DDI 4 and Data Structures

Executive Summary

While there were many use cases that have contributed to the development of DDI 4, perhaps data management and, through it, the evolution of data beyond data collection has been the most compelling one.

In this connection a number of capabilities or, again, features were introduced in DDI 4 that grow data description beyond what is possible in DDI 3.x.

The purpose of this document is to bring these features together in one place in a format that can be readily reviewed by the DDI community and its leadership. To this end the document starts with the Summary Table below. This is followed by the body. In the body the reader will find a short introduction and a series of examples that illustrate the features called out by the summary.

The Summary Table

Feature	DDI 3.3	DDI 4	Comments
Support for NCubes	☑	•	The DDI 4 prototype does not yet support a dimensional data structure. Something like a DimensionalViewpoint should be completed soon though.
Explicit support for BPMN and ETL platforms	•	☑	DDI 4 introduces the DataManagementView. It strings together business processes in a data pipeline in line with the GSBPM and GLBPM. It passes records from a data store between the business processes. The pipeline need not be linear: it can have multiple entry and exit points.
In line with BPMN and ETL support, support for transformation acts based on standard rules languages like VTL (SDMX) and SDTL as opposed to ComputationActions commonly used in statistical packages like SPSS, Stata, SAS and R	•	☑	DDI 4 introduces a MetadataDrivenAction alongside the DDI 3.x ComputationAction which can be used to describe rules-driven data transformations
Represents other record types besides the UnitDataRecord and NCubes	•	☑	Leveraging the Collection Pattern, DDI 4 supports hierarchical data structures like the C struct and JSON objects.
Introduces new types of relationships between variables in support of 21st century data analysis	•	☑	Also, using the Collection Pattern DDI 4 represents network relationships between variables and their concepts in support of qualitative data analysis, concept maps, and causal data analysis
Represents health data structures that interleave single and multiple measures together with their contexts in line with openEHR[1], HL7 and CIMI "archetypes"	•	☑	In DDI 4 we can use the InstanceVariableRelationStructure collection to weave measures and attributes together in complex hierarchical and network-like data structures

[1] EHR = Electronic Health Record

Supports both legacy and advanced forms of event representation	•	☑	Legacy event representation refers to star schemas. Here events are facts with foreign keys that link the facts to entities with which the facts are associated. DDI 3.x uses a RecordRelation construct to establish relationships between facts and their entity contexts. In advanced forms of event representation events and their context are maintained in business objects specific to each type of event. There are no tables and no joins. Instead events come ready to use out of object data stores. See the example in the Event data section below.
Big data support	•	☑	Workflow in DDI 3.x is "retail". With the addition of business processes in DDI 4, workflow supports "wholesale" transformations like the transformation from a "wide" table to a "skinny" one and vice versa. That's because in a DDI 4 business process inputs and outputs are records, not variables.
More natural support for object-oriented metadata structures.	•	☑	The class oriented development of the DDI 4 model offers a more direct fit to object-oriented representations in languages like Java, and even R. This has the potential to open up operations on metadata with operators on Collections like "sum" and "difference", useful for tasks like harmonization.

Data Management and Data Structures in DDI 4

Introduction

The initial versions of DDI [2] were designed to describe tabular data structures. With DDI 2, the ability to describe dimensional structures (NCubes) was added. DDI 3 enhanced the capability of describing hierarchical record relationships and enhanced the ability to describe the physical layout of these structures. DDI 4 is being designed to allow for the description of a much wider array of data structures, some of which are described below.

New tools

DDI 4 will also make use of a couple of new tools to ensure consistency in the way these sorts of relationships are described.

The Collection Pattern

The Collection pattern is a powerful new tool that allows the description of a collection of objects and their interrelationships. It can be used to describe a simple ordered or unordered list of objects, or to describe any sort of interrelationship among members of the collection.
DDI 2 and 3 can describe hierarchies from the top down (this level contains that level…). DDI 4 uses a RelationStructure to provide a more general network type description, detailing how each member connects to other members. This allows for descriptions of new structures like conceptual networks or social networks.

[2] See Vardigan, Mary DDI TimeLine. IASSIST Quarterly Vol 37 No 1 (2014): http://www.iassistdata.org/sites/default/files/iqvol371_4_vardigan2.pdf

ViewPoints

Another new feature of DDI 4 is that of ViewPoints. This tool allows for the description of roles of variables: as identifiers, measures, or attributes. The examples below will show how this can be useful for data like event data and can be used to describe the physical layout of aggregate data.

Logical data description

Logical data structure is described in a hierarchy of classes in DDI 4. A DataStoreLibrary is a collection of DataStores. A DataStore is a collection of LogicalRecords. A LogicalRecord is an ordered list of InstanceVariables. An InstanceVariable describes a DataPoint. A DataPoint has a Datum. RelationStructures allow for the description of complex relationships among LogicalRecords within a DataStore and DataStores within a DataStoreLibrary. A RecordRelation further describes the relationships by allowing the mapping of InstanceVariables as keys among records.

Data ultimately have a physical representation. In a CSV file, for example, a number is represented by a string of numerals and possibly some other characters like a period, a comma, a dollar sign, and so on. The physical data description describes how that physical representation is done at both a record level and a DataPoint level. These classes can be found in the FormatDescription package.

A PhysicalDataSet formats a DataStore and contains PhysicalRecordSegments. These may have complex relationships described by a PhysicalOrderRelationStructure. A PhysicalRecordSegment contains DataPoints which can have complex relationships described by a DataPointRelationStructure. A PhysicalRecordSegment also has a PhysicalSegmentLayout that describes the details of the physical representation (e.g. is it delimited? what is the delimiter? What string ends lines? Etc.). A PhysicalSegmentLayout also contains ValueMappings. A ValueMapping formats a DataPoint (e.g. what decimal separator character does the physical string representing the value use? Is there a regular expression or a W3C number pattern for that string?)

Multiline rectangular data

It is not uncommon for a simple (logical) rectangular table to be represented physically with multiple lines for each logical record. This was particularly common when there were length restrictions on physical records (think Hollerith cards). The DDI 4 physical layout classes can describe these data.

Hierarchical record structures

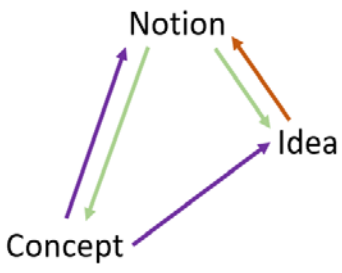
Hierarchical records and their relationships can be described at both the logical and physical level. At the logical level, parent-child relationships can be described along with the variables that link parents and children. The physical level description can delineate how the records are laid out in segments and their order.

Relational structures

Hierarchies are commonly represented as sets of related tables as in a relational database. The same tools that can describe hierarchical structures can be used to describe a relational structure and can also preserve the conceptual description of, for example, parent-child relationships.

Networks

A new feature for DDI 4 is the ability to describe relationships more complex than hierarchical ones. An example would be a network relationship among Concepts. This is a common type of data in qualitative data analysis or more generally in concept maps. The example below describes a simple conceptual network, defined by the MS Word synonym relationships among three words. DDI 4 uses three MemberRelations to describe this network.



MemberRelation
Source: Idea
Target: Notion
Semantic: Synonym

MemberRelation
Source: Notion
Target: Concept
Target: Idea
Semantic: Synonym

MemberRelation
Source: Concept
Target: Notion
Target: Idea
Semantic: Synonym

Tall skinny data

Many software platforms have tools to transpose data from "short wide" layouts to "tall skinny" layouts (e.g. R's reshape 2 package, SAS Proc Transpose). None do a good job of preserving meaning of variables across these transformations.

Event data

Event data are sometimes represented in a tall layout, with the unit for the record as an event. Here again the Viewpoint can be used to have identifiers for the event and for measures in the event. Multiple measure variables can be described, and each can be grouped with one or more attribute variables that point to InstanceVariables and ValueMappings metadata.

The table below is arranged as a "wide" table, the sort of table DDI has traditionally been able to describe.

Event	Temperature	Systolic	Diastolic	Posture	DateTime	Patient
1	37	120	80	1	2018-08-04T10:43:42+00:00	1
2	38				2018-08-04T16:23:22+00:00	2
3	38.5	125	83	1	2018-08-05T10:44:53+00:00	1

The same data are shown below in a "tall" format.

Event	Value	VariablePointer
1	"37"	UrnTemperatureValueMapping
1	"120"	URNSystolicValueMapping
1	"80"	URNDiastolicValueMapping
1	"upright"	URNPostureValueMapping
1	"2018-08-04T10:43:42+00:00"	URNDateTimeValueMapping
1	"1"	URNPatientIDValueMapping
2	"38"	UrnTemperatureValueMapping
2	"2018-08-04T16:23:22+00:00"	URNDateTimeValueMapping
2	"2"	URNPatientIDValueMapping
3	"38.5"	UrnTemperatureValueMapping
3	"125"	URNSystolicValueMapping
3	"83"	URNDiastolicValueMapping
3	"reclining"	URNPostureValueMapping
3	"2018-08-05T10:44:53+00:00"	URNDateTimeValueMapping
3	"1"	URNPatientIDValueMapping

What we have discussed but not yet added to DDI4 is a datatype of pointer to **metadata** for a variable and its physical representation, shown above as the VariablePointer variable. To have a machine actionable description of a general tall table like this, what is needed is not only the logical level variable information for each cell in the value column, but also the details of each associated physical representation. The column "Value" is not a variable in the sense of traditional DDI variables.

In the table above, for example, a program would need to know the format of the datetime DataPoints in order to read it correctly. Note also that there are two types of Unit involved in this example, “event” and “patient”. For events DateTime is an attribute variable. If patients are of interest, records might have PatientID and DateTime as record identifiers, with event as an attribute.

More Complex Event Data

More complex event data may involve different types of data for each event. This may involve nested forms of data, using for example JSON. The Superheroes example in the complex datatypes section below shows how DDI4 might be used in these instances.

Aggregate data

While not currently in the DDI 4 model, the description of aggregate data will be possible, in part through using the ViewPoint. The thinking is that dimensions of a data cube can be described as identifiers, variables within cells of a cube can be described as measures, and additional variables can be assigned to cells as attributes.
A cube can be laid out in a tall fashion with one cell per record. In the following example a two by two cube is laid out with one row per cell. There are two dimensions: Family, and ContactInFamily. There is one measure: NumberOfCalls. There are two attributes: Audited, and SinglePerson. ContactInFamily, the primary or secondary contact person, is nested within Family. This relationship can be described with a VariableRelationStucture. The top row shows the assignment of variables to ViewpointRoles within a ViewPoint. The value in each cell is a Code. Associated Categories for ContactInFamily would be 1 = “Primary Contact”, 2 = “Secondary Contact”.

Identifiers		Measures	Attributes		*Viewpoint*
Family	**ContactInFamily**	**NumberOfCalls**	**Audited**	**SinglePerson**	***Variables***
1	1	5	TRUE	TRUE	
1	2	2	TRUE	FALSE	
2	1	6	TRUE	TRUE	
2	2	1	FALSE	TRUE	

Single datum

DDI 4 is designed to describe data down to the DataPoint and Datum level. A DataPoint is a container for a Datum (e.g. a cell in a spreadsheet that could be empty. The Datum is the actual representation of some measurement. The goal of modeling to this fine-grained level is to be able to attach metadata in data stores that break data down into single observations, e.g. a data lake.

Complex, or Aggregate DataTypes

In terms of ISO/IEC 11404, an InstanceVariableRelationStructure is able to create "aggregate datatypes". These aggregate datatypes may be homogeneous if all the component datatypes are the same one, or non-homogeneous if the component datatypes vary. Aggregate datatypes may be hierarchical or not and they may be recursive or not. Under IS0/IEC 11404, the InstanceVariableRelationStructure qualifies as a "datatype generator". For example, the HL7 FHIR (Fast Healthcare Interoperability Resources) EHR is a non-homogeneous, hierarchical, recursive aggregate datatype. The openEHR archetypes that are one component in an EHR are non-homogeneous, hierarchical, non-recursive aggregate datatypes.

A “blood pressure” would be an example of such a complex datatype. At the design stage of a study one might specify that a blood pressure be taken. This is an aggregate structure of two (or more) ConceptualVariables with at least systolic and diastolic pressures. At the InstanceVariable level there might also be attribute variables (delineated by a Viewpoint) as part of the aggregate (e.g. patient posture – sitting or recumbent).

Lists or Lists of Lists

There are many examples of list datatypes in data that will need to be described by DDI. R, for example has atomic vector and list datatypes. CSV on the Web allows for lists in cells of tables. A VariableRelationStructure can be used to describe a list as a composition of variables (or even sub-lists).

Named Lists of Lists

R, JSON, Python and others also allow for named lists of lists. Internal organization varies (simple sequences like R named lists, or hash tables for more efficient retrieval), but the concept is the same as a key-value structure. Here the keys can be directly related to a DDI variable.

Hierarchical Structures

active
age
formed
homeTown
members
memberItem
name
powers
powersItem
secretBase
secretIdentity
squadName

Aggregate Datatypes might form a hierarchy. An example might be “Rainfall”, with AnnualRainfall at the top of the Hierarchy and JanuaryRainfall…DecemberRainfall variables at a lower level.

Superheroes Example

The JSON file below, taken from the MDN web docs page: “Working with JSON”, shows an example of data with aggregate datatypes. There are 11 keys, shown at right. Most are atomic, traditional variables with basic datatypes like string, Boolean, numeric. The whole collection might be considered as a “superHeroes” variable. It forms a **set** with variables squadName, homeTown, formed, secretBase, active, and members.

The variable “members”, though is an aggregate. It must be described as a **list** of what we’ll call memberItems. A memberItem is a **set** of name, age, secretIdentity, and powers. Roles of each of these component variables could also be described with a ViewPoint.

The variable powers is, in turn a list of something we’ll call PowersItems. PowersItems can be described itself as a codelist.

The ability in DDI4 to describe complex datatypes like member will allow metadata to quickly adapt to “schemaless” practices, yet still provide the ability to attach important metadata.

SuperHeroes JSON file

from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON


```
{
 "squadName": "Super hero squad",
 "homeTown": "Metro City",
 "formed": 2016,
 "secretBase": "Super tower",
 "active": true,
 "members": [
  {
   "name": "Molecule Man",
   "age": 29,
   "secretIdentity": "Dan Jukes",
   "powers": [
    "Radiation resistance",
    "Turning tiny",
    "Radiation blast"
   ]
  },
  {
   "name": "Madame Uppercut",
   "age": 39,
   "secretIdentity": "Jane Wilson",
   "powers": [
    "Million tonne punch",
    "Damage resistance",
    "Superhuman reflexes"
   ]
  },
  {
   "name": "Eternal Flame",
   "age": 1000000,
   "secretIdentity": "Unknown",
   "powers": [
    "Immortality",
    "Heat Immunity",
    "Inferno",
    "Teleportation",
    "Interdimensional travel"
   ]
  }
 ]
}
```


Metadata Processing

DDI 4 has been developed as a set of interrelated classes. This offers a natural fit to object-oriented representations of metadata that have built-in validation of cardinalities and datatype as well as functions that can search for existence of referenced objects both in local and distributed repositories. An object-oriented approach also offers the possibility of developing operators that work directly on the metadata objects. Two

collections could be merged with a “sum” operator. The difference between two collections (e.g. two CodeLists) could also be computed with a “difference” operator. Changes between two objects of the same type could also be computed directly.