Introduction to the DDI4 Logical Data Description Package

# Introduction

In our lifetimes we have traversed three great waves of data representation: relational, data warehouse (dimensional) and elemental (RDF and NoSQL). Along the way the future has seemingly turned the past upside down. I refer to the rise of pipelines in which relational and data warehouse data representations have become so many endpoints in a flow that commences with elemental data.

Quoting Alice, there is little doubt that the future of data representation and data stores is becoming curiouser and curiouser.

In the mist of these developments DDI4 offers a logical data description model that can turn relational and/or dimensional and/or elemental in a [New York Minute](http://www.urbandictionary.com/define.php?term=New%20York%20Minute) (we think). To be honest, so far we have not proven this. Indeed, after two years, the logical data description model remains a work in progress. That being said, we have no shame, and we want to take you on a tour.

# Our Tour

Our tour consists of a number of traversals of the LogicalDataDescription *package*. Note that a *package* in the [Unified Modeling Language](https://en.wikipedia.org/wiki/Unified_Modeling_Language) is used "to group elements, and to provide a namespace for the grouped elements"[[1]](#footnote-1) Also, note that a traversal of a package corresponds to a view. The view / traversal circumscribes a specific function that the package can perform[[2]](#footnote-2).

We start with the Big Picture (package). We will decompose this picture through a series of traversals that begins with the relational store and progresses through a series of knowledge stores that are more or less elemental.

Our story begins like many Big Pictures do with a creation myth. Call this creation myth the big bang of data representation.

First, however, see our big picture.

# The Big Picture

|  |
| --- |
| Figure 1: The LogicalDataDescription Package |

Mostly the traversals we are contemplating make flows within this Big Picture. We will, however, as needed, reference another DDI4 package. This is [FormatDescription](http://lion.ddialliance.org/package/formatdescription). Corresponding to a number of the elements here, FormatDescription encapsulates their physical counterparts. For example, corresponding to each LogicalRecordLayout, there is a PhysicalLayout. FormatDescription is [where the rubber hits the road](http://www.urbandictionary.com/define.php?term=Where%20the%20rubber%20meets%20the%20road). Using FormatDescription, a software agent can, for example, produce actual .csv files or, perhaps in the future, a JSON instance or an RDF graph.

# The Big Bang of Data Representation

|  |
| --- |
| Figure 2: From Observation to Information |

In the beginning there was the Observation. Sort of…

The Observation is performed by an Actor on a Unit and yields one or more Datum. Datum might include not just one or more measures but the location of the Observation in space/time, the identity and state of the Actor and so forth. The one or more Datum are collected in DataPoints, one for each Datum. Empty or not, each DataPoint is described by an InstanceVariable which is ultimately related to a Concept. Finally, a collection of DataPoints form a DataRecord.

The magic here is the Datum. An Observation occurs in space/time. A DataPoint is an information object. The Datum, like a recording, passes in between.

# The Relational DataStore

|  |
| --- |
| Figure 3: Proposed Relational View |

Strangely, tables and table groups are not specifically represented here in line the [W3C Model for Tabular Data and Metadata](http://www.w3.org/TR/tabular-metadata/#schemas) on the Web. Instead both the DataRecord and DataStore “realize” collections. Now consider that:

* The DataRecord has a schema (LogicalRecordLayout)
* The LogicalRecordLayout is a collection of InstanceVariables
* The RecordRelation may specify variable relationships within a LogicalRecordLayout (imagine variable hierarchies) or between LogicalRecordLayouts (imagine primary/foreign key relationships)
* Many DataRecord collections each with their own LogicalRecordLayout may be hosted in a DataStore. Imagine tables.
* The DataStore is also a collection. Imagine table groups.

Seemingly this Relational View of the LogicalDataDescription package is in conformance with the W3C standard.

"tableSchema": {

"columns": [{

"name": "givenName"

}, {

"name": "familyName"

}, ... ],

"primaryKey": [ "givenName", "familyName" ]

}

"tables": [{

"url": "https://example.org/countries.csv",

"tableSchema": "https://example.org/countries.json"

}, {

"url": "https://example.org/country\_slice.csv",

"tableSchema": "https://example.org/country\_slice.json"

}]

What we don’t show here is that the LogicalRecordLayout has a corresponding PhysicalLayout through which a .csv might be generated by a software agent. PhysicalLayout belongs to another DDI4 package called [FormatDescription](http://lion.ddialliance.org/package/formatdescription). We can go there in the future but right now we prefer to be discrete and not [boil the ocean](http://www.urbandictionary.com/define.php?term=%22boiling%20the%20ocean%22) with other packages and views.

# The Data Warehouse DataStore

|  |
| --- |
| Figure 4: Proposed Data Warehouse View |

The difference in this Big Picture fragment is the addition of the ViewPoint.

* In a fact table most variables play identifier roles. One variable plays a measure role
* Attributes of a measurement are not easily handled
* In a dimension table most variables play measure roles and one or more variables play an identifier role
* Leveraging identifiers across fact and dimension tables, each dimension table enjoys a primary/foreign key relationship with one or more fact tables

# [Thunking](http://www.urbandictionary.com/define.php?term=thunking) Non-Tabular Data: JSON and RDF

|  |
| --- |
| Figure 5: The Elementary View - A Cross Between Key/Value Pairs and Triples |

* Before a LogicalRecordLayout can express other types of schema it needs to be able to host sub-schema. That is the purpose of the “nests” relationship through which a LogicalRecordLayout can contain a LogicalRecordLayout. With this relationship in place LogicalRecordLayout can be a model for other types of data in addition to tabular data
  + So the LogicalRecordLayout is inclusive of the schema for an RDF assertions as defined [here](http://www.w3.org/TR/WD-rdf-syntax-971002/#model) in the seminal W3C RDF Model and Syntax (1997) and made more popular and usable in [N3](http://www.w3.org/TeamSubmission/n3/) (2011)

|  |
| --- |
| A collection of triples with the same resource is called an *assertions*. Assertions are particularly useful when describing a number of properties of the same resource. Assertions are diagrammed as follows:  [resource R] ----property P1----> [value Vp1]  |  ----property P2----> [value Vp2]  An RDF *assertions* can be a resource itself and can therefore be described by properties; that is, an *assertions* can itself be used as the source node of an arc. |

* + Also, this LogicalRecordLayout can be used to model structures like the [c struct](https://en.wikipedia.org/wiki/Struct_(C_programming_language)) and the JavaScript Objects that are the subject of JavaScript Object Notation (JSON)

|  |
| --- |
| **{**  **"title":** **"**Example JSON Schema**",**  **"type":** **"**object**",**  **"properties":** **{**  *"firstName"***:** **{**  **"type":** **"**string**"**  **},**  *"lastName"***:** **{**  **"type":** **"**string**"**  **},**  *"age"***:** **{**  **"description":** **"**Age in years**",**  **"type":** **"**integer**",**  **"minimum":** **0**  **}**  **},**  **"required":** **["**firstName**",** **"**lastName**"]** |
|  |

Here properties correspond to instance variables. In terms of “required” each LogicalRecordLayout has a PhysicalLayout which describes the format of the schema. And the PhysicalLayout contains for each instance variable its physical description which includes “required”.

Note that a DataRecord can have many LogicalRecordLayouts and as such the LogicalRecordLayout is specified as a Collection. A Collection in turn considers MemberCorrespondence, so we can weigh pairwise the correspondence between different schema representations like JSON and RDF. See [Can JSON and RDF be friends?](http://milicicvuk.com/blog/2014/08/26/can-json-and-rdf-be-friends/) Using this article we might describe the CorrespondenceType between the two schema representations:

|  |
| --- |
|  |

**Thunk**: *What is it that makes a bunch of key-value pairs appear so different from a set of triples? Add the subject to key-value pairs and you get triples. Or vice-versa — group the triples around the common subject and you get the key-value pairs. Well, in practice, it’s a bit more complicated…* [from [Can JSON and RDF be friends?]](http://milicicvuk.com/blog/2014/08/26/can-json-and-rdf-be-friends/)

**Thunk again***: The fact that a tree is a special type of graph doesn’t necessarily help in losslessly translating between JSON and RDF. In both directions constraints are needed on the source before we can translate to the target* [sounds responsible]

**Thunk at last***: The Elementary view can be used to model JSON without qualification [an unproven claim] but this is not the case with RDF. With RDF the Elementary view is only able to model one of its dialects.*

# The openEHR Use Case

|  |
| --- |
| Figure 6: The openEHR View |

* A Viewpoint groups the instance variables defined in a schema by role. Viewpoints have been introduced because Observation is complicated:
  + Observation yields multiple datum (data), each of which is bounded by an instance variable.
  + Some of the instance variables go to “who”: the identity of a subject.
  + Some of the instance variables go to “what” was observed: they are the measurements.
  + And some of the instance variables go to “how”, “when” and “where”: they are the context that qualifies the measurement(s).
  + openEHR does something comparable with the schema it has defined for health observations
  + In its Observation class, openEHR groups attributes into four parts:
    - a DATA part which contains the core information e.g. the systolic and diastolic pressures when measuring a blood pressure.
    - a STATE part which contains information about the subject of data at the time the information was collected, and this information is required for safe clinical interpretation of the core information. An example is the position of the patient at the time of measuring a blood pressure.
    - a PROTOCOL part which contains information on how the information was gathered or measured, and any other information that is not required for safe clinical interpretation of the core information. By default, this information will not be displayed in the primary view of the EHR.
    - a HISTORY part which contains information about the timing of the observation and the 'width' of the information.

|  |
| --- |
|  |

* Using the Viewpoint, DDI4 can, for the most part, partition the LogicalRecordLayout and define health observation schemata that are equivalent to the ones that openEHR builds using [ADL](http://www.openehr.org/releases/trunk/architecture/am/adl2.pdf) (Archetype Definition Language)

# The FHIR Use Case

FHIR consists of resources that are typically combined.

*FHIR modeling uses a composition approach. In comparison,* [*HL7 v3 *](https://www.hl7.org/implement/standards/product_brief.cfm?product_id=186) *modeling is based on "model by constraint" (see* [*Comparing FHIR to other HL7 standards*](https://www.hl7.org/fhir/comparison.html)*). With FHIR, specific use cases are usually implemented by combining resources together through the use of* [*resource references*](https://www.hl7.org/fhir/references.html)*. Although a single resource might be useful by itself for a given use case, it is more common that resources will be combined and tailored to meet use case specific requirement*

Consider the Composition resource:

|  |
| --- |
|  |

* Note that the Composition resource consists of not one but several classes, that within each class there are resource references and that some resource references (“Any”) are open-ended / untyped
* DDI may be able to account for much of this “architecture”

|  |
| --- |
| Figure 7: The FHIR View |

* A LogicalRecordLayout, because it nests, can mimic the architecture of resources like the Composition. The challenge here is that there needs to be a mechanism for circumscribing a set LogicalRecordLayouts to form a resource
* LogicalRecordLayout inherits from AnnotatedIdentifiable, as do many classes in the DDI4 model. Perhaps it is the job of either the as-is or a to-be AnnotatedIdentifiable to support the identification of resources:

|  |
| --- |
|  |

* When it comes to supporting resource references in general, perhaps the reference is just an instance variable that takes a complex datatype which conforms to how FHIR models references:

|  |
| --- |
|  |

* + There remain nuances like inline or “contained” references that are not specifically identifiable and “Any” references that may be the subject of late binding. Managing the nuances, so long as they are explicitly identified, could be part of the work of the serializer. In fact, in DDI4 during serialization all nesting is managed by reference: nested objects are pulled out and nesting objects are given references to their children.
  + Much of the context of a FHIR Observation resource is captured elsewhere. In discussing its “Boundaries and Relationships” this is what FHIR says:

*In contrast to the Observation resource, the* [*DiagnosticReport*](https://www.hl7.org/fhir/diagnosticreport.html) *resource typically includes additional clinical context and some mix of atomic results, images, imaging reports, textual and coded interpretation, and formatted representations. Laboratory reports, pathology reports, and imaging reports should be represented using the DiagnosticReport resource. The Observation resource is referenced by the DiagnosticReport to provide the atomic results for a particular investigation.*

Arguably FHIR manages at least some of the context of an Observation in a way that is different from the openEHR approach. FHIR goes on to say:

*The Observation resources should not be used to record diagnosis or clinical assessments about a patient or subject that are typically captured in the* [*Condition*](https://www.hl7.org/fhir/condition.html) *resource or the* [*ClinicalImpression resource*](https://www.hl7.org/fhir/clinicalimpression.html)*. However, the Observation resource is often referenced by the Condition resource to provide specific subjective and objective data to support its assertions. There are other resources that can be considered "specializations" of the Observation resource and should be used for those specific contexts and use cases. They include* [*AllergyIntolerance*](https://www.hl7.org/fhir/allergyintolerance.html) *resource,* [*FamilyMemberHistory*](https://www.hl7.org/fhir/familymemberhistory.html) *resource,* [*Procedure*](https://www.hl7.org/fhir/procedure.html) *resource, and* [*Questionnaire*](https://www.hl7.org/fhir/questionnaire.html) *resource.*

FHIR then approaches observation systematically. It both supports specializations of an observation and places observations in various clinical contexts. This has significance for the DDI use of ViewPoint. As we move from Observation and its specializations in FHIR to their contexts, we imagine there will be more “attributes” and fewer “measures” in each resource. In openEHR DDI was exposed to one way of classifying context. In FHIR there are specific resources in which atomic results may be embedded. So FHIR, by and large, takes the approach of representing contexts as their own resources. If DDI is interested classifying health information attributes in its AttributeRole, it could learn much from FHIR.

# Conformance with GSIM

|  |
| --- |
|  |

In GSIM data description revolves around Data Structure whereas in DDI4 data description revolves around LogicalRecordLayout.

* Recall that a LogicalRecordLayout is synonymous with a schema, and a schema is a formalism or, again, a language for describing structures
* DDI4 uses this language to discern structures, while GSIM both posits dimensional and unit data structures and uses the distinction between identifier, measure and attribute components to describe them
* The difference here is that DDI4, using the LogicalRecordLayout, doesn’t get to unit and dimensional data on the basis of apriori distinctions.
* Instead a DataStore can take DataRecords with different LogicalRecordLayouts with a Process in between them.
  + The Process takes as input a set of unit DataRecords its reads with one LogicalRecordLayout and outputs a set of dimensional DataRecords guided by a second LogicalRecordLayout
  + The Process is typically performed by a software agent and generates / provides derived Observations defined by Datum that are in turned swept up into new DataPoints and DataRecords

|  |
| --- |
| Figure 8: The Dialectic of Unit and Dimensional Data |

* Note that in this arrangement the LogicalRecordLayout can describe other types of records too. Unit and dimensional are members of a class of rectangular layouts. The LogicalRecordLayout is recursive, so it can describe non-rectangular layouts too.

This is not to say that GSIM and DDI4 don’t have a lot in common. Conceptually, they do. It’s just that in additional to being a conceptual model that adopts a number of GSIM classes, DDI4 is also a logical model.

As a logical model DDI4 doesn’t just detail a conceptual model stopping at physical implementation. A logical model can reveal when a conceptual model is over-architected which seems to be the case with the GSIM unit and dimensional classes.

# Afterword

I am uncertain what these traversals actually “prove”.

Maybe DDI4 has sufficient promise even now to describe the same data that other standards do. I am not sure that the team that developed this standard was aiming at a “theory of everything”. As a member of the team, I can say it was our intention to subsume relational and dimensional data within a single representation. Also, because of a [national experiment underway in Norway](http://www.raird.no/) in which at least one of our team members participated, we became very interested in including use cases for atomic results and their context. As a result, we began to think more elementally. And this interest dovetailed with efforts of a few of us who were engaged in building [health information data processing pipelines](http://finance.yahoo.com/news/splunk-showcases-big-data-healthcare-130000152.html) that began with unstructured information and used learning techniques to turn this information into knowledge and ultimately present this knowledge in relational databases, date warehouses and various visualization tools for high dimensional data.

My opinion is that we remain at the beginning. But I think this is less a reflection on DDI4 and more a reflection on the state of data representation which, to quote Alice and Lewis Carroll once more, has become “curiouser and curiouser”.

I end with three additional Carroll quotes now, courtesy of Google’s [Knowledge Graph](https://www.google.com/intl/es419/insidesearch/features/search/knowledge.html):

* *Sometimes I've believed as many as six impossible things before breakfast.*
* *I can't go back to yesterday - because I was a different person then.*
* *If you don't know where you are going, any road will take you there.*

1. [OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.1](http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF) p.158. [↑](#footnote-ref-1)
2. See [The Structure of DDI 4](https://ddi-alliance.atlassian.net/wiki/display/DDI4/Structure+of+DDI+4) for an in depth discussion of functional views. [↑](#footnote-ref-2)