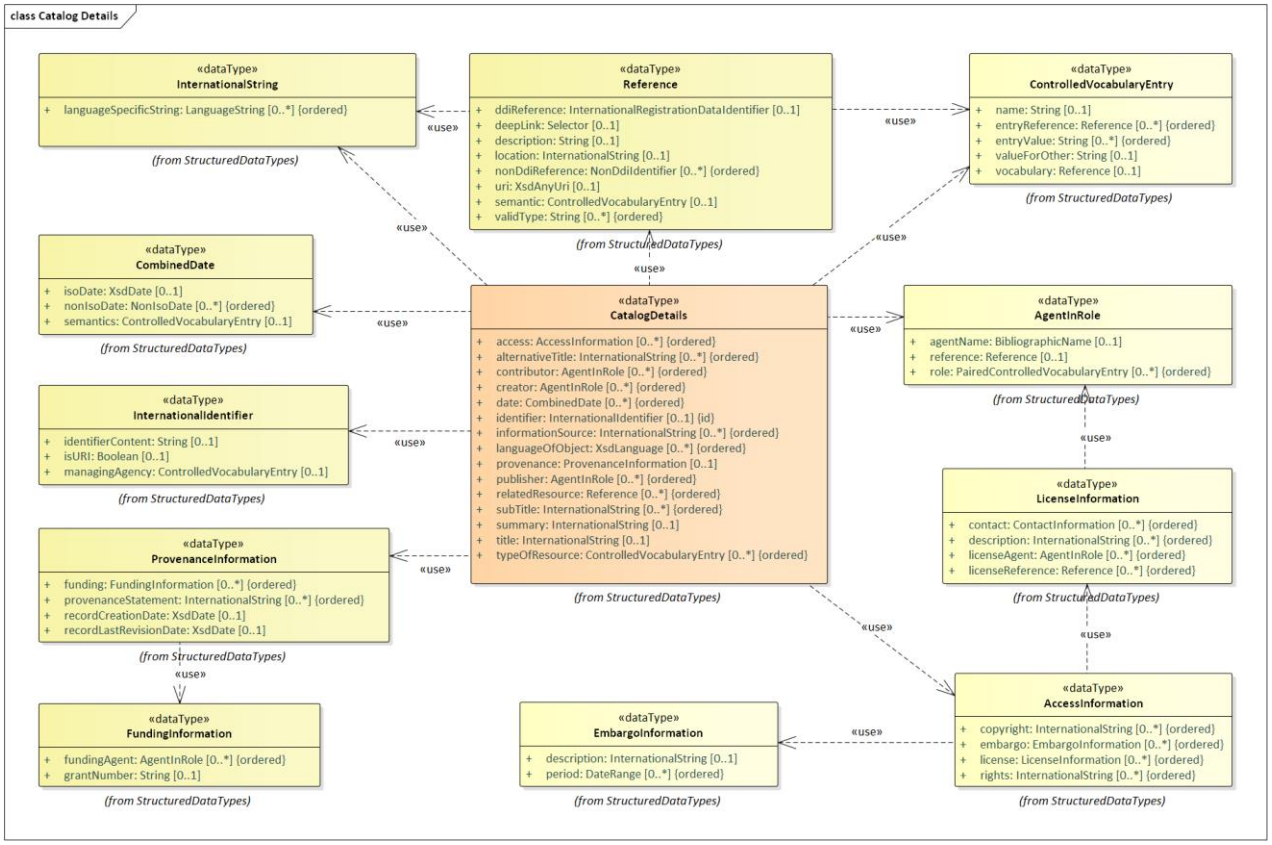


Figure 2: Catalog Details

VI. DDI - CDI Content: Summary and Business Perspective

The DDI - CDI model offers a range of different metadata classes by which different aspects of data can be described. Ultimately, DDI - CDI aims to support an understanding of any given data by providing a complete set of information regarding its structure, provenance, meaning, and access through the models it provides and by connecting to other standard models.

That said, the specific portion of the DDI - CDI model which will be used in any given application or community depends on what information is needed to support exchange, reuse, and integration, and what elements of that information set already has a standard expression which can be readily consumed by all relevant users. DDI - CDI is intended to be useful in filling these gaps, and pulling together otherwise unconnected information regarding data and the processes from which it comes.

This section of the document provides a high-level overview of how DDI - CDI describes the business-level processes around data, and then looks at how the data itself is described. The following sections will cover the foundational constructs used, and provide greater depth of detail on both process and data description.

A. The Process Model

DDI - CDI process description starts with the environment in which the process occurs, represented by the Production Environment class.

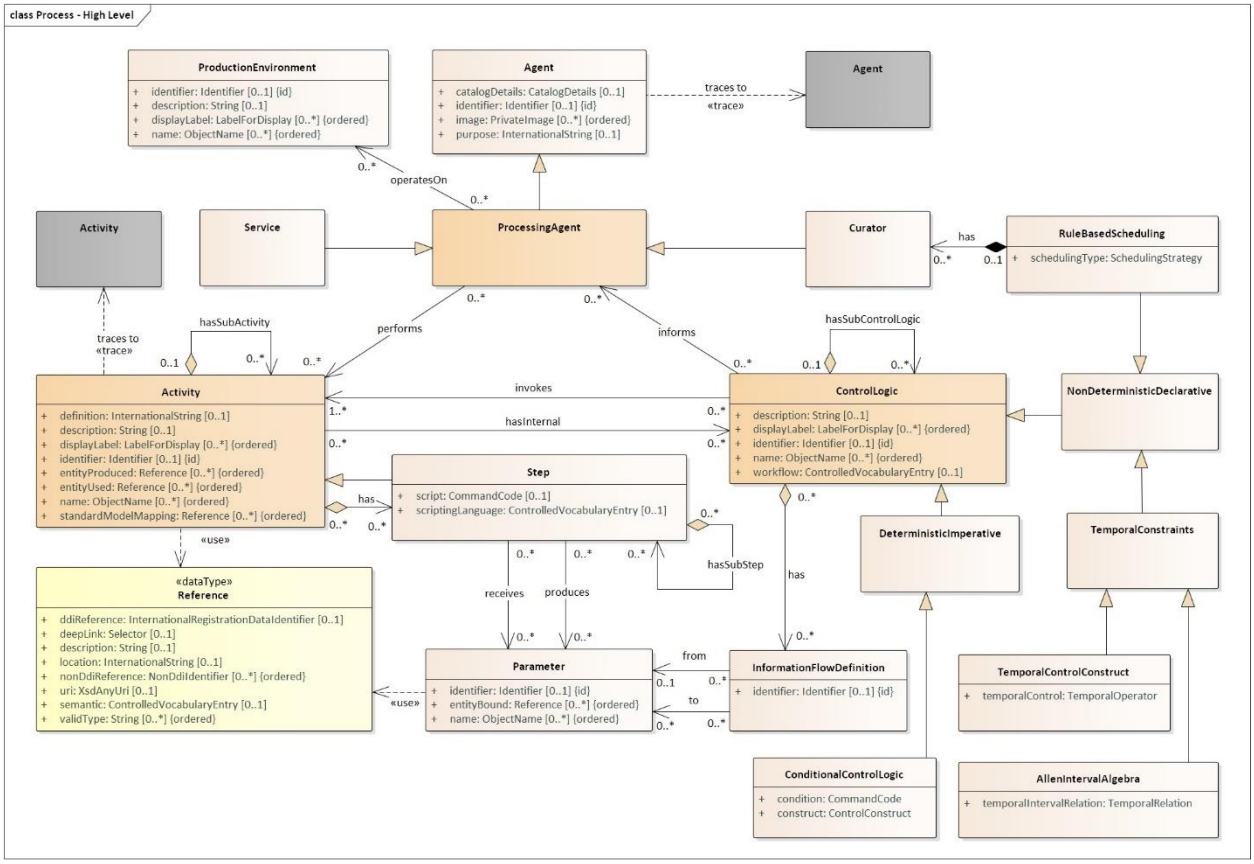


Figure 3: Process Model High Level

The DDI - CDI Process Model is intended to be usable in every type of Production Environment. It supports the description of control systems employing both deterministic and non-deterministic sequencing. Deterministic control systems use deterministic control logic. Non-deterministic control systems include rules-based systems and systems that employ machine learning.

Note that the DDI Core Process Model traces to PROV-O (and thus can be connected to extensions of PROV-O such as ProvONE). This is because the process descriptions in the DDI – CDI model are intended to support the description of process as provenance documentation (as well as to potentially describe/drive execution, which goes beyond what PROV-O is intended for). This is significant – in past versions of DDI, the process model has been designed to both describe workflows (DDI 4 "Moving Forward" Prototype) and the flow of questionnaires and related processing (DDI Lifecycle, DDI 4 "Moving Forward" Prototype) in sufficient detail that they can drive the execution of data processing workflows and questionnaires. In DDI – CDI the focus of the process description is primarily the documentation of provenance, but that it employs a similar model to other DDI specifications, and can

potentially support some of the execution cases as well. Note that DDI – CDI uses References to point to the “entity” in PROV-O, which can be to data at a granular level (variables, datums, etc.), the documentation of specific workflow types, and the evolution of workflows as occurs in machine learning.

B. Describing Data

The DataStore comprises data, potentially of different types. Significant among these are data sets. DDI - CDI provides a detailed description of data and the structures which are used by different types of data sets.

Some of the most significant information classes – those related to describing data – are a major focus if the DDI - CDI model. The diagram below shows a more detailed view of how data fits into the production system.

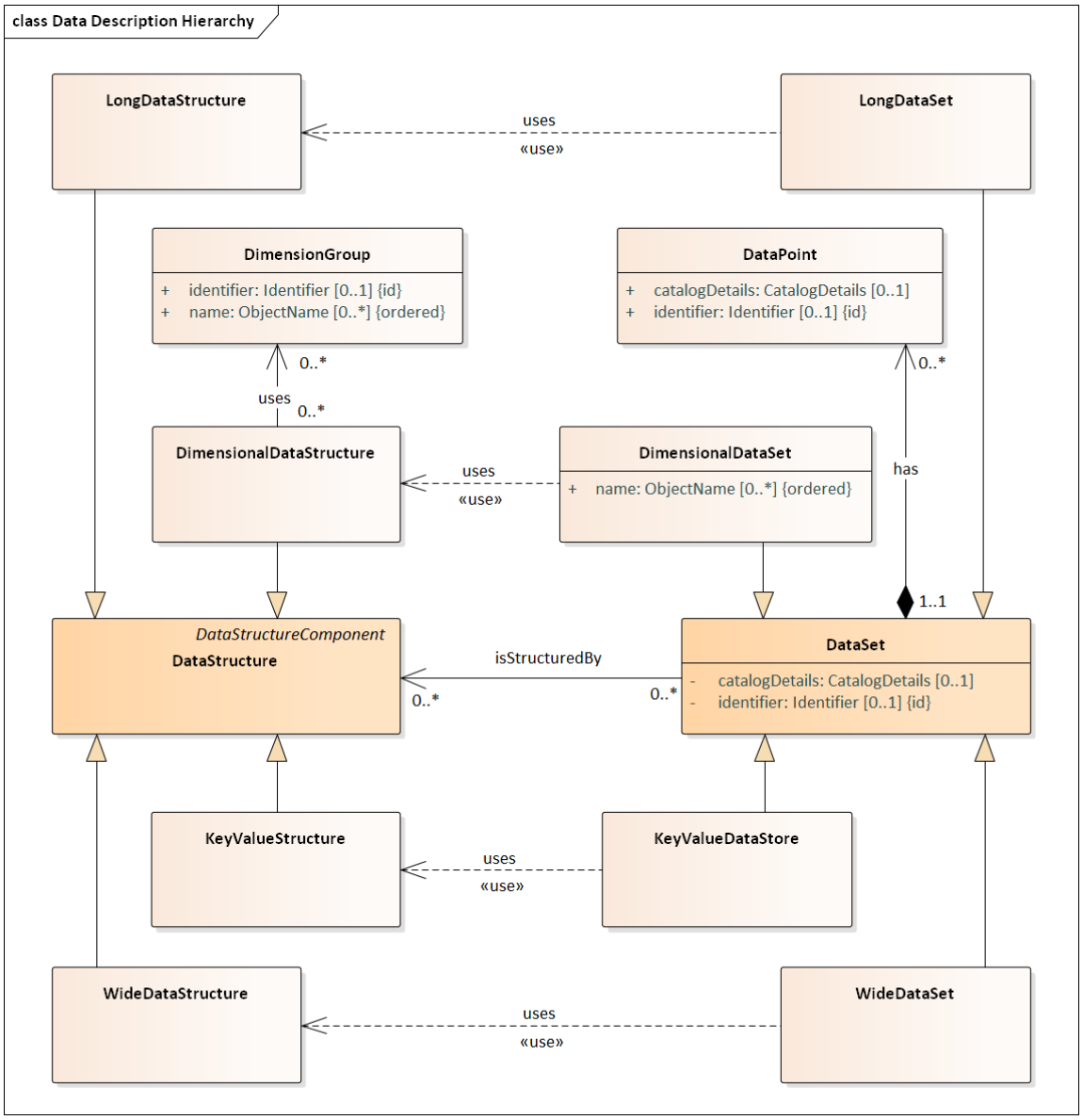


Figure 4: Data Description Hierarchy

The mechanism for describing data structures in the DDI - CDI model provides four basic types: Wide Data, Long Data, Multi-Dimensional Data, and Key-Value Data. These types represent different styles of describing data structures, using a consistent set of components for identification, grouping observations into records, adding descriptive fields, and so on. The differences between each type of data description are found in the roles played by these different components.

The four types are characterized as:

> ***Wide Data:*** This is a way of describing traditional rectangular unit-record data sets. Each record has a set of observations about a single unit. The record has a unit identifier and a set of

measures and/or descriptors which are the same for each unit. The unit identifier can be used as an identifier for the record, because each unit has only one.

Long Data: This is a technique for describing many common types of data, including sensor data, event data, and spell data. In this form, each record has a unit identifier and a set of measures and/or descriptors, but there may be multiple records for any given (observed) unit. The identification of the record is a combination of the unit identifier and one or more other fields. These are sometimes referred to as "tall" data sets.

There are two refinements of the Long Data class which further constrain it to correspond with Event Data (defined as Long Data for which an observation time is provided as a single point) , and Spell Data (Long Data with fields for start and end times bounding a period). In both of these, the record identification involves time as well as the unit identifier.

Multidimensional Data: Multi-dimensional data is data in which observations can be identified using a set of dimensions. These values both identify the cell and serve to describe the measured population. Records may be organized in various ways and may include descriptors as well as their dimensions and measures. It is common to view such data sets as multi-dimensional Cubes, and also to describe them as Time Series. These specific approaches are defined as sub-types of DDI - CDI's multi-dimensional data.

Key-Value Data: Key-Value Data is data which consists of a set of measures, each of which is paired with an identifier. Descriptors may also be attached to these pairs. Such data is often organized in more complex ways when it is used but may be stored or exchanged using this simple construction.

It should be noted that relational data is not an explicit type in DDI - CDI, but can be described using the model. Relational data is typically a set of Wide Data tables connected through the use of keys. While it may be preferable for interoperability to describe the results of queries as single data sets in DDI - CDI, it is still necessary in some scenarios to describe the full relational structure. DDI - CDI uses the Primary Key and Foreign Key classes to capture the relationships between tables.

To understand how DDI - CDI provides a generic description of data across these different types, it is useful to consider how they are built from the perspective of DataPoints. DataSets are collections of DataPoints that can be further organized and described by a DataStructure.

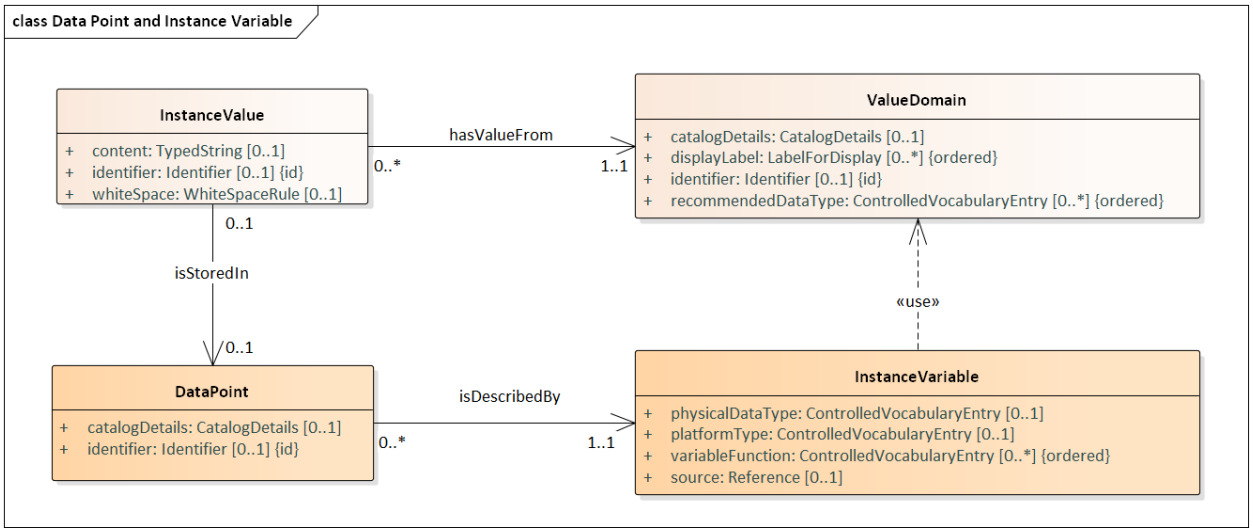


Figure 5: Data Point and Instance Variable

A DataPoint is described by an InstanceVariable and is a placeholder for a value in its ValueDomain. A value in our model is called InstanceValue to distinguish it from the ConceptualValue of which the InstanceValue is the representation (more on this below).

Consider this example:

Name	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8

This (wide) table shows a fragment of a DataSet with six InstanceVariables (column headers), and 12 DataPoints (the cells in gray), and 12 InstanceValues (the content of the cells). For instance, “Female” is an InstanceValue of the Sex InstanceVariable, and “73.7” is an InstanceValue of the Longevity InstanceVariable.

At this point the InstanceVariables may or may not have a physical datatype. At first glance, the “Female” InstanceValue seems to be a String whereas “73.2” seems to be a Decimal. However, there are two caveats here: (i) each platform has different specific datatypes for similar syntax representations, e.g. VARCHAR, String, etc. and we don’t know at this point the platform nor the syntax representation, and (ii) the data type might be indeterminate, in which case the InstanceValue could default to the *universal base class*[1] of the underlying platform/syntax representation.

[1] A universal base class is a term used in programming languages to denote the top or universal type of a type system, usually represented by T. The notion of universal base class is also present in multiple syntax representations. Some common ones are:

As the data is analyzed and processed, and its meaning is clarified and better understood, the InstanceValue's universal types are cast to more detailed data types in successive DataStructures. The data doesn't need to change in this process, only our understanding of it. Using universal base classes it is not necessary to have precise data type definition to start doing analysis and processing. The model provides a few extensions of InstanceValue, e.g., ValueString and RangeValue, which can be used once the datatypes are better understood. Other extensions will be considered (or may be specified by user communities in specific implementation guides).

Another application of top types is for multimedia content, in which DataPoints can contain pretty much anything that can be represented in a platform, from images and videos to sound and complex data structures. There are multiple instances of this in health care, satellite images, maps, etc. in which an Instance Value could be multimedia or data organized in a structure not-known in advance, e.g. XHTML. (It should be noted that DDI - CDI is not designed to support these cases in the current release, but that support for these cases is anticipated in future.)

InstanceValues and DataPoints can be designed in different ways in the different syntax representations. For instance, in XML Schema a DataPoint can be an element containing an InstanceValue attribute or sub-element, which can be defined as "any", in its various XSD forms, or as more specific XSD data types for ValueString and others when the understanding of the datatype is clear. In RDF, it could be a triple in which the DataPoint is the subject, InstanceValue the object, and the association between them the actual predicate. InstanceValue could be just owl:Thing for the most generic datatype case, or any more specific OWL datatypes as necessary.

As we mentioned before, there is a conceptual side to the InstanceValues. The concept associated with an InstanceValue is called a ConceptualValue. A ConceptualValue is just a Concept with a computational model associated with it, reflecting its meaning. This is important in comparing concepts and determining when they are the same.

- Object – in Java, JavaScript, C#, Smalltalk;
- object – in Python;
- Any – in Scala;
- <xsd:any>, <xsd:anyAttribute> and xsd:anyType – in XML Schema and related languages;
- owl:Thing – in the OWL Web Ontology Language.

Resorting to top types is critical in many applications in which data is either heterogenous or its type is unknown, like in many Big Data applications, Hadoop, and other modern data processing platforms. For instance, when ingesting data from outside an organization it is common to use data lakes and related technologies to persist the data in raw form and then apply schema on-read when necessary (i.e., when the data needs to be used, as opposed to when the data is stored).

An InstanceValue is a value from the ValueDomain of an InstanceVariable (the InstanceVariable associated with the DataPoint where the InstanceValue is stored). Similarly, the ConceptualValue it represents is a Concept from the ConceptualDomain of the associated ConceptualVariable.

In DDI - CDI, then, a datum is the combination of an InstanceValue and a ConceptualValue. This may be a point of confusion when comparing CDI to other standards where a datum is defined more like the CDI InstanceValue.

VII. Foundational Metadata: Concept, Datum, and Variable

A. Introduction

This section considers how data can be understood and modelled from a granular perspective. Such a topic can become very technical, and a high degree of precision is required for completeness. The Appendixes provide a description of the DDI - CDI model at this level. Here, we will approach this topic in as understandable a fashion as possible, accepting that a more-complete formalism requires some of the detail we will not address here.

Working upward from the individual values found in data, we will explain how concepts are attached, and how these are arranged in modelling variables. The “variable cascade” is a nuanced model of variables which is at the core of how DDI - CDI describes data, and how it can then be arranged in different structures without losing its meaning.

B. Data

Data is modeled as a set of organized atomic structures, which perform different roles in relation to the data, but which can be associated with concepts to give them meaning. If we consider the most granular bits of data – something termed a Datum in DDI - CDI – we can build up from this to understand how the data is arranged in meaningful, reusable sets. These reusable sets – variables – can then provide the basis for describing higher-level constructs composed from them.

A Datum can be thought of as the contents of an individual cell in a table, or the value stored in a single field in a data set or stream. As described in the section above, a Datum has an encoded representation in a physical sense: what is termed an InstanceValue. The Datum associates this with the meaning of that representation, which is termed a ConceptualValue. In the DDI - CDI model, we see that the Datum combines the InstanceValue with the ConceptualValue that it denotes. Thus, the Datum becomes an easy way to point simultaneously to both the granular representation appearing physically in the data and the direct meaning of that representation.

(The term “instance” is used in opposition to the term “conceptual” when describing Datums, because one of the primary functions of DDI - CDI is the description of how data can be re-arranged into different data structures, which relies on similar Datums being grouped into variables. An “instance” is one appearance of the member of such a group. This in turn allows us to describe now the many instances of these groups can form parts of higher-level constructs.)

If we have a string “2” in our data – an InstanceValue – we can associate this with the number 2 - the corresponding ConceptualValue – rather than being a code which represents the idea of being “married”

or some other categorical concept. The Datum is a single class which describes – in this example - objects such as the value "2" logically being the corresponding number. (A ConceptualValue is defined by a formal Concept in DDI - CDI, but this is only one of the many roles which concepts perform in relation to data.)

The Datum should not be understood as providing a "datatype" for the field – that is a more complex set of relationships which will become apparent as we more fully describe how variables fit into the picture. Similarly, complex sets of coded categorical values will also be described elsewhere in the model. The Datum is a simple joining of the InstanceValue and the ConceptualValue.

Any given Datum exists within the bounds of an InstanceVariable: a description of all of the similar Datums existing within a given structural context. We can think of this as a column appearing in a table, where all of the values in the column are of a similar type, and share a similar form of representation. In our example, the column might be the variable Age, indicated with a number represented as an integer.

Variables are commonly understood as groups of similar values in this way, but we can describe several different types of variables (see Section VII. C.).

InstanceValues may also perform other functions in relation to data, but their pairing with ConceptualValues through the Datum is their primary role within DDI - CDI vis-à-vis the assignment of meaning to data values.

C. Variables: General Description

Variables are groups of values which measure or describe the characteristics of real-world objects, which we term "units". They function both as a way of helping to organize where such values are located (as a column in a table, for example) and as the potential for such values – a variable exists independent of whether it has already been populated or not, as a sort of template for a particular type of value.

Further, Variables are used to assign values to each unit in a group of units of the same type (termed a "population"). The values represent measurements taken on the units, based on a characteristic of the population, which defines the Variable. For example, a population of US children might have age as a characteristic. The age Variable is used to record an age for each child.

Describing variables is, therefore, a vital part of understanding data. There are three levels in the description of variables in DDI - CDI, corresponding to different aspects or uses of the variable.

The Datum is directly associated with an InstanceVariable: within that specific set of data, all the Datums within a group which performs a single function (that is, all the values in our Age column) will be grouped as a single InstanceVariable.

Some properties of the InstanceVariable only exist within the context of the data set in which it appears. Thus – because the data measures a discrete set of objects (the Units which make up the Population in DDI - CDI terms) – there is a relationship between the InstanceVariable and the Population which it describes.

For each Unit in the data – say, a specific child – the Age variable will provide a value. The set of children being measured would constitute the Population in our example, and the data would be made up of measurements of that group of children.

When we have a similar variable in a different data set, it is no longer identical to our InstanceVariable: it may share many other properties, but specifics about each data set are going to differ.

One possible way the variables differ is the Population. Suppose age is measured for children in two school districts, A and B. For the children in School District A, I will have a set of data which has an Age column, where my Datums contain an integer, one value for each child in the Population. For School District B, I will have a similar set of data, differing only in the Population it describes (it is a different group of children): my Datums in that column will again be integers giving the age of each child.

This reusable form of the variable description – the one which is the same across our two data sets – is termed a RepresentedVariable in DDI - CDI. Every InstanceVariable inherently contains the description of a RepresentedVariable, but adds some context-specific information to it (such as the specific Population it measures).

In many data sets, there may be variables which measure the same thing, but do so in different ways. For example, I might choose to represent the Age column for the children in School Districts A and B as an integer, but to use a code for those values in the same type of data for School District C, where I have grouped the integers into three-year cohorts (1-3 years, 4-6 years, 7-9 years, etc.). Further, I might assign codes to these types of cohorts.

When two data sets contain InstanceVariables which are represented in the same way, this is termed a RepresentedVariable. Thus, for the example above, the data sets for School Districts A and B would contain the same RepresentedVariable – a measurement of age as an integer – while the data set for School District C would not, because the Population *and* the way in which the values are represented are different. (Note, however, that the Universe of these two data sets would still be the same – see Section VII. D. 5.)

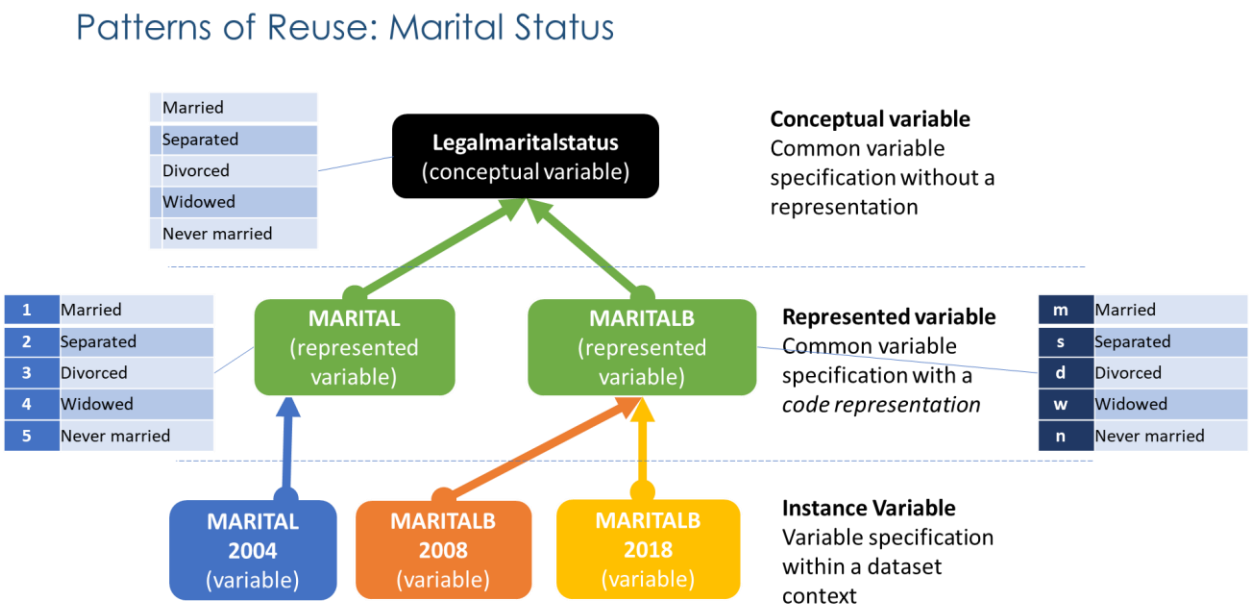
All three data sets still share a high degree of similarity, however: they all measure a characteristic of children – Age – and they all describe the same real-world Units (children, the type of the members of each Population). This level of a variable description is termed a ConceptualVariable in DDI - CDI. While the representation of values may differ the characteristic being measured, and the UnitType of that measurement, are the same.

Again, inherent in every InstanceVariable is a ConceptualVariable. This is also true of RepresentedVariables – each adds some specific information (the representation of the values) to the information needed to describe a ConceptualVariable (the Concept of the characteristic being measured for a particular UnitType).

Systems may choose to only manage InstanceVariables, only Instance and RepresentedVariables, or variables at all three levels. This depends on the kinds of reuse the system is designed to support, but we can see very distinct patterns of reuse at the different levels: InstanceVariables exist only within their data set context, for a particular Population. RepresentedVariables may appear across many different

data sets, and across data sets with different structures, so long as they represent the values in the same way. ConceptualVariables are reusable so long as they are measuring the same characteristic of the same type of Unit.

The diagram below shows how these patterns of reuse occur in related sets of data, in this example for a set of studies containing variables measuring marital status:



It should be noted that some data includes specific measurements of individual units, while other data – aggregates – describes groups of individuals. These types of measurement both use the kinds of data we describe, although they are often structured differently (see the Appendixes).

This three-level model of variables is termed the "variable cascade," and it is central to how DDI - CDI describes data. We will explore this model in greater detail in the next section.

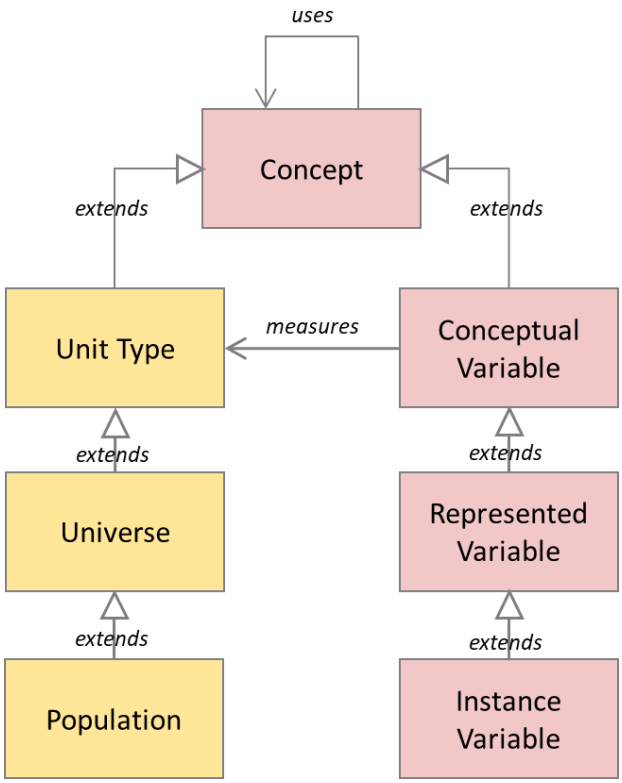
D. The Variable Cascade

In DDI - CDI, the variable cascade is the way the descriptions of variables are managed. The main purpose of the cascade is to increase the reuse of metadata. The features defined at each level of the cascade do not depend on features at any of the lower levels. Because of this, the descriptions at each level are reusable.

The cascade consists of four levels, each level corresponding to an ever-increasing descriptive detail. The levels in the cascade are

- Concept
- Conceptual variable
- Represented variable
- Instance variable

The diagram below shows these levels and some related classes:



The names of the levels indicate to the user what the main focus of the description is at each. The Concept and Conceptual Variable provide details about the concepts employed. The Represented Variable and Instance Variable provide the details about the codes, characters, and numbers representing the concepts at the higher levels.

We will describe these levels and show how they fit into the terminological approach in the following sections. In tables in each section, we illustrate the approach with two examples. The attributes are taken from the class diagram of DDI - CDI. We only illustrate the attributes at each level. The inherited ones from the level above are assumed.

1. Concept

The variables about some subject share that subject as common among them all. For example, all variables in use in data sets in a research library about marital status share that Concept. There may be little in common about the marital status as measured in each variable, but marital status itself – the fact there are statuses across societies or cultures – is a common characteristic. The Concept expressing this commonality is the purpose of this highest level.

The Concept at this level is very generic, because it must account for all possible variations of the more specialized versions attached to each variable that makes use of it. The table below gives some examples.

Concept

ID	*Name*	*Definition*
1	Marital status	Category of current marital arrangement
2	Age	Whole number of years of operation
3	Velocity	Change of position per unit of time

2. ConceptualVariable

The ConceptualVariable is the level at which most of the Concepts used to describe a variable are applied. The main Concepts are specialized applications of the Concept to a type of Unit, and the nature of the measurement or description to be made. In our marital status example, the main Concepts are:

- Specialized application: marital status
  - The specialized nature of this Concept is that it is applied to people living in the US (for instance)
- Measurement: kinds of marital status
  - Single
  - Married
  - Divorced
  - Widowed

Both the Concept being measured (marital status) and the distinct measurements (kinds of marital status) are Concepts. The values used to capture the measurement are known in DDI - CDI as "substantive values."

Additional Concepts are those associated with missing data. These are known as "sentinel values." The two most common examples seen in survey data (and expressed in the statistical packages such as SPSS and Stata) are "missing" and "refused". There may be others, depending on the processing system and the data being described.

3. RepresentedVariable

The main addition at the RepresentedVariable level are the substantive categories. In our example, we might end up with the following designations:

- <s, single>
- <m, married>
- <d, divorced>
- <w, widowed>

The set of these designations is a SubstantiveValueDomain. These Concepts are associated with the subject matter of the variable, not with processing. A SubstantiveValueDomain can be used by many RepresentedVariables, so it is important to identify and manage them.

4. InstanceVariable

Moving further down the chain to data, we get to the InstanceVariable. An InstanceVariable is intended to be a variable used in a data set. For each data set, new InstanceVariables are created.

The main addition in specificity is giving the sentinel categories designations. Further, the list of sentinel values (designations) are managed in one set, the SentinelValueDomain. Separating the substantive and sentinel value domains eases the burden on metadata management. Changes needed in one kind of value domain do not affect the other.

An example of the designations in a sentinel value domain is:

- <m, missing>
- <r, refused>

Since, the InstanceVariable is associated with data in a data set, then the datatype of the data for that variable is necessary information as well.

5. Relationships between Concepts, Variables, Unit Types, Universes, and Populations

In the diagram above, we see the variable cascade, and set of classes which describe the Units, Universes, and Populations for which the variables describe characteristics. There are important relationships between them that must be understood, but which may not be immediately obvious.

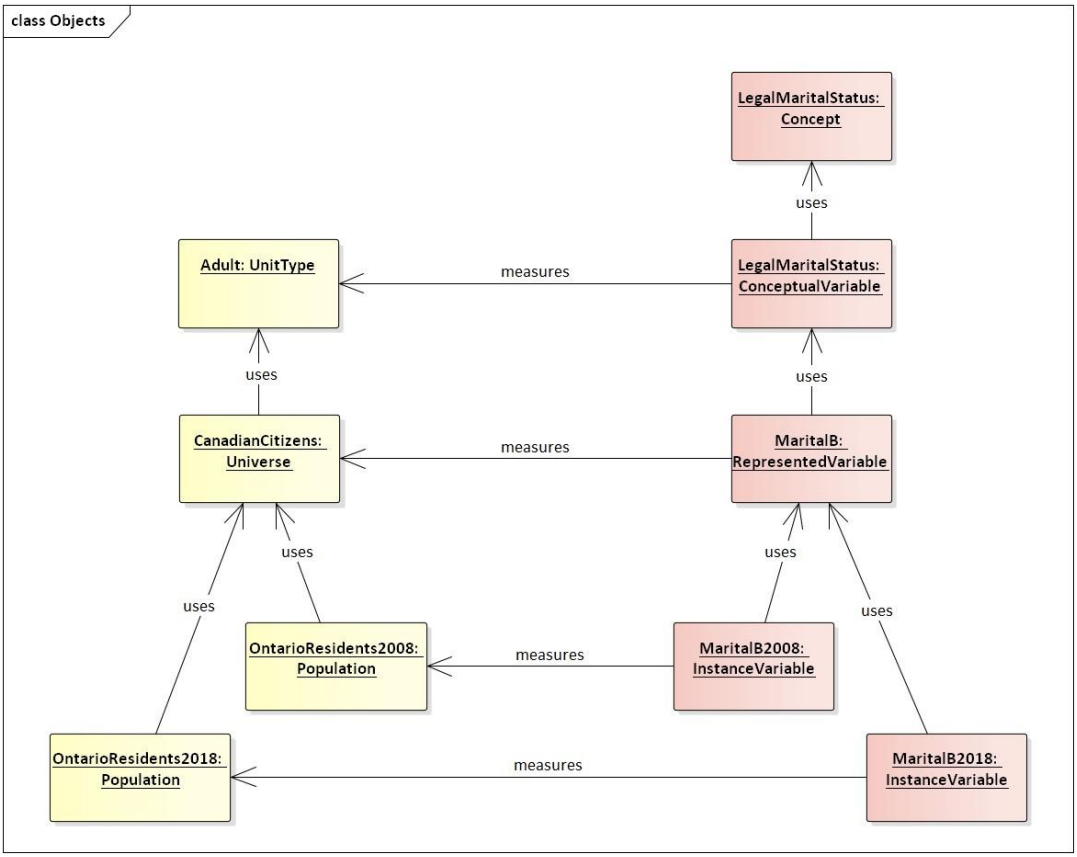


Figure 6: Inheritance chains for Concepts: Examples of Instances

The list of intended uses of the "uses" and "measures" associations is as follows:

- A ConceptualVariable uses a Concept
- A RepresentedVariable uses a ConceptualVariable
- An InstanceVariable uses a RepresentedVariable
- An InstanceVariable uses a ConceptualVariable
- A Universe uses a UnitType
- A Population uses a Universe
- A ConceptualVariable measures a UnitType
- A RepresentedVariable measures a Universe
- An InstanceVariable measures a Population

The different types of variables – ConceptualVariable, RepresentedVariable, InstanceVariable – are related via the inheritance (also called "specialization") chain. An InstanceVariable is a specialization of a RepresentedVariable, which itself is a specialization of a ConceptualVariable. All of these classes are specializations of the Concept class.

A similar chain of inheritance is in operation among the UnitType, Universe, and Population classes (described more fully below). All of these are Concepts, and they form a similar chain of specialization. A

Unit is associated with a UnitType (so, any particular child from our example above be of the type "student" – its UnitType). The set of Units of a given type can be qualified by many different characteristics, among them temporal and geographical ones ("time and space"). The Universe represents this qualified set, using all characteristics *other than* temporal and geographic ones. The Population is a Universe with the additional qualifying characteristics of time and space added, expressed as a selection of Units (the represented sample in cases where not every member of a population is measured).

Note that the measures association is always "horizontal" between levels when the two inheritance chains are considered (with the levels being ConceptualVariable-UnitType, RepresentedVariable-Universe, and Instancevariable-Population).

Note also that multiple InstanceVariables can have "measures" associations with the same Population, as they may reflect differences in sampling or encoding (they might have different SentinelValues.)

DDI - CDI does not describe the qualification of UnitTypes, Universes, or Populations in a very structured way – it provides only for the definition of these Concepts and an indication of how they relate. Their primary role is in understanding and navigating the variables associated with them.

Because all of the variables in the variable cascade, and the UnitType, Universe, and Population classes are all specializations of Concept, they all share the "uses" association. This is intended for specific purposes. Likewise, all of the variables lower in the variable cascade inherit the "measures" association from the ConceptualVariable class, which is likewise intended for expressing a specific type of relationship between the variables in the cascade and the Concepts used as associated UnitTypes, Universes, and Populations.

The "uses" association exists between any two Concepts in the model, but is intended to express very specific relationships. These connect the specific set of instances within the two specialization chains: ConceptualVariable-RepresentedVariable-Instancevariable and UnitType-Universe-Population. An instance of a ConceptualVariable (Age in the example above) would be used by one of more RepresentedVariables, and these in turn would be used by one or more InstanceVariables. These relationships are captured with the "uses" association in the model, with instances in this chain "using" the instance of the class which is closer to the Concept super-class. Thus, a RepresentedVariable which expresses the ConceptualVariable Age as an integer would be associated with it via the "uses" association. Likewise, an InstanceVariable which expresses Age as an integer would relate to that RepresentedVariable with a "uses" association. (It is also possible for InstanceVariables to have "uses" relationships with ConceptualVariables directly, to express the fact that they are all measuring the same Concept.) This function of the "uses" association also exists within the "UnitType-Universe-Population" chain: each instance of these classes should be related to the instance of the class which is closer to Concept in the specialization chain through the "uses" association.

This set of relationships demands that there be a corresponding set of relationships expressed with the "measures" associations. In our example, the Age ConceptualVariable describes a characteristic of students (the UnitType). It would express this relationship through a "measures" association with the "student" instance of the UnitType class. Any RepresentedVariable which uses the Age

ConceptualVariable will have a “measures” association to a Universe which itself has a “uses” relationship with the “student” UnitType. Similarly, any instance of the InstanceVariable class which “uses” an instance of RepresentedVariable should have a “measures” association to an instance of the Population class which “uses” the instance of the Universe measured by that RepresentedVariable instance.

This model has some implications for the description of UnitTypes, Universes, and Populations. These classes are described more fully below.

6. Physical Datatypes

The Physical Datatype addresses the kind of data as written on a file, and therefore is often an approximation of what is needed to describe values. (The actual use of the values depends more on the Intended Datatype at the Represented Variable level.) The value $2.60 (two dollars and sixty cents) is often written as a real number with 2 decimal places. But monetary amounts don’t follow all the rules for real numbers. The amounts at the third decimal place or after are truncated. The values are not rounded, as real numbers will be. This has an effect on computations, as the following example illustrates:

Take the average of $1.50, $1.30, and $1.00. The arithmetic average is $1.2666. The rounded real number average is $1.27, and the monetary, or scaled number, rounded average is $1.26. (The fractional penny is dropped in the scaled situation.) The rules for scaled numbers correspond to how banks handle money, for example.

E. Populations, Units, and Unit Types

One important feature of the variable cascade is the levels at which relationships exist to the phenomenon being measured, as discussed above. A Datum is associated directly with a Unit, and an InstanceVariable with a specific group of Units – a Population. At the level of the RepresentedVariable, a Universe is associated with the data. A UnitType – the class of the Unit – is associated at the ConceptualVariable.

The UnitType, Universe, and Population represent a “cascade” which shadows the variable cascade itself: the Concept is applied to a particular type of unit to define a ConceptualVariable (the Age of a Person). When the application of this concept is specialized (narrowed) for use as a representation (an Integer between 0 and 20 in whole years for children, for example), this defines a Universe of those types of Unit. When a particular set of Units from that Universe are selected, these form a Population (the children in School District A).

Units are perhaps a less obvious feature of data when it is not focused on people (as in the examples above). The same ideas are in operation regardless of the domain, however: we can consider the situation where sensors are measuring salinity and other properties of sea-water. Salinity is a Concept; Salinity of Sea-Water is a ConceptualVariable (Sea-Water is a UnitType). We can define a Universe such as the Salinity of Sea Water in the Baltic (a specialization) and then select the locations of the sensors to make up our Population (a set of locations of Sea Water in the Baltic being our Units).

The way in which UnitTypes, Universes, and Populations are described is left very open in the DDI - CDI model: the definitions are descriptive, rather than being fully structured. These descriptions should be crafted with the relationships to variables in mind, as the breadth of UnitTypes and Universes will limit how broadly they may be used, and this will have a direct impact on how they interact with the data they help systems to organize.

F. Concepts, Codelists, and Classifications

The variable cascade shows the use of Concepts in several places, and it is important to understand how these uses (and reuses) of Concepts work in the DDI - CDI model.

A Concept is simply the formal definition of an idea: a unit of thought differentiated by characteristics. Typically, this is expressed as a formally defined term (minimally, the Name and Definition properties of the Concept).

Concepts are very important in navigating and comparing different data, but they play several roles. If we are to perform navigation and comparison in an automated fashion, it is important to understand these different roles. The DDI - CDI model provides this.

In addition to being associated with the definition of a variable, Concepts also define Universes and UnitTypes, as described above. Further, they are often used as Categories in the representation of ValueDomains.

Consider a data set where there is a ConceptualVariable “Gender,” represented by one of a set of Categories (“Male,” “Female,” etc.). This might be comparable – via some form of transformation – with a data set which has a variable “IsMale” – based on the same Concept “Male” used as a Category in the first data set – represented with a binary “Yes” or “No”. The fact that the same Concept “Male” is used in these two roles may provide an indication that the two data sets may be comparable, or useful in answering a research question. The DDI - CDI model provides for a formalization in which this situation could be detected and acted on by a machine.

The use of Concepts as Categories is important, because there we see how they are associated with specific Units within a Population. Even when representations of variables is different, it is possible for the same Concepts to exist as ConceptualValues. This is the case in the marital status example shown above: the same sets of Categories – that is, the same Concepts – may be represented using different Codes. This level of detail is again significant when operating on the data to understand what is and is not potentially comparable.

It is worth noting that the use of Categories ties a Concept to two important structures in the DDI - CDI model: in describing data, they provide the meanings of Codes, which are often used as ConceptualValues in the Datums described by variables. They also provide the definitions for ClassificationItems – the nodes in Classifications. Categories may carry additional value in the context of a Classification or Codelist in informing the meaning of the Code for human users as well as machines.

Concepts are also used in a number of other roles in the DDI - CDI model, but the mentioned uses are significant in understanding how the variable cascade functions.

VIII. Data Description

A. Introduction: Reading the Model

The DDI - CDI model is defined as a Unified Modeling Language (UML) model. Figure 7 below shows a core portion of that model. The elements (classes) of the model appear as boxes with a name at the top and a list of properties below the name. Properties, listed in the bottom half of the box for the class, contain the payload of the class. Sometimes the value of a property is complex. The "definition" property of a Concept, for example, has the datatype of "InternationalString", which will have a text string, but also other properties such as whether it is translated and from what language it is translated. (This complexity is the result of many years of incorporation of use cases into the model.)

Classes may also have associations with other identifiable classes. In the diagram below a Datum has a simple association named "denotes" with a ConceptualValue. This relationship is read as "a Datum denotes a ConceptualValue". This relationship is read as "a Datum denotes a ConceptualValue". It is displayed in the diagram as an arrow that indicates the order in which the association is to be read. Classes that can be the target (object) of an association have a unique identifier and are reusable. The target end is indicated by an open arrowhead.

Some classes inherit from others. This is indicated by the filled-in triangular arrowhead on the parent end.

Some associations indicate containership. A ConceptSystem aggregates (has) a set of Concepts. This is indicated by the diamond on the containing end of the relationship line and is read as "a ConceptSystem has Concept".

In the diagram below the content property of InstanceValue is a TypedString, which holds a physical representation (and optionally a code describing its type taken from a user-defined controlled vocabulary). There can be a chain of these complex datatypes as seen in the diagram where Concept uses InternationalString which in turn uses LanguageString. Introduction of the ConceptualValue allows for the description of multiple representations of the same measurement across multiple platforms. A height, for example could be recorded as a decimal string or a binary string.

Physical structures (like files) are made up of DataPoints, each of which contains one InstanceValue.

The general idea in DDI - CDI is to be able to attach metadata at the "cellular" level, rather than at the structural level, and to allow those "cells" to be arranged into different structures without loss of descriptive information.

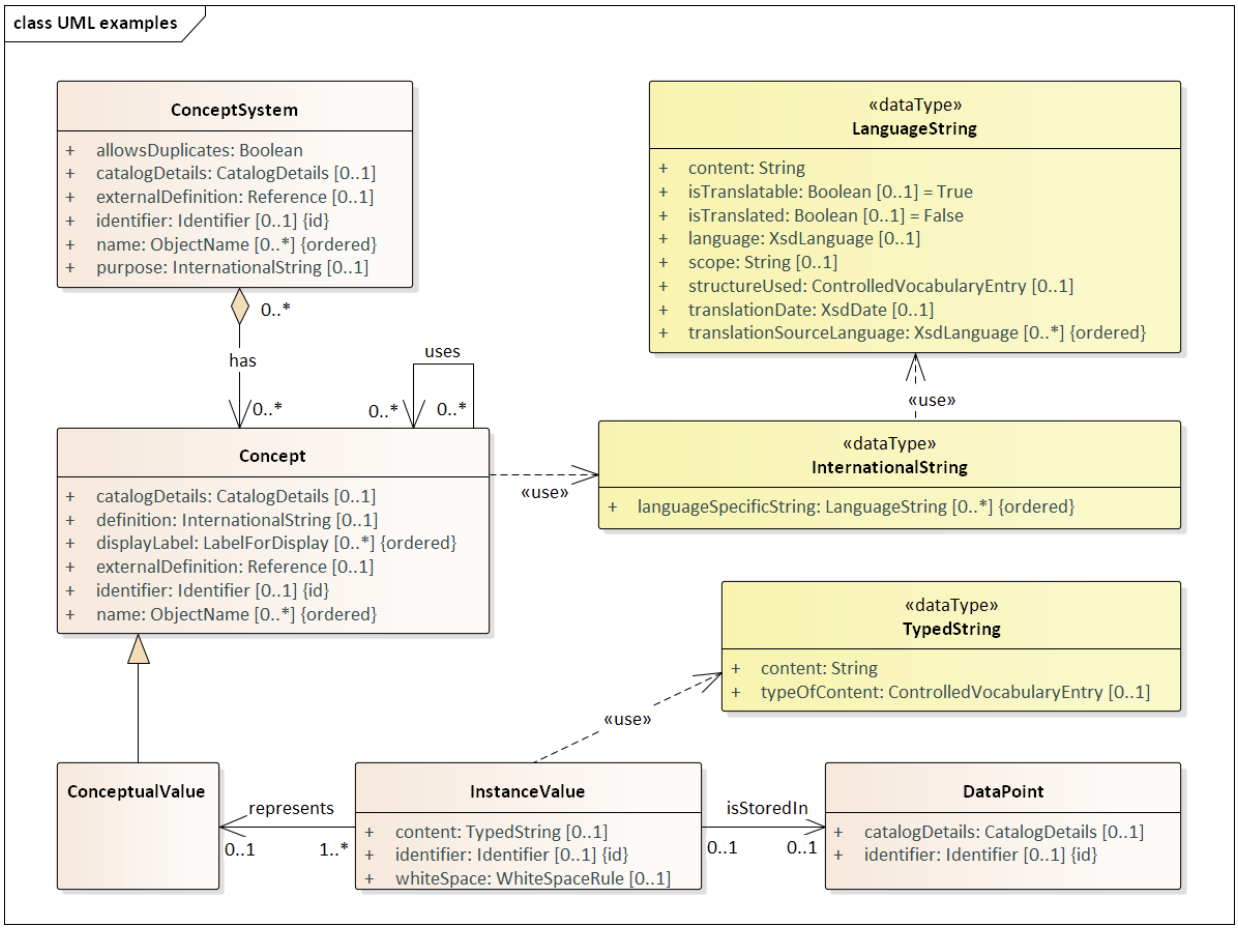


Figure 7: Example Taken from the DDI – CDI Model

B. Scope

The DDI – CDI Data Description provides the basis for describing a broad range of data structures using a core set of metadata elements. The model separates data structure and content in such a way as to allow the same data to be structured flexibly, appearing in different forms without losing its meaning or identity.

This section describes the general approach of DDI – CDI before going through the details for a selected set of data structures. The goal for DDI – CDI is to describe various data structures, both legacy structures such as rectangular data sets, multi-dimensional data, and event data, but also more recent ones like data streams or data lakes. The approach is independent of any specific domain or discipline, as similar data structures are used broadly in a range of research settings.

The model has structures for documenting different data structures and the transformations between them.

Data structures are a way to organize data for processing by software programs. The current DDI - CDI model describes data from different data structures using a Datum-based approach. This approach

involves describing each “cell” in a granular fashion, such that the same values can be recognized when occurring in data sets and streams which have different structures.

Each of the four data structure types – Wide, Long, Dimensional, and Key-Value – while slightly different, share some common features (see section VI B above). Before going into how each of them can be structured a set of related common components will be presented, applicable across the range of data structures described.

It should be noted that relational structures can be described as a set of tables with keys indicating their relationships, and so are not supported as a separate type.

C. Basic Concepts

Before explaining about the four data structure types some basic shared concepts require explanation.

1. Variables and Values

The variable cascade has been described above. In this section we discuss how values interact with this model of variables. Central to this discussion are two aspects of the value: the domains from which it draws its meaning and representation, and the way in which the values are associated with the higher-level structures.

Domains provide representations and contexts for the data values. The SubstantiveConceptualDomain specifies the set of valid concepts for the ConceptualVariable, while the SubstantiveValueDomain specifies the set of values for the corresponding InstanceValues.

It is important to understand that substantive concepts and values are not the only ones which require description. We also have a class of “sentinel” concepts and values which describe the status of a variable, or reflect a meaningful aspect of the process.

Conceptual values comprise the meaning of the data – they are divided into “substantive” (that is, those relevant to the scientific use of the data) and “sentinel” (those describing technical or process-related aspects of the data). When describing variables, both of these are used, as appropriate. A variable which measures altitude might have as values a measure of height above sea level (a substantive value), or it might contain an indication that the altitude was unspecified (a sentinel value). These concepts would be described by the SubstantiveConceptualDomain and SentinelConceptualDomain, respectively.

In the data itself, the InstanceValue might be a number (the measure) or a code (i.e., “unspecified”). These would be described as valid values in the SubstantiveValueDomain and the SentinelValueDomain, respectively.

To understand how these values are modelled in relation to the variable, we must look at DataPoints, Datums, and InstanceValues.

A DataPoint is the space within which a value may reside. You can think of a data point as a cell in a table: it may be populated with a value or not, but it still exists as a place in the structure where a value can be held. Data points are *not* the values themselves – they are the place where the values are stored, and in this sense are purely structural.

As described above, an InstanceValue conveys the meaning of a ConceptualValue by representing it. Thus, it is the InstanceValue which resides in the DataPoint, even though it is conveying the meaning described by the associated ConceptualValue. In a table, we might see a “3” in the “Marital Status” column, and we understand that “3” is a code representing the status “Unmarried”.

In the CDI model we have a class which represents the union of these ideas: the Datum. A Datum is a granular bit of data which uses an instance value – a representation – to convey a conceptual value. It is the union of these two, and so is the most granular meaningful object in the description of data.

An InstanceVariable describes the DataPoints which appear in a given data set (or similar collection). In our notional table, the “Marital Status” column is an InstanceVariable: it provides the template for the values in that column, each of which is a Datum (a ConceptualValue represented by an InstanceValue). All of the Datums in the column will have values coming from the SubstantiveValueDomain or the SentinelValueDomain associated with the Datum’s InstanceValue, and these will in turn represent ConceptualValues coming from the SubstantiveConceptualDomain or SentinelConceptualDomain.

When navigating the model, it is important to remember that ConceptualValues are both substantive and sentinel, and that ConceptualDomains are likewise both substantive and sentinel. (The conceptual relationships are conveyed using the sub-classing feature of UML.)

In the diagram below, we can see that the InstanceVariable brings together the Datum and the DataPoint, thus showing how the cells in our table are populated.

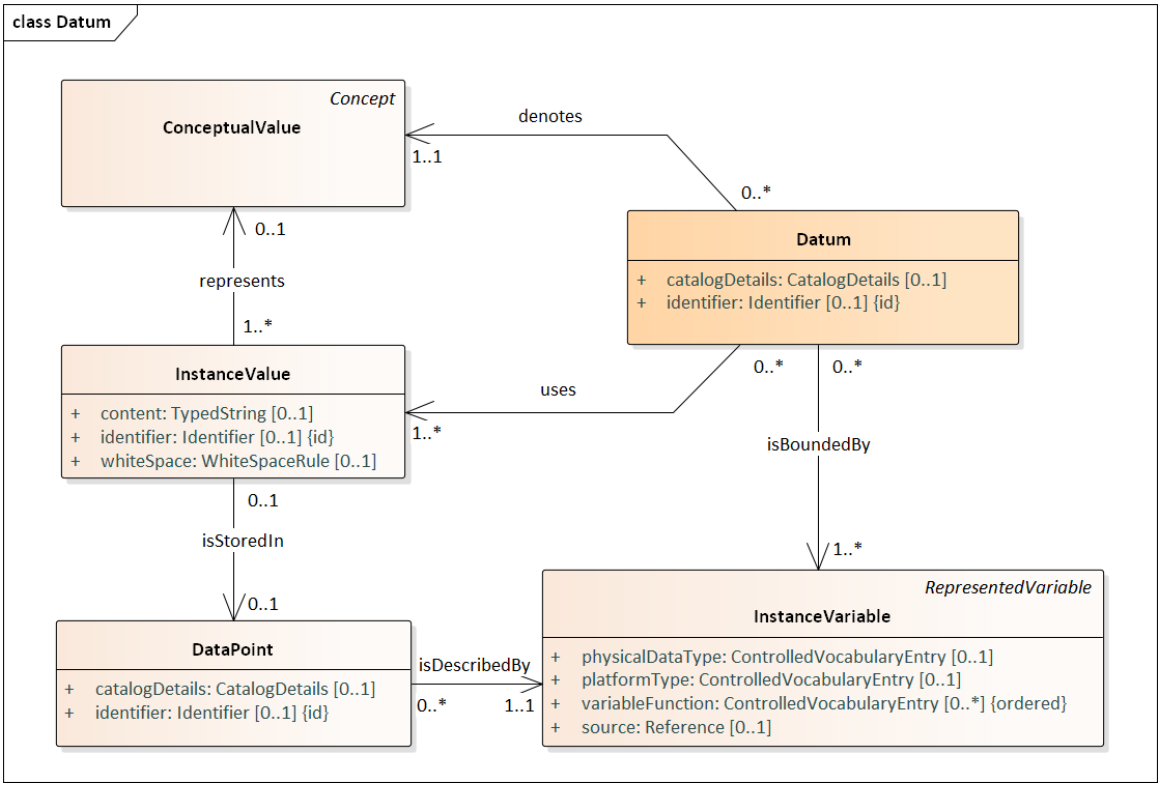
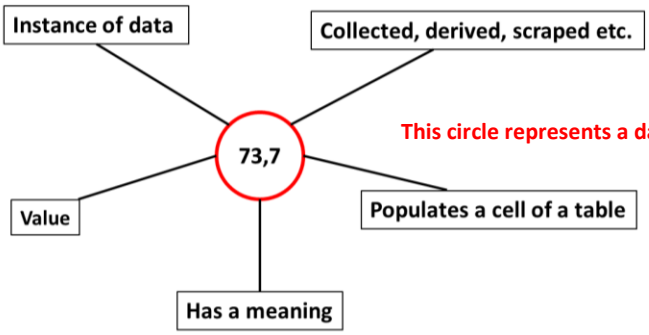


Figure 8: Datum

The role of the Datum as the most granular meaningful construct in the model is significant: it allows the single Datum to be be 'followed' across different data sets/streams with different structures.[2] A Datum populates a cell of a dataset, database table etc. The general idea in DDI - CDI is to be able to attach all necessary metadata to the single Datum so that this information is not lost when it appears in different places with different data structures.

This concept is summarized in the figure below: the Datum can be thought of as a central building block in the DDI - CDI model, with its associated variable (it is connected through the InstanceVariable to the reusable RepresentedVariable and ConceptualVariable, as per the variable cascade described above.)

[2] This differs from approaches used in many other similar models, including other DDI products (e.g., DDI Codebook, DDI Lifecycle) where some of this information was attached at a higher level (typically the data set or record). DDI - CDI is a little more explicit than GSIM in describing the conceptual value and its representation, the instance value. As an information model, GSIM does not deal with the details of physical representation of data. DDI - CDI needs to be able to describe representation in more detail.



2. Keys

Another central concept for Data Description is that of the Key. In the model a Key is used for the identification of data and may comprise a set of Key members or be a unitary value. Figure 11 below shows how an InstanceValue is linked to a Unit (an individual or object of interest) via a Key that identifies the DataPoint where the InstanceValue is stored.

A Key is defined as a collection of data instances that uniquely identify one or more data points in a given data structure. A KeyMember is a single data instance that is a part of an aggregate key. (Note that the members of a Key are themselves specialized InstanceValues stored in DataPoints.)

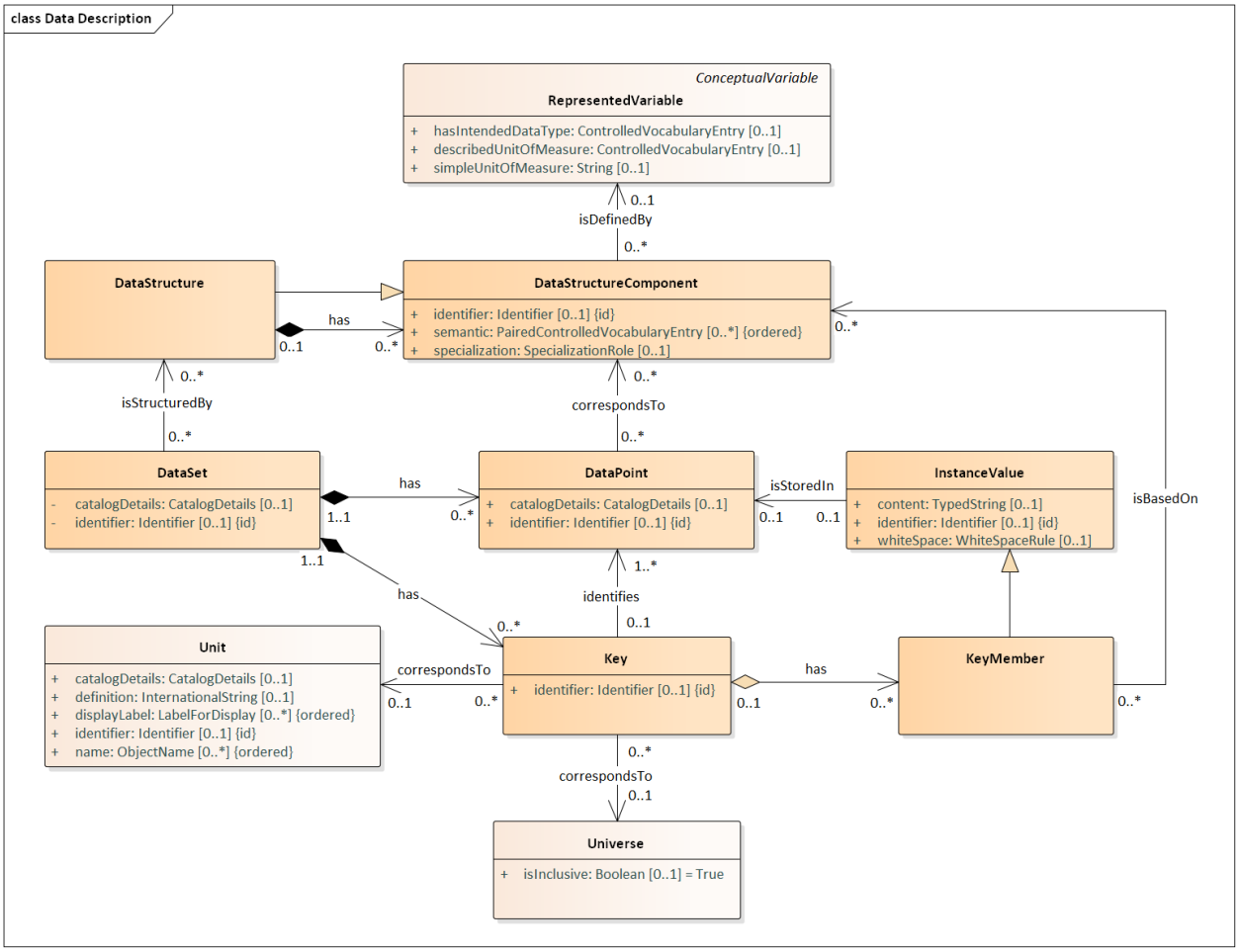


Figure 9: Core Data Structures and Related Classes

3. Data Structure Components

A third important feature of the model are DataStructureComponents, allowing the RepresentedVariables to take different roles. Each data structure type - Wide, Long, Multi-Dimensional and Key-Value - has its own set of DataStructureComponents, with some being common across data structure types. The components which reflect roles are the IdentifierComponent, AttributeComponent and MeasureComponent. The roles allow a RepresentedVariable to serve as a Measure in one context and as an Identifier in a different context, for example (where the “context” is a particular data set/stream with its own structure). This will be detailed below in the description of the different data structure types.

Roles allow users to assign different structural functions to variables according to their context of use. Roles are not inherent in variables but are imposed on them, as appropriate. In DDI - CDI there are currently three “common” roles used across all data structures, and several additional ones which are used only in some of the possible data structure descriptions:

Common Components:

- **Identifier** - An identifier role that serves to identify a unit. More than one variable may be used in combination to produce a compound identifier for the unit of a particular observation.
- **Measure** – Variables tagged with the measure role represent the values of interest.
- **Attribute** – The attribute role serves to provide information about the measures of interest. Variables might, for example, describe the conditions of a measurement. This way attributes can be used to link metadata or paradata to the Measure of interest.

Additional Comnponents:

- **Dimension** – Acts as a field in a compound identifier for disambiguating the cells in a multidimensional table or "cube". Specific to multi-dimensional data structures.
- **Primary Key** – Acts like a primary key attribute in a relational model. Used in conjunction with a ForeignKeyComponent in another table structure to indicate a relationship between the two structures.
- **Foreign Key** - Acts like a foreign key attribute in a relational model (e.g., a column). Used in conjunction with a PrimaryKeyComponent in another table structure to indicate a relationship between the two structures.
- **Variable Descriptor** – Acts in a record to identify the variable to which the value in the accompanying VariableMeasure component applies. Specific to Long data structures.
- **Variable Measure** – Holds values for the variable in a record which is specified by the VariableDescriptorComponent. Specific to Long data structures.
- **Synthetic ID** – Used to hold a PID in some types of data structures, to complement or supplant ordinary identifying components. Populated with a GUID, UUID, or other, similar identifier. (SyntheticIdComponents are not intended as the primary form of identification structures for data in DDI - CDI, but are sometimes needed/useful.)
- **Contextual** – Used to indicate the scope within which a Key is unique within a Key-Value Data Store. Specific to Key-Value data structures.

A variable may take on different roles in different contexts.

Data structure components have two attributes which can serve important functions in some cases: semantic and specializationRole. These are described below:

Semantic – this attribute provides a value from a controlled vocabulary to further inform the role indicated by the data structure component type. This could be taken from an external controlled

vocabulary (such as an ontology) in cases where the function of the role needs further descriptive semantics attached to it.

SpecializationRole – This attribute is required for the automation of some types of data integration and harmonization, by indicting additional functions performed by the data structure component. These provide further information about time, geography, or other aspects of the component which are needed in processing to identify how data sets might be joined. (For example, in a data set which has multiple variables holding time values, the TimeRole attribute might hold a value of "ReferencePeriod" to indicate that it is useful for temporal integration.)

D. Wide Format (Unit Record Data Structure)

1. Example

A Unit Record data table, as shown in Figure 12, is a common way to organize data. Each record has a set of observations about a single unit. The record has a unit identifier (a variable being used as an IdentifierComponent which holds an identifier for the Unit) and a set of measures and/or descriptors which are the same for each unit. The unit identifier can be used as an identifier for the record, because each unit has only one. This structure is also referred to as a rectangular data file.

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8
etc.					

The objects of the Wide format Unit Record data table are unit data records, variables and InstanceValues. In the Wide format the rows correspond to each unit record, which is a set of InstanceValues for one entity (Unit). The columns correspond to each variable measure or categorization. Cell entries are InstanceValues.

A cell in the Unit record table is an intersection between a column representing a variable and a row representing a measurement unit. See for example '8.1.1929' in Figure 13 (yellow highlighting).

Each cell of the table contains an InstanceValue. 'Marie' and 'Henry' (green highlighting) are identifiers for each of the records. 'Sex', 'Longevity' etc. are variables (blue) and 'Female' and '78.8' are example of InstanceValues (red).

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2006	Cardiff	76.8
etc.					

The WideDataSet contains DataPoints, all the 'cells' in the table. Columns contain the set of values from individual variables, and each row contain the DataPoints for one Unit. Some of the DataPoints contain values keys that identify the DataPoints common to an individual row of the table. A WideKey can have more than one Member - e.g. more DataPoints which act as identifiers. This will be further explained below.

2. Discussion of Structure and Diagrams – Wide

A Wide table row is further structured by three DataStructureComponents types:

- IdentifierComponents - the DataPoints which serve to identify the row.
- MeasurementComponents - the DataPoints in each row which contain the measures of interest.
- AttributeComponents - DataPoints which provide context for the MeasureComponents.

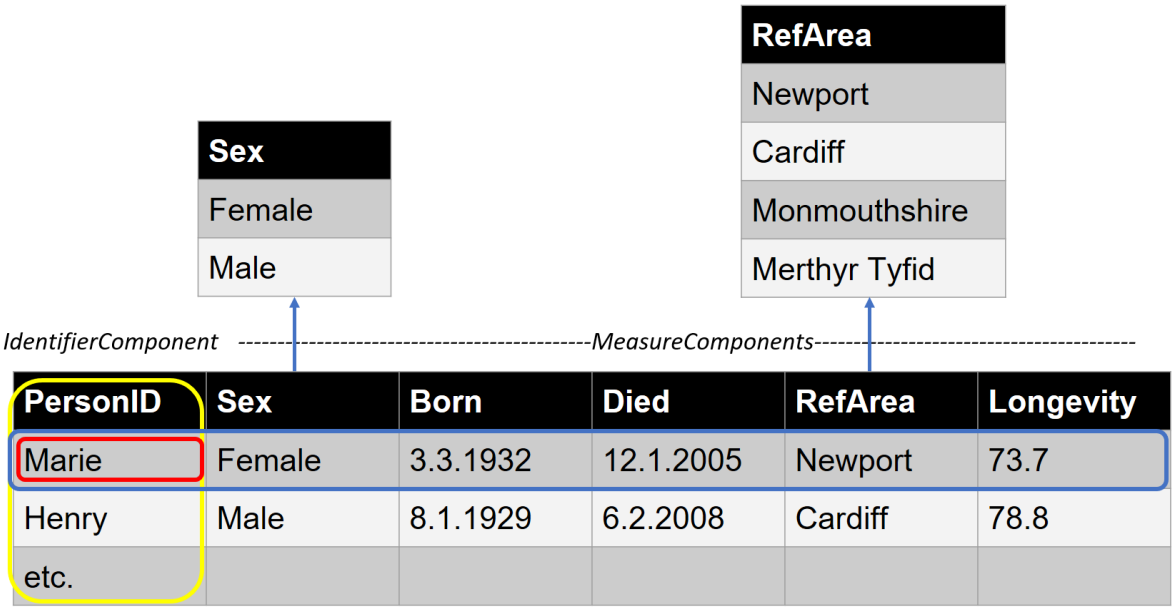
RepresentedVariables provide SubstantiveValues for a WideKeyMember.

In the example dataset displayed in Figure 14 below the "PersonID" column contains DataPoints that contain the key values that identify a row and also correspond to a Unit.

The DataPoint in the upper left of the table contains the key value "Marie". That DataPoint identifies the other DataPoints also associated with the person named "Marie", the DataPoints in the first row of the table.

A WideKey can be composed of more than one WideKeyMember. Our table might, for instance, have contained another column like "Family" so that we could identify the Marie in a particular family. (This might be important in a data set which had more than one unit named "Marie", requiring a further value to disambiguate.)

These are defined by RepresentedVariables, which in turn provide the SubstantiveValueDomain (often a Codelist) for a WideKeyMember.



In the figure above, PersonID is an identifier for a person, Sex, Born, Died, and Longevity and RefArea are the measures of interest.

These roles are not fixed, but are dependent on how the role functions in the context of the structure. For another purpose, RefArea might be considered an attribute of the measures. Roles are often slightly different when the same data is viewed using different formats (PersonID is the only identifier needed for the Unit Record format above – when expressed in a Long format, it would be only one needed component of a compound identifier – more than one variable would take on the role of identifier. (See the microdata.no example, below).

Note that at least one variable functioning as an IdentifierComponent must be specified. (It would also be unusual not to have at least one MeasureComponent in a data structure.)

The diagram in Figure 15 below shows the DDI - CDI classes used to represent unit data in wide format. This is probably the most common layout for data – the traditional table of data as used in many statistical packages and spreadsheet programs. Columns are variables and each row contain the DataPoints for one Unit.

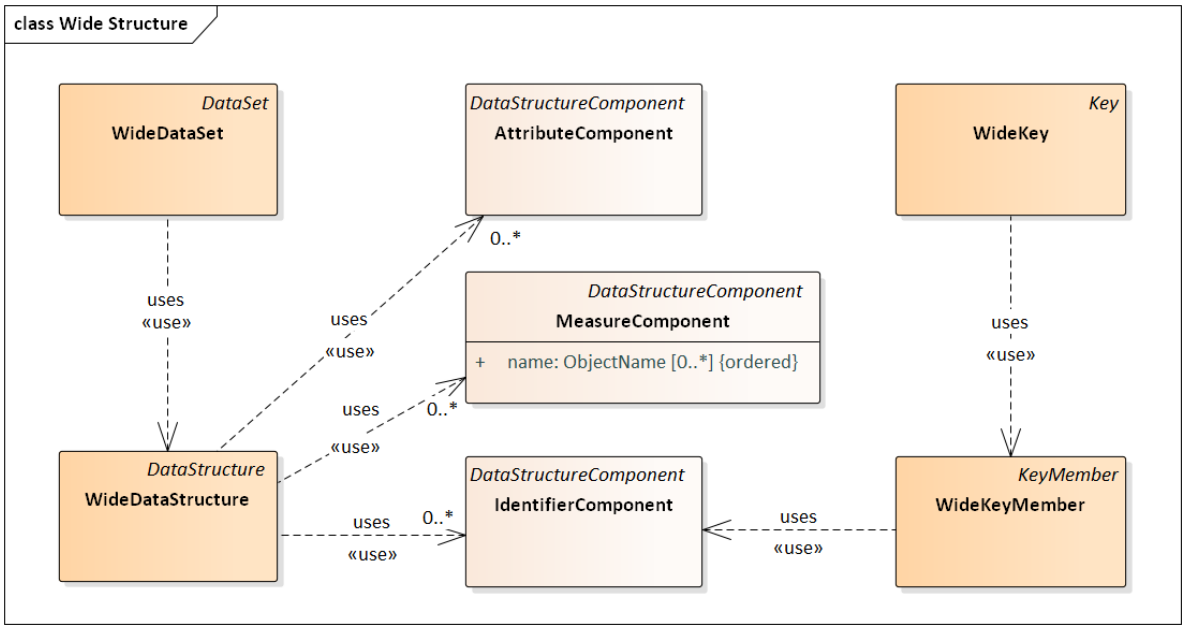


Figure 10: Wide Structure

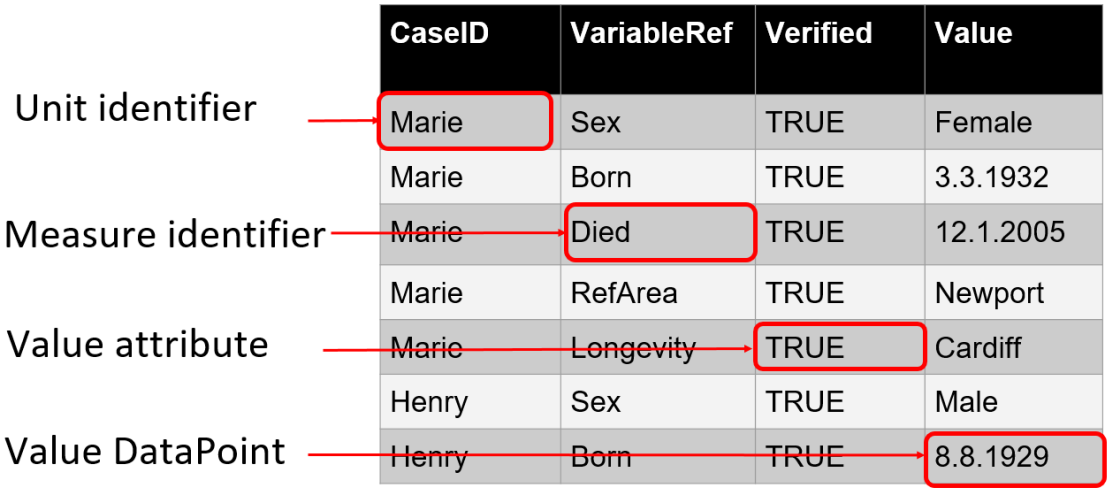
E. Long Data Format

1. Example

The same data as in the Wide example can be expressed in a different format called Long as shown in Figure 16 below. This format is often used to express event data which is collected as an on-going "stream" over time.

In the Long format columns correspond to each kind of object in a Wide (unit record) description. Each row now contains (at a minimum) a unit identifier, a variable identifier, and a data point with an Instance Value.

The rows correspond to each value of each (non-identifying) variable for each Wide record. Note that for this example, a process value has been added, indicating whether the value has been subjected to a verification check – this is just for purposes of illustration, to show how an attribute component functions in the Long structure.



CaseID	VariableRef	Verified	Value
Marie	Sex	TRUE	Female
Marie	Born	TRUE	3.3.1932
Marie	Died	TRUE	12.1.2005
Marie	RefArea	TRUE	Newport
Marie	Longevity	TRUE	Cardiff
Henry	Sex	TRUE	Male
Henry	Born	TRUE	8.8.1929

In pure form, each row of a Long structure contains a DataPoint with the value of interest (the InstanceValue) along with identifiers for a unit and a column with a code that identifies the variable (VariableRef above) that associates with the value in the value DataPoint. In the figure above the Value column contains DataPoints with values from more than one variable (Sex, Born, Died, RefArea, and Longevity). Note that there may be many rows for a unit (like for “Marie”). There can also be columns containing attribute values. The “Verified” column is an attribute that indicates whether the value in the Value column has been verified.

CaseID	VariableRef	Value
Marie	Sex	Female
Marie	Born	3.3.1932
Marie	Died	12.1.2005
Marie	RefArea	Newport
Marie	Longevity	73.7
Henry	Born	8.1.1929
Henry	Died	6.2.2008
Etc.		

Here we can see how a complete record for one unit from our Wide example might be represented: each column in the Wide format for a single row becomes a row in the Long format (see Transformations between Data Structures, Examples, below).

2. Discussion of Structure and Diagrams – Long

The high-level view of the LongDataStructure model is shown below. Each DataPoint in the dataset is based on one of five data structure components. Each component is associated with a RepresentedVariable that can define a column in the long table.

These perform the following functions:

• **IdentifierComponent** – one of possibly several components that together identify the Unit associated with the measures and attributes. In the example above this is the CaseID column in Figure 16.

• **MeasureComponent** – a measure just like in the wide layout. This allows a hybrid wide-long layout. There is no such column in the example above - if there were the values for the Marie rows in Figure 16 would all be the same.

• **AttributeComponent** – an attribute that annotates the associated measure values. This is the Verified column in Figure 16.

• **VariableDescriptorComponent** – an indicator of the InstanceVariable in each associated VariableValueComponent DataPoint (see Diagram). This is the VariableRef column above. In the first row the code “Sex” indicates that the value “Female” is associated with the variable named “Sex” used in the Wide table. Note that this component has an association to a specific VariableValueComponent.

• **VariableValueComponent** – defines a column that has a value associated with the value in the VariableDescriptorComponent. This is the Value column above. The “3.3.1932” is interpreted as the date that Marie was born. This column will have to have a datatype as generic as needed to hold all of the values from the set of variables indicated in the VariableDescriptorComponent. In the example above there is a mix of numeric (Longevity), Date (Born, Died), character (Sex), and geographic codes (RefArea) variables. A character datatype for the associated RepresentedVariable would be required. In many statistical platforms there are tools to reshape data between wide and long format. Many have restrictions that would force all of the measure values to have the same datatype (e.g. all numeric).

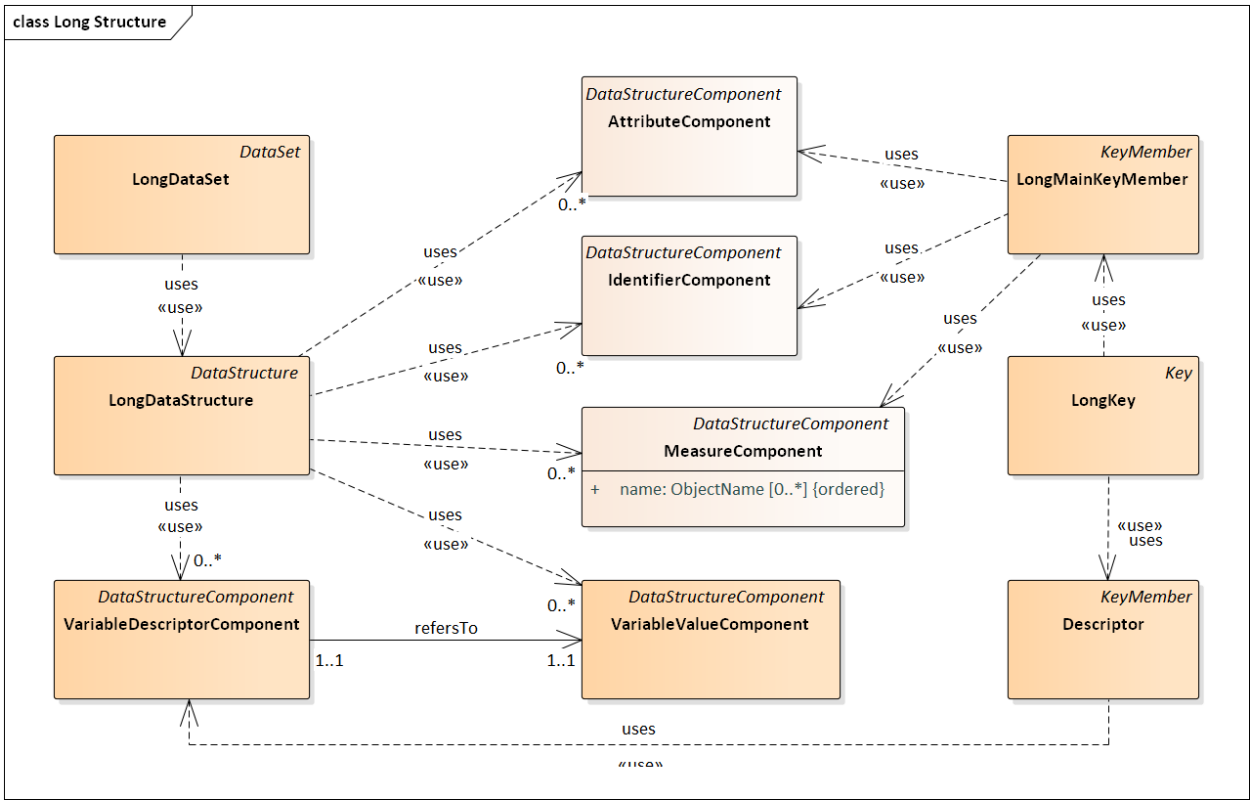


Figure 11: Long Structure – Overall Diagram

Figure 11 shows the different data structure components described above.

The diagram in Figure 11 conceals some of the complexity involving the association between the LongKey on the one hand, and the LongMainKeyMember and the Descriptor classes on the other. The LongKey is actually a composite of LongMainKeyMembers and Descriptors, each of which is based on one of the five component types. A LongKey could include, for example, two IdentifierComponents, such as variables “Household” and “personInHousehold”, assuming that both of these are present in a single record having two IdentifierComponents.

The Long layout brings out the utility of the Datum based approach and the use of keys to describe data. In the Long dataset example the values of the “Value” column are in a different conceptual domain in each row. A traditional (“wide”) variable having one conceptual and one value domain makes no sense for the column, as it fails to provide a sufficiently complete description of the values for many purposes.

If we again consider the table, the VariableRef column contains the VariableDescriptorComponent of the compound key describing the InstanceValue in each row of the value column. The column VariableRef itself is a DescriptorVariable (see Figure 12) that can be described as having codes that point to InstanceVariables. In the highlighted cell in that column “Born” is a code for an InstanceVariable that describes dates of birth. The other two columns are associated with InstanceVariables that could appear in a wide layout. CaseID contains id values each of which is an IdentifierComponent of the compound

key. Verified contains the AttributeComponent of the key. Together the compound key of “Marie”, “Born”, and “TRUE” provides context for the highlighted InstanceValue of “3.3.1992”. They allow it to associate it with the “3.3.1992” in the “Marie” row of the “Born” column of the wide example table above.

The VariableDescriptorComponent diagram below shows how the VariableDescriptorComponent relates to other components of the model.

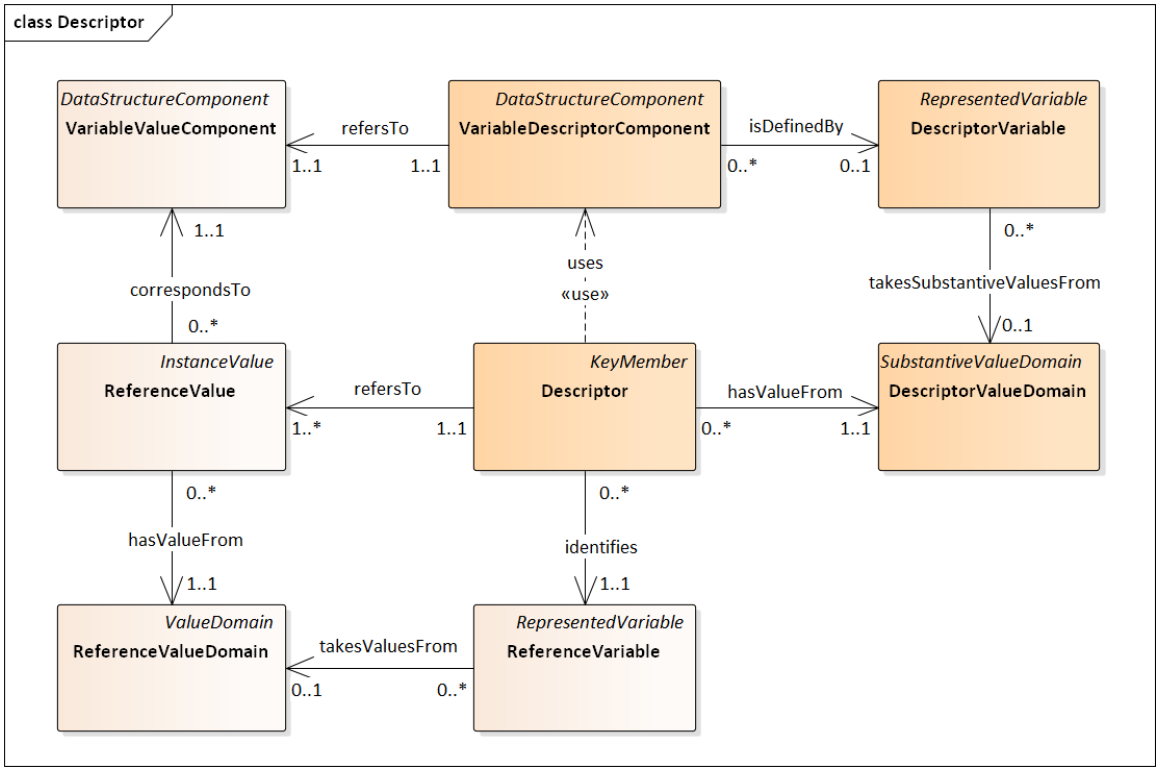


Figure 12: Descriptor Diagram

Each record in a LongStructure must have one component providing the value of the measure (the VariableValueComponent) and one which assigns that value to a variable (the VariableDescriptorComponent). You may not have multiple variables containing measures in a single record in a Long record, as the record itself implies the connection between these two when data is structured in this way.

It is very common for one or more AttributeComponents to contain time values (such as those which specify a point in time when the value was obtained, or for the start and end points of an event occurring over a period of time). The time values will frequently contribute to the identification of the measured or observed value, and must be included in the Key.

F. Multi-Dimensional Format

1. Example

Sometimes data are presented in dimensional form. In the example below (Figure 21) there are three dimensions: geographic, with Categories of Newport, Cardiff, Monmouthshire, and Merthyr Tydfil.; temporal, with categories like 2004-2006; and gender, with categories of Male and Female. The numeric values in the cells are often aggregates computed on some variable or combination of variables, in this case the mean of longevity. Cells might also contain direct measurements such as with data from an experiment with a factorial design. Dimensional data are commonly displayed in a dimensional table like a pivot table.

	2004 - 2006		2005 - 2007		2006 - 2008	
	Male	Female	Male	Female	Male	Female
Newport	76.7	80.7	77.1	80.9	77.0	81.5
Cardiff	78.7	83.3	78.6	83.7	78.7	83.4
Monmouthshire	76.6	81.3	76.5	81.5	76.6	81.7
Merthyr Tyfid	75.5	79.1	75.5	79.4	74.9	70.6

2. Discussion of Structure and Diagrams – Dimensional

A cube is a multi-dimensional array of cells (DataPoints). Values in the cells may be the result of an aggregate computation or a direct measurement.

At a logical level the structure of the cube is defined by a set of Dimensions (the DimensionalDataStructure in the diagram below).

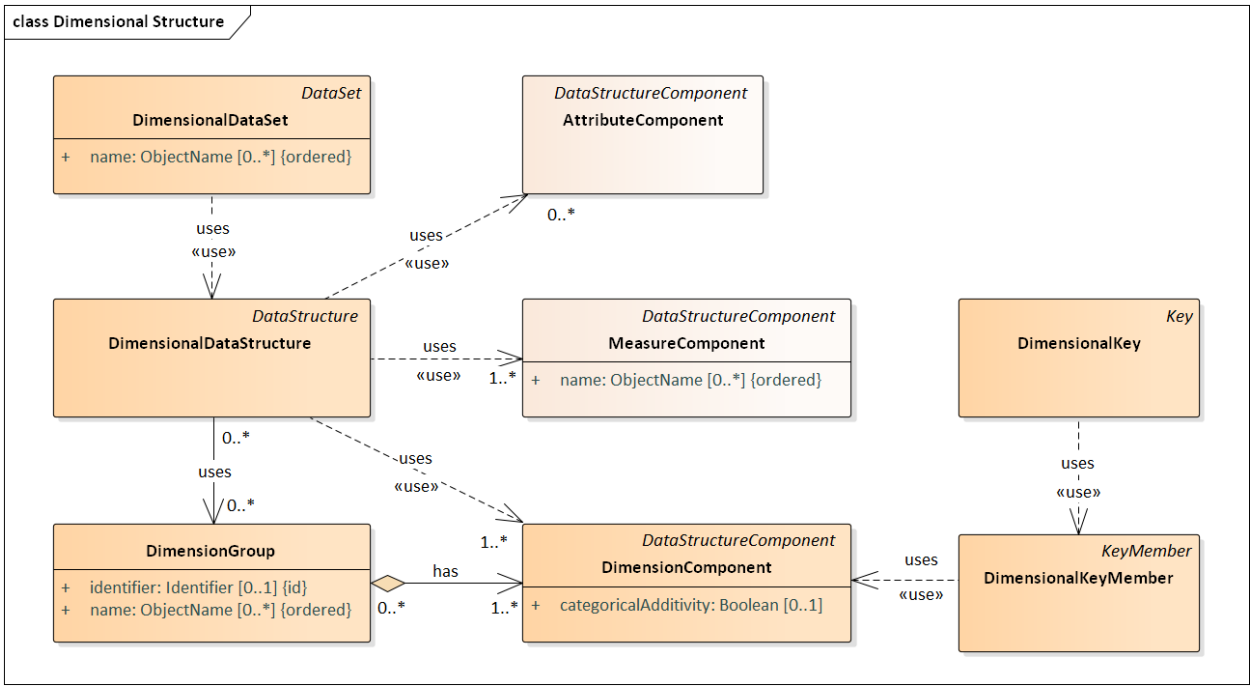


Figure 13: Dimensional Data Structure

Each dimension (DimensionComponent) is, in turn structured by a SubstantiveValueDomain and defined by a RepresentedVariable. The latter also brings along the specification of a Universe and a Concept. A Dimension can be categorical, for example ("Male", "Female"). In this case the SubstantiveValueDomain would consist of a Codelist. Typically cubes containing aggregate data would have primarily (or only) categorical dimensions. A DimensionComponent might also have a described value domain. Experimental data might, for instance, employ an independent variable measured as a real number (e.g. person's weight).

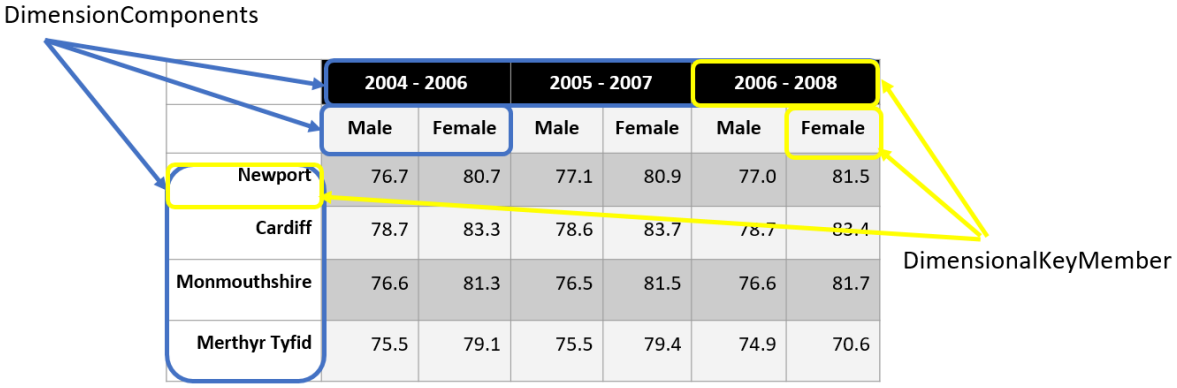
While there may be some underlying continuous variable for a Dimension, a Dimension may often be delineated by discrete dimensional categories. Time, for example, is a continuous measure. In our cube example, though, it has been transformed into a set of three-year categories like 2004-2006. This, along with the other two dimensions (gender, and geography), allows for the delineation of discrete cells in a table. Note that the time periods in this example (Figure 22) overlap.

The DimensionalComponents form the basis for keys. A DimensionalKey is a composite of one value from each SubstantiveValueDomain of a DimensionComponent. This composite DimensionalKey identifies the location of a DataPoint in the dimensional structure. Our example cube, for example, contains mean longevity data measured on people of Wales. The DataPoint (cell) identified by the key value (2006 – 2008, Female, Newport) is associated with that subset of people.

Partial Keys – in which only a subset of the DimensionalKeyMembers have values specified for them – can be used to refer to regions (or "slices") within the cube. (DDI - CDI does not explicitly model this; it is left to implementations to handle partial Keys if this is useful or required.)

Each DimensionalKeyMember (InstanceValue) of the DimensionalKey is also associated with the concept 'Male' in a ConceptualValueDomain. This would provide meaning for the DimensionalKeyMember in the case of an aggregate data set.

Categories within the Dimension may be additive or not. In our example the geographic areas could be combined to create larger areas. The year range categories could not be combined in a straightforward fashion given that they overlap.



In addition to structure a cube has content. The CubeDataStructure also includes a MeasureComponent and an AttributeComponent. The MeasureComponent is defined by a variable for that value.

A QualifiedMeasure as the measure for the whole cube (e.g. mean of longevity), while a ScopedMeasure is for each cell in a cube as its Population narrows the Universe of the Qualified measure

There might also be Attributes associated with each cell in a cube. One example of an attribute might indicate whether the measure for the cell was imputed.

The DDI - CDI model bundles a number of information elements into an Instance Variable. While a cube like our example may have a measure with a single concept, each cell in the cube has a different Population. The upper left cell in the example has a mean of longevity for Males in Newport in 2004-2006. The cell just to the right of it has mean of longevity for Females in Newport in 2004-2006. The DDI - CDI Dimensional model includes the notion of a ScopedMeasure for the InstanceVarriable for each cell in a cube and a QualifiedMeasure as the measure for the whole cube. The ScopedMeasure has a Population which narrows the Universe for the QualifiedMeasure.

The table below shows a long representation of a cube with three DimensionalComponents, one MeasureComponent, and two AttributeComponents. The attributes in this case indicate revised data in the cells of the cube, identified by vintage, and with an indication of what revision process took place.

DimensionalComponents *QualifiedMeasure* *AttributeComponents*

RefArea	Sex	TimePeriod	AverageLongevity	Vintage	Vintage Reason	Revision process
Newport	Female	2004 - 2006	80.7	**Aug-09**	additional data	recalculate mean
Newport	Female	2005 - 2007	80.9	**Aug-09**	additional data	recalculate mean
Newport	Female	2006 - 2007	81.5	**Aug-09**	no change	none
Newport	Male	2004 - 2006	76.7	**Aug-09**	additional data	recalculate mean
Newport	Male	2005 - 2007	77.1	**Aug-09**	additional data	recalculate mean
Newport	Male	2006 - 2007	77.0	**Aug-09**	additional data	recalculate mean
Cardiff	Male	2004 - 2006	78,7	**Aug-09**	additional data	recalculate mean
Etc.						

In the diagram below, we can see that multiple Datums can exist for those cases where there are revisions: these would share a Key but would be distinguished by the vintage property associated with each RevisableDatum.

While such revisions can be handled in other ways using this model (a time stamp associated with the observation – observation period - functioning as a dimension, for example) many systems use the approach modeled here, and do not manage revisions as part of the dimensionality of their data. The requirement is that two values with identical Keys be distinguishable – this model includes RevisableDatum to support those systems which require it.

G. Key-Value Format

1. Example

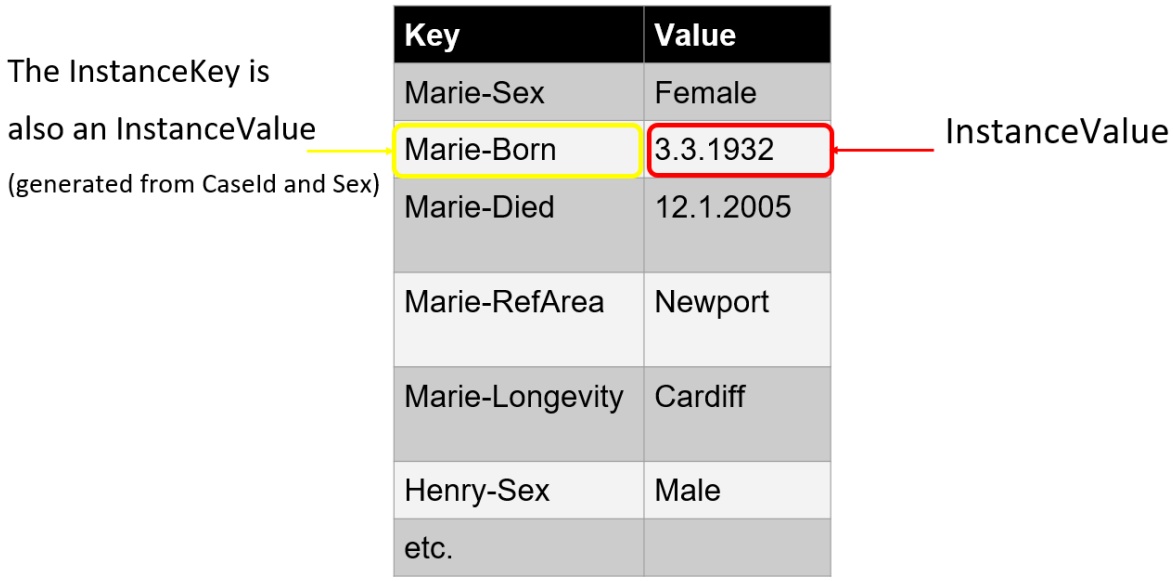
A Key-Value store represents a repository holding data as a set of pairs, a key – the InstanceKey - and its associated value, a DataPoint. The DDI - CDI model is shown in Figure 15. A key is a unique value that allows look-up of its linked value. The DDI - CDI model includes a KeyValueDataStore which contains the key-value pairs.

There are many possible ways to compose keys. The KeyValueDataStore may be divided into contexts, within which all of the subordinate keys are unique. The subject of the data – either a Unit or Population – can be contained as a component of the key. When this is a population, this portion of the key may itself be composed of the dimensional identifiers of the population, as for multi-dimensional data. Time may serve as a component of the key. Reference values may be used, as may variables. If needed, a "synthetic" component may be used, which holds no meaning but is unique within the context of the key.

In the example below the data are stored as key-value pairs. The Key column contains InstanceKey values that identify the associated DataPoints. Looking at the data in Figure 27, the value "3.3.1992" could be associated with a key "Marie-Born" combining the unit identifier ("Marie") and the variable

name (“Born”). The date 3.3.1932, for example is described by the InstanceKey “Marie-Born”. The cell containing 3.3.1932 is the DataPoint identified by the Key. This table, if combined with other data with keys composed in different ways, add a context – a Contextual component – to the key to distinguish between the different ways in which data are being composed within the repository.

The KeyValue structure can be used for data in data lakes, No SQL systems, and other forms of big data.



Key	Value
Marie-Sex	Female
Marie-Born	3.3.1932
Marie-Died	12.1.2005
Marie-RefArea	Newport
Marie-Longevity	Cardiff
Henry-Sex	Male
etc.	

2. Discussion of Structure and Diagrams – Key-Value

At its heart the Key-Value model is simple. A key identifies a value, and a set of these are held in a KeyValueDataStore. The key is represented in DDI - CDI as an InstanceKey, the value as a DataPoint. The structure of the KeyValueDataStore is known from the KeyValueStructure with which it is associated.

It is possible to have more than one scheme for the composition of keys, by including in each a component which represents that scheme – or “context” – within which the key is unique.

The diagram below gives an overview of the relevant classes in DDI - CDI:

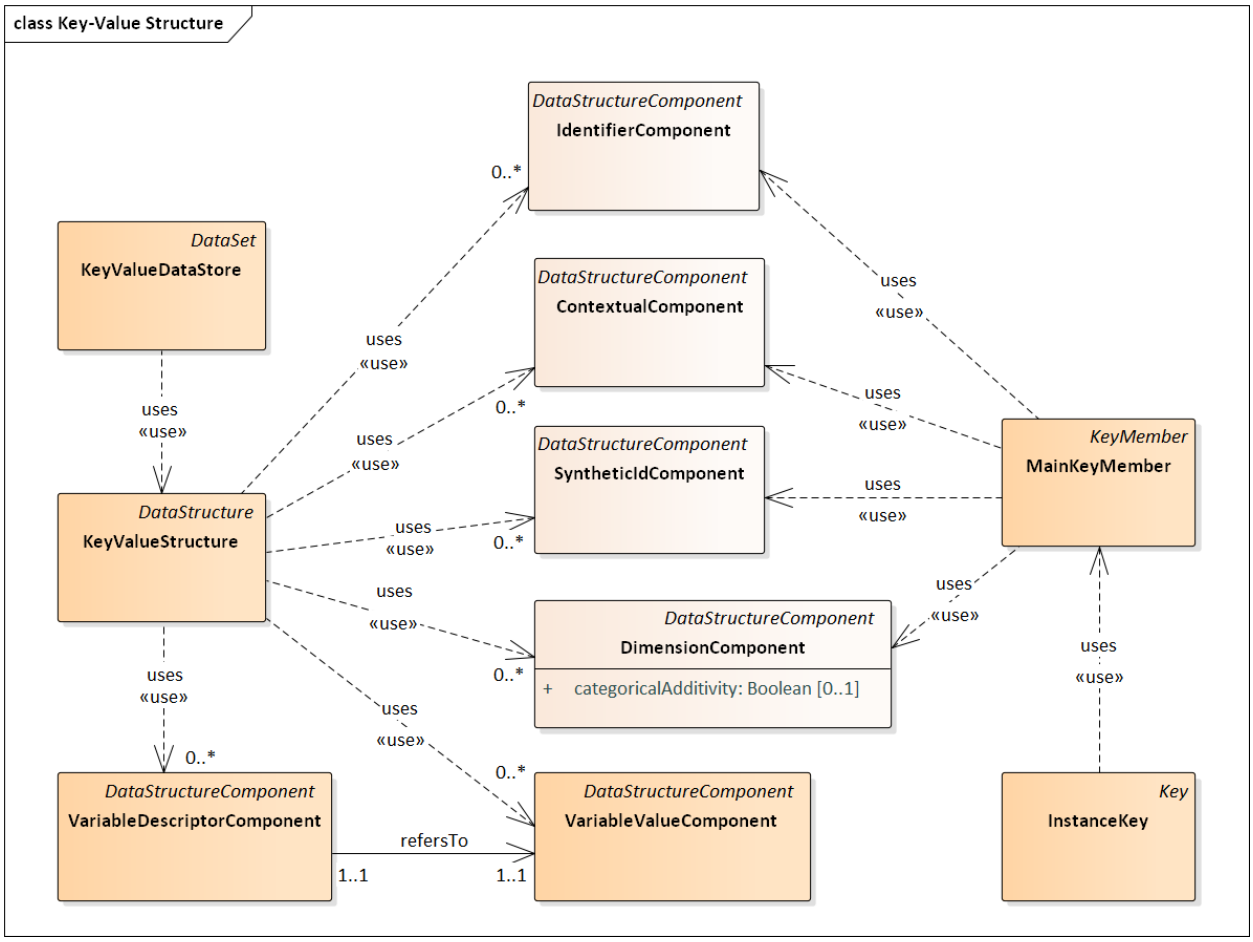


Figure 14: Key Value Overall Diagram

InstanceKeys may be composed of a variety of different members: MainKeyMember, TimeKeyMember, and Descriptor are all used. These members are in turn composed of different StructureComponents according to rules which guarantee their uniqueness.

The members which are used to compose an InstanceKey are shown in the diagram below:

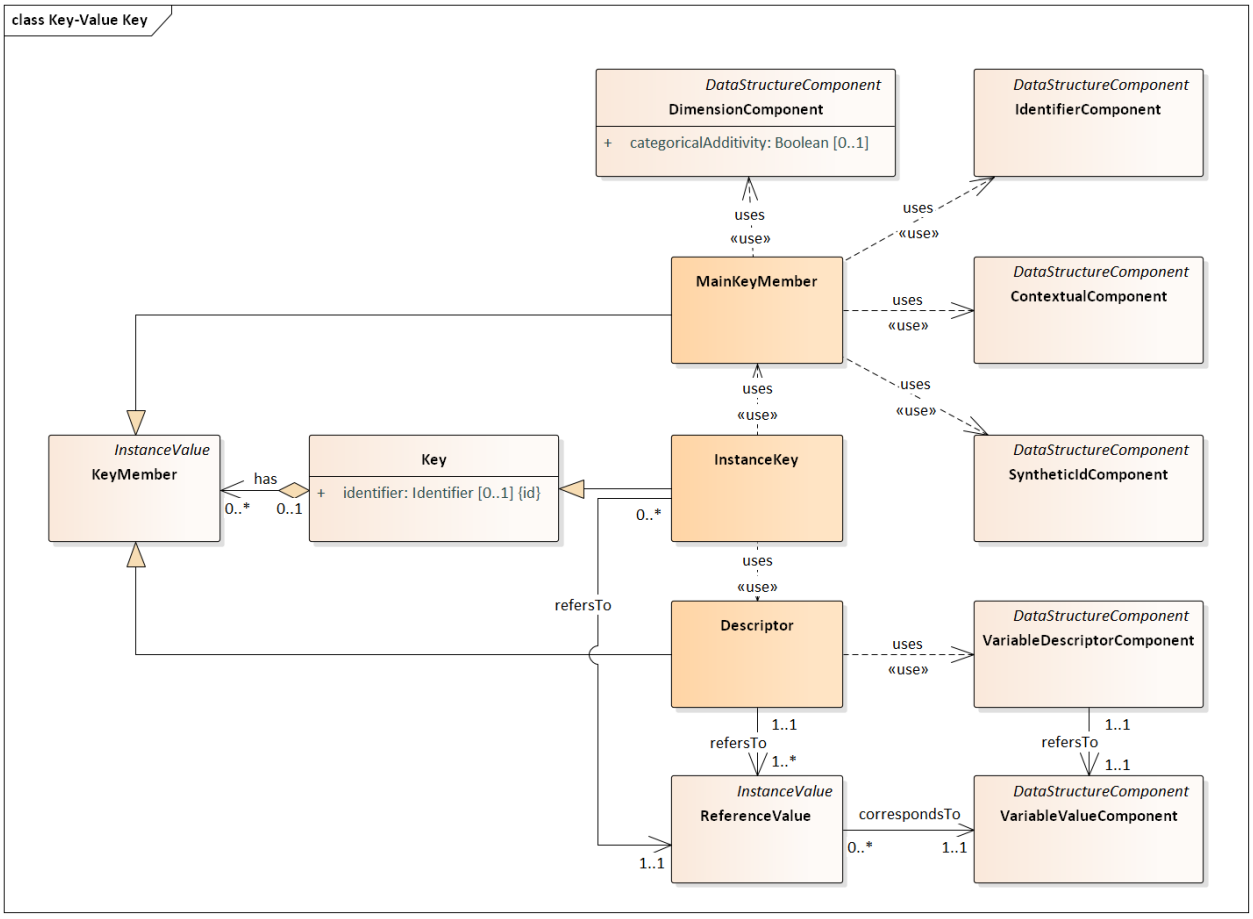


Figure 15: Key Value Instance Key

The MainKeyMember is the most complex one. In the simplest case, it may be composed of a SyntheticIdComponent, which might be a GUID or similar identifier which is guaranteed to be unique, but serves no other purpose. (The SyntheticComponent could be a primary key in a lookup table having other key components along with the SyntheticComponent.) IdentifierComponents may be used to provide unitary values which identify the Units of the value (that is, their subject). Similarly, Units and Populations may be identified using DimensionComponents, providing a compound key structure like that found for multi-dimensional data. If more than one approach to composing keys is used, each may be established as a "context", and this can be added to the keys using the ContextualComponent.

TimeKeyMembers are made up of TimeComponents, which may be anything with a temporal association (this can be an enumerated value such as "Valid", a timestamp, or any other time-related value.)

Descriptors use the VariableDescriptorComponent, which brings together AttributeComponents and MeasureComponents (as for the Long Data structure). Descriptors are associated with a ReferenceValue – that is, the value held as an instance of the component being used to compose the key. (In our

example, the variable “Born” could be a column in a Wide table, or a value in a Long table in the VariableDescriptor column. For Key-Value data, it is used as a Member in composing the Key.)

A Key has a structure consisting of all of these components.

H. Physical Data Set (Wide Format)

The PhysicalDataSet diagram below shows the relationship of the PhysicalDataSet to other classes. A PhysicalDataset is a set of record segments (PhysicalRecordSegments). In older data files it was common to have a record (a row of a table) that was represented as a sequence of shorter records (e.g. strings) due to constraints imposed by the physical media. A record, for example, of 150 characters required two 80 column cards. A property of the PhysicalDataSet signifies the number of segments per record.

The order of the PhysicalRecordSegments is specified by a set of PhysicalRecordSegmentPositions, each of which having an integer describing the position of the segment in the dataset.

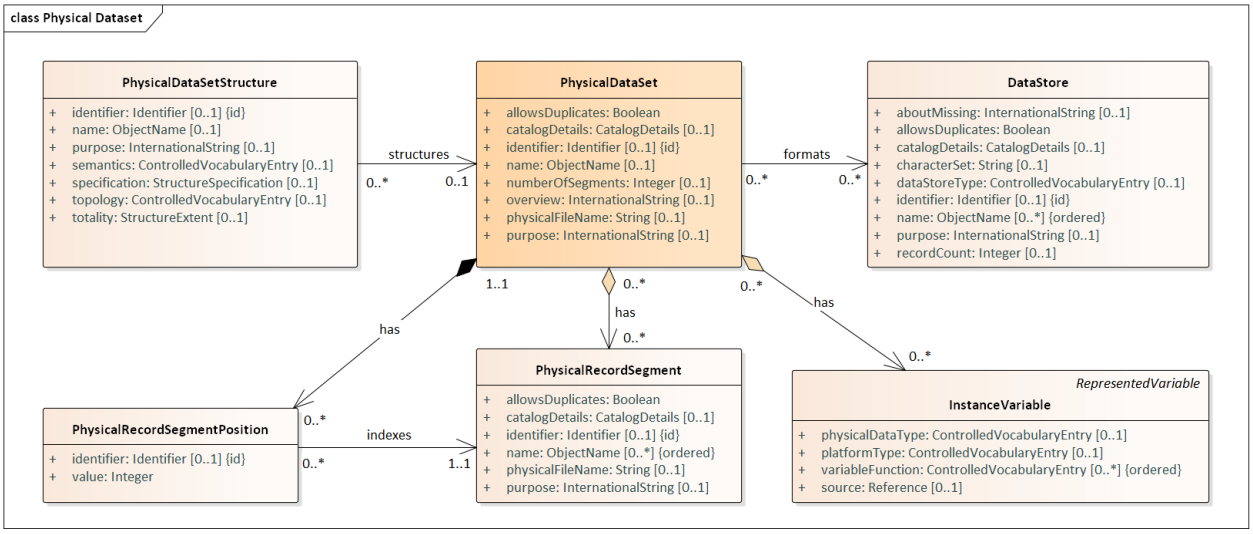


Figure 16: Physical Data Set - Overall Diagram

The PhysicalRecordSegment is composed of DataPoints. A DataPoint contains an InstanceValue. In a text file the InstanceValue would be a substring of the string comprising the PhysicalRecordSegment. In a binary file it would be a sequence of bits within a larger sequence of bits. A DataPoint is described conceptually by an InstanceVariable. It is identified and set into context by a Key. The example below, for a traditional rectangular table, uses a WideKey.

The DataPoint is also described by a ValueMapping. For a string representation this contains information like the separator used for the decimal part of a number (defaultDecimalSeparator), or the maximum length of the string (maximumLength), etc.

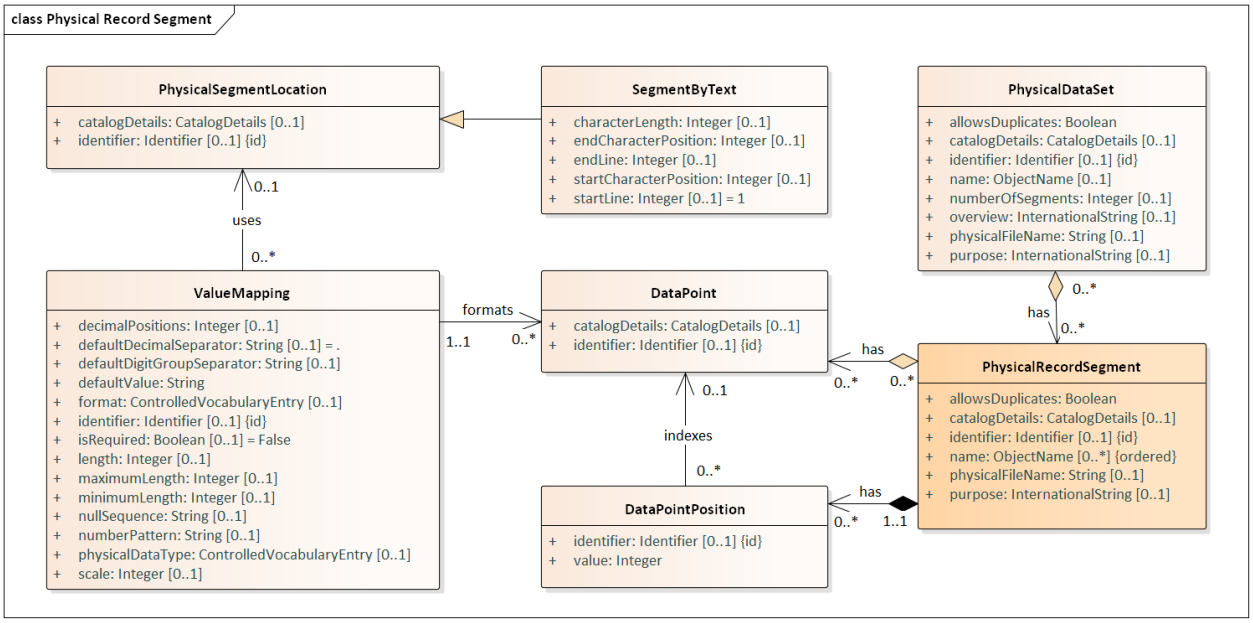


Figure 17: Physical Record Segment Diagram

In a text file the InstanceValue in a DataPoint is a substring of the PhysicalRecordSegment string itself. In a delimited file like a CSV file, the separation of those sequential substrings is indicated by delimiters. The PhysicalSegmentLayout contains information about those delimiters, the encoding of the record segment, whether text values are enclosed in quotes, etc.

For a fixed width file the ValueMapping can point to a SegmentByText object that contains information like the starting position (startCharacterPosition) and ending position (endCharacterPosition) of the substring within the segment. There is a parent class, PhysicalSegmentLocation, that will allow for description of data location in other types of media than text files. In a binary file this might be starting byte number and ending byte number. A video clip within a larger video file might be described by a start time and end time or by start and end frame number.

I. Relational Structures using Primary and Foreign Key

It is possible to describe a relational structure using the data structure descriptions in DDI - CDI, as a set of tables (that is, data structures) related through the use of foreign keys. Typically, such tables are in Wide or Long form. Figure 18 shows the relevant classes:

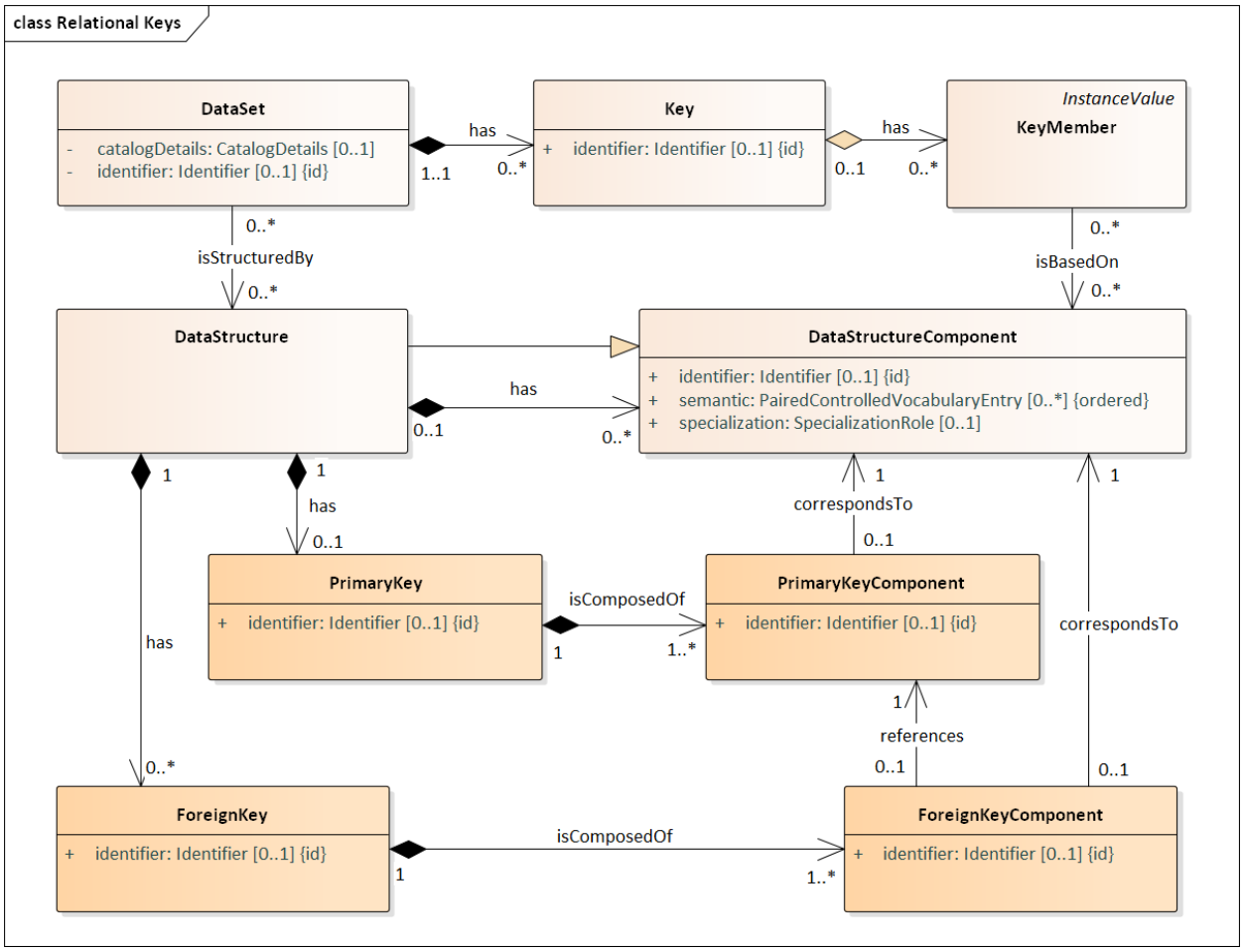


Figure 18: Relational Keys

DDI - CDI includes a construct to capture the notions of keys and foreign keys from the classical relational model. This construct provides the functionality of linking datasets via their associated data structures. The construct consists of a DataStructureKey class which is specialized into two sub-classes: PrimaryKey and ForeignKey. A DataStructureKey is composed of DataStructureComponents.

A DataStructure may have multiple DataStructureKeys, but only one of them can be a PrimaryKey while the others can be ForeignKeys. (No other type of relational keys is considered in this version.)

DDI - CDI also has the notion of Keys, as explained in VIII.C.2 Keys. Note that there is a constraint that needs to be satisfied: Keys in a DataSet should be consistent with the PrimaryKey. Essentially, that means that KeyMembers must be based only on DataStructureComponents that constitute a PrimaryKey of an associated DataStructure. This is in line with the relational model.

Once a DataStructure has a PrimaryKey it can be linked to from another DataStructure. This is done by means of a ForeignKey, which references an external PrimaryKey, i.e. a PrimaryKey of another DataStructure. This is also in line with the relational model, and provides a simple and yet powerful mechanism of linking data across DataSets.

J. Transformations between Formats/Examples

1. Wide and long: Correspondence between unit record data and data in a long format

The example below shows the mapping between the Wide Unit record format and the Long format. We see that all combinations of variables and values for each unit record identifier are retained. Each value in the record for Marie now has its own row, with a second value – the VariableRef – telling us what the value is (the column in the Wide table). The cell value is the InstanceValue.

Name	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8

CaseID	VariableRef	Value
Marie	Sex	Female
Marie	Born	3.3.1932
Marie	Died	12.1.2005
Marie	RefArea	Newport
Marie	Longevity	73.7
Henry	Born	8.1.1929
Henry	Died	6.2.2008
Etc.		

The VariableDescriptorComponent allows for the tracking of Datums between traditional wide layouts like the unit record format and Long layouts as shown in the Figure 34 example. All of the popular data analysis platforms have procedures like the R and Stata “reshape” function, or SAS PROC Transpose, that transform data tables back and forth between the two layouts. DDI - CDI provides a way to record this metadata which is not typically supported by non-proprietary formats.

Some types of data, like event data, typically employ Long layouts for the flexibility of adding measures and for the ability to represent sparse structures economically. Columns like “Value” in the Long layout example cannot be described as a traditional variable with a single value domain. They are instead a set of DataPoints having different conceptual domains and representations. For each DataPoint an associated Datum may populate DataPoints in other structures.

The ValueMapping attached to the DataPoint allows for description of the physical representation of the generic representations in the Value column. That column as a whole must have a common representation, like a text string or bit string, that is capable of representing all of the value types for the set of underlying InstanceVariable. (Note that some platforms, like Python Pandas, may allow multiple datatypes in a column.)

2. Wide and Dimensional: Unit Record Data Tabulated into an Aggregate Data Cube

Unit record data can be tabulated into cubes (aggregate/dimensional data). Data from the individual units contribute to the aggregates of a cube. We see that 'Mary', 'Henry' and the others contribute to the aggregate statistics of the cube. The appropriate Unit record datum is averaged, producing the datum for the cube cell. In the cube below Marie contributes to two different cells due to overlapping time periods, while Henry only contributes to one cell.

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8
Etc.					

RefArea	Sex	TimePeriod	AverageLongevity
Newport	Male	2004-2006	76.7
Newport	Male	2005-2007	77.1
Newport	Male	2006-2008	77.0
Newport	Female	2004-2006	80.7
Newport	Female	2005-2007	80.9
Newport	Female	2006-2008	81.5
Cardiff	Male	2004-2006	78.7
Cardiff	Male	2005-2007	78.6
Cardiff	Male	2006-2008	78.7
Etc.			

When computing a cube from the unit record data the value domains of some of the variables listed as measures above will correspond to dimensions of the cube. The categories of Sex, for example define the Sex dimension in the cube example. A computation on Died above would produce the time categories for the cube. The combination of dimension values for each unit (person here) would determine which set of units would contribute to the computation of the measure (Longevity here).

The SQL query that follows computes the cube data from the unit data:

```
create table WalesCube as
select Sex,RefArea,
    case
     when died >= '1jan2004'd and  died <= '31Dec2006'd   then "2004-2006"
     when died >= '1jan2005'd and  died <= '31Dec2007'd   then "2005-2007"
     when died >= '1jan2006'd and  died <= '31Dec2008'd   then "2006-2008"
    else " "
    end as TimePeriod,
    mean(Longevity) as Longevity

 from WalesUnitData
 group by sex, TimePeriod, RefArea
 ;
```

The processing code used to perform aggregations can be expressed in many different forms, both standard and proprietary. The Process model of DDI – CDI is designed to work with these to connect the metadata describing the data (both pre- and post-transformation) with the relevant processing code.

3. Long and Dimensional: Dimensional Data Represented in a Long Data Format

As noted before dimensional data can be represented in a Long layout. In this case the measure corresponds to the QualifiedMeasure in the model. Its population is the whole set of observations in the cube. There could be an extra column to represent the vintage instance for the associated measure. The DDI - CDI model includes classes that can assign roles to variables. In this example the first three variables take on the role as an IdentifierComponent. The values (codes), like “Newport”, or “2005-2007”in those columns are the representations of IdentifierComponent in the model. The longevity variable has a MeasureComponent, and the revision variable is an AttributeComponent. The values (codes) like “Newport”, or “2005-2007” in those columns are the representations of the IdentifierComponents in the model.

	IdentifierComponent		*QualifiedMeasure*	
Geography	**Gender**	**Time**	**Longevity**	**Vintage**
NewPort	Male	2004-2006	76.7	**Aug-09**
NewPort	Female	2004-2006	80.7	**Aug-09**
NewPort	Male	2005-2007	77.1	**Aug-09**
NewPort	Female	2005-2007	80.9	**Aug-09**
NewPort	Male	2006-2008	77	**Aug-09**
NewPort	Female	2006-2008	81.5	**Aug-09**
Cardiff	Male	2004-2006	78.7	**Aug-09**
Cardiff	Female	2004-2006	83.3	**Aug-09**
…	…	…	…	..
Merthyr	Male	2006-2008		**Jul-09**
Merthyr	Female	2006-2008		**Jul-09**

4. Key-Value and Wide: Key-Value Stores in microdata.no

The example below shows what a possible dataset based on the RAIRD information model might look like. (The RAIRD Information Model was developed for microdata.no, a project involving the compilation of data from a set of administrative registers in Norway into a resource which can be used securely through an online analysis package by researchers. The central compiled data store is similar to the example given here, but researchers perform analysis on Wide data sets derived from it. The data is a form of "event history" data, giving information about specific events and periods for the Units it describes.)

Microdata.no uses a hybrid form of Long and Wide layouts in that they add StartDate and EndDate as attributes that identify a value. In Figure 37 we recognize the crosswalk from the Wide Unit record data format to Long. StartDate and EndDate variables for each value are added additionally.

The KeyValue table expresses the collection of variables in a possible microdata.no data set and how they are ordered. Key values link roles to each of them.

PersonID	Sex	Born	Died	RefArea	Longevity
Marie	Female	3.3.1932	12.1.2005	Newport	73.7
Henry	Male	8.1.1929	6.2.2008	Cardiff	78.8

raird:keyValue

InstanceVariableID	IndexValue	Role
CaseID	1	Identifier
VariableReference	2	Identifier
Value	3	Measure
StartDate	4	Attribute
EndDate	5	Attribute

RAIRD DataSet

CaseID	VariableRef	Value	StartDate	EndDate
Marie	Sex	Female	3.3.1932	12.1.2005
Marie	Born	3.3.1932	3.3.1932	12.1.2005
Marie	Died	12.1.2005	12.1.2005	12.1.2005
Marie	RefArea	Newport	1.1.2003	12.1.2005
Marie	Longevity	73.7	12.1.2005	12.1.2005
Henry	Born	8.1.1929	8.1.1929	
Henry	Died	6.2.2008	6.2.2008	
Etc.				

Here, we see that both CaseID and VariableRef function as identifiers – taken together, they uniquely identify a record in the Long format, and indeed as the identifier for a specific measure (the Value).

5. Time series

With time as an attribute dimension in a full cube, a time series can be seen as a slice of the cube, holding the structural identifier values constant. In the example below geography and Gender are held constant and time varies across its possible values. The vintage column is added to indicate which revision of the data is being reported.

CellDefinition			*QualifiedMeasure*	
geography	**Gender**	**Time**	**longevity**	**vintage**
NewPort	Male	2004-2006	76.7	**Aug-09**
NewPort	Male	2005-2007	77.1	**Aug-09**
NewPort	Male	2006-2008	77	**Aug-09**

6. Key-Value Stores and Streams

Streaming data may involve a flexible set of measures arriving at unpredictable times. Structures that may be useful for streaming data include the long structure (like for event data) or a key value store. With a long structure, measure variables may be associated with identifier variables (such as a sensor identifier) and attribute variables (such as time of measurement, time of receipt, and location of measurement).

Measures may involve datatypes not currently described in DDI - CDI (images, sound recording, etc.) but envisioned as potential candidates for inclusion in future.

An example sensor observation from the W3C Semantic Sensor Network Ontology (SSN) (https://www.w3.org/TR/vocab-ssn/#iphone_barometer-sosa) is of a barometric pressure taken by an iPhone. The SSN RDF for the Observation is:

```
<Observation/346344> rdf:type sosa:Observation ;
  sosa:observedProperty <sensor/35-207306-844818-
0/BMP282/atmosphericPressure> ;
  sosa:hasFeatureOfInterest  <earthAtmosphere> ;
  sosa:madeBySensor <sensor/35-207306-844818-0/BMP282> ;
  sosa:hasSimpleResult "1021.45 hPa"^^cdt:ucum ;
  sosa:resultTime "2017-06-06T12:36:12Z"^^xsd:dateTime .
```

A long representation for the data might look like this, where the value atmosphericPressurehPa is a code that points to a variable that links to the Concept "`earthAtmosphere`" in units of hectoPascal (hPa).

SensorID	Property	Time	ResultingValue
sensor/35-207306-844818-0/BMP282	atmosphericPressurehPa	2017-06-06T12:36:12Z	1021.45

A Key-Value representation might look like this. The SensorID and Property are concatenated into a single Key. The Key could be decomposed into the SensorID and Property components as needed.

Key	Time	ResultingValue
sensor/35-207306-844818-0/BMP282/atmosphericPressure	2017-06-06T12:36:12Z	1021.45

IX. The Process Model

A. Introduction

The DDI - CDI Process model is a generic process model able to describe retrospectively a succession of activities. These activities may be a set of business processes described at a conceptual level and/or a set of concrete steps (and their sub-steps, ad infinitum) that take different information objects as parameters. This basic model can be applied at any level: a top-level Activity is a business function which covers the full scope of the process. The sub-Activities and Steps are specific subordinate parts of the process. This could be used to describe the entire production of a database, or something as specific as the succession of questions in a questionnaire. Parameters may include, data, structured metadata, and computer programs, or any other input which can be identified.

Although primarily intended for retrospective use, the model can also be used to describe *intended* process flows: prospective process. Additionally, it can be the basis for computational replication. The

model itself does not indicate its application in any given system: this should be made clear by implementers.

Several forms of "succession" can be described. They fall into two categories – deterministic and non-deterministic. Deterministic succession may be parallel or sequential. Non-deterministic succession may be temporally ordered using Allen's interval algebra. Alternatively, non-deterministic succession may be governed by inference engines that form the basis for rule-based systems.

Generally speaking, each type of succession is supported by a set of control constructs. Together the control constructs form a plan or program that orchestrates a workflow. Depending on the control constructs, there are a myriad of workflow patterns. It is possible within a single process to combine both forms of succession, using the ControlLogic to pass between them.

B. Process Model Conceptual Model Overview

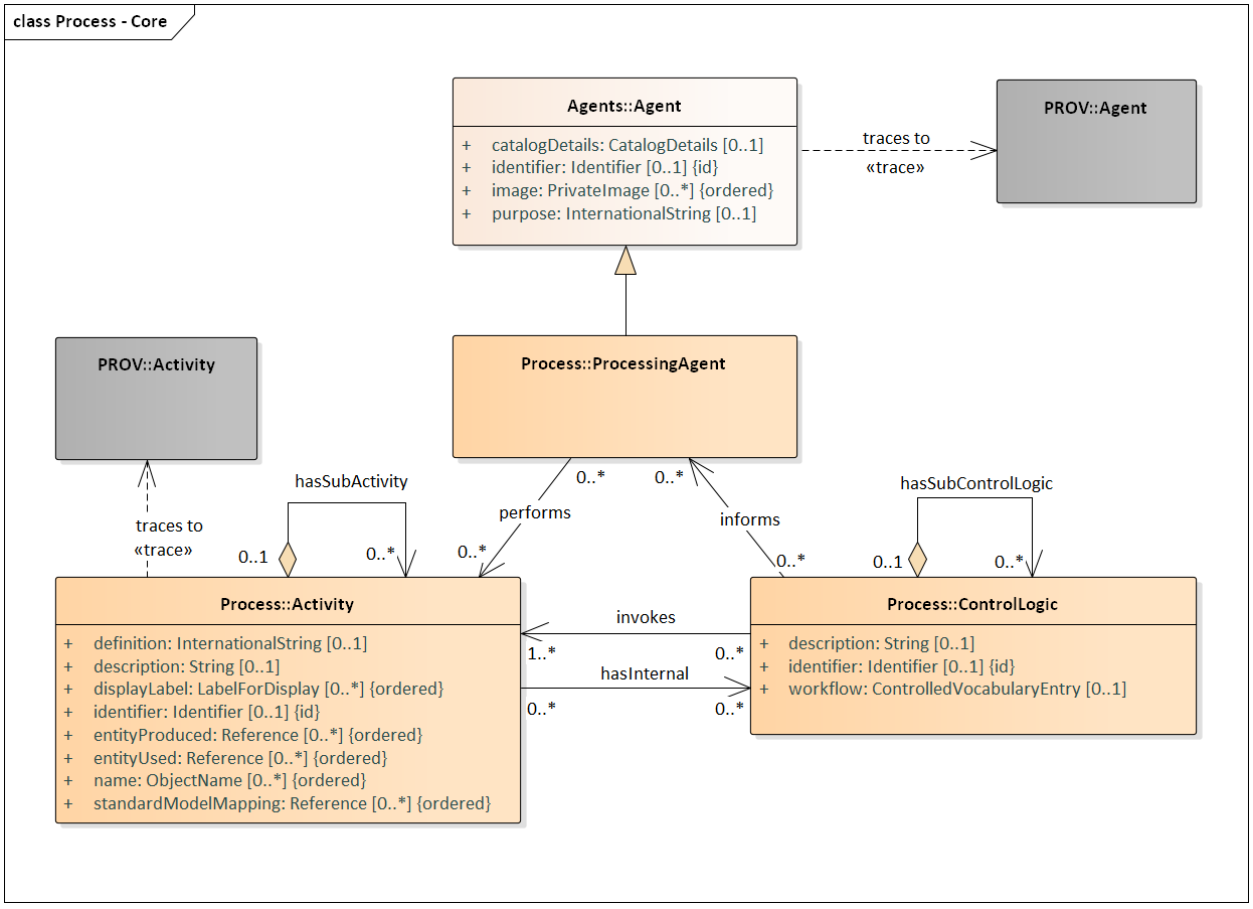
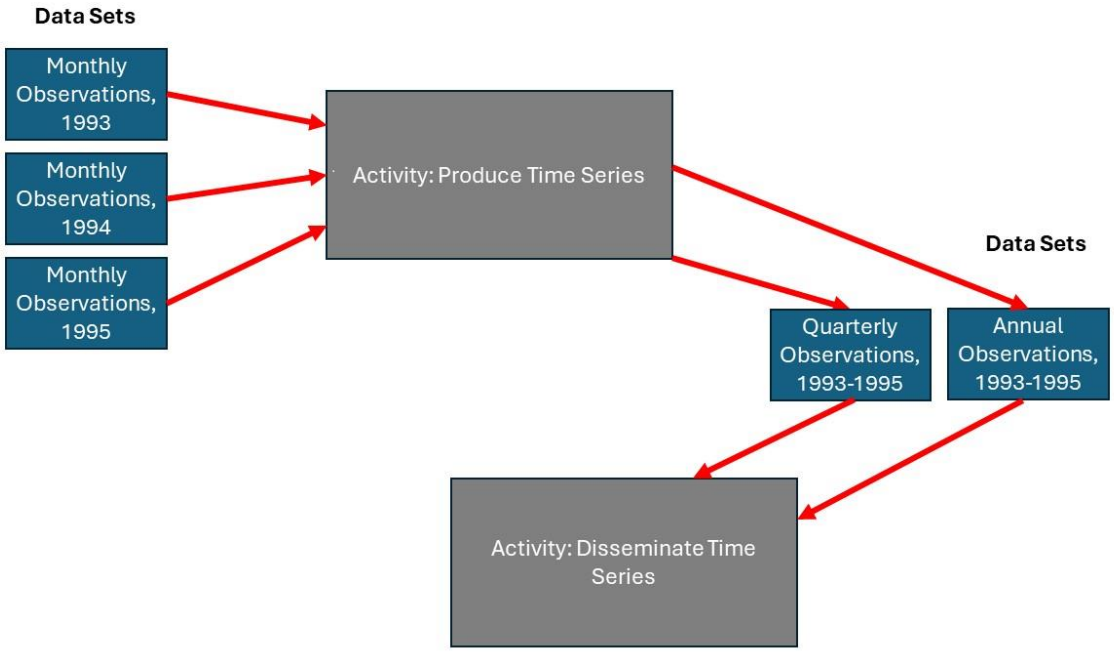


Figure 19: DDI - CDI Process Model Overview

In the DDI - CDI Process model processes are performed by ProcessingAgents, which can be machines or other types of actors. ProcessingAgents perform business functions – Activities – in accordance with the ControlLogic associated with that Activity. Activities have inputs and outputs, which are attributes

termed “entityUsed” and “entityProduced”, which correspond to the resources consumed and produced at the business level described by the Activity, using references to whatever resources are meaningful in the context of the implementation. The details of the specific inputs and outputs needed – the information objects used as Parameters (see Section IX. B. 1.) – are associated to the Activity’s ControlLogic through InformationFlowDefinition objects, which connect the outputs of one Activity to the inputs of another. This allows for the description of specific workflows within and between Activities, either at a very general level (using entityUsed and entityProduced) or at a more detailed level, as Parameters, in reference to resources described according to the DDI-CDI model.

To give an example, consider the workflow illustrated below:



Here, we have a business process which takes several data sets giving monthly measurements over the course of a specific year, and produces quarterly and annual times series from them, which are then disseminated. The “Produce Time Series” Activity will take the monthly observations and produce new data sets at quarterly and annual frequencies (“time series”) across several years. These new data sets would then be passed to another Activity which would disseminate them.

If our monthly data sets are proprietary files from a statistical package, and our quarterly and annual time series are to be produced in a standard Statistical Data and Metadata Exchange (SDMX) format, I could describe this workflow at the business level using entityUsed and entityProduced attributes on my Produce Time Series Activity, using file names or URLs for the monthly inputs, and SDMX identifiers for my quarterly and annual outputs. The subsequent Disseminate Time Series Activity would then point to the SDMX files which it uses as inputs, again using the SDMX identifiers.

If a description of the sequence of the two Activities was needed, I could create a higher-level Activity with the ones shown being sub-Activiities, and describe the sequence using an instance of the ControlLogic class. Other classes such as Sequence and SequencePosition exist to more fully describe relationships between sub-Activities.

This would permit the creation of a high-level description of the business process, which might be useful. If I wish to have more detail, to understand, for example, how the quarterly and annual time series are calculated from the monthly observations, then I would need to use other features of the DDI-CDI model: Steps, Parameters, and InformationFlowDefinitions. This more-detailed approach is described in the following section.

1. Steps and ControlLogic

Activities can comprise several Steps, which take specific input and output Parameters. The relationship to specific Paraneters is indicated with the receives and produces associations. Parameters are objects described according to the DDI-CDI model, and can be any data or metadata so described.

Steps can be made up of sub-Steps, allowing the Parameters to be at whatever level of granularity makes sense: you might wish to refer to an entire DataSet, or you might choose to reference particular Datums within that DataSet from a more-granular sub-Step. The level of granularity is not dictated by the DDI-CDI model, but by the implementation of it. Rules about how Parameters at the Step and sub-Step levels relate to each other (i.e., the Datums referenced by a sub-Step must be found within a DataSet referenced by a higher-level Step) are likewise left up to implementers to determine.

Connection of the Parameters between Steps, where the Parameter produced by one Step is received for use as an input to another Step, can be deterimed by examining the set of associations and objects described. In managing workflows, however, it is sometimes useful to have objects which identify these points of connection. For this, the InformationFlowDefinition class provides a link between the ControlLogic and its component Steps, and the to and from Parameters which are related through them.

The figure below shows this portion of the model.

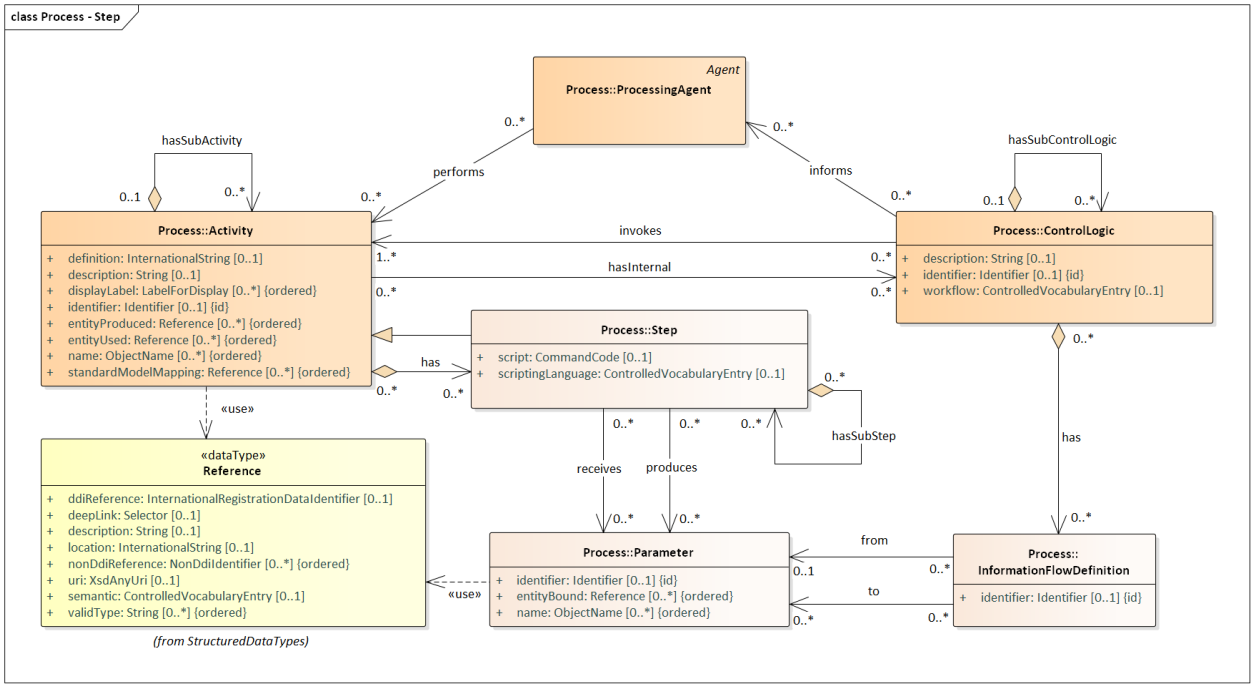


Figure 20: Process Model Steps and Activities

The flow of a process is governed by ControlLogic. Figure 21 shows how ControlLogic is associated with other significant classes for describing different aspects of process flow.

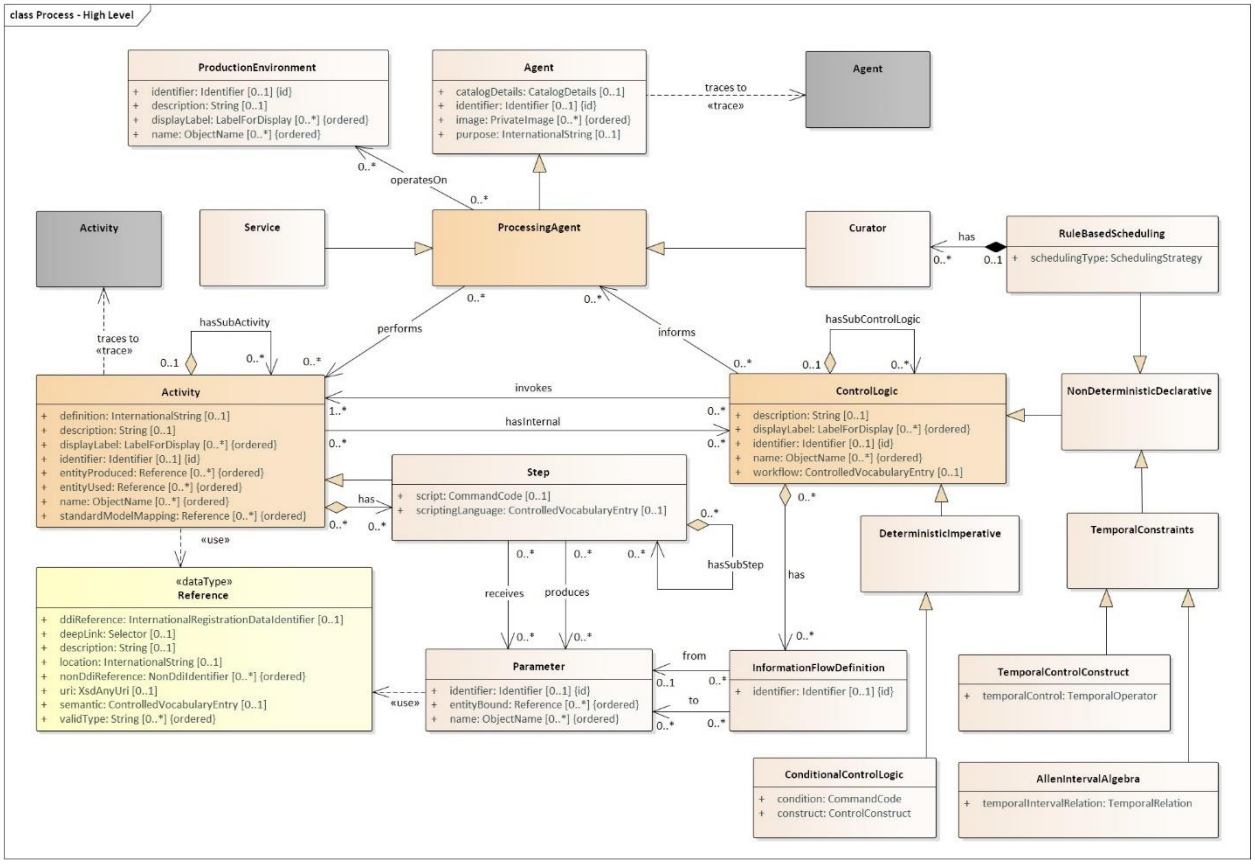


Figure 21: Detailed Process Model

Deterministic Control Logic consists of Sequences and Conditional Control Logic. Sequences may themselves contain Sequences and Conditional Control Logic. Conditional Control Logic comes in several types or flavors including *If Then*, *Else*, *Loop*, *Repeat Until*, and *Repeat While*. Conditional Control Logic also includes logical expressions (the Condition attribute) that evaluate to *true* or *false,* indicating whether executable code associated with a Step (Command Code) or subordinate Control Logic constructs are to be actioned. Command Code may be provided in whatever form implementers wish – for Steps, this is indicated in the Scripting Language attribute of the Step. Finally, Conditional Control Logic may contain Sequences.

For Non-Deterministic Control Logic, we cannot know exactly how the process will be organized in terms of sequential execution, so instead we describe the resources which are provided to the processing engine.

Non-Deterministic Control Logic has two subtypes – Temporal Constraints and Rule-Based Scheduling. Temporal Constraints in turn has two subtypes – Allen Interval Algebra and Temporal Control Construct. Both Allen Interval Algebra and Temporal Control Construct use enumerations to qualify their type further. Note that Allen Interval Algebra is a calculus for temporal reasoning useful in describing complex pairwise temporal relationships across a group of Activities. Temporal Control Constructs, on the other hand, are useful in describing parallel processing.

Rule-Based Scheduling takes a Rule Set and Information Objects as input and produces Information Objects as output. Rule-Based Scheduling may employ the assistance of one or more domain-specific Curators to match the Rule's conditions with real world facts or the current state of Information Objects.

Note that Control Logic is recursive, so that Non-Deterministic Declarative and Deterministic Imperative constructs can invoke each other, passing the processing off to other types of systems. These interactions need to be described according to each type of process, such that it invokes the other using a Step (for Deterministic Imperative) or is indicated in the Rule Set in an appropriate fashion for the Non-Deterministic Declarative Processing Agent being employed (etc.)

2. Relation to Other Standards

There are several models currently in use which provide a strong basis for the DDI - CDI Process model. PROV-O is perhaps the best-known of these, giving us a basic set of classes describing *Activities* (the things which are done), *Agents* (the people and organizations which do things), and *Entities* (the resources which are operated on/with and produced). This is an extremely general model, and one which was designed to be made more specific for use in specific applications.

A good example of this is ProvONE. PROV-O has been extended by ProvONE. ProvONE to make it data- and computer-program-specific. In PROV-O, entities didn't distinguish data at different level of specificity. 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:

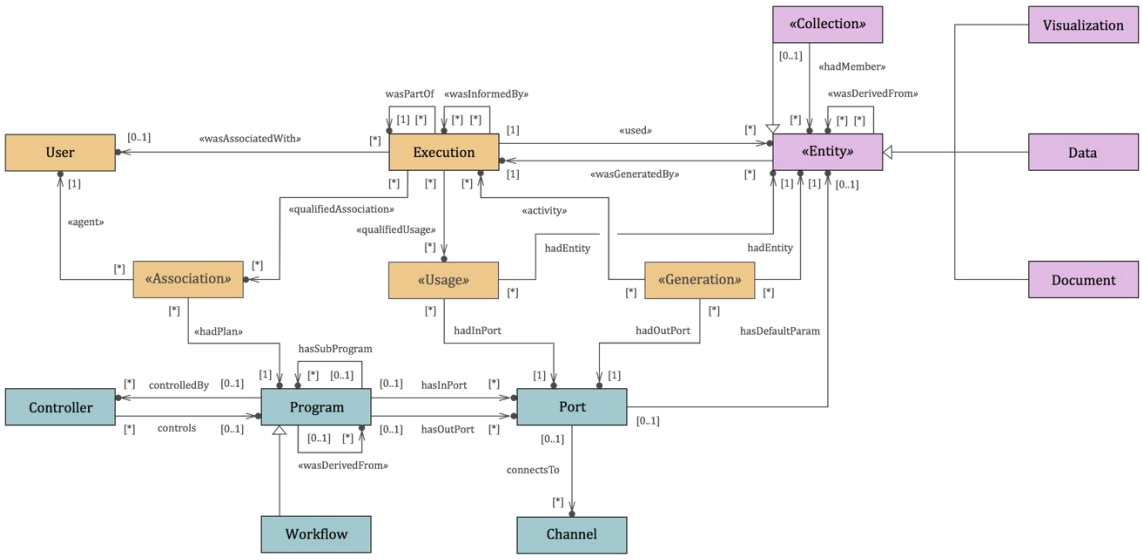


Figure 22: The ProvONE Conceptual Model

DDI - CDI process descriptions can be understood as extensions of PROV-O, and could be mapped to similar models such as ProvONE. Trace relationships to some of the key PROV classes are indicated in the DDI-CDI model.

These extensions mostly take the form of Control Constructs which DDI - CDI has borrowed from other products in the family of DDI specifications, notably DDI Lifecycle. DDI Lifecycle process components borrowed heavily from OWL-S. (Notably, the Control Construct is a central feature of how DDI Lifecycle describes questionnaire flows.)

3. Aspects covered by the DDI - CDI Process model

Currently “prospective provenance” and “process provenance” are not in scope. In prospective provenance plans and programs have a hand in guiding execution. Process provenance is about workflow evolution over time. Workflow evolution is integral to machine learning experiments which might evaluate a succession of workflows. (Workflow evolution may be addressed by DDI – CDI in future.)

For now, the focus is “retrospective provenance” or, again, “data lineage”. When data lineage enumerates a set of beginning and intermediate on-ramps in a workflow, it is *backward data lineage*. When data lineage enumerates a set of off ramps for InformationObjects that have entered the workflow upstream, this is *forward data lineage*. The DDI - CDI Process model aims to be able to describe both backward and forward data lineage.

4. Implementation Syntaxes for Command Code

In DDI-CDI, the process description relies on executable syntaxes for the automation of processes. In the Deterministic Imperative processes, this includes scripts or executables which are triggered through Command Code. There are two options for how such Command Code can be implemented.

Proprietary syntaxes for the executing program can be used, or standard expressions of the desired processing – requiring translation into platform-specific syntax – may be employed. In this latter category, there are two alternatives which deserve mention.

The first of these is the Statistical Data Transformation Language (SDTL). This was developed by the C2Metadata project, but is now a specification developed and maintained by the DDI Alliance. The second is the Validation and Transformation Language, created by SDMX.

Implementation choices are specific to application of the model, but should be clearly specified for a given community of users. It is assumed in the DDI-CDI model that an agreed syntax will be used for the conditions associated with Conditional Control Logic, as the process descriptions themselves will not be interoperable without such an agreement. Because Steps may be executed using a variety of different languages, even within a single system, the option to indicate what language is used is provided.

V. General Topics

A. Model Features

This section addresses some features of the model which are not specific to the description of data, but are pertinent to the style of the model itself.

1. The UML subset

DDI-CDI employs the UML Class Model Interoperable Subset (UCMIS), a subset of UML class diagram items, which is intended for data modeling. It focuses on core concepts that are familiar from object-oriented programming. The subset focuses on items that describe classes, describe their relationships to each other, and their attributes.

This subset supports the interoperability of a model, particularly in the form of Canonical XMI.

The UCMIS Git repository including a description of further details can be found at https://bitbucket.org/ddi-alliance/ucmis/.

UCMIS was developed to include a relatively small set of features that is well-supported by UML tools, and that leverages Canonical XMI a specific constrained format of XMI that minimizes variability and provides predictable identification and ordering (see Annex B of the XMI specification: https://www.omg.org/spec/XMI/2.5.1/PDF).

2. Design patterns

DDI-CDI employs a set of formal design patterns to help guide the consistency of the model generally, and as a useful resource for modelers. The design patterns are not intended to be directly implemented, but are connected to the classes in the DDI-CDI Library using <<Refine>> relationships. They function as "template" models.

Collections Pattern

The Collections Pattern describes groups of members, their organization, and their relationships. A Collection is made up of Members, which may be Individual Members (atomic ones) or sub-Collections. Members may have relationships (expressed with Member Relationships) and may be mapped to members in other Collections (expressed with Maps). Lists are a subtype of Collection, which may be ordered (using Position).

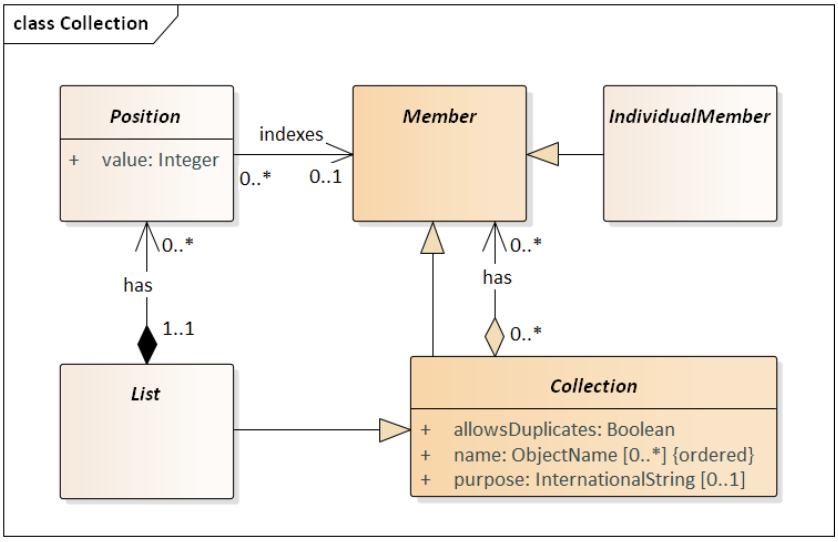


Figure 23: Collections

Various types of structural arrangements can be described (with Structure), including a directed graph of members and their relationships.

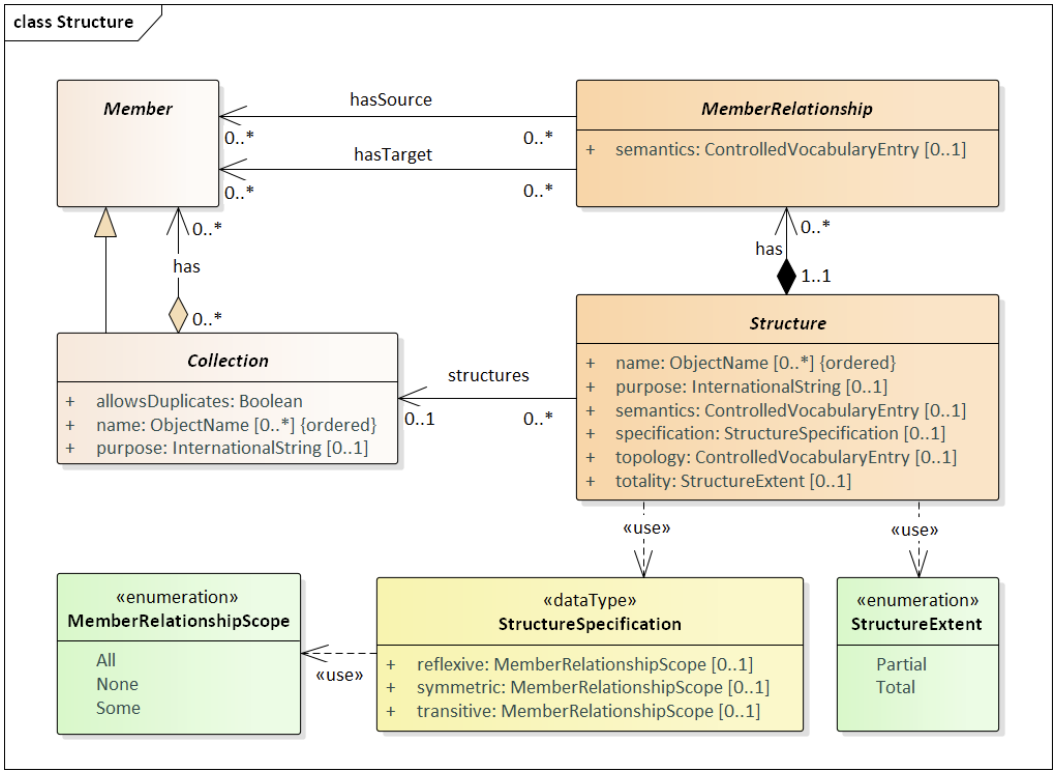


Figure 24: Structure

Data Description Pattern

The description of data in DDI-CDI uses a set of constructs for identifying Data Points. These are described using the Key as shown in the Data Description Pattern. A Key Definition is a Collection of the Concepts which serve to identify the Data Points in a Data Set. It is made up of Key Definition Members, which are a subclass of Conceptual Value. The Key members are the Instance Values found in the data which form the Key, and correspond to the Conceptual Values of the Key Definition members.

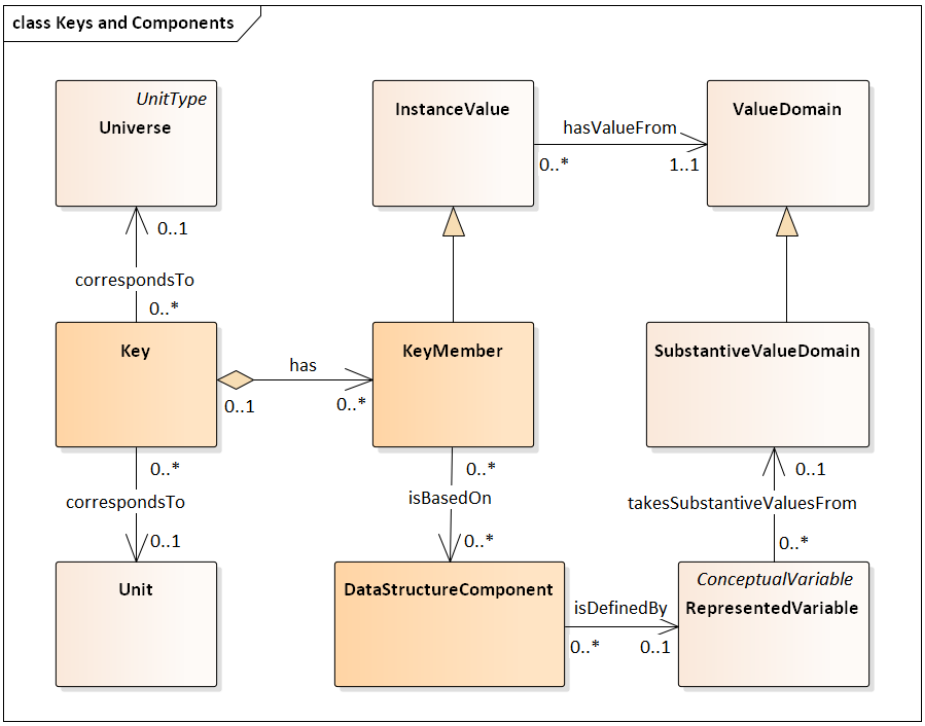


Figure 25: Keys and Components

Signification Pattern

The Signification Pattern describes how conceptual representations function in the DDI-CDI model. A Signifier – the perceivable object – is paired with a Signified – the concept or object of thought – by a Sign.

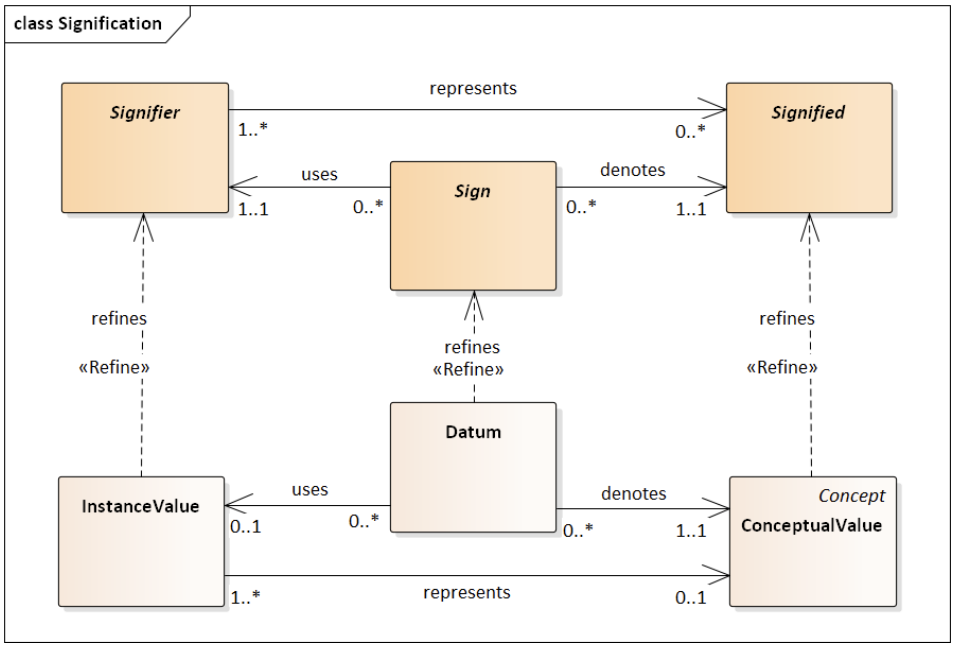


Figure 26: Signification

3. Trace relationships and other standards

The Context package of the DDI-CDI model indicates places where constructs from other specifications have been incorporated. This is done using the informal UML <<Trace>> relationship which indicates that the same concept is used in different models. It connects the DDI-CDI model to a set of other standards, including DCMI Metadata Terms (https://www.dublincore.org/specifications/dublin-core/dcmi-terms/), PROV (https://www.w3.org/TR/prov-overview/), the Resource Description Framework (RDF - https://www.w3.org/RDF/), and XML Schema (https://www.w3.org/XML/Schema).

There is no attempt to provide a full mapping to any other specification in this section of the model – places where DDI-CDI constructs intentionally correspond with those in other models or specifications have been highlighted, to assist users in understanding and implementing the model.

4. Cardinalities and Validation

The DDI-CDI model uses a subset of the overall features of UML, but does not use the aligned Object Constraint Language (OCL) for validation purposes (i.e., choices, conditional dependencies). This supports the interoperability of the model. Even the cardinalities used in the model may not be sufficient in some cases for validation purposes.

Such information is included in DDI-CDI in the form of UML Notes. Some cardinalities may seem too open, allowing for the absence of properties or relationships which is counter-intuitive. This is an intentional style used within DDI-CDI: often, such cardinalities are very specific to the implementation of the model within a particular system. Users should be aware that some fields which are not required may need to be required for their own use of the model, and that some of the constraints found in notes should be implemented in the application of DDI-CDI, as they are not described formally in DDI-CDI itself.

5. Inheritance

DDI-CDI is intended to be implemented in a variety of different syntaxes with differing expressive capabilities. For this reason, some typical aspects of object models are included in a consistent but perhaps not typical fashion. In DDI-CDI, abstract classes are found only in the Design Patterns, and not in the DDI-CDI Library. UCMIS does not use multiple inheritance, and as a consequence only single inheritance is found in the DDI-CDI model. This supports the interoperability of the model with syntax representations.

An example is the Selector structured datatype. The deepLink attribute of the Reference datatype has the datatype of Selector. Selector itself has no attributes. Instead there can be multiple types of Selectors (currently Text Position Selector and Object Attribute Selector. A deepLink should be able to use either one, each with its own attributes, for instance a Text Position Selector with start and end attributes.

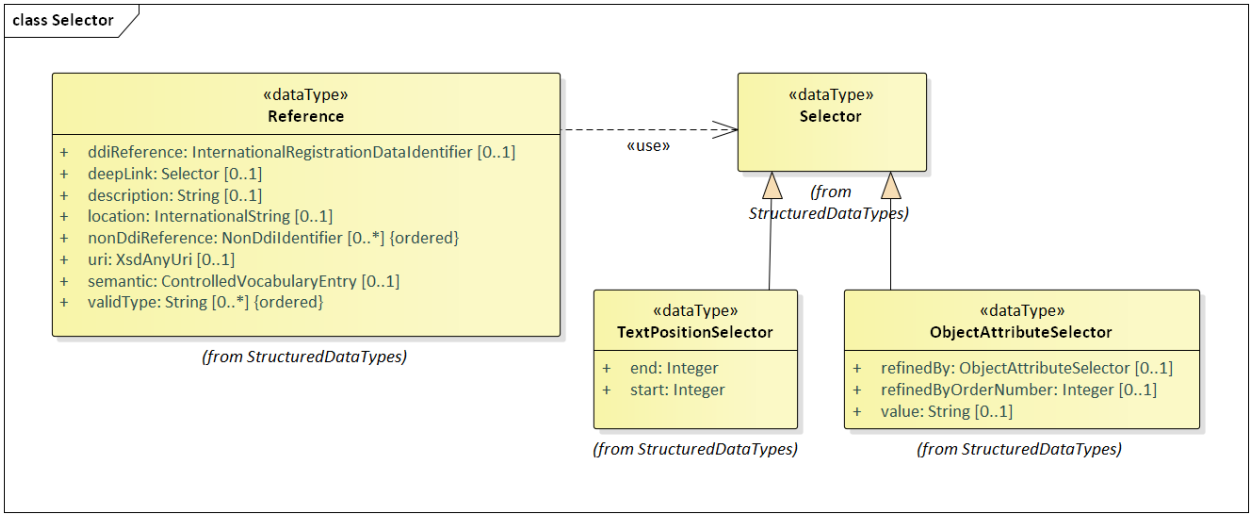


Figure 27: Selector Classes

B. Syntax Representation

A model like DDI-CDI cannot in itself guarantee system-level interoperability – it is intended to provide a basis for interoperable implementations. In recognition of this limitation, it is assumed that implementers of DDI-CDI will produce implementation guidance to specify which parts of the DDI-CDI model are being employed. Such implementation guidance should also include necessary information about controlled vocabularies, syntax representation, and so on.

DDI-CDI provides syntax representations for XML (expressed as XML Schema) and also has RDF representations described in OWL/Turtle and JSON-LD. These are intended to be reference syntax representations, and to assist implementers in their work: the DDI-CDI model can and will be implemented in different ways in different systems.

Appendix I: Theory of Terminology[3]

The theory of terminology for special language is described in ISO 704:2000 – *Terminology – Principles and methods*. This appendix is a reformulation of that standard to data and variables. The ideas of concept and object are used throughout.

Readers might find this overall section very philosophical. Even so, we attempt to make all the ideas accessible. We adopt a mentalist position for concepts and nothing more. This corresponds to experience. Likewise, objects are very generally defined, and they correspond to things in the world that people and systems use. We hope this approach allows the reader to maintain an intuitive understanding.

A. Objects

In our way of thinking, each thing, physical or not, is an object. A more technical definition is an **object** is anything perceivable or conceivable. What does this mean? We will describe what is meant by perceivable and conceivable in the next paragraphs. For now, accept that the idea of an object is our most generic one. Each thing is an object.

Perceivable objects are those detectable through one of the five human senses or some other (human-made) detector, such as a thermometer or a voltmeter. Any physical object in the world is perceivable, mostly through sight and touch, but the other senses may be used as well. For instance, a sound is perceivable through hearing. An object may also be perceived through some detector. Examples include voltage and current (in electricity), and they are perceived through instruments.

Objects may also be **conceivable**, and these come in two main kinds: abstract and imaginary. Examples of abstract conceivable objects are variables, laws, and numbers. Examples of imaginary conceivable objects are unicorns and hallucinations.

B. Properties and Characteristics

We make distinctions among objects by noticing features of them. We refer to these noticeable features as properties. It is through properties that we distinguish objects from one another. By determining their properties, we can make distinctions among objects. For instance, one person may be 185 cm tall, have brown colored eyes and hair, and have medium brown colored skin. Another may be 170 cm tall, have blue colored eyes and blond hair, and have very light brown colored skin. These properties of each person serve to help distinguish between the two people.

More technically then, a **property**[4] is the result of a determination either directly or indirectly about some object. One form of determination is through observation – something humans perceive through their senses. Noticing the color of a person's eyes is an observation or direct determination of the eye color of that person. Another form of determination is through detection by an instrument. An oral thermometer is an instrument that detects internal body temperature of a person. Observing a reading

[3] This Appendix is based on unpublished research by Frank Farance (President, Farance Inc) and Dan Gillman (Information Scientist, US Bureau of Labor Statistics).

[4] The term property is not defined in ISO 704.

on the thermometer is an indirect determination about the internal temperature of a person. The specific observed eye color and internal body temperature are properties of a person.

Since the examples above use perceivable objects, it is important to note that conceivable objects have properties, too. For instance, consider the rational numbers “three and fourteen hundredths” and “negative seventeen”. In the same way as with perceivable objects, properties of conceivable objects are the results of determinations about these objects. Here, the “sign” (in the mathematical sense) of the numbers is a property of them. The “sign” of 3.14 is positive, and the “sign” of -17 is negative.

When we notice features of an object, it is helpful and often necessary to know in advance what to look for. We can ask questions, often unconsciously, about objects, such as how tall a person is, the color of fur on a bear, the sign of a number, the definition of a variable, and so on. These questions can be asked of all objects in a set of similar objects: all people, all bears, all numbers, all variables. We use the term characteristic to refer to these questions.

So, a **characteristic** is an answerable question, capable of being determined, ascertained, or decided upon. Height, for instance, is an example, and applied to a person begs the question of how tall that person is. It is capable of being ascertained by measuring the distance from the bottom of the foot to the top of a person’s head. Properties are the answers to the questions posed by characteristics. Characteristics of a person are height, eye color, hair color, and skin tone. Example characteristic / property (question / answer) pairs, taken from the paragraph above, are height has the properties 185 cm and 170 cm; eye color has the properties brown and blue; hair color has the properties brown and blond; and skin tone has the properties medium brown and very light brown.

The set of possible properties is said to correspond to a characteristic. These properties (those in a set) form an extensional definition (See sub-section E on definitions below) of the characteristic they correspond to. In Examples 5 and 6, different sets of properties may correspond to the same characteristic, depending on needs. In addition, the same property may correspond to two characteristics. The following Example 1 illustrates this.

EXAMPLE 1: A property may correspond to two characteristics. Consider the following characteristics: height (of a person) and length of the diagonal (of a television screen). The property 60 inches (5 feet or 152.40 cm) corresponds to both characteristics. Some people are 60 inches tall and some large widescreen television sets measure 60 inches diagonally across the screen.

C. Concepts

A **concept** is a unit of thought, and we use characteristics to differentiate them. Consider the concept “person”. The characteristics of a person include being designed to stand upright on two legs, ability to talk, age, marital status, height, eye color, and skin tone. There are many others.

Some characteristics are indispensable for understanding a concept. These are the **essential characteristics**. An essential characteristic of persons is that they are designed to stand upright. A **delimiting characteristic** is a characteristic used to distinguish it from a generic concept. For example, being designed to stand and walk upright distinguishes people from other primates. The **intension** of a concept is the set of characteristics associated with the concept. The **extension** of a concept is the

totality of objects to which a concept corresponds. We will address what it means for objects to correspond to concepts a little later. Figure 29 addresses the issue pictorially.

Characteristics and properties are themselves concepts, and each kind plays a role. The role is how the ideas are distinguished. The role for a characteristic is the capability of being determined, and that for a property is the result of a determination. Some concepts can play each role in different situations. Consider the concept of ball. Balls have color, and some balls are red in color. In this case red is a property. But if we specialize the concept ball to red ball, then red takes on the role of a characteristic. For this concept, all the balls corresponding to it are red. It is no longer meaningful to wonder what the color of a red ball is. Instead, one needs to know if a ball is red colored. There are no other choices. This puts red on the question (characteristic) side, rather than the answer (property) side.

Example 2 illustrates the importance of establishing essential characteristics for a concept. In particular, the addition of a single characteristic may have profound influences on the objects in the extension of the concept. Adding or removing characteristics often affects the meaning of a given concept, changing the concept itself. Thus, the extension would be expected to change.

EXAMPLE 2:

The concept of planet was revised in 2006 by the International Astronomical Union. This revision resulted in the elimination of Pluto as one of the planets in the solar system. Pluto was long considered the ninth planet in the solar system, but some astronomers questioned this classification. Several properties Pluto possesses differ markedly from those of the other planets. Additionally, recent advances in astronomy - much better telescopes and vastly improved computation - showed there are many more celestial bodies that could be considered planets if Pluto remained one. Therefore, a concerted effort was made to define “planet” in a more limiting way.

The concept of a planet is now defined by these four essential characteristics: A planet is a celestial body that

1 Is in orbit around a star
2 Contains sufficient mass to maintain a nearly spherical shape due to its own gravity
3 Is not massive enough to cause thermonuclear fusion in its core
4 Has “cleared the neighborhood”, i.e., become gravitationally dominant, so the only other bodies in its vicinity are its satellites

This fourth characteristic is what eliminated Pluto.

A **general concept** is a concept which corresponds to an indeterminate number of objects which form a group by reason of common properties. An example is the concept “planets in our solar system”. An **individual concept** is a concept which corresponds to only one object. An example is the concept “Saturn”. In other words, a general concept may have any number of objects in its extension, and an individual concept must have exactly one object in its extension.

Note, a concept might be so defined that there exists only one object in its extension even though the possibility for more exists. This is still a general concept. For example, the notion "all planets with one moon" is a general concept. Even though there is one known planet with one moon – Earth – the possibility there are more cannot be ruled out.

The following Figure 28 shows the relationships between concepts and characteristics on the one hand and objects and properties on the other. The figure illustrates the correspondence between a concept and all the objects in its extension. The parallels between this construction and how data are obtained through surveys, experiments, clinical settings, and other kinds of observations is clear. This parallel will be discussed in Appendix II.

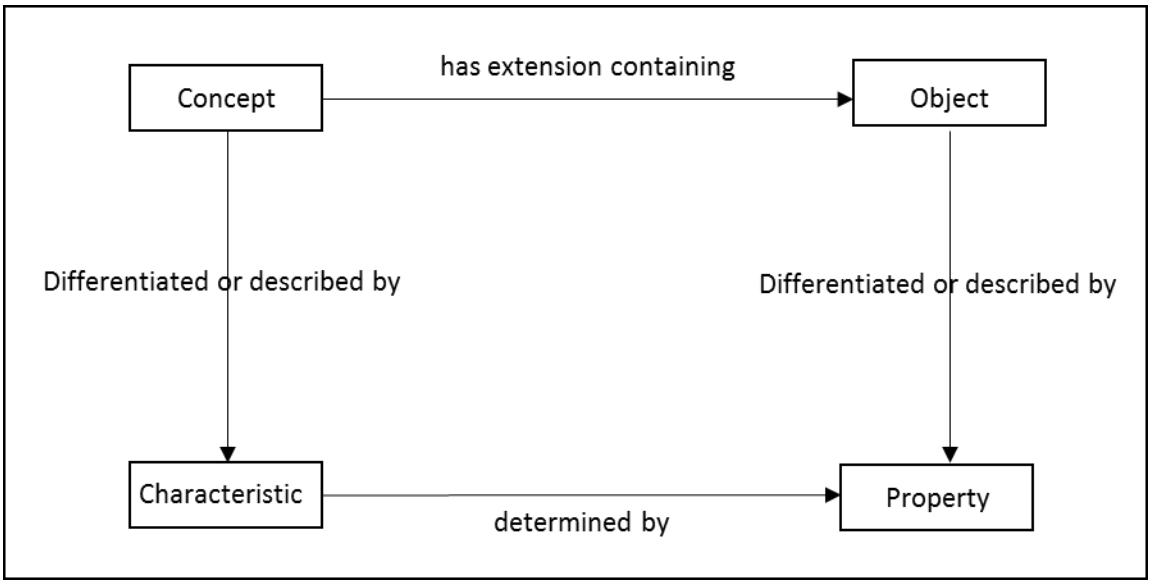


Figure 28: Concept - Object Correspondence

D. Signifier, Signified, and Sign[5]

A signifier is what is written down in place of, to denote, an object. We refer to objects through **signifiers**, and a **signified** is the object we refer to. For instance, the numerals **5** and **6** are often used to denote the numbers five and six. Here, the numerals are signifiers, and the numbers are signifieds. Note here that numbers are concepts, but we are also saying they are objects. This is true in general – concepts are conceivable objects.

It is possible for one signifier to denote several objects (homographs), and it is possible for more than one signifier to denote a single object (synonyms). When a signifier denotes a signified, we refer to this association as a **sign**. See Figure 29 for a pictorial explanation of signs, consistent with the wording in this section.

[5] This is outside the scope of ISO 704.

The following is a list of kinds of signs for which the signified is an object:

- A **label** is a linguistic sign for an object
- A **name** is a non-linguistic sign for an object, where the signifier is an alphanumeric string
- An **identifier** is a label or name intended to be used for dereferencing
- A **locator** is an identifier with a known dereferencing mechanism

The following is a list of kinds of signs for which the signified is a concept:

- A **designation** is a sign for a concept
- A **code** is a non-linguistic designation
- An **appellation** is a linguistic designation for an individual concept
- A **term** is a linguistic designation for a general concept
- A **numeral** is a code for a number, where a number is a concept

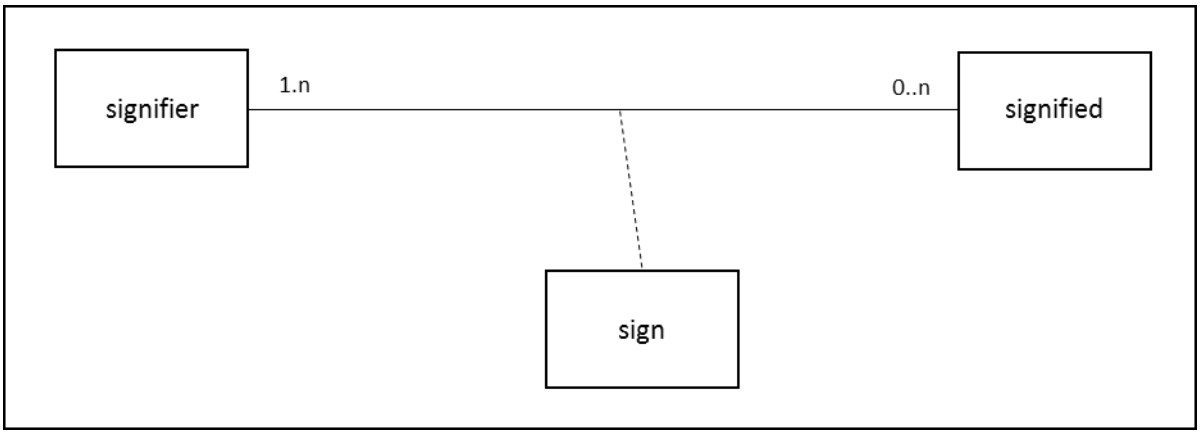


Figure 29: Structure of Signs

E. Definitions

A **definition** is a descriptive statement which serves to convey the meaning for a concept, and it differentiates it from related concepts. There are 2 kinds of definitions. An **intensional definition** is a definition that describes the intension of a concept by stating the superordinate concept and the delimiting characteristics. An example of an intensional definition is the one just above for defining the term intensional definition. An **extensional definition** is a definition of a concept formed by enumerating its subordinate concepts under one criterion of subdivision. An example of an extensional definition is to define a planet in our solar system as Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, or Neptune. Note, both kinds of definitions depend on knowing the definitions of other concepts to fully understand the concept under study.

Appendix II: Datums and Variables

A. Introduction

Here in Appendix II, we explain data and variables from the perspective of the terminology theory we laid out in Appendix I. We use the terms defined in Appendix I to create a framework for data and variables. Note, we will define data using the singular, datum. When referring to more than one datum, we use datums. This Appendix provides a thorough understanding of data and variables as used in DDI-CDI.

The underlying theory for understanding data and variables is the same as that for concepts and terms. We start by defining a datum as a kind of designation and develop the connections from there. The connection between concepts and their corresponding objects is precisely what variables do.

Data are often described by what they do, the operations and statistics available to process them. And from the point of view of a collection of data, this might be all one can say. The terminological approach is an attempt to define what a datum is.

Variables are described in DDI-CDI through levels of specificity. This is known as the variable cascade, and it enhances reuse of metadata, an important principle of metadata management. How the cascade ties back into the terminological view of data and variables is also described.

B. Data

This section contains a description of the connection between data and terminology. A datum is defined as a kind of designation. The idea of a designation is defined in Appendix I.D (Signifier, Signified, and Sign).

A Datum is a designation of a value, where a **value** is a concept with a notion of equality defined.

A fundamental requirement for a datum is that it can be copied. Whether a system managing data uses a computer or is based solely on paper and pencil, data are copied from one storage medium to another regularly. For instance, in computers data are moved from disk to internal memory to internal registers and back during the execution of a process. Copies need to be made faithfully, and the only way to ensure this is to compare a copy with its original. If a copy is equal to its original, the copying process is faithful.

The ability to copy faithfully is dependent on establishing equality between an original and a copy. In a paper and pencil system, this is done by visual comparison. On a computer, this is accomplished as the result of the engineering behind the design of a machine. Faithful copying is inherent to the successful operation of a computer.

Each time a copy of a datum is made, it is rather trivial to compare the signifiers. But the associated concepts need to be the same too. Any concept may have an equality operation defined for it. For a set of values, the same equality operation is sometimes defined for the entire set, and this leads to the construction of datatypes. See ISO/IEC 11404 – *General purpose datatypes*. Assigning an equality operation to a concept implies that if, say, two people say they have that concept, a determination of

equality between them can be made. For example, two people agree they have the same gender. This operation may be different depending on the situation. In fact, more than one measure of equality can be defined for any given concept. See Example 3.

> EXAMPLE 3: Consider the natural number "seventeen". It is a concept, and its extension is all situations of 17 objects. Equality may be defined as it is commonly understood for natural numbers. Another way to define equality for natural numbers, including "seventeen", is to ask if the number is even or odd. In this situation, all odd numbers are equal, and all even numbers are equal.

Example 4 illustrates values associated with social conventions.

> EXAMPLE 4: M for married, as in some person is married. Married is a value since marriage is a social and legal status controlled by the state. Equality may be determined by referencing the meaning in common law.

A datum is often generated in some context, and this context is what connects it to Figure 29 and to the connection between concepts and objects. Suppose we consider the object Donald Trump, and we determine he has orange hair. Donald Trump is an object, and we can find a concept for which he is in its extension. We know, for instance, he was president of the United States, so he is in the extension of the concept of presidents of the US. This concept has characteristics, and one of them is hair color (of a president). For argument's sake, suppose all the possible hair colors of presidents are Brown, Gray, and Orange. Thus, each president (each object in the extension of the concept presidents of the US) has one of the possible hair colors. Washington's hair color was Gray, and Obama's is Brown. In each case, the appropriate one must be determined. So, the possible hair colors are properties corresponding to the characteristic hair color.

Now, assuming that the hair colors Brown, Gray, and Orange are all the ones possible, every president is assigned one and only one color. Hair color divides the extension of presidents of the US into subsets that are disjoint, i.e., each president is assigned to one and only one of the subsets – each president has one hair color. The subsets are defined by the properties. In this case, there are 3 of them: Brown, Gray, and Orange. As stated above, no president belongs to more than one subset, and every president belongs to at least one.

When we determine the hair color of a president, we might want to record that, so we assign signifiers to each of the possible properties: for instance, b for Brown, g for Gray, and o for Orange, and through this assignment we create designations called codes. Again, by observation, we have a way to decide if two presidents have the same hair color, and this is based on light reflectivity and color reception in the judge's eyes. So, there is an equality operation for each of these properties. This means each of the properties is a value, each code is a designation, and when we assign a hair color and write down a signifier representing the property, a datum is produced.

Examples 5, 6, and 7 illustrate the same ideas presented above on hair color, this time using marital status and winning probabilities.

EXAMPLE 5: Subdividing people based on marital status
Concept = people of the UK
Characteristic = marital status
Properties = {single, married, divorced, widowed}, where "single" means never married and the rest correspond to their usual meanings. The signifiers S, M, D, and W designate these concepts, respectively.

EXAMPLE 6: Second example of subdividing people based on marital status
Concept = people of the UK
Characteristic = marital status
Properties = {single, married}, where "single" means not married and married takes its usual meaning. The signs S and M designate these concepts, respectively. The purpose of the example is to show that more than one set of properties may apply to a characteristic of a concept.

EXAMPLE 7: Subdividing gambling casino games based on probability of winning
Concept = gambling casino games
Characteristic = probability of winning
Properties = $\{x \mid 0 < x \leq 1\}$ (the set of all numbers, x, such that x is greater than zero and less than or equal to one), where x is a probability. The signs are the numeric strings that designate the numbers, to some agreed upon precision, fixing the lengths of the strings. Here it is possible that each possible probability has no more than one game assigned to it. Some may have none.

C. Variables and Aggregates

Variables and aggregates are characteristics in the sense previously described. Variables are mostly characteristics for general concepts, and aggregates are mostly characteristics of individual ones. This corresponds to the notion that a variable is a mapping between some collection of units (the extension of the general concept for which the variable is a characteristic) to a set of values (properties). An aggregate does the same, except the concept is an individual one, so there is one unit – the aggregate.

There are some exceptions. In socio-economic statistics, a household income is the sum (an aggregation) of the incomes of each of the individuals in that household. This aggregate applies to a general concept (i.e., households).

Table 1 shows how the terminological constructs described correspond to common notions about data found in socio-economic data.

Socio-Economic Data	Terminology
Unit Type or Universe	Concept
Microdata	General concept
Macrodata	Individual concept
Frame	Extension
Variable or aggregate	Characteristic
Unit	Object (in the extension of the concept)
Observation (or estimation)	Property

Table 1: Socio-Economic Data versus Terminology

D. Variable Cascade

In DDI-CDI, the variable cascade is the way the descriptions of variables are organized. The main purpose of the cascade is to increase the reuse of metadata. The features defined at each level of the cascade don't depend on features at any of the lower levels. Because of this, the descriptions at each level are reusable.

The cascade consists of four levels, each level corresponding to an ever-increasing descriptive detail. The levels in the cascade are

- Concept
- Conceptual Variable
- Represented Variable
- Instance Variable

The names of the levels indicate to the user what the focus of the description is at each. The Concept and Conceptual Variable provide details about the concepts employed. The Represented Variable and Instance Variable provide the details about the codes, characters, and numbers representing the concepts at the higher levels.

We will describe these levels and show how they fit into the terminological approach in the following sections. In tables in each section, we illustrate the approach with two examples. The attributes are taken from the class diagram of DDI-CDI. We only illustrate the attributes at each level. The inherited ones from the level above are assumed. See section VII-D for a description of the variable cascade from the point of view of the DDI-CDI model.

1. Concept

The variables about some subject share that subject as common among them all. For example, all variables in use in data sets in a research library about marital status share that concept among them all. There may be little in common about the marital status as measured in each variable, but marital status

itself – the fact there are statuses across societies or cultures – is a common characteristic. The concept expressing this commonality is the purpose of this highest level.

The concept at this level is very generic because it must account for all possible variations of the more specialized versions attached to each variable that makes use of it.

Concept

ID	*Name*	*Definition*
1	Marital status	Category of current marital arrangement
2	Age	Whole number of years of operation

2. Conceptual Variable

The Conceptual Variable is the level at which most of the concepts used to describe a variable are applied. The main concepts are the characteristic, which defines a variable, and the properties corresponding to the characteristic. In our marital status example, the main concepts are:

- Characteristic: marital status
  - The specialized nature of this concept is that it is applied to people living in the US (for instance)
- Properties: kinds of marital status
  - Single
  - Married
  - Divorced
  - Widowed

This example emphasizes that at the conceptual variable, the characteristic and properties are concepts. Suitable properties form an extensional definition for the characteristic. In our case, single, married, divorced, and widowed form an extensional definition for marital status. The properties are known as substantive values in DDI-CDI.

Additional concepts are those associated with missing data. These are known as sentinel values. The two most important ones that the statistical packages use are “missing” and “refused”. There might be others, depending on processing needs.

Conceptual Variable (Links to Concept)

Name	Marital Status	Age
Concept	Concept #1	Concept #2
Unit type	Person	Business establishment
Substantive Conceptual Domain	Single Married Divorced Widowed	Count (of years), top coded at 25
Sentinel Conceptual Domain	Missing Refused	Missing Refused

In this table, the names of the categories for marital status in the substantive conceptual domain are there in place of actual concepts. The only way to write down a concept is either through providing a definition or providing an unambiguous term or word denoting it.

3. Represented Variable

The main addition at the Represented Variable level is the signifiers for the properties or substantive values. Assigning signifiers to concepts turns them into designations. So, in our example, we might end up with the following designations:

- <s, single>
- <m, married>
- <d, divorced>
- <w, widowed>

The set of these designations is a substantive value domain. As discussed, the underlying concepts form an extensional definition for the characteristic, the concept associated with the variable. So, these values (properties) are associated with the subject matter of the variable, not with processing. A substantive value domain can be used by many Represented Variables, so it is important to identify and manage them.

Represented Variable (Inherits from Conceptual Variable)

Name	Marital Status	Age
Universe	Deer Hunters	Gun Shops
Substantive Value Domain	<s, Single> <m, Married> <d, Divorced> <w, Widowed>	Count (of years) represented with 2-digit Arabic numeral
Unit of Measure	N/A	years
Intended Datatype	Nominal	Quantitative discrete

The Intended Datatype Family attribute above needs some explanation. The interpretation of datatypes in this document is contained in the international standard *ISO/IEC 11404 – General purpose datatypes.* See https://standards.iso.org/ittf/PubliclyAvailableStandards/c039479_ISO_IEC_11404_2007(E).zip for a freely available copy of the standard.

We can interpret the use of the idea datatype in two ways. The physical (or application) datatype tells us how data are stored on some medium. In some data store, the byte containing the bits 00110010 can be interpreted in two ways. With lowest order bit on the right, this byte either represents the number 50 or the numeral 2 in the ASCII character set[6]. A datatype of integer tells us to interpret the byte as the number 50. As a character, we interpret it as the numeral 2. Therefore, the physical datatype is often just an approximation of what is needed to describe values. Instead, it corresponds to how the values are written in a file. The actual use of the values depends more on the Intended Datatype at the Represented Variable level.

For the reader of data in some application, there is a secondary issue. How do we use the data we read? Let's consider the FIPS[7] State Codes, an encoding of the states in the US. The codes are numeric, and there are 56 entries, including the 50 states; Washington, DC; and other areas. The state of Vermont has code 50. These could be stored as integers or numeric strings. In our example above, we show 50 stored as an integer. But, either way, the interpretation of the data is that they represent categories. Codes have no arithmetic properties. The intended datatype shows the user how to interpret the data. In this case the intended datatype is nominal (categorical data with no order).

Nominal, Ordinal, Interval, Ratio, Quantitative, Qualitative, Discrete, and Continuous are names of intended datatypes typically used in the statistics.

[6] For this character data, we use little-endian to interpret these bits.

[7] Federal Information Processing Standards in the US.

4. Instance Variable

Moving further down the chain to data, we get to the Instance Variable. An Instance Variable is intended to be a variable used in a data set. For each data set, new Instance Variables are created.

The main addition in specificity is turning the sentinel categories into designations. Further, the list of sentinel values (designations) is managed in one set, the sentinel value domain. Separating the substantive and sentinel value domains eases the burden on metadata management. Changes needed in one kind of value domain do not affect the other.

An example of the designations in a sentinel value domain is:

- <m, missing>
- <r, refused>

Since, the Instance Variable is associated with data in a data set, then the physical datatype of the data for that variable is necessary information as well.

Instance Variable (Inherits from Represented Variable)

Name	Marital Status	Age1
Population	US Deer Hunters in 2019	US Gun Shops in 2019
Sentinel Value Domain	<1, Missing> <2, Refused>	<1, Missing> <2, Refused>
Function	demographic	establishment
Physical Datatype	1-character	2-integer

The codes used to designate the sentinel categories are often provided by each statistical package. This topic will not be addressed in detail here.

The Physical Datatype addresses the kind of data as written on a file. The value $2.60 (two dollars and sixty cents) is often written as a real number with 2 decimal places. But monetary amounts don't follow all the rules for real numbers. The amounts at the third decimal place or after are truncated. The values are not rounded, as real numbers will be. This influences computations, as the following example illustrates:

Take the average of $1.50, $1.30, and $1.00. The arithmetic average is $1.2666. The rounded real number average is $1.27, and the monetary, or scaled number, rounded average is $1.26. The reason is the fractional penny is dropped in the scaled situation. And the rules for scaled numbers correspond to how banks handle money.