DDI Moving Forward Project: Structure of DDI 4 and the process to create it

Version 0.1
Produced at Vancouver Sprint
March 28 2014

Executive Summary

DDI 4 model will consist of two parts – a Library of objects and functional views of the model. The library encompasses the entire DDI 4.0 model. The objects in the library are the building blocks used to construct the Functional Views. These functional views are in essence profiles of the full specification oriented around specific user needs.

This document provides details regarding the model constructs in the library and how they can be extended, as well as information on how the library will be managed and versioned. It also explains the production process and framework that will be used to create the DDI 4 model.

I. Introduction

In 2012 the DDI Alliance established the Moving Forward project to create a model-based specification for DDI (DDI 4) in the Lifecycle development line. This information model approach has the benefits of improved communication with other disciplines and standards efforts, flexibility in terms of technical expressions of the model, and streamlined development and maintenance, among others. The project was launched in October 2012 at Schloss Dagstuhl where a team developed design principles (see Appendix A) and requirements for the new specification. Outcomes of the meeting were summarized in a paper.

The project is now progressing through a series of "sprints" and virtual teams. Project participants include domain specialists and modelers who interact to ensure that appropriate content is captured and that it is modeled optimally.

This document is intended to provide a summary of the project during the early phases of development to enable the wider community to understand how the project is taking shape, the outcomes envisioned, and the DDI 4 development process.

II. The Structure of the DDI 4 Model

A. Overview

This section describes the overall architecture and structure of the DDI 4.0 Model, as shown in Figure 1, including the relationships among its various parts.

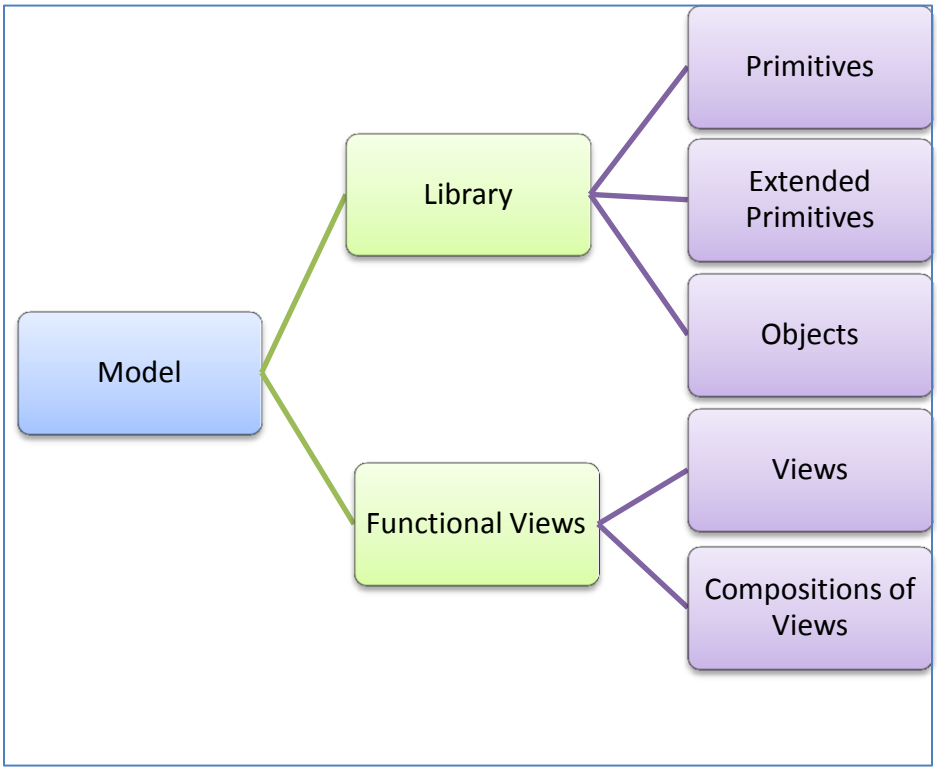

Figure 1. Library of Objects and Views in DDI 4

B. Library of Objects

The Library of Objects encompasses the entire DDI 4.0 model, but without any specific schemas or vocabularies for Functional Views. The objects in the library contain primitives and extended primitives and are the building blocks used to construct the Functional Views. Objects are organized into packages in the Library.

C. Functional Views

From the *Library* are constructed *Functional Views*, which are made up of a set of references to the objects in the library. *Functional views* are subsets of the model grouped to support a specific application (for example the description of a questionnaire). The functional views are divided into sections. Each section loosely corresponds to DDI lifecycle business area. Within each business area section there are separate subsections for views and compositions. Note that views may include placeholders like an abstract class that need to be substituted before the view can actually be used.

Functional views are always a strict subset of the existing published (or, for customization, extended) model packages. A functional view identifies a set of objects that are needed to perform a specific task. It primarily consists of a set of references to specific versions of objects.

Views are the method used to restrict the portions of the model that are used, and as such they function very much like DDI profiles in DDI 3.*. One may (1) restrict the use of non-mandatory properties on an object; (2) restrict the cardinality of an object's relationships and properties; and (3) restrict the use of non-mandatory relationships. Restrictions may never be made that would violate the mandatory inclusion of a relationship or property.

Views may combine objects from any package or set of packages needed. The creation of views thus has no dependency on the organization of metadata objects within the packaging structure.

There are three types of functional views: (1) *Instantiated Views* - those which are used directly to produce XML schemas and RDF vocabularies for implementation; (2) *Template Views,* used as extension bases for Instantiated Views, and (3) *Composition Views,* which combine two or more Instantiated Views.

Interoperability of Functional Views

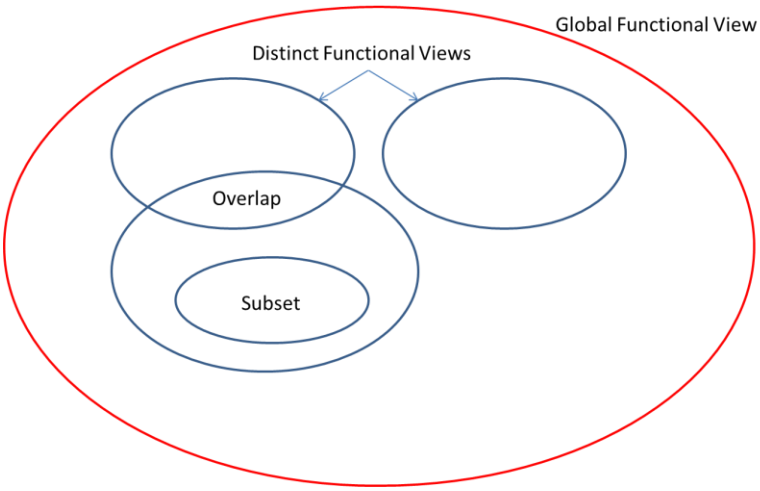

Figure 2. Interoperability of views

As shown in Figure 2, Each functional view is a subset of the objects in the library. Functional views might be distinct, overlapping in their function or a subset respectively superset of another functional view. Interoperability between two functional views is only given for the library objects which are used in both functional views.

A global functional view could be created which comprehends all objects in the library. It represents all functionality of the object in the library. Each functional view would be interoperable to this global functional view.

D. Model Constructs and Their Relationships

Figure 3 below shows the basic relationships between the types of constructs in the model. At the lowest level, we have the primitives. These are used directly by objects, and are also used to create extended primitives. The extended primitives are also used by objects. Objects themselves can relate to other objects, building increasingly complex structures. The objects – along with the primitives and extended primitives – form the Object Library.

Objects can relate to each other in two ways: an object may have a “direct” relationship (composition, aggregation) with another object, or it may have an inheritance relationship. In this latter case, the DDI model uses additive extension. One object may extend another by inheriting all of its properties and relationships, to which the new object may add additional properties and relationships. This mechanism is used to take more generic objects and alter them for a more specific purpose. Extension is explained more fully below.

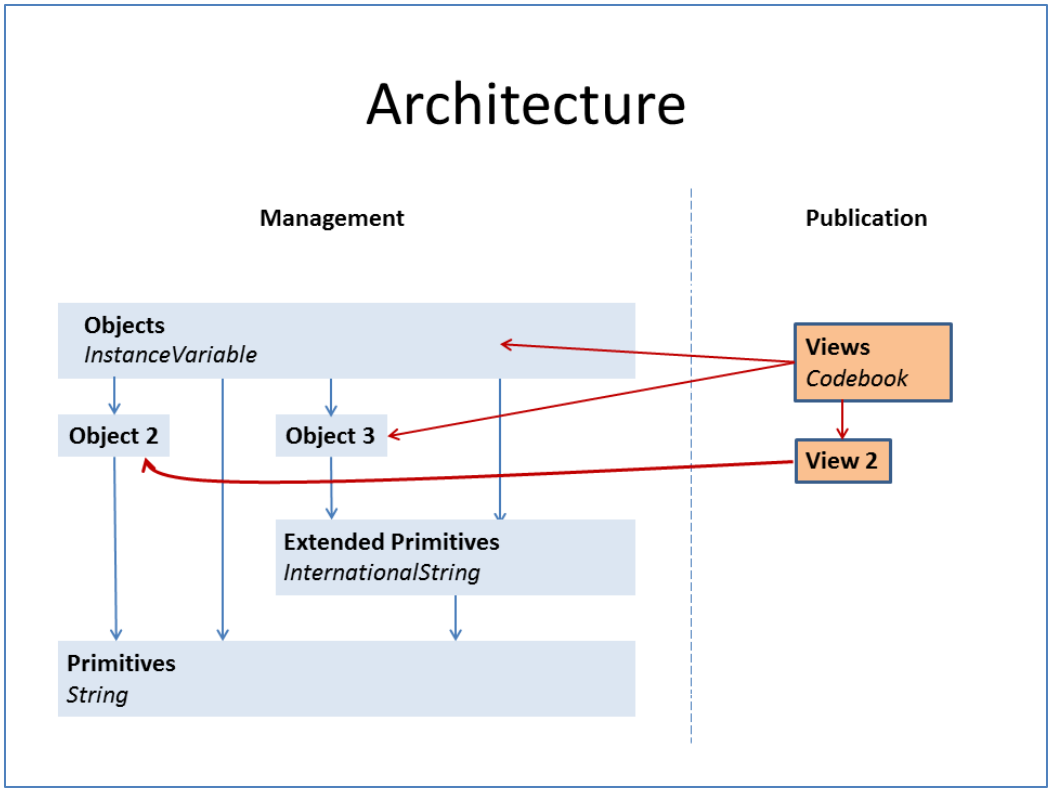

Figure 3. DDI 4 Architecture

E. Extension

Extension is the inheritance of one object's properties and relationships from another object. It also has a semantic relationship – an extending object provides a specialized use of the extended object.

Extensions are used within the DDI-published packages to provide relationships between objects as they increase in complexity to meet increasingly complex functionality. Thus, a "simple" version of a questionnaire object might be extended into a more complex object, describing a more complex questionnaire.

Some objects exist only for the purpose of extension, and are declared abstract. A functional view may never include an abstract object. Non-abstract objects may never have direct relationships with abstract objects.

Extension is illustrated in Figure 4 below.

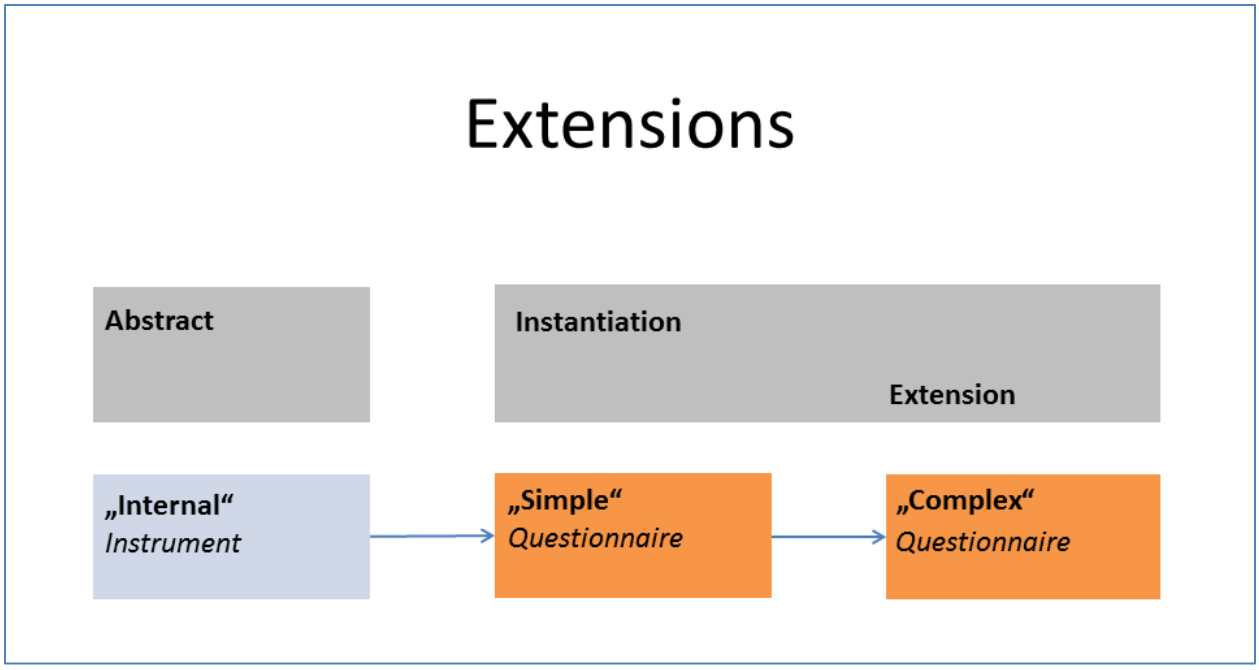

Figure 4. Extensions in DDI 4

Here, an abstract class – Instrument, which is any tool used to collect data – is extended by Simple Questionnaire, which is itself extended by Complex Questionnaire. As we proceed through this chain of extension, the objects have increasingly large numbers of properties and relationships.

For example, if an Instrument object has a name property, a description property, and an ID property, these would be inherited by Simple Questionnaire, which might add a relationship to one or more Question objects. The Complex Questionnaire in turn might add a relationship to a Questionnaire Flow object, to add conditional logic to the questionnaire.

The second use of extension in the DDI model is to allow users to add needed metadata fields for the purposes of customization. Thus, a specific user community may decide to have a standard set of additional properties, objects, and relationships and create their own model package which contains objects extending the objects in the DDI-published packages. The creator of the extensions is the owner and maintainer of the extended objects and packages – this is not the business of the DDI Alliance.

Extension in DDI is strictly defined: you are able to add new properties to existing objects, and add new relationships to existing objects. Extension is always done on an object which is referenced and inherited from: that is, a new object is declared which inherits all the properties and relationships of an existing object. New properties and relationships are then declared for it. Extension is always *additive* extension. There is no concept of refinement – that is handled using Functional Views.

Those creating their own custom packages based on extensions to the DDI model may also declare entirely new objects which are not extension of DDI objects.

Extensions made by those customizing the DDI model are expressed using the same modeling techniques and information that are used for the development of the DDI-published model itself. As a

result of this, the same tools for the creation of documentation and syntax artifacts (XML schemas, RDF vocabularies) could potentially be used.

F. Managing the Library

In order to manage the library effectively, the *objects,* together with *primitives* and *extended primitives*, are grouped into *packages.* The *packages* are organised into a hierarchy, according to the types of constructs. The DDI model is organized in a hierarchy of packages, arranged according to the types of constructs. Higher-level packages contain only other packages – at the lowest level, the packages contain the various types of model constructs (objects, primitives, views, etc.). The Figure 5 below shows how the packages are arranged.

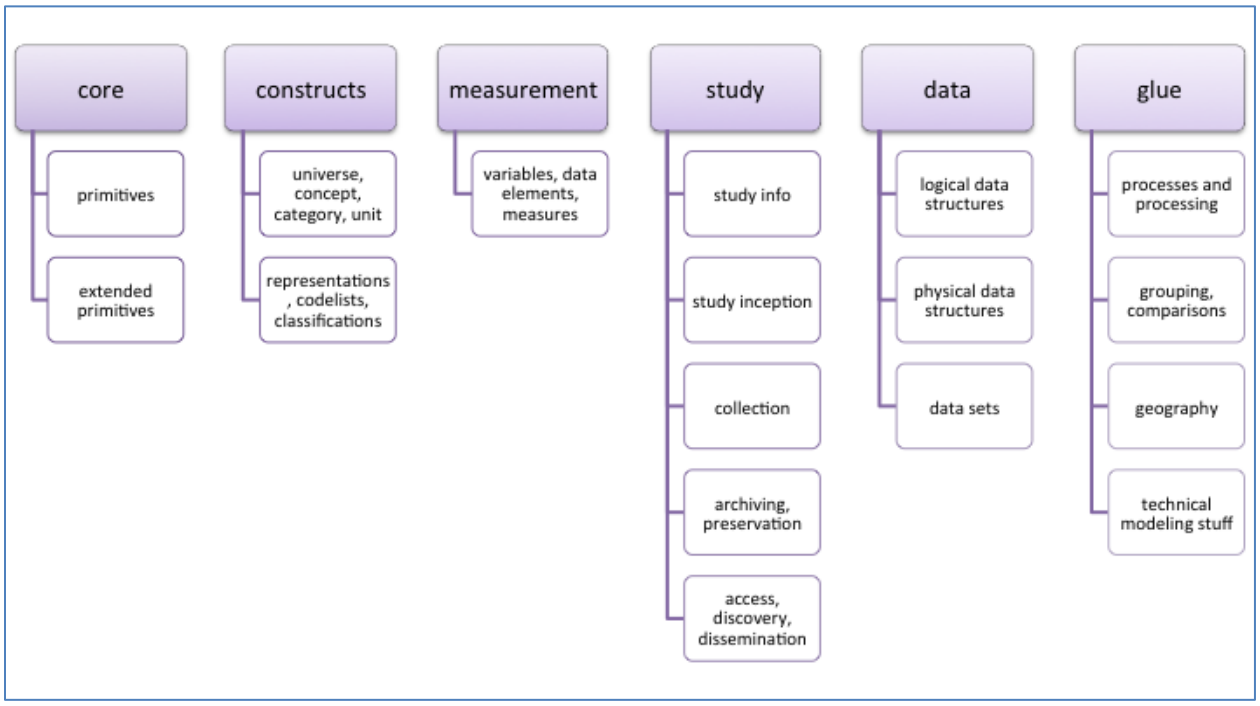

Figure 5. Packages inside the Library

Packages are mutually exclusive and comprehensive. They are organic entities with a logical organization and are labeled in an accessible way so that developers and modelers can easily understand their content. They are stable and should not be changed often.

G. Versioning the Library

The Library itself is versioned as releases are made. Each package within the Library is similarly versioned, the objects within each package are themselves versioned, and the Functional Views are versioned.

The versioning rule is that if the contents of a versioned object change, it is versioned. This means that versions "trickle up" – a new object is added to a package, which versions the package; the new version of the package drives a new version of the Library, and so on.

However, if an object does not change, it does not version, even if the package within which it lives is versioned. Once published, an object is always available for use within Functional Views, even if it is not the latest version of the object. (If the old version of an object is good enough, it is still available for use in a new version of a Functional View, etc.) Once published, objects are never removed from the Library. This has the effect of de-coupling the dependencies created by the use of extensions to add new things to the model. Decisions about what release packages consist of are driven by the needs of users and marketing considerations, and not by the chain of dependencies between objects, packages, etc.

It is foreseen that at least initially, the Library will be released alongside sets of useful Functional Views, but incremental releases are possible without causing problems – a new version of the Library is released, but it will always contain all objects already in use.

III. Production Process and Framework

A. Project Deliverables

The Design Principles for DDI 4 emphasise the need for the model to be easily understood and driven by the user community. In alignment with this design requirement, the aim is for DDI objects to be managed and organized in a logical manner to ensure that the user and developers community can understand the specification and its parts.

The DDI user community has quite diverse needs. There are two target audiences for DDI 4 deliverables:

- The "typical" implementer, who uses the standard UML model, XML schemas, and/or RDF vocabularies as published by the DDI Alliance;

- The "sophisticated" users who might want to extend the official DDI model for the use of a particular community, or who may want to customize the XML schemas or RDF vocabularies. This would include the creation of entirely new Functional Views of the model, to support processes or applications that the DDI Alliance does not support with its official products.

In order to meet the needs of the diverse audience, the DDI Alliance will release two types of products.

The first type of product will be *Functional Views*. These functional views are in essence profiles of the full specification oriented around specific user needs – for example, simple data description, simple codebook, and discovery are functional views.

The second class of products – the *Library of Objects* -- is intended for use by more sophisticated users, as it requires a much deeper understanding of the DDI model as a whole, and the techniques for working with it.

The Library itself would also be an official product of the Alliance published for the purposes of those communities of users who needed additional unofficial views. The creation of these might involve making unofficial extensions to the official Library.

Thus, users can interact with a small portion of the model without needing to learn about the entire thing, making it more accessible. Most users would interact only with the Functional View(s) that interest them.

The DDI Alliance will publish a set of official standard views. It is expected that most implementers would use the official views as published by the Alliance.

A Functional View includes:

1. A documented model, including UML diagrams and narrative
2. An EA file and an XMI file
3. An XML schema with:
   - XSD file with inline documentation
   - HTML documentation
4. An OWL ontology, with:
   - RDF expressions
   - HTML documentation

In the DDI Moving Forward project, the following Functional Views will be created:

- Discovery
- Simple Instrument
- Simple Data Description
- Collection Management
- Data Management Plan
- Study Inception
- Simple Codebook
- Complex Survey
- Complex Data Description
- Comparison/Harmonisation
- Classification
- Qualitative
- Survey Development
- Data Capture Methodology
- Field Work Management

B. Project Teams

In the DDI Moving Forward project, there are a number of virtual content task teams. Some of these teams (for example, the conceptual team and the process/provenance team) are creating the core objects in the library which are needed by many views. Other teams are creating views that reflect common use cases and user stories.

Each content team has a team leader and a lead modeler. The modeler is assigned to the content team by the modeling team. The focus of the modeling team is on integrating objects in the Library into metadata management packages for the purposes of managing the model.

C. Production Process

In DDI 4, a number of outputs are being created. These are shown at a high level in Figure 6 below.

Instances (of XML Schema and OWL/RDF representations)	Usage
Model Representations (of subsets of model) in XML Schema and OWL/RDF	Specification
Functional Views (subsets of model)	
Object Library (model in EA)	
Workbench (in EA)	Development
Drupal	

Figure 6. Production outputs

In order to create these outputs, a production workflow has been created. The agreed process is described below showing the workflows of the content and modeling teams (see Figure 7). Most of the development work will occur in the Drupal content management system used by the Alliance to model objects and packages (see lion.ddialliance.org). Enterprise Architect will be used to validate the UML Model and create the UML Diagrams.

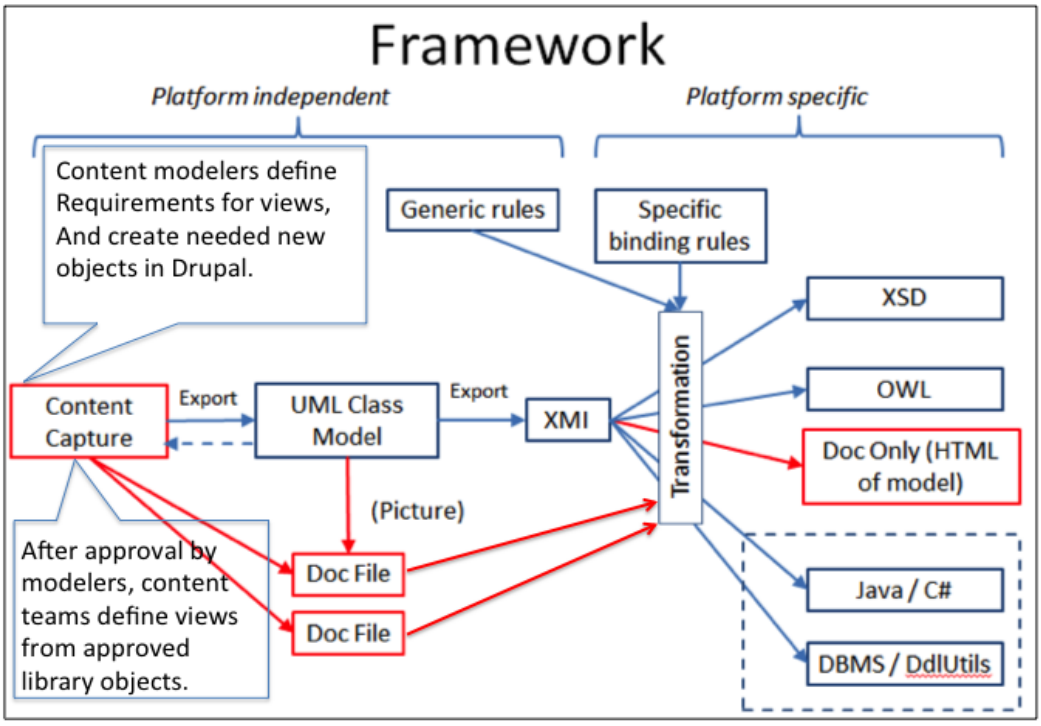

Figure 7. Production framework

Workflow Steps

Step 1: The content team agrees the scope of the work:

- The set of objects to be developed and /or requirements for a Functional View

Step 2: The content team assesses the availability of needed objects:

- Imported DDI 3.2 objects (or other external sources)

- Library packages
- Other objects needing to be created

Step 3: The content team with the Modeler enters the object descriptions
To capture content in the form of machine-processable object descriptions, a Drupal site has been established (lion.ddialliance.org). Content teams enter the descriptions using specified fields. The content teams work with their Modeler to create proposed sets of new objects as well as review and edit existing 3.2 objects.

When new objects are created, they are placed in a temporary container within Drupal for submission to the UML Class Model step, currently Enterprise Architect (EA), for modeling and approval.

When a team is working on a Functional View, it will also create a container that contains a list of all of the necessary objects (by reference). The listing of each referenced object may also include restrictions on properties and relationships. Only objects in the Library can be referenced (not objects being created by other content teams).

Data Modelers will review and comment on the submitted Working Package (not on the integration of objects into the Library). Corrections are always made in Drupal until objects are accepted. Once approved, modelers integrate new objects into the Library.

The object container and possibly the Functional View container are placed into the EA Workbench from which the objects are moved into their final UML packages by the data modelers.

Step 4: The Modeling Team reviews and integrates objects
The organization of the Library inside Enterprise Architect includes a Workbench area which is not published as part of the distributed Library. This is an area where proposed objects to be incorporated into the model are brought from Drupal.

Review and integration of new and edited objects by modeling team, working in EA (see Figure 8):
- Objects and library packages are imported into EA into a holding area (EA Workbench)
- Objects are given a home in an existing or new library package
- Quality and integrity checking is undertaken and changes of objects and versions are made as appropriate
- Objects are approved
- Objects are exported out of EA and Drupal is updated / synched with the EA version

Once the objects are approved in EA, this new Library structure is passed back to Drupal (shown by the dashed line in the figure). We can export a simple EA report to update Drupal (e.g., Library Package name and version, Object Name and version) instead of a huge XMI file.

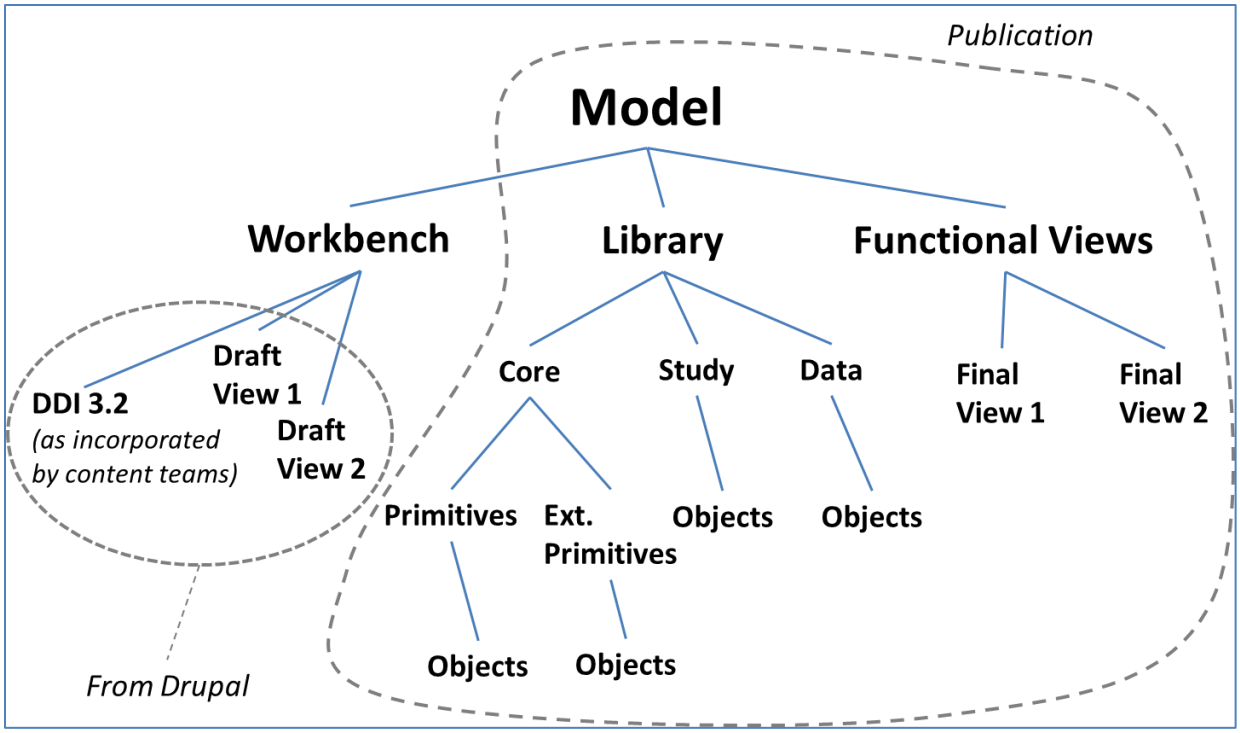

Figure 8. Library organization inside Enterprise Architect to support production workflow

Step 5: The content team creates a draft Functional View

- A functional view is created in Drupal and the description of this as references to its objects and versions are exported as XMI for import into EA.

Step 6: The Modeling Team reviews the Functional Package

- Quality and integrity checking is undertaken in EA
- Functional View model is reviewed and approved
- XMI output from EA
- Functional View Diagrams output from EA

Step 7: The Modeling Team synchs the Functional View from Drupal and outputs documentation of the Functional View

- Drupal is synched with EA
- Functional View-level documentation (DocBook format) is drafted and exported from Drupal
- Documentation is attached to the Functional View package in the XMI
- Full documentation of the objects is encouraged to ensure that end-user documentation will be robust and accessible. The models are then expressed as syntax-bound XML schemas and RDF vocabularies.

Step 8: The Modeling Team produces the Functional View deliverable

- Functional View documentation and EA XMI and diagrams are merged
- All automated outputs are produced for review, approval, and publication

Known System Issues:

1. Functionality will need to be added to Drupal to import XMI from EA and correctly update, add and 'delete' existing objects in line with the remodeling
2. The XMI coming out of Drupal must include Functional View contents as included by reference
3. There is a need to implement a way of locking (possibly putting object to "read only") when XMI is output so we retain data integrity; release of the lock would be triggered on receipt of an update from the Modeling Team.

Conclusion

The DDI Moving Forward Project is driven by the needs of the user community. The best way to influence the outcome of the project is get involved. The project is resourced by volunteers. If you are interested in being involved, there are a number of ways you can contribute.

Several virtual task teams are now running. Each task team is focused on developing a different area of the model. These teams meet regularly via web conference. **Please contact ddisecretariat@umich.edu to express your interest in the task teams.**

You can view the progress of the project by looking at the project wiki:
http://www1.unece.org/stat/platform/display/DDI4/DDI+Moving+Forward+Project+Home