## **Modelling recommendations**

## **Associations**

### Requirements

* Associations should be used in a precise and consistent way. This applies especially for the several available UML options to qualify associations and/or document them.
* Associations are rendered in UML diagrams by connecting classes with a line. This should be used efficiently to indicate the relationship of classes and the kind of relationship
* Restricted associations should only be used with a clear need and reasoning

Associations and properties (defined by other classes or data types) have a similar meaning in UML.

**Decision of record**: associations should be viewed as predicates between a subject and an object. The name of the association should reflect that. The direction of the association should also be in line with the predicate so that it points from the subject (class) to the object (class). Additionally, the direction can be used for versioning purposes, i.e. changes to the object will trigger a new version of the subject. For this reason, associations should always be unidirectional.

It’s important to note that uni-directionality in the conceptual model doesn’t prevent bi-directionality at the representation level.

We don’t see any semantics in aggregations not already covered by common associations with appropriate directed names, but it could provide a way of easily visualizing whole-part relationships.

**Task**: We need to provide guidance on when to use Collections as oppose to simple aggregations. Collections are basically a specialized type of aggregation with additional semantics (order, structures). One case in which Collections come in handy is when we deal with order, either lists/arrays or more complex structures. Otherwise, for unordered Collections, we might just want to use aggregation.

Composition might make sense for a very few cases in which there is a strong lifecycle dependency, e.g. a cell in an array cannot exists without the array. However, it could provide a way of easily visualizing strong lifecycle dependency.

**Decision of record**: to use UML aggregation type and navigability (direction), but not ownership.

**Decision of record**: to have mandatory cardinalities on both ends of associations.

**Task**: we need a review of the existing aggregations/compositions, and any association with minimal cardinality of 1.

**UML note**: association names must be unique within a package (likely, UML uses namespaces as part of association names). The current model has some name clashes in which two associations share the same name within a package.

**Decision of record**: association names (the predicate) should be semantically meaningful, i.e. we should avoid generic names, e.g. has, is, contains, whenever possible.

**Decision of record**: to have unnamed aggregations and associations for those generic cases where the association name is redundant, e.g. “has” and “contains” are implicit in the meaning of aggregations/associations. Transformations will add appropriate names when required, e.g. for the RDF representation all unnamed aggregations/compositions will be represented as “has”, which can be reused across packages.

**Note on Namespaces**: Names in the UML model, both class and association ones, are fully qualified by their namespace, e.g. the package they belong to. In other words, qualified name = namespace + local name, where the local name is the actual name of the class/association.

**UML Note**: qualified names of associations should be unique across the model, local names for associations should be unique within their namespaces (packages) and can repeat across packages.

**Decision of record**: associations with the same local name have the same semantics across the entire model.

**Decision of record**: property names have to be unique across the model.

**Decision of record**: local class names have to be unique across the model.

**Note:** UML roles can be used to clarify the use of a class in different contexts. For instance, in a self-referent association, like isParentOf, roles can be used to identify the source and target classes as parent and child, respectively.

**Data Types**

### Requirements

* Data types are important to assure consistency in the model.
* The definition and selection of primitive data types play a crucial role because more complex data types are based on these.
* Data types should be as consistent as possible across the model and the representations. This minimizes mapping requirements and supports easy round tripping of metadata between different representation instances.

**UML note**: [data types](https://www.uml-diagrams.org/data-type.html) in UML are related to class definitions but are their own structural item. Data types can have properties but not relationships.

**Decision of record**: classes that have no associations to other classes should be data types. Currently, there are classes in the structured data types package that have associations to other classes, they should be moved elsewhere.

**Decision of record**: when a property is defined by a class rather than a data type, and the class has associations to other classes (hence it’s not a UML data type) we should either (i) make the property into an association or (ii) remodel the “data type” class into a real UML data type by removing associations, etc.

Data types include primitives, structured data types, and enumerations.

Primitives include a set of [5 UML primitives](https://www.uml-diagrams.org/data-type.html#primitive-type) and user-defined primitives. A primitive has a definition which is outside of UML, i.e. cannot be defined further by UML.

Structured data types have a structure which are defined by properties. The properties can be defined by other data types

**Question**: How many levels of reuse of other data types is reasonable? Too many levels are not easily understandable and create a network of dependencies.

**Recommendation**: Data types should be the same across different technologies, i.e. in the model and the representations. If this is not possible, they should be easily mappable without the risk of information loss of the content/values.

From [XML Schema Part 2: Datatypes Second Edition](https://www.w3.org/TR/xmlschema-2/): “The framework has been influenced by the [[ISO 11404]](https://www.w3.org/TR/xmlschema-2/#ISO11404) standard on language-independent datatypes as well as the datatypes for [[SQL]](https://www.w3.org/TR/xmlschema-2/#SQL) and for programming languages such as Java.”

**Decision of record**: to use UML primitive data types in the UML model and XML Schema data types in the representations. Four out of five primitive data types map directly to XML Schema data types at transformation time. The fifth one, the literal unlimited natural, needs to be handled separately to different representations.

**Decision of record**: for XML Schema data types beyond the UML primitive data types we should define new UML primitives (approach followed by Eclipse developers). That way we can integrate XML Schema data types into the UML model. If we ever need data types that are not in XML Schema we propose to define new UML primitives by using the rules specified in ISO 11404. Additional primitive data types might need a special mapping to data types in the representations which would require additional work.

Additional primitive data types can be added as needed which might be required for special cases. From [Wikipedia](https://en.wikipedia.org/wiki/ISO/IEC_11404): “ISO/IEC 11404, General Purpose Datatypes (GPD), are a collection of datatypes defined independently of any particular programming language or implementation.” This standard might be helpful for creating additional data types. But the definition might be outside of UML. UML primitive data types have a definition beyond UML, i.e. the definition could be just a reference to a section in a document.

**Note**: the collection pattern could be applied for the data type aggregator ISO 11404.

The overall understanding here is that the set of XML Schema data types and the [ISO/IEC 11404 General Purpose Datatypes](http://standards.iso.org/ittf/PubliclyAvailableStandards/c039479_ISO_IEC_11404_2007(E).zip) could be used in a complementary way.

**Proposal**: XML Schema data types are in the xsd namespace. We should maintain the namespace and use it in the UML to avoid name clashes. We need to check whether namespaces are included in canonical XMI.

Any other structured data types could be built on the basis of primitive data types.

This approach would support the use of data types in many representations as the XML Schema data types are an acknowledged set across languages not just for XML. A mapping would not be required for these. They could be immediately used.

Structural data types which are built on the basis of primitive data types can be automatically generated for the representations according to representation specific rules

**Question**: If the XML Schema data types are used, another question needs to be decided: should the 5 UML primitives be used or only the XML Schema data types. UML UnlimitedNatural is not available in the XML Schema data types.

### **Regular expressions for data type definitions**

Regular expressions are constraints on string values. A property defined as string can have a related constraint in UML. The constraint can be defined in a chosen language which is usually indicated by a prefix. Common languages are OCL and English. Regular expressions don’t seem to exist in OCL. Therefore a specific regular expression syntax can be chosen for this purpose.

**Proposal**: Usage of [XML Schema regular expression syntax](https://www.w3.org/TR/xmlschema-2/#regexs). This syntax is a common subset of many others used in Perl, Python, Java, etc. The usage of a common subset would enable immediate use of the regular expression in many representations without adoption to local regular expression flavors.   
Example constraint - regexpr:a+

Citation from <https://www.regular-expressions.info/xml.html>: “Particularly noteworthy is the complete absence of anchors like the caret and dollar, word boundaries, and lookaround. XML schema always implicitly anchors the entire regular expression. The regex must match the whole element for the element to be considered valid.”

This is not really a constraint for regular expressions in the context of the DDI 4 UML model.

### Post-prototype and Prototype-review issues related to Data Types

Master issue [DMT-220](https://ddi-alliance.atlassian.net/browse/DMT-220) Data Types

* [DMT-200](https://ddi-alliance.atlassian.net/browse/DMT-200) Classes without relations should be defined as datatypes
* [DMT-205](https://ddi-alliance.atlassian.net/browse/DMT-205) Regular Expression rules for model
* [DMT-209](https://ddi-alliance.atlassian.net/browse/DMT-209) Datatypes - where are they appropriate
* [TC-62](https://ddi-alliance.atlassian.net/browse/TC-62) XML Schema Datatypes
* [XMI-1](https://ddi-alliance.atlassian.net/browse/XMI-1) How can XSD datatypes be included in the UML model?
* [TC-45](https://ddi-alliance.atlassian.net/browse/TC-45) Datatypes: clarification and improvement of organization

Documents with content of each issue (2018-11-28)

<https://drive.google.com/open?id=1jbNrTmx0XLK2l3vRpCS7PzY85c-Hf9RW>

**Structural and integrity constrains**

**Proposal**: we should use OCL to define constraints only when they are relevant and can be expressed in the different representations. We should just document them in English otherwise.

Formalizations of constraints are understood to be important. For the purposes of the core, however, they will be documented in English.

**Note**: To capture the unbound cardinality in associations and properties UML uses “\*”, which is represented as LiteralUnlimitedNatural in canonical XMI. In XML Schema it should be represented as maxOccurs="unbounded". We need to check how that can be represented in RDF, e.g. ShEx

Additional notes

We need to remove multiple inheritance as much as possible. One way of doing that would be to make the Process Pattern extend the Collections Pattern.