DDI Prototype: TC Review comments

UML Model

The UML model is core to DDI4. Work on the Canonical XMI (PIM) has raised a number of naming and organization issues. Work on a second binding (RDF) has highlighted where structures in the model have been modified to more directly support XML or RDF. This shows where we have blurred the lines between what the UML should express (binding neutral) and what the binding specific views (PSMs) should express. In addition, documentation of the model, specifically of the patterns, brought up a number of modeling issues [filed and labeled Post-Prototype] that go to the core of what the UML model should express, what classes and relationships should be used, and what rules should be enforced so that bindings can be generated by consistent transformation rules. In addition, we do not currently provide users with a clear visualization of the model as a whole either as a full complex coverage or as a simplification. Work has been done on providing the ability to click through the model starting at a class and moving through the relationships but that feature is not currently functional. The Canonical XMI can be loaded into a variety of modeling tools but these are not available to all users.

The following is a summary of the issues relating to the UML model that have been classed as post-prototype. These should be part of the TC review comments so that reviewers understand that we have identified a number of things that need review and clarification in terms of the model itself.

Questions raised concerning the use of UML:

- What should the UML be used to express?
- Where are the lines between a conceptual model, and information model, and the implementation model (bindings) and where does the DDI UML apply?
- Is there a better way to express patterns in UML?
- How can/should regular expressions be captured in the UML?
- Name uniqueness - within model, within package
- Are there UML conventions that would simplify our work (collection types for multiplicity elements, property modifier "id", UML profiles, UML isOrdered property)
- What is the role of Packages as used in Lion - internal only, organization for end user, mixed?
- What is the role of Functional Views within the UML model and how does this affect how they are expressed in bindings?

Expression of different relationships from the UML to bindings

- Relationship targets should not all require identifiers (example the target of a composition)
- Clarify the roles of property (target of a primitive or a set of primitives?)
- Which Structured Data Types should be treated as the target of a composition relationship?
- Are we capturing the needed information on relationships (type, source cardinality, target cardinality, directionality)

Lion/Binding constraints that have creeped into modeling rules

- A class can realize only one class type
- Currently done by pattern classes extending from another pattern class
- Lion rules of "overwriting" relationships with the same name means only a single class can be realized
- Overwriting inherited content with another relationship/property of the same name

Packages – should express regions of the interrelated content

- The stated purpose of packages was to support internal management and provide some groupings of closely related classes for users. Currently they are closely related to development work groups
    - Large set of classes are "shared" across the model either as structural pieces or patterns
    - Due to limitations of Lion, packages were created to development areas and pieces added for the use of a particular Functional View
    - There are no rules to constitute what goes into a package which can result in sub-sets of unrelated classes within a package or confusion when classes relate across package lines.
    - The model is a complex network and interrelations are not well presented
    - Patterns are clearly separated and all pattern classes are within the pattern package, even those that would be considered Structured Data Types

COMMENT: Need to revisit the Package structure in conjunction with the review of the UML modeling use. Clarify the role and use of Packages in the Class Library for modelers and for end users.

Class Construction Rules

- A class supports a single usage/activity – it cannot be used in multiple ways (in terms of intended content and role)
- An activity/use should be represented by a single class
- Classes realizing a pattern class must have all the properties and relationships of the realized class and may restrict cardinality, target class to sub-class [or specific realization], or a more constrained relationship type
- Pattern realizations have been reviewed and errors corrected
- Property names are associated with a single datatype [or realization sub-type] across the model
- Relationship names related to patterns are consistent and are unique [noted used outside of pattern structure/realization]
- Relationship naming conventions are not stated or consistent

COMMENT: This approach may have led to too many classes because we've tied semantics into the naming of classes – this should be reviewed. Can classes be more generic and use the relationship name to provide the semantic?

Review what should be properties. In particular the set of Structured Data Types which contain relationships. Should these be targets of relationships of type Composition with identification needs handled in binding?

Review the idea of full identifiers in the UML model as identification is handled differently in various bindings (i.e. RDF items all have identifiers, while in XML only classes that are targets of relationships have identifiers).

Review UML rules regarding uniqueness of names – in relation to the identifiers within the Canonical XMI

Functional Views

The Functional View is intended to present a clear sub-set of classes needed for a common use case or set of use cases. In the model this ends up being a simple listing of classes with documentation on the boundaries of set (listed as restrictions in the availability of target classes for any relationship from an included class). The content of Functional Views may be stored in a class content library that provides the content for a generated Functional View in a specified binding. As such there needs to be a clear means of identifying all the classes that go into an instance of a Functional View, either by identifying one or more "top" classes which result in a hierarchy or graph of included content, or by defining some selection filter. The primary questions for review are whether the selection of classes for any given Functional View achieve the goal of that Functional View, then whether the model of the Functional View is appropriately expressed in the various bindings.

- If Functional Views are seen as profiles of a subset of classes with discussion of where the boundaries are located the UML expression seems reasonable. Further investigation should be made of UML features that may express these more accurately or consistently.
  - For example using UML means of presenting sub areas of the model through a "view"
- Current Functional Views were created by hand, identifying relationships and following them through to additional classes, identifying restrictions on relationships, verifying that no orphan (unconnected) classes were included, and identification of potential entry points (top of hierarchy).
  - In creating Functional Views we found many difficulties in determining end points (defining boundaries). We could not anticipate user needs even for the examples and had to add classes throughout the review process.
  - It isn't clear within an instance where to "enter" the content. Document Information was an attempt to clarify information about the instance itself and to clarify entry points. However, not all instances are "documents" so this does not seem to be the ideal solution.
- In looking at whether the representations in the various bindings provide similar support we find that in RDF Functional Views were handled by putting them into an inheritance tree which allowed you to limit the view but had no effect on validation.
- The use of XML schema to express Functional Views created a number of problems:
  - Created "hard" boundaries where relationships were blocked
    - External references were allowed but in effect the schema would not be able to process the content of the reference
    - Creates difficulty in the ability to share across a contributed environment. Not all uses of a class support all relationships raising the question of how this additional material is handled. For example if an Instance Variable created for a Descriptive Codebook instance were used by a Data Management Process instance the Data Management Process could not contain all of the content of the Instance Variable resulting in essentially 2 versions of the Instance Variable (one with all content and one with limited content).
    - This situation also effects the ability to "round-trip" between bindings and/or repositories.
    - RECOMMENDATION: TC has found a number of issues with the XML rendering of Functional Views and reviews should provide recommendations for how

Functional Views should be surfaced by bindings. (see TC-75 for a fuller description)

General Binding Issues

- Do the bindings accurately reflect the model?
    - Currently the Lion modeling platform does not export information on regular expressions or default values into the XMI. This has been noted as a requirement for the COGS exports
    - UML has a limited set of primitives (primarily numbers and dates) that it supports resulting in inaccuracies in the bindings (i.e. Integers and Real numbers can be expressed in UML but DDI further differentiates Decimal, Double, Non Negative, etc.).
    - RECOMMENDATION: Explore the needs of the DDI for this level of detail and determine how this could be addressed within the binding transformation rules if required.
    - All round-trip transformations need to be informed by the individual binding transformation rules. This implies that they should be as simple and consistent as possible. However, differentiation between what the UML model should represent and what is handled by the transformation to the bindings implies additional complexity in these rules. Transformation rules should be simplified without pushing restrictions back in the UML model/
        - Example of issues going from RDF to XML
            - Rules for what becomes an attribute in XML
            - How the property "content" is handled within XML
- Is the approach correct?
    - UML: Lines between what should be in UML and what should be in translation rules have become blurred (see more about this in the model discussion)
    - XML: Bindings need to be revisited in terms of schema structure and transformation rules to simplify and clarify the resulting schema structure
    - RDF: Binding has not been thoroughly reviewed due to time constraints

Production Process

- Lion to Canonical to bindings
    - Use issues identified regarding the expression of Lion content in the PIM and in the creation of a Canonical XMI as an information source for preparing the rules for the COGS to Canonical export
        - COGS currently produces Normative 2.4.2 XMI and 2.5 with diagramming
    - Currently the process from the PIM to the Canonical XMI is outside of the daily automated build process. This needs to be brought into the automated production process as we move to COGS.
    - RDF currently builds from the Canonical XMI and is not in the automated build process. It does not create an intermediary PSM as defined in the production framework.
    - XML PSM cannot be used by UML modeling tools as it replicates id's in the flattening process
    - We are currently not following production framework as defined in the documentation
- Move from Lion to COGS
    - Changes in the production tools require a rethink of the production process

    - Both Lion and COGS are modeling tools
        - What do we need to facilitate the modeling?
        - What is needed in terms of visualization? As output and as modeling work support
    - COGS requires rethinking of how input to COGS is created (csv by hand, webform, and/or from EA or other XMI)
    - Determine the appropriate production flow and reinstitute the production process
    - Sphinx and ReadTheDocs is used by both systems
        - Resolve differences between RST generated/used by these systems
        - Production of Sphinx into ReadTheDocs has run into a size issue and must currently be generated on a local computer rather than in the cloud

Complexity

Overall questions regarding the complexity of the model were raised during the TC review and a number of factors have led to the appearance of greater complexity than may be there. These need to be sorted out to present a clear, shared sense of what the model covers and the complexity expressed in each layer (UML, Documentation, and Bindings).

- The model is complex because it is describing a complex topic, but can this complexity be simplified?
- What are the requirements for the model overall and in each area?
- Can some of the apparent complexity be resolved with better organization and management of the Class Library? Clearer differentiation between the UML and binding specific requirements (PIM and PSM approach)?
- Given the options for creating the input format to COGS and the development of the Canonical XMI, does this allow for the use of UML modeling tools that were rejected earlier due to limited outputs and non-interoperability issues?
- Can the use of Patterns (intended to provide a consistent means of modeling similar activities such as managing collections, planning/design, and processing) help in managing complexity in terms of the idea of patterns and also how they are implemented?

The issues surrounding complexity seem to be broad and will require a well-integrated solution covering, requirements, production processes, production tools, development work, and modeling approaches.

Coverage of DDI 3.3 and GSIM

One of the initial requirements of DDI 4 was to cover all DDI 3.2 elements, and areas of GSIM that are relevant to DDI. A preliminary review of the classes in DDI 4, DDI 3.3, and GSIM was made by several members of the TC and is found in the document comparison.xlsx. Note that DDI 3.3 has been used for the listing as it has just completed a public review for publication. Only “primary” classes were reviewed. In DDI 4 this includes the published packages excluding Primitives, Regular Expressions, Enumerations, and Structured Data Types (198 classes). In DDI 3.3 only identifiable classes were used (231 classes). Relevant classes in GSIM (110 classes). Also, this is a preliminary review checking for conceptual similarity, that is that the two classes intend by definition to capture metadata about the same thing. It does not provide a detailed comparison to see if, for example, the DDI 4 class contains all the properties and relationships of the GSIM class.

In summary, DDI 4 covers 43% of the DDI 3.3 classes and 44% of the GSIM classes. The DDI 4 Prototype does not claim to be complete and this should be taken into consideration when looking at coverage. These are ball-park estimates of content coverage and help to understand the ability of DDI 4 to address the same content areas. For example, as a conceptual model GSIM may express a content area in fewer classes whereas DDI 4 is intended to implement the GSIM conceptual model.