mvdXML

Specification of a standardized format to define and exchange Model View Definitions with Exchange Requirements and Validation Rules

Developed by
Model Support Group (MSG) of buildingSMART International Ltd.

Authors
Chipman, Tim; Liebich, Thomas; Weise, Matthias

Version 1.1 Final
15. 02. 2016
## Document history

<table>
<thead>
<tr>
<th>Version</th>
<th>Change description</th>
<th>Date</th>
</tr>
</thead>
</table>
| 1.1 Final | **schema extensions**  
- namespace updated to: [http://buildingsmart-tech.org/mvd/XML/1.1](http://buildingsmart-tech.org/mvd/XML/1.1)  
- **RuleId** - new simple type to restrict EntityRule.Reference.@IdPrefix, EntityRule.@RuleId and AttributeRule.@RuleId  
- EntityRule.Reference.@IdPrefix - changed to RuleId (was normalizedString)  
- EntityRule.@RuleId - changed to RuleID (was normalizedString)  
- AttributeRule.@RuleId - changed to RuleID (was normalizedString)  
- Copyright – changed to normalizedString (was anyURI)  
- documentation  
  - examples updated to the latest grammar  
  - documentation updated and improved | 2016-02-15 |
| 1.1 RC   | **schema extensions**  
- EntityRule.References.Template - new element that allows to reference other templates as partial templates, it allows to reuse common, smaller ConceptTemplate definitions  
- EntityRule.References.Template.@ref - reference to the partial template by uuid  
- EntityRule.References.@IdPrefix - an optional prefix for the RuleId name, used to prevent ambiguous RuleId, if the same partial template is referenced twice in a concept template tree  
- Concept.TemplateRules - new element and tree structure to define a logical tree (with Boolean operators) to combine several template rules  
- ConceptRoot.Applicability - new element to check, whether the instance of the applicableRootEntity is applicable, allows for more conditions (like certain property values)  
- ConceptTemplate.@applicableSchema - defined as a list of extensible enumeration of standard IFC schema identifiers, or any other schema name.  
- ModelView.@applicableSchema – defined as a single string, being an extensible enumeration of standard IFC schema identifiers, or any other schema name name  
- TemplateRules – new element that is declared in a recursive way, allowing other TemplateRules, or individual TemplateRule as child elements. It allows to establish a Boolean tree, where at each TemplateRules a logical operator is defined,  
- TemplateRules.@operator – new attribute that defines the logical operator to combine the logic results of its children,  
- Requirement.@requirement – enhancement of the enumerators to include recommended, not-relevant (was "not relevant” and “optional”) and not-recommended.  
- schema changes - strict version: removed, transitional version: deprecated  
- AbstractRule - abstract element and complexType removed, attributes moved to AttributeRule and EntityRule  
- ConceptTemplate.Rules - restricted to AttributeRule, was an agreement in V1.0, now enforced by schema  
- AttributeRule.@Cardinality - removed: this attribute shall not be used to impose a restriction on the cardinality, restrictions are all handled by template rules  
- EntityRule.@Cardinality - removed: this attribute shall not be used to impose a restriction on the cardinality, restrictions are all handled by template rules  
- EntityRule.EntityRules - removed: There is no usage for an EntityRule to directly contain other EntityRules, without an intermediate AttributeRules  
- ConceptRoot.Requirements - removed: requirements are only valid for concepts, not for a root concept | 2015-08-18 |
### Version 1.1 beta

**Change description**
- First revision of mvdXML with following corrections, changes and clarifications:

**Schema extensions:**
- cardinality attribute of `AttributeRule` and `EntityRule` extended to support definition of any min and max settings
- `BaseConcept` and `Override` attribute added to `Concept`
- `tags` attribute added to `Definitions`

**Rule grammar:**
- `mvdXML 1.1 beta` provides a grammar for defining constraints to simplify rule parsing and to enable logical “or” combination of rules

**Schema improvements:**
- new complex type `GenericReference` (used by `ModelView` and `Concept`)
- simplified definition of `EntityRule` and `TemplateRule`
- definition of applicability attribute changed for `ExchangeRequirement` and `Requirement`
- `minOccurs` changed from 1 to 0 for `ModelView.Roots` and `mvdXML.Templates`
- `maxOccurs` added to several definitions, mainly for clarification
- definition and use of applicability (was `xs:attribute` is now `xs:simpleType`)
- `ConceptTemplate.applicableSchema` changed to a list of String types
- `ConceptRoot.applicableRootEntity` now mandatory

**Improved and extended documentation:**
- Use of sub-templates and sub-concepts clarified
- Several improvements and corrections

**Date:** 2013-11-01

### Version 1.0

**Change description**
- Final release of mvdXML. Accepted by bSI ITM committee as the official buildingSMART specification for publishing Model View Definitions
- NOTE: This release does not yet focus on model validation

**Date:** 2012-05-14

### Version 0.9.4

**Change description**
- The following changes were made in this draft:

**Date:** 2012-05-11

### Version 0.9.3

**Change description**
- The following changes were made in this draft:
  - `ModelView.BaseView` added for indicating add-on views.
  - `ExchangeRequirement.applicability` attribute added.
  - `ConceptTemplate.ApplicableEntities` renamed to `ApplicableEntity`.

**Date:** 2012-05-07

### Version 0.9.2

**Change description**
- The following changes were made in this draft:
  - `ConceptLeafNode` was renamed to `Concept`, with `SubConcepts` added.
  - `ApplicableSchema` attributes use string instead of enumeration for version flexibility.
  - Cardinality includes "..asSchema" to indicate default cardinality.

**Date:** 2012-04-20
<table>
<thead>
<tr>
<th>Version</th>
<th>Change description</th>
<th>Date</th>
</tr>
</thead>
</table>
| 0.9.1   | Combined mvdXML schema proposal incorporating the original mvdXML 0.5 with the proposed phase 2 extension, several simplifications:  
  ▪ The ConceptNode entity was deleted.  
  ▪ The Concept abstract entity was deleted, since ConceptNode was deleted, where the only attribute was moved to the subtype ConceptLeafNode.  
  ▪ The ConceptRootNode Category attribute was deleted   | 2012-03-27  |
| 0.8     | Proposal for phase 2 of a formal mvdXML format: Mapping of MVD concepts to IFC definitions as appendix to mvdXML 0.5                                                                                                   | 2011-05-20  |
| 0.5     | First buildingSMART release, no other changes of content                                                                                                                                                              | 2011-06-19  |
| 0.4     | Public release, first release after acceptance of mvdXML by buildingSMART ITM group, following changes have been made:  
  Incorporation of the formally defined (IFC) schema, that describes the formal subschema corresponding to the Model View Definition.  
  Minor changes as result of first prototype developments.                                                                                             | 2011-05-05  |
| 0.3     | Public release, incorporating feedback from buildingSMART MSG  
  Restructuring of document content, adding MVD history.  
  Adding general objectives, motivation and relation to MVD methodology.  
  Minor corrections in XSD Version 0.3 (key/keyref and href for Definition).                                                                            | 2011-03-04  |
| 0.2     | Restricted release to buildingSMART MSG and TechCom, XSD Version 0.2                                                                                                                                                | 2011-02-16  |
| 0.1     | Internal release, XSD version 0.1                                                                                                                                                                                   | 2011-02-07  |
Table of Content

1 Overview ................................................................................................................................. 7
  1.1 Purpose .............................................................................................................................. 7
  1.2 Methodology ..................................................................................................................... 7
  1.3 Usage ................................................................................................................................. 8

2 Schema.......................................................................................................................................... 9
  2.1 Quick overview .................................................................................................................. 10
  2.2 Simple example .............................................................................................................. 14

3 Description of mvdXML schema elements and types .............................................................. 18
  3.1 mvdXML ............................................................................................................................ 18
  3.2 Concept Template ............................................................................................................. 18
    3.2.1 Attribute Rule ............................................................................................................. 19
    3.2.2 Entity Rule ................................................................................................................ 20
    3.2.3 Constraint .................................................................................................................. 20
  3.3 Model View ....................................................................................................................... 21
    3.3.1 Exchange Requirement ............................................................................................. 21
    3.3.2 Concept Root ............................................................................................................. 22
    3.3.3 Applicability .............................................................................................................. 22
    3.3.4 Concept ..................................................................................................................... 23
    3.3.5 Requirement .............................................................................................................. 24
    3.3.6 TemplateRules ......................................................................................................... 25
    3.3.7 TemplateRule .......................................................................................................... 26
  3.4 Common type and attribute definitions ............................................................................. 26
    3.4.1 Identity ...................................................................................................................... 26
    3.4.2 Definition ................................................................................................................... 27
    3.4.3 Body .......................................................................................................................... 27
    3.4.4 Link ............................................................................................................................ 28

4 Rule Grammar ...................................................................................................................... 29

5 mvdXML Use Cases ................................................................................................................ 32
  5.1 MVD Documentation ...................................................................................................... 32
  5.2 Specification of subset schemas ...................................................................................... 32
  5.3 Data Filtering .................................................................................................................. 33
  5.4 Data Validation .............................................................................................................. 33

6 Glossary ...................................................................................................................................... 34
  6.1 ER ................................................................................................................................. 34
  6.2 ERM ............................................................................................................................... 34
  6.3 MVD .............................................................................................................................. 34

7 Examples ..................................................................................................................................... 35
  7.1 Example for MVD documentation ................................................................................. 35
  7.2 Example for MVD validation ......................................................................................... 41

8 XSD Listing .............................................................................................................................. 49
Table of Figures

Figure 1: Graphical representation of mvdXML schema ................................................................. 9
Figure 2: Basic structure of the mvdXML schema ........................................................................ 10
Figure 3: main mvdXML elements for ConceptTemplate ............................................................ 10
Figure 4: main mvdXML elements for ModelView ..................................................................... 13
Figure 5: HTML documentation of IFC4 .................................................................................... 32
Figure 6: Graphical representation of the concept template “Nesting” ....................................... 35
Figure 7: graphical representation of the concept template “port nesting” ................................. 36
Figure 8: rendering of HTML tables to document the exchange requirements for the different ports .... 36
Figure 9: graphical representation of the usage of concept “port-nesting” at the root concept IfcHeatExchanger ......................................................................................................................... 37

Table of Tables

Table 1: Common element references defined in the element mvdXML ........................................ 18
Table 2: Common attributes and element references defined in the element ConceptTemplate .... 19
Table 3: Common attributes and element references defined in the element AttributeRule ........ 20
Table 4: Common attributes and element references defined in the element EntityRule ............... 20
Table 5: Common attributes defined in the element Constraint .................................................. 20
Table 6: Common attributes and element references defined in the element ModelView .......... 21
Table 7: Common attributes defined in the element ExchangeRequirement ............................... 22
Table 8: Common attributes and element references defined in the element ConceptRoot ........ 22
Table 9: Common element references defined in the element Concept ....................................... 23
Table 10: Common element references defined in the element Requirement ............................. 24
Table 11: Common attributes defined in the element TemplateRules ......................................... 25
Table 12: Truth table for operator attribute .................................................................................. 25
Table 13: Common attributes defined in the element TemplateRule .......................................... 26
Table 14: Common attributes defined in the attributeGroup identity ........................................ 27
Table 15: Common element references defined in the element Definition .................................. 27
Table 16: Common attributes defined in the element Body ......................................................... 28
Table 17: Common attributes defined in the element Link .......................................................... 28
Table 18: Description of metric values .......................................................................................... 30
Table 19: Description of operators ............................................................................................... 30
Table 20: Description of operators that can be applied to different data types ............................ 31
1 Overview

A Model View Definition (MVD) describes the subset of a data schema that is required to exchange the data required in specific data exchange scenarios. An Exchange Requirement (ER) defined the required population (the data that is actually provided in an exchange file) of such a sub schema.

The buildingSMART standard mvdXML refers to an electronic format for representing such Model View Definitions and associated Exchange requirements. The purpose of this document is to describe the structure and usage of mvdXML.

While a data schema describes available data structures and type information, a model view definition describes graphs of such data structures to be used in particular scenarios with particular constraints. While mvdXML is a generic structure that could be applied to any data schema, the primary intention and documentation herein describes its use relative to the Industry Foundation Classes (IFC) data schema, the ISO16739 standard.

1.1 Purpose

The mvdXML format serves several purposes:

- To define the sub schema for the MVD, based on the base schema of IFC
- To support automated validation of IFC data sets for quality assurance and software certification.
- To generate documentation for specific model views and the IFC specification itself.
- To support software vendors providing filtering of IFC data based on model views.
- To limit the scope of IFC to well-defined subsets applicable for particular applications.

NOTE: If mvdXML shall be used for one specific purpose only not all features of this specification might be of interest. For instance, data filtering and data validation not necessarily require detailed end-user documentation or any meta-data like status, owner etc. Accordingly, depending on the main use case a subset of mvdXML might be sufficient to cover required functionality. More details about suggested use cases and evaluation of mvdXML are discussed in chapter 5.

1.2 Methodology

The underlying methodology of mvdXML is the definition of concept templates and concepts.

- A Concept Template is a graph that starts with a root entity and consists of attribute and other entity definition, all are required to represent a functional unit required to exchange specific data
  - An example is the concept template “property sets for objects”, that describes the graph, starting at the applicable supertype IfcObject, and describing the graph down to the assigned IfcPropertySet and further to the individual properties, such as IfcPropertySingleValue.
  - The official IFC specification lists within its chapter 4 “fundamental concepts and assumptions” those concept templates already defined. Developers of Model View Definitions are encouraged to use these concept templates, but may enhance the existing or define new ones.
- A Concept is the reference to such a concept template for each entity (as subtypes of the applicable root entity of the concept template) and describes the particular constraints and usages within the scope of the entity.
  - An example is the definition of all applicable property sets, such as Pset_BeamCommon for the entity IfcBeam as a particular usage of the concept template “property sets for objects”.
A model view indirectly describes the set of IFC entities within scope based on entities referenced from each root concept, and entities or types (and subtypes as defined) used within instance graphs. All such IFC entities maybe combined and published as the IFC schema subset corresponding to the model view. The IFC schema subset has to be a valid schema by itself.

1.3 Usage

While it is still being possible to write an mvdXML based Model View Definition by using any text editor, it is anticipated that specific software applications are used to read and write mvdXML data sets. Software for working with mvdXML may include the following.

- The IFC Documentation Generator (IFCDOC.EXE) is a free tool issued by buildingSMART that reads and writes mvdXML, and provides a graphical user interface for defining all content within mvdXML. It can be preloaded with a particular IFC release specification and allows access to all parts of the IFC specification when developing the mvdXML concepts and constraints. This tool may also auto-generate instantiation diagrams, output HTML documentation for model views, and is also used for generating the IFC4 documentation.
- XML/XSD editors such as Microsoft Visual Studio and Eclipse may edit mvdXML in raw format, just as any other XSD-based schema.
- Testing servers may read mvdXML and use such information to validate submitted IFC files for conformance.
- IFC-based software applications may read mvdXML for automatically filtering and validating data to conform to the specified constraints. It is also possible for IFC-based software applications to write mvdXML to enable users to define custom exchange scenarios.
- Requirement management tools may support configuration of data exchange requirements that, if based on existing mvdXML snippets, could be exported as an mvdXML document to be used for filtering and validating IFC data.
2 Schema

Figure 1: Graphical representation of mvdXML schema
The overall mvdXML Schema is introduced with a quick overview and by a simple example first. The next chapter 3 is the reference for all XSD elements and types.

2.1 Quick overview

An mvdXML document contains an instance of mvd:mvdXML as the only single valid root element. The mvdXML element defines two main sub elements:

- **mvd:Templates**: a list of reusable concept templates, mvd:ConceptTemplate, that define the graph within the base IFC schema representing the entities and attributes needed to support the functional unit addressed by the concept
- **mvd:Views**: a list of model view definitions, mvd:ModelView that contains the necessary entities and associated concepts to define the sub schema of the base schema to support the exchange requirements.

Figure 2: Basic structure of the mvdXML schema

An mvd:ConceptTemplate defines the graph, starting from an applicable root entity, following attribute and entity links, down to the individual attributes, which contains all schema information for a particular unit of functionality – or “concept template”, the term used within the Model View Definition methodology.

Figure 3: main mvdXML elements for ConceptTemplate
EXAMPLE: The attachment of property sets with particular properties to an element is the unit of functionality a concept template “Property Sets for Objects”. The definition of an assembly structure, where the assembly, such as an element wall has building element parts is another example “Element Decomposition”.

Each mvd:ConceptTemplate starts with the applicable entity, the root of this unit of functionality. In most cases, it is a subtype of IfcObject, being an occurrence of a model element subject to validation.

Then the attribute(s) used for expressing the unit of functionality are declared, then the type of the attribute, in case of an entity type, it can have own attribute definitions again. All together it defines a tree structure that describes the portion of the IFC schema needed for this unit of functionality. The ConceptTemplate element contains:

- @applicableSchema, such as IFC2X3 or IFC4,
- @applicableEntity, the root entity of the concept template, often IfcObject, or a subtype, like IfcProduct, or IfcGroup, IfcProject and other high level entities, deriving from IfcRoot
  - NOTE: In case of partial concept templates, which are reused at many concept templates, may have root entities that do not derive from IfcRoot. An example is the partial concept template of swept solid geometry with a root element IfcSweptAreaSolid, which is reused in several other concept templates describing element shape representation
- Description, a general element to include potentially multilingual documentation and links to figures, diagrams, examples and other external documents. It is mainly used for the mvdXML purpose of generating MVD documentation
- Rules, a list of attribute definitions, being direct attribute or relationships of the root entity, or attributes defined at the level of its subtypes, that are part of the concept template tree
  - NOTE: In many cases, the inverse attribute is used here to navigate to relationship entities
  - EXAMPLE: a common example of an attribute rule that relate to an attribute that is only defined at a subtype level is PredefinedType.
- SubTemplates, a concept template that extents the definition of the main concept, it is used to group related concept templates, e.g. all concept templates that relate to element geometry may have a common parent concept template “Product Geometric Representation”, and then extent to box geometry, footprint geometry and body geometry.
  - NOTE: If a template with subtemplates is used in an exchange requirement, then the applicableEntity decides which template is used for model checking.
  - EXAMPLE: A template is defined for IfcSimpleProperty with subtemplates for IfcPropertySingleValue and IfcPropertyEnumeratedValue. It is referenced by a Concept for checking properties. If an instance of IfcPropertySingleValue is to be checked, then the template with best matching applicableEntity is selected from the subtemplates.

The tree structure of the Rules section at ConceptTemplate consists of AttributeRules, referring to EntityRule, referring to AttributeRules, and so on. Each AttributeRule has:

- an @AttributeName, the name of the attribute, relationship or inverse relationship in the IFC schema
- an @RuleID, if present, it defines an ID which is used in the model view definition to document specific usage for particular entities, or to validate its values according to exchange requirements,
- a Constraints, a list of Constraint on the schema population, if used for this concept template

---

1 See: http://www.buildingsmart-tech.org/ifc/IFC4/Add1/html/link/property-sets-for-objects.htm
EXAMPLE: a concept template for swept solid geometry would enforce, that the value of
RepresentationType of entity IfcShapeRepresentation is always “SweptSolid”, independently of its particular
usage in a model view definition later. This can be encoded as a Constraint. Similarly the cardinality of sets
or lists might be constrained.

- EXAMPLE: a concept template for swept solid geometry would enforce, that the value of
RepresentationType of entity IfcShapeRepresentation is always “SweptSolid”, independently of its particular
usage in a model view definition later. This can be encoded as a Constraint. Similarly the cardinality of sets
or lists might be constrained.

- NOTE: the Constraints are a way to flexibly enhance the WHERE rules within the EXPRESS definition of
IFC, and to add such rules, when using ifcXML (that cannot include such WHERE rules).

- an EntityRules element, containing a list of EntityRule, relating to the underlying type of the attribute

An EntityRule refers to an entity, an enumeration, a derived or simple type (based on the EXPRESS
definition of IFC). Each EntityRule has:

- an @EntityName, the name of the underlying type
  - NOTE: it shall not be a SELECT type, those have to be expanded to the selected types
- an @RuleID, see above
- a Constraints, see above
- an AttributeRules element, containing a list of AttributeRule, relating to the attributes, relationships or
  inverse relationships, if the EntityRule represents an entity itself
- a References element, if present, it links to a partial concept template that shall be used to expand the
  concept template further.
  - EXAMPLE: the definition of a property set is used in different concept templates, for property sets on
    occurrences and for property sets on types. Hence it can be defined once as a partial concept template that
    is referenced from the main concept template through References.
  - NOTE: The underlying type of the EntityRule, defined by the EntityName attribute, and the applicableEntity
    of the referenced template should be the same.

The mvd:ModelView element describes how the concept templates are used in a view and contains:

- @applicableSchema, such as IFC2X3 or IFC4,
- mvd:BaseView definition if it is an add-on view that only defines restrictions or extensions on top of
  another model view definition,
- mvd:ExchangeRequirements, a list of mvd:ExchangeRequirement, that stipulate if the template rules
  imposed on concepts, declared for each ConceptRoot, have to be fulfilled for the individual exchange
  requirements,
- mvd:Roots, a list of mvd:ConceptRoot, that defined the concepts applicable to each entity instance in
  an IFC data set together with the template rules
Figure 4: main mvdXML elements for ModelView

An mvd:ConceptRoot references a specific IFC entity, e.g. IfcWall, representing a major and individually testable model element\(^3\) in a MVD. Each concept root contains

- **mvd:Concepts**: a set of concepts, mvd:Concept, which describe template rules for common subsets of information (e.g. material usage) within the context of the particular concept root.
  - **mvd:Template**: Each concept is backed by a template, mvd:ConceptTemplate, describing a graph of object instances, relationships, and constraints, where the concept may provide a set of template rules containing the parameters that apply to the referenced rule ID at the concept template. The mvd:Template provides a link, based on the uuid, to that mvd:ConceptTemplate.
  - **mvd:Requirements**: a list of mvd:Requirement, each linking to the mvd:ExchangeRequirement by uuid, to declare that this Concept is stipulated for this exchange.
  - **mvd:TemplateRules**: a tree of mvd:TemplateRule that creates a Boolean logic between individual template rules (applying and, or, and other Boolean operators). The outermost TemplateRules element has to validate to true for this concept to pass validation.
  - **mvd:Applicability**: a list of TemplateRules with a link to the applicable ConceptTemplate via the Template element. It optionally applies additional constraints on the applicable entity that needs to be fulfilled by the entity instance before the Concepts are validated.

NOTE The elements Applicability and the tree structure of TemplateRules are the main extensions of mvdXML1.1 (since Release Candidate) in order to support the purpose “support automated validation of IFC data sets for quality assurance”.

---

\(^3\) The IFC schema differentiates between root entities, all entities derived from IfcRoot, and resource entities, all other entity definitions. A resource entity shall always be used (referenced) by a root entity. Therefore, entities defined in an mvd:ConceptRoot should be an IFC root entity.
2.2 Simple example

The following simple example shows the use of mvdXML for validation purposes. It defines a necessary concept template describing the unit of functionality of how to associate a port to a distribution element in IFC, and a hypothetical model view definition, that enforces that every sensor within submitted IFC data complying with the MVD shall have at least one port that submits signals.

Header section:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<mdvXML
    name="example MVD for mvdXML documentation – sensor signals"
    uuid="4af8ba8b-0b61-4ff8-9863-c10690fe06f2"
    xmlns="http://buildingsmart-tech.org/mvd/XML/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://buildingsmart-tech.org/mvd/XML/1.1 ../mvdXML_V1.1.xsd">

A single mvdXML element shall be the single root element within an mvdXML file. The name space of the current mvdXML version shall be http://buildingsmart-tech.org/mvd/XML/1.1. The schema location is provided locally.

```xml
<Templates>
    <ConceptTemplate uuid="bafe93b7-d0e2-42d8-84cf-5da20ee1480a"
        name="Port Assignment" applicableSchema="IFC4"
        applicableEntity="IfcDistributionElement">
        <Definitions>
            <Definition>
                <Body><![CDATA[
                Distribution ports are defined by <i>IfcDistributionPort</i> and attached by the <i>IfcRelNests</i> relationship. Ports can be distinguished by the <i>IfcDistributionPort</i> attributes <i>Name</i>, <i>PredefinedType</i>, and <i>FlowDirection</i>:
                </i></i>]]>
            </Definition>
        </Definitions>
    </ConceptTemplate>
</Templates>

A single ConceptTemplate is declared with the name “Port Assignment”, based on the schema definition of IFC4. The applicable entity, and root entity of the concept template, is IfcDistributionElement. The definition of the attribute rules start from that root entity.

```xml
<Rules>
    <AttributeRule AttributeName="IsNestedBy">
        <EntityRules>
            <AttributeRule AttributeName="RelatedObjects">
                <EntityRules>
                    <AttributeRule AttributeName="IfcDistributionPort">
                        <AttributeRules>
                            <AttributeRule AttributeName="Name" RuleID="Name"/>
                            <AttributeRule AttributeName="PredefinedType" RuleID="Type"/>
                            <AttributeRule AttributeName="FlowDirection" RuleID="Flow"/>
                        </AttributeRules>
                    </EntityRule>
                </EntityRules>
            </AttributeRules>
        </EntityRules>
    </AttributeRule>
</Rules>
```
The concept template tree follows the IFC definition for that portion that is required to describe, how a distribution port is associated to a distribution element. It is defined by a path starting from the applicable entity.

- **IfcDistributionElement** –IsNestedBy → **IfcRelNests** –RelatedObjects → **IfcDistributionPort**

At the related **IfcDistributionPort**, the three necessary attributes Name, PredefinedType and FlowDirection are declared. Since the instantiation of those attributes shall be checkable, they each have a @RuleID. The @RuleID attributes shall be unique within the scope of its usage in a ConceptTemplate. The validation rules at the individual concepts use the @RuleID strings as variable names within the formal grammar.

A single Model View Definition MVD is defined within the mvdXML file, it is applicable to the IFC4 schema and has a name and code. In case of fully defined and published MVD’s, the name is the full name as published, and the code is the same abbreviation, as used in the IFC HEADER Section.

EXAMPLE: name="IFC4 Reference View Version 1.0" and code="IFC4 RV V1.0" see official documentation

For the MVD there is one exchange requirement defined. Each exchange requirement has an own selection of validation rules, so that data requirements and data completeness can be described specifically for an exchange.

NOTE: An example for an exchange requirement is the import requirements for a BIM usage or purpose.

---

4 http://www.buildingsmart-tech.org/specifications/ifc-view-definition/ifc4-reference-view
A sensor is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument.

The MVD (or the MVD part that shall be validated) consists of a single root concept, i.e. a single IFC model element (or occurrence object), here IfcSensor.

The ConceptTemplate "Port Assignment", which is applicable to IfcSensor, since IfcSensor is a subtype of the @applicableEntity IfcDistributionElement, for which the ConceptTemplate is declared, is assigned as a Concept with the same name “Port Assignment”.

The assignment is declared using an ID/IDREF pair, based on the uuid for the ConceptTemplate.

NOTE: If using mvdXML for MVD definition and documentation purposes, this statement means that any implementation of IfcSensor has to support the functionality to assign ports to the sensor in order to comply with the requirements of that MVD. A certification process for that MVD would impose tests to make sure that ports are assigned to sensors for import and/or export.

Reference to the exchange requirement where additional constraints apply to the data provided for the Concept "Port Assignment". The link is declared using an ID/IDREF pair, based on the uuid for the ExchangeRequirement. The logical results created by the TemplateRules are interpreted following the @requirement attribute.

EXAMPLE: The requirement="mandatory" stipulates, that the outcome of the single outermost TemplateRule shall be true, otherwise an error is reported.

A rule, referring to the template definition, hence the name “TemplateRule” is imposed, and an mvdXML compliant validator would check that each instance of IfcSensor in the IFC file would have an assigned IfcDistributionPort with the attributes and corresponding values Name="Output", Type="SIGNAL", and Flow="SOURCE".

NOTE: The parameter syntax of mvdXML 1.0 is using a semicolon between the parameters for defining AND combination. The use of semicolon is still supported, but it is recommended to use AND instead. The metric
[Value] checks the value of the attribute. It is the default metric, therefore Name='Output' is identical to Name[Value]="Output".

```xml
  </Concept>
  </Concepts>
  </ConceptRoot>
  </Roots>
  </ModelView>
  </Views>
  </mvdXML>
```
3 Description of mvdXML schema elements and types

This section documents the individual mvdXML element, type and attribute definitions.

3.1 mvdXML

This element comprises the scope of the mvdXML document; it includes zero-to-many mvd:ModelView and zero-to-many mvd:ConceptTemplate (as a minimum, all concept templates that are referenced in the included model view(s)).

It is recommended to include all concept templates that are referenced by included model view(s), or else distribute the mvdXML file along with other mvdXML files containing such templates.

NOTE: A particular usage of mvdXML is to publish concept templates only. In this case, the ModelView element remains empty.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity attributes</td>
<td>See section 3.4.1</td>
<td></td>
</tr>
<tr>
<td>Templates</td>
<td>ConceptTemplate [0:?]</td>
<td>Set of templates, which may be exchanged with or without referencing model view definitions.</td>
</tr>
</tbody>
</table>
| Views | ModelView [0:?] | List of model view definitions, in order of listing in generated documentation.  
  - If empty, the mvdXML file is only used to exchange concept templates and cannot be used to fulfill other purposes such as data validation |

Table 1: Common element references defined in the element mvdXML.

3.2 Concept Template

This element represents the reusable concepts as templates; it has zero-to-many mvd:SubTemplates and thereby may form a tree of related reusable concept templates. Within the tree it may refer to shared partial concepts. Each mvd:ConceptTemplate has an applicable schema and may have applicable root entities (i.e. concept roots to which the mvd:ConceptTemplate applies).

NOTE: For buildingSMART compliant MVD documentation generation, each mvd:ConceptTemplate appears in Chapter 4 of the resulting HTML based documentation, with descriptive text and diagram generated from rules.

EXAMPLE: Decomposition (the re-usable concept of decomposing elements into parts)
### @applicableEntity

String [0:?]

Indicates the IfcRoot-based entities, including all derived entities, for which the concept applies. It is recommended to use a single base class (e.g. IfcElement). This value provides the context for any attribute rules and is used within MVD tools to filter the list of available templates for particular entities. For a sub-template, the applicable entity must be the same type or a subtype of the outer template. This value may be blank to indicate an abstract template that cannot be instantiated, containing sub-templates for specific entities.

### @isPartial

Boolean (opt)

A flag, indicating whether the concept template is a partial template, which shall only be used inside another concept template, or not.

### SubTemplates

ConceptTemplate [0:?]

Set of sub-templates, having a subset of applicable entities, which further define a concept template for particular usage. For example, a template for material usage may have sub-templates for material layer sets, material profile sets, and material constituent sets.

NOTE: Sub-templates have to repeat rule definitions from super-templates in case they apply. Further restrictions can be added if necessary. If rules are not repeated, they do not apply for the sub-templates.

### Rules

AttributeRule [0:?]

Set of attributes defined at applicableEntity, where each attribute may have value constraints and/or graphs of object instances defined. If an attribute is not defined, then the requirements are the same as indicated for the schema.

[v1.0] Restricted to AttributeRule by informal agreement for uniform usage.

[v1.1] Schema changed to enforce AttributeRule.

NOTE: For each attribute there should be no more than one AttributeRule.

NOTE: It is allowed to define rules for attributes that are defined in a subtype of applicableEntity. This feature can be used in IFC for instance for the PredefinedType attribute defined for each subtype of IfcElement.

NOTE: For generating a subset schema it is mandatory to add an AttributeRule for each optional attribute that shall be included in the subset schema. Otherwise this attribute will be removed.

### Table 2: Common attributes and element references defined in the element ConceptTemplate.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@AttributeName</td>
<td>String</td>
<td>The case-sensitive name of the attribute relative to the enclosing EntityRule (if exists) or the enclosing applicableEntity of the ConceptTemplate.</td>
</tr>
<tr>
<td>@RuleID</td>
<td>String (opt)</td>
<td>Identifies the rule for referencing at template rules defined within concepts, where specific parameters are applied for this rule.</td>
</tr>
<tr>
<td>@Description</td>
<td>String (opt)</td>
<td>Optional description of the rule.</td>
</tr>
</tbody>
</table>
### Entity Rules

- **EntityRule**
  - `[0:?]`
  - An empty list indicates that any type may be used according to the schema. If one or more entities or types are defined, then instances must match one of the entries. The list of entries is expanded by each referencing **TemplateRule** defined at a **Concept**, where downstream rules apply according to the matching entity rule.

  **EXAMPLE:** An attribute rule “Quantities” for **IfcElementQuantity** could add entity rules for **IfcQuantityLength**, **IfcQuantityArea**, **IfcQuantityVolume** etc. Each entity rule then defines an own scope depending on the referenced quantity type.

### Constraints

- **Constraint**
  - `[0:?]`
  - Set of expressions, which all must evaluate to **True** for the referenced attribute. This implies a Boolean AND combination.

### 3.2.2 Entity Rule

This element represents the specification of an entity (or value type) referenced by an attribute, either as a scalar reference or a reference from within a collection.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@EntityName</td>
<td>String</td>
<td>The case-sensitive name of the entity (e.g., “IfcBeam”) which must be assignable to the enclosing <strong>AttributeRule</strong> (i.e., entity subtype or select member).</td>
</tr>
</tbody>
</table>
| @RuleID              | String (opt) | Identifies the rule for referencing at template rules defined within concepts, where specific parameters are applied for this rule.  
                         **NOTE:** The same **RuleID** might be used multiple times within a concept template definition, but it must be unique within the scope of its usage. See also description of **AttributeRule.@RuleID**. |
| @Description         | String (opt) | Optional description of the rule. |
| References           | ConceptTemplate [0:1] | Optional reference to a partial template. An optional attribute “IdPrefix” can be given to ensure that RuleIDs of partial templates are unique within the scope of its usage. This attribute is used as a prefix for all referenced RuleIDs. |
| AttributeRules       | AttributeRule [0:?] | Indicates a list of attributes included in the concept template and potentially constrained on the referenced entity. |
| Constraints          | Constraint [0:?] | Set of expressions, which all must evaluate to **True** for the referenced entity. This implies a Boolean **AND** combination. |

### 3.2.3 Constraint

This element is defined within the elements **mvd:EntityRule** and **mvd:AttributeRule** and represents a restriction on an attribute, which may require the value, type, or collection size to have equality (or other comparison) to a literal value or referenced value.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Expression           | String        | [v1.1] A grammar is used to simplify parsing of expressions and to introduce new features like **AND**, **OR** and **XOR**. With minor changes the form "(Metric)(Operator)(Benchmark)" from **mvdXML 1.0** is still valid.  
                         The rule grammar is defined in chapter 4.  
                         **NOTE:** One major difference to 1.0 is that it is not possible to use (Benchmark) only, i.e. to omit (Metric) and (Operator). |

Table 3: Common attributes and element references defined in the element **AttributeRule**.

Table 4: Common attributes and element references defined in the element **EntityRule**.

Table 5: Common attributes defined in the element **Constraint**.
3.3 Model View

This element represents the description of a Model View Definition (MVD); it is specific to an IFC schema release and contains zero-to-many mvd:ConceptRoot elements. It also includes the reference to zero-to-many applicable mvd:ExchangeRequirement elements. Multiple model views from potentially different schema releases may be contained in the same file.

The set of entities and types regarded to be within scope of a model view is not explicitly defined; rather it is indirectly determined by constructing a graph of mvd:ConceptRoot elements and following the set of rules indicating referenced entities within scope. Thus, describing the set of rules automatically determines what is in or out of scope, preventing the possible mismatch of missing data structures that are required, or included data structures that are not documented for use.

EXAMPLE: The "CoordinationView_V2.0" is a Model View Definition; it is captured by an mvd:ModelView element. It has the @name="CoordinationView_V2.0", the @applicableSchema="IFC2X3", and a reference to the four exchange requirements currently defined for the Coordination View Version 2.0.

HISTORY: Roots changed to optional in mvdXML 1.1 to allow "incomplete" model view definitions with meta-data only.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity attributes</td>
<td></td>
<td>See section 3.4.1</td>
</tr>
<tr>
<td>Definitions</td>
<td></td>
<td>See section 3.4.2</td>
</tr>
<tr>
<td>applicableSchema</td>
<td>String</td>
<td>Identifies the schema using the ISO 10303 schema identifier, such as IFC2X_FINAL, IFC2X2_FINAL, IFC2X3, or IFC4. NOTE: In future versions it might be of interest to support more than one IFC release. This can be supported by using a semicolon as schema name delimiter.</td>
</tr>
<tr>
<td>BaseView</td>
<td>uuid</td>
<td></td>
</tr>
<tr>
<td>Exchange Requirements</td>
<td>Exchange Requirement [0:?]</td>
<td>List of exchange requirements defined within this model view. They should appear in logical order.</td>
</tr>
<tr>
<td>Roots</td>
<td>ConceptRoot [0:?]</td>
<td>List of root concepts defined within scope of the model view.</td>
</tr>
</tbody>
</table>

Table 6: Common attributes and element references defined in the element ModelView.

3.3.1 Exchange Requirement

This element is the description of an Exchange Requirement Model (ERM) that is covered by the MVD. An ERM covers the Exchange Requirements (ER) that are identified for a particular exchange scenario that is covered by the MVD. ERM's may add additional constraints to the use of concepts and are an important part of later certification and validation processes.

An ERM can be referenced from an mvd:Concept to impose specific constraints for exchanges that reference this ERM. An ERM can be specifically declared to be only applicable for import, export or both scenarios using the attribute applicability.

EXAMPLE: The ERM "Architecture" capturing the ER for exporting an architectural building model is an exchange requirement model within the CoordinationView_V2.0. It is captured by an mvd:ExchangeRequirement element. It has the @name="Architecture", and the @applicability="export".

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity attributes</td>
<td></td>
<td>See section 3.4.1</td>
</tr>
</tbody>
</table>


### Definitions

| applicability | Enum (opt) | Identifies if the ERM is specific for  
| | | • import  
| | | • export  
| | | • both  

If such value is provided, then any referencing requirements must match; for example, if such value indicates export, then referencing requirements may use export but not import; if such value is not provided, then referencing requirements may use any value.

NOTE: The differentiation between import and export origins from software certification and does not have any meaning for data checking applications.

Export means that some application must be able to create a data set that fulfills defined requirements.

If an exchange requirement is defined for import only, it defines the data set that must be properly processed by an application.

Table 7: Common attributes defined in the element ExchangeRequirement.

#### 3.3.2 Concept Root

This element represents the root element (other terms are "leaf node class", "variable concept") that represent the fundamental parts of an MVD that is represented by a collection of supported concepts.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identity attributes</td>
<td></td>
<td>See section 3.4.1</td>
</tr>
<tr>
<td>Definitions</td>
<td></td>
<td>See section 3.4.2</td>
</tr>
</tbody>
</table>
| applicable RootEntity | String | Identifies the class or data type of instance being described or validated, i.e. the IFC entity (deriving from IfcRoot) for which the concepts apply. The concepts apply to this IFC entity or its subtypes (respectively instances of those classes in case of validation).  

NOTE that non-rooted entities are described by referencing rules, as such instances cannot exist on their own where usage is always dependent upon the referencing IfcRoot-based instance. |

| Applicability | Applicability | A set of TemplateRules, based on a concept template, which describe the conditions, under which the concepts apply to the applicableRootEntity. Those conditions need to validate to true as a prerequisite for checking the TemplateRules imposed at the concepts.  

NOTE the Applicability has been added to mvdXML1.1 in order to better support data validation. It is used to control the applicability of concepts to particular configurations of root entities, e.g. to only apply for load bearing walls, instead of any wall (declared by the applicable IFC entity IfcWall). |

| Concepts | Concept [0:?] | List of concepts for the applicable root entity. The order of elements indicates the sequence displayed in generated documentation. |

Table 8: Common attributes and element references defined in the element ConceptRoot.

#### 3.3.3 Applicability

This element defines those rules, being TemplateRules with a reference to a ConceptTemplate, that need to be validated before the concepts associated to the ConceptRoot are checked.
### 3.3.4 Concept

This element represents a use definition for a particular entity with specific rules to be enforced.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definitions</td>
<td></td>
<td>See section 3.4.2</td>
</tr>
<tr>
<td>Template</td>
<td>TemplateRef</td>
<td>Mandatory reference to the ConceptTemplate by uuid, where such template may be defined within the same file (by @ref) or an external file (by @href). NOTE Current usage of mvdXML imposes the inclusion off all concept templates within the same data file. The external reference by @href is reserved for future usage.</td>
</tr>
<tr>
<td>TemplateRules</td>
<td>TemplateRules</td>
<td>Tree structure of rules indicating how template applies to particular entity. Each TemplateRules element consists of a set of other TemplateRules or TemplateRule element and a logical operator. Each TemplateRule element defines the @Parameter that refer to the RuleID of the referenced Template. NOTE Added in mvdXML to define any logical combination of rules, allowing for AND, OR, NOT, NAND, NOR, XOR, and NXOR logic.</td>
</tr>
</tbody>
</table>

### Table 9: Common element references defined in the element Concept.

NOTE: The following options are possible for using BaseConcept and Override:

- an empty concept with BaseConcept="idref" (Override="false") to indicate that it applies with no change
- an non-empty concept with BaseConcept="idref" (Override="false") to indicate that it has additional rules (by extension)
- an empty concept with BaseConcept="idref" and Override="true" to indicate that it does not apply at all (overridden)
- an non-empty concept with BaseConcept="idref" and Override="true" to indicate that it has replacement rules (by restriction - no inherited rules apply, all are declared new)

**EXAMPLE:**
- a rule for property sets may list each applicable property set; a rule for ports may list the name, type, and direction of each port.
- For LIST types, multiple rules combined by an AND operator indicate a sequence of instances which must match the order of the rules. For SET types, multiple rules indicate a set of instances which ALL must be included according to the rules.
- For SELECT, ENTITY, and all other types, multiple rules combined by an OR operator indicate any valid state (it is valid for the attribute to reference objects that match one of the elaborated configurations).

### 3.3.5 Requirement

This element represents a use definition for a particular entity with specific rules to be enforced.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@exchange</td>
<td>Requirement</td>
<td>Identifies the <em>ExchangeRequirement</em> by GUID within the same Model View Definition.</td>
</tr>
</tbody>
</table>
| @requirement      | Enum      | Describes the interpretation of the result of the outermost *TemplateRule* specific for one exchange requirements.  
|                   |           | - **mandatory**: must be true, otherwise create an error  
|                   |           | - **recommended**: should be true, otherwise create a warning  
|                   |           | - **not-relevant**: no requirement;  
|                   |           | - **not-recommended**: should not be true, otherwise create a warning  
|                   |           | - **excluded**: must not be true, otherwise create an error                     |
| @applicability    | Enum (opt) | Identifies if the requirement applies to  
|                   |           | - import  
|                   |           | - export or  
|                   |           | - both  

**NOTE** If such value is provided, then it must match, if given, with the applicability setting of the exchange requirement in which it is used.

<table>
<thead>
<tr>
<th>mvdXML 1.1</th>
<th>Meaning for validation</th>
<th>mvdXML 1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>mandatory</td>
<td>error, if outer template rule validates to false</td>
<td>mandatory</td>
</tr>
<tr>
<td>recommended</td>
<td>warning, if outer template rule validates to false</td>
<td></td>
</tr>
<tr>
<td>not-relevant</td>
<td>no reporting (no check needs to be executed)</td>
<td>optional, not-relevant</td>
</tr>
<tr>
<td>not-recommended</td>
<td>warning, if outer template rule validates to true</td>
<td></td>
</tr>
<tr>
<td>excluded</td>
<td>error, if outer template rule validates to true</td>
<td>excluded</td>
</tr>
</tbody>
</table>

**Table 10: Common element references defined in the element Requirement.**

Note: The different enumerators of @requirement have the following meaning, if mvdXML is used for validating IFC data sets. The table also shows the comparison with the mvdXML1.0 definitions.
Note: For a standard existence check (e.g. is the property value provided, it means: mandatory – error, if no value is provided, recommended – warning, if no value is provided, not-relevant – no check, not-recommended – warning, if a value is provided, excluded – error, if a value is provided.

### 3.3.6 TemplateRules

This element establishes the possibility to define a tree of logical expressions. Individual TemplateRule are grouped under a TemplateRules element and are logically interpreted by the @operator attribute.

NOTE: This improves the previous way to embed the logical operator in the @Parameter string at the TemplateRule. Due to its tree structure realized by the recursive definition of TemplateRules, the logical operators can be nested.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td></td>
<td>See section 3.4.2</td>
</tr>
</tbody>
</table>
| @operator         | Enum  | The logical operator, which is used to combine the nested TemplateRules and TemplateRule. The Boolean results of the nested rules are combined by the logical operation according to the Truth table. The following logical operators are defined:  
  - AND  
  - OR  
  - NOT  
  - NAND  
  - NOR  
  - XOR  
  - NXOR  
  NOTE: The valid number of the nested elements depend on the value of the @operator. E.g. for the operator NOT, only one nested element shall exist. |

Table 11: Common attributes defined in the element TemplateRules.

The following truth tables are to be used with the @operator.

#### Table 12: Truth table for operator attribute

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A AND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A OR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A NAND B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A NOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A XOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A XNOR B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
3.3.7 TemplateRule

This element represents an instantiation of a rule with specified parameters. It allows repetitive definitions to be efficiently represented, such as lists of applicable ports, materials, units, property sets, etc. The @RuleID used in the Parameters of the template rule serves as a reference to the @RuleID of an mvd:AttributeRule or mvd:EntityRule at the referenced mvd:ConceptTemplate to be instantiated.

If the referenced mvd:EntityRule is part of a SET-based attribute, then the instance is required to uniquely exist once (having unique combination of defined parameters), but without regard for order (as a SET has no implied order). If the referenced mvd:EntityRule is part of a LIST-based attribute, then the instance is required to occur at the relative position of the mvd:TemplateRule.

<table>
<thead>
<tr>
<th>Element/Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| Parameters        | String   | [v1.1] mvdXML introduces a grammar definition for the Parameters string that is harmonized with the Expression string of mvd:Constraint. With minor changes the form “{Parameter}={Value};” from mvdXML 1.0 is still valid. In the new grammar the Parameters string is defined by expression where each boolean_term requires a parameter. The parameter corresponds to the RuleID of an AttributeRule or EntityRule at the referenced ConceptTemplate. The operator of the grammar is now more flexible and not only supports Equals as in mvdXML 1.0. The Value is also enhanced. It now supports the use of a parameter, not only a value. This enables to replace the agreement for the definition of parameter values using a ‘#’ sign. This agreement is no longer supported. Finally, each expression can be grouped and combined through AND, OR and XOR logic. 

NOTE: The differentiation between conditions and constraints as used in mvdXML 1.0 is no longer available. |

Table 13: Common attributes defined in the element TemplateRule.

3.4 Common type and attribute definitions

The mvd:Definition, and elements referenced by the element Definition, mvd:Body, and mvd:Link elements provide the capability to add multi-lingual descriptions at any element with own identity. Such elements are:

- mvd:ModelView
- mvd:ExchangeRequirement
- mvd:ConceptTemplate
- mvd:ConceptRoot
- mvd:Concept

The information provided by this element is mainly used for documentation purposes, in particular to generate HTML documentation as used by buildingSMART for the IFC data model.

3.4.1 Identity

Similar to IFC, the mvdXML schema makes a distinction between elements having identity and those that do not. All elements with identity have the following attributes and sub elements defined. The information provided in this attribute group is used for management purposes.

NOTE: The mvdXML.xsd does not incorporate an mvd:Identity abstract class, the common attributes are defined in the attributeGroup name="identity" and the definitions in the element name="Definition".
HISTORY: The attribute ‘status’ has been changed from string to an enumeration that includes the previously recommended values in mvdXML 1.1.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uuid</td>
<td>uuid</td>
<td>Universally unique identifier. This is used as a persistent identifier, and must never change. It is string type with a fixed length of 36 characters, which should follow a specific pattern.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Human readable name. This is used as the header of the section and entry within table of contents when generating documentation. The name is also reported for a validation against this MVD, if assigned to concepts checked against the MVD.</td>
</tr>
<tr>
<td>code</td>
<td>String (opt)</td>
<td>Human readable reference value of this element of the MVD definition</td>
</tr>
<tr>
<td>version</td>
<td>String (opt)</td>
<td>Sequential version number of this element of the MVD definition.</td>
</tr>
<tr>
<td>status</td>
<td>enumeration base: String (opt)</td>
<td>The status information of this element of the MVD definition. It has the following enumerators: Sample, Proposal, Draft, Candidate, Final, Deprecated</td>
</tr>
<tr>
<td>author</td>
<td>String (opt)</td>
<td>The author(s) of his element of the MVD definition. Authors are separated by semicolon.</td>
</tr>
<tr>
<td>owner</td>
<td>String (opt)</td>
<td>The legal owner of this element of the MVD definition.</td>
</tr>
<tr>
<td>copyright</td>
<td>String (opt)</td>
<td>The copyright under which the work is published.</td>
</tr>
</tbody>
</table>

Table 14: Common attributes defined in the attributeGroup identity.

3.4.2 Definition

The element mvd:Definition groups definition text and links to additional figures, diagrams, examples, and other external documents.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body</td>
<td>Body (opt)</td>
<td>HTML-formatted description of the concept in the default language.</td>
</tr>
<tr>
<td>Links</td>
<td>Link [0:?]</td>
<td>List of additional content, each of which may be in separate languages.</td>
</tr>
</tbody>
</table>

Table 15: Common element references defined in the element Definition.

3.4.3 Body

The element mvd:Body holds the definition text or explanatory remarks. It is qualified by a language tag. It also holds tags that further classify the nature of the definition or remark.

NOTE: In order to correctly encapsulate the HTML formatted text, the content shall be tagged by <![CDATA[   ]]> to preserve the HTML code.
HISTORY: tags attribute available since mvdXML 1.1.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lang</td>
<td>String (opt)</td>
<td>Locale identifier based on RFC 1766 language codes to indicate the default locale. Examples are 'en', 'de', 'en-GB', 'de-CH'.</td>
</tr>
<tr>
<td>tags</td>
<td>String [0:?]</td>
<td>List of tags that classify the element. All tags are separated through whitespaces per default. A semicolon must be used if given tags consists of multiple words.</td>
</tr>
<tr>
<td>(content)</td>
<td>String</td>
<td>HTML-formatted content for generating documentation. Content within should be encapsulated by paragraph tags (&quot;&lt;p&gt;&quot;) and/or list tags (&quot;&lt;ul&gt;&quot;). Images should not be contained within; rather they should be specifically referenced by the Link element (allowing for automatic figure numbering).</td>
</tr>
</tbody>
</table>

Table 16: Common attributes defined in the element Body.

3.4.4 Link

The element mvd:Link holds all links to additional documentation content.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lang</td>
<td>language</td>
<td>Locale identifier based on RFC 1766 language codes to indicate the default locale. Examples are 'en', 'de', 'en-GB', 'de-CH'.</td>
</tr>
<tr>
<td>title</td>
<td>String (opt)</td>
<td>Human readable name. This is used as the header of the link content and entry within table of contents when generating documentation</td>
</tr>
<tr>
<td>category</td>
<td>enumeration</td>
<td>Indication about the category of the linked content. definition: formatted as documented definition in alternate locale agreement: formatted as NOTE in documentation diagram: formatted as custom figure in documentation based on href instantiation: formatted as instance diagram figure based on href example: formatted as EXAMPLE in documentation based on href</td>
</tr>
<tr>
<td>href</td>
<td>anyURI</td>
<td>URL to referenced content, particularly for diagrams and examples that are manually generated. This is used to reference any external files such that they are included when generating documentation. NOTE: URL's local to the file system shall be relative.</td>
</tr>
<tr>
<td>(content)</td>
<td>String</td>
<td>HTML-formatted description in specified language.</td>
</tr>
</tbody>
</table>

Table 17: Common attributes defined in the element Link.
4 Rule Grammar

The grammar for parsing expression strings is defined below. It is used to specify:

- `mvd:TemplateRule.Parameters`, and
- `mvd:Constraint.Expression`.

```
grammar mvdXMLv1_1;
/*---------------- * PARSER RULES *----------------*/
expression    : boolean_expression    ;
boolean_expression    : boolean_term (logical_interconnection boolean_term)*  ;
boolean_term             : ( parameter ( metric )? | metric ) operator ( value | parameter ( metric )? )  
                         | ( LPAREN boolean_expression RPAREN );
parameter               : SIMPLEID    ;
metric                  : '[Value]' | '[Size]' | '[Type]' | '[Unique]' | '[Exists]';
logical_interconnection : AND | OR | XOR | NAND | NOR | NXOR ;
operator                : EQUAL | NOT_EQUAL | GREATER_THAN | GREATER_THAN_OR_EQUAL | LESS_THAN | LESS_THAN_OR_EQUAL;
value                  : logical_literal | real_literal | string_literal | regular_expression;
logical_literal        : FALSE | TRUE | UNKNOWN ;
real_literal            : (sign)? (DIGIT | INT) (',')? ( (DIGIT | INT) )? ( 'e' (sign)? (DIGIT | INT) )? ;
string_literal          : STRING ;
regular_expression      : 'reg' STRING ;
sign                    : '+' | '-' ;
/*----------------* LEXER RULES *----------------*/
AND                      : 'AND' | 'and' | '&' | ';' ;
OR                      : 'OR' | 'or' | '|' ;
XOR                     : 'XOR' | 'xor' ;
NAND                    : 'NAND' | 'nand' ;
NOR                     : 'NOR' | 'nor' ;
NXOR                    : 'NXOR' | 'nxor' ;
EQUAL                   : '=' ;
NOT_EQUAL              : '!=' ;
GREATER_THAN            : '>' ;
GREATER_THAN_OR_EQUAL   : '>=' ;
LESS_THAN               : '<' ;
```
The following tables describe more details about the meaning of keywords.

<table>
<thead>
<tr>
<th>metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Indicates the value of the attribute (value uses syntax according to the attribute type, defined below).</td>
</tr>
<tr>
<td>Size</td>
<td>Indicates the size of a collection or STRING (value is an INTEGER).</td>
</tr>
<tr>
<td>Type</td>
<td>Indicates the type of the value assigned to the attribute (value is a STRING).</td>
</tr>
<tr>
<td>Unique</td>
<td>Indicates whether value must be unique within the population of instances described within the Concept Template (BOOLEAN).</td>
</tr>
</tbody>
</table>

Table 18: Description of metric values.

<table>
<thead>
<tr>
<th>operator</th>
<th>XML Escaped</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>=</td>
<td>Equal.</td>
</tr>
<tr>
<td>!=</td>
<td>!=</td>
<td>Not Equal.</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Greater Than.</td>
</tr>
<tr>
<td>&gt;=</td>
<td>&gt;=</td>
<td>Greater Than Or Equal.</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Less Than.</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&lt;=</td>
<td>Less Than Or Equal</td>
</tr>
</tbody>
</table>

Table 19: Description of operators.

Benchmark may be either a literal value or a parameter. The syntax of literal values varies according to the EXPRESS attribute type and follows ifcXML (ISO-10303-28) format:
<table>
<thead>
<tr>
<th>Type</th>
<th>Operators</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
<td>=, !=, &gt;, &gt;=, &lt;, &lt;=</td>
<td>The integer value.</td>
</tr>
<tr>
<td>REAL</td>
<td>=, !=, &gt;, &gt;=, &lt;, &lt;=</td>
<td>The real value including decimal point, where equality is exact (no epsilon offset).</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>=, !=</td>
<td>The boolean value as “true” or “false”.</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>=, !=</td>
<td>The logical value as “true”, “false”, or “unknown”.</td>
</tr>
<tr>
<td>ENUM</td>
<td>=, !=</td>
<td>The enumeration value by case-insensitive name.</td>
</tr>
<tr>
<td>STRING</td>
<td>=, !=, &gt;, &gt;=, &lt;, &lt;=</td>
<td>The string value, which may optionally be enclosed by single quotes (if escaping required). Comparison operators indicate alphabetical sorting (e.g. “&gt;=” can indicate “must start with” such as the scope of a classification reference, or earliest date/time).</td>
</tr>
<tr>
<td>BINARY</td>
<td>=, !=</td>
<td>The binary value encoded as hexadecimal prefixed by “%” and number of unused bits.</td>
</tr>
<tr>
<td>ENTITY</td>
<td>=, !=, &gt;, &gt;=, &lt;, &lt;=</td>
<td>The name of the entity type (e.g. “IfcWall”). Equality means exact type match; “&gt;” means subtype of; “&gt;=” means same type or subtype; “&lt;” means supertype of; “&lt;=” means same type or supertype.</td>
</tr>
</tbody>
</table>

Table 20: Description of operators that can be applied to different data types.
5 mvdXML Use Cases

This chapter describes typical use cases of mvdXML. Although the main structure of a Model View Definition is always the same there are differences regarding mandatory, optional and not relevant data. This review should enable to focus on features of mvdXML that are needed to support those use cases.

5.1 MVD Documentation

On basis of an mvdXML definition it is possible to generate a set of interlinked HTML files. The IfcDoc tool from buildingSMART for instance enables to export HTML files using the style of the IFC4 documentation (see Figure 5). The main focus of such documentation is to describe how a subset of IFC must be implemented and used to fulfil specific requirements. Such kind of information is needed by software developers for proper interpretation of the IFC specification.

The following parts of mvdXML are of main interest:

- mvd:Definition attached to mvd:ModelView, mvd:ExchangeRequirement, mvd:ConceptTemplate, mvd:ConceptRoot and mvd:Concept for storing definition text and links to additional figures, diagrams, examples, and other external documents

- mvd:ConceptTemplate definitions that enable to generate instance diagrams as shown in Figure 5

NOTE: For documentation purposes it is not necessary to specify an mvd:ConceptTemplate in full detail. Instead it is sufficient to focus on elements that shall be shown in an instance diagram. This is different to other use cases, in particular for data filtering and data validation, where all required elements must be defined.

5.2 Specification of subset schemas

A subset schema includes only those parts of IFC that are relevant for implementation. Such subset schema, which can be defined in EXPRESS or XML Schema, supports software implementation as it enables to generate software code for file parsing (ifc or ifcXML), data management and data serialization.
But a subset schema does not include additional constraints that define proper use of the subset schema. Therefore, the use of mvdXML can be simplified as it is only necessary to select entities and attributes that shall be part of the subset schema. Also, no documentation or other meta-data is required as it cannot be exported to EXPRESS or XML Schema.

5.3 Data Filtering

Data filtering creates a model subset and is similar to generation of a subset schema. But a data filter is working on instance level, i.e. with data instead of schema definitions. Many instances can be defined for a data type, but not all instances must be part of a Model View Definition. Accordingly, data filtering may require adding further conditions that enable to differentiate instances of the same data type. This is leading to more complex definitions. For example, a condition is necessary if space properties must be distinguished from wall properties in an IFC model, because both are defined by IfcPropertySet instances.

5.4 Data Validation

Data validation checks if a data set fulfils all constraints of an exchange requirement. If required data is missing or wrong then the check fails. In addition to data filtering it is therefore necessary to specify Concept.Requirements, i.e. to differentiate between mandatory and excluded data, and to restrict possible values of instances by using TemplateRule.Parameters of a Concept and Constraint.Expression of AttributeRule and EntityRule used by ConceptTemplate. Similar to generation of subset schemas and data filtering no documentation or other meta-data is required for this use case.
6 Glossary

6.1 ER

ER = Exchange Requirement; defines the data that is needed to fulfil a specific task. If not all mandatory data is available then the task cannot be carried out. Exchange requirement definitions are independent from a technical solution.

6.2 ERM

ERM = Exchange Requirement Model; implementation of an Exchange Requirement by an IT specification.

6.3 MVD

MVD = Model View Definition; defines a subset of an IT specification that is able to store data for a set of Exchange Requirements.
7 Examples

This sections shows a series of small mvdXML example files, used for the different purposes for which the mvdXML standard is developed.

7.1 Example for MVD documentation

The following example shows an mvdXML file to be used for documenting a model view definition, it is a direct output of the IFC document generator ifcDoc.

From this mvdXML file, in conjunction with the IFC schema definition, the following output is rendered.

![Graphical representation of the concept template “Nesting”](image)

Figure 6: Graphical representation of the concept template “Nesting”
Figure 7: graphical representation of the concept template “port nesting”

Figure 8: rendering of HTML tables to document the exchange requirements for the different ports
Figure 9: graphical representation of the usage of concept “port-nesting” at the root concept IfcHeatExchanger

<?xml version="1.0"?>
<mvdXML xmlns="http://buildingsmart-tech.org/mvd/XML/1.1" uuid="ch830d34-5696-4263-a7e9-2259ea343117" name="example 7.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://buildingsmart-tech.org/mvd/XML/1.1 ../mvdXML_V1.1.xsd">
  <Templates>
    <ConceptTemplate uuid="5098cd13-bf4b-473a-a846-a60f69e9b738" name="Object Composition" code=" applicableSchema="IFC4"
      applicableEntity="IfcObjectDefinition">
      <Definitions>
        <Definition>
Objects may be composed into parts to indicate levels of detail, such as a building having multiple storeys, a framed wall having studs, or a task having subtasks. Composition may form a hierarchy of multiple levels, where an object must have a single parent, or if a top-level object then declared within the single project or a project library.

A nesting indicates an external ordered part composition relationship between the hosting structure, referred to as the "host", and the attached components, referred to as the "hosted elements". The concept of nesting is used in various ways. Examples are:

- Nesting is used on product elements to indicate external connectable parts such as faucets mounted on a sink, or switches within a junction box.
- Nesting is used on control objects to indicate specification hierarchies.
- Nesting is used on process objects to indicate subordinate processes which may occur in parallel or in series.
- Nesting is used on resource objects to indicate subordinate resource allocations which may occur in parallel or in series.

Nesting is a bi-directional relationship, the relationship from the hosting structure to its attached components is called Nesting, and the relationship from the components to their containing structure is called Hosting.

Ports indicate possible connections to other objects according to specified system types, flow direction, and connection properties. Ports are typically connected between devices via cables, pipes, or ducts. Ports may have placement defined indicating the position and outward orientation of the port relative to the product or product type. Ports may also have material profile sets defined indicating the flow area and connection enclosure.
<AttributeRule AttributeName="IsNestedBy">
  <EntityRules>
    <EntityRule EntityName="IfcRelNests">
      <AttributeRules>
        <AttributeRule AttributeName="RelatedObjects">
          <EntityRules>
            <EntityRule EntityName="IfcDistributionPort">
              <AttributeRules>
                <AttributeRule RuleID="Name" Description="The name of the port." AttributeName="Name">
                  <EntityRules>
                    <EntityRule EntityName="IfcLabel" />
                  </EntityRules>
                </AttributeRule>
                <AttributeRule RuleID="Flow" Description="The flow direction of the port." AttributeName="FlowDirection">
                  <EntityRules>
                    <EntityRule EntityName="IfcFlowDirectionEnum" />
                  </EntityRules>
                </AttributeRule>
                <AttributeRule RuleID="Type" AttributeName="SystemType">
                  <EntityRules>
                    <EntityRule EntityName="IfcDistributionSystemEnum" />
                  </EntityRules>
                </AttributeRule>
                <AttributeRule AttributeName="ObjectPlacement">
                  <EntityRules>
                    <AttributeRule AttributeName="RelativePlacement">
                      <EntityRules>
                        <EntityRule EntityName="IfcAxis2Placement3D">
                          <AttributeRules>
                            <AttributeRule AttributeName="Location">
                              <EntityRules>
                                <EntityRule EntityName="IfcCartesianPoint" />
                              </EntityRules>
                            </AttributeRule>
                            <AttributeRule AttributeName="Axis">
                              <EntityRules>
                                <EntityRule EntityName="IfcDirection" />
                              </EntityRules>
                            </AttributeRule>
                          </AttributeRules>
                        </EntityRule>
                      </EntityRules>
                    </AttributeRule>
                  </EntityRules>
                </AttributeRule>
              </AttributeRules>
            </EntityRule>
          </EntityRules>
        </AttributeRule>
      </AttributeRules>
    </EntityRule>
  </EntityRules>
</AttributeRule>
<EntityRule>
  <EntityRules>
    <EntityRule EntityName="IfcDirection" />
  </EntityRules>
  <AttributeRule AttributeName="RefDirection">
    <EntityRule EntityName="IfcDirection" />
  </AttributeRule>
</EntityRules>

<AttributeRule AttributeName="IsDeclaredBy">
  <EntityRule EntityName="IfcDistributionPortTypeEnum" />
</AttributeRule>

<AttributeRule AttributeName="PredefinedType">
  <EntityRule EntityName="IfcDistributionPortTypeEnum" />
</AttributeRule>

<AttributeRule AttributeName="ApplicableSchema">
  <EntityRule EntityName="IfcDistributionPortTypeEnum" />
</AttributeRule>

<ExchangeRequirements>
</ExchangeRequirements>
<ExchangeRequirement uuid="a5846830-de9a-4195-9339-31169ecb7b0e" name="HVAC Sample Exchange" applicability="both"/>
</ExchangeRequirements>

<Roots>
  <ConceptRoot uuid="3ca6a49f-81c4-4010-9589-578cab9d4428" name="" applicableRootEntity="IfcHeatExchanger">
    <Concepts>
      <Concept uuid="875138d3-6911-4e3a-9369-2d273f331250" name="Port" override="false">
        <Template ref="bafc93b7-d0e2-42d8-84cf-5da20ee1480a"/>
      </Concept>
      <Requirements>
        <Requirement applicability="both" requirement="mandatory" exchangeRequirement="a5846830-de9a-4195-9339-31169ecb7b0e"/>
      </Requirements>
      <TemplateRules operator="and">
        <TemplateRule Description="Inlet of substance to be heated.
Parameters="Name[Value]="HeatingInlet' AND Flow[Value]='SINK' AND Type[Value]="NOTDEFINED'"/>
        <TemplateRule Description="Outlet of substance to be heated.
Parameters="Name[Value]="HeatingOutlet' AND Flow[Value]='SOURCE' AND Type[Value]="NOTDEFINED'"/>
        <TemplateRule Description="Inlet of substance to be cooled.
Parameters="Name[Value]="CoolingInlet' AND Flow[Value]='SINK' AND Type[Value]="NOTDEFINED'"/>
        <TemplateRule Description="Outlet of substance to be cooled.
Parameters="Name[Value]="CoolingOutlet' AND Flow[Value]='SOURCE' AND Type[Value]="NOTDEFINED'"/>
      </TemplateRules>
    </Concept>
  </Concepts>
</ConceptRoot>
</Roots>
</mvdXML>

7.2 Example for MVD validation

The following example shows an mvdXML file to be used to validate the completeness of IFC data in an IFC file. In the particular case, it checks that for every load bearing and external wall the following is true: a property set “Pset_WallCommon” with the following properties “FireRating” and “ThermalTransmittance” are provided, and that the direct optional attribute “PredefinedType” is given. It is permissible to either assign the properties on the occurrence object or at the type object.
Therefore a ConceptTemplate "Property Sets for Objects and Types" is included, that declares that part of the overall IFC structure, which is used to assign a property set to an occurrence object, a type object to an occurrence object, and a property set to the type object. Since the definition of property set (and the referenced definition of the property) is used twice (for occurrence and type object) it is declared as a partial concept template and referenced twice from the main concept template.

As the stipulated completeness checks for “FireRating”, “ThermalTransmittance” and “PredefinedType” are only applicable to those walls, represented by IfcWall or its subtypes, that are external and load bearing, a separate applicability check is performed as a precondition, before validating the rules themselves.

Each rule checking for the provision of the properties “FireRating” and “ThermalTransmittance” need to check, whether they are assigned to the occurrence (instance of IfcWall) or the type (associated instance of IfcWallType). A recursive structure of TemplateRules with a @operator attribute is used to hold the logical combinations.

<?xml version="1.0" encoding="UTF-8"?>
<mvXML xmlns="http://buildingsmart-tech.org/mvd/XML/1.1" uuid="8a70d456-c609-4ef7-b496-b92fd1e12796" name="example 7.2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://buildingsmart-tech.org/mvd/XML/1.1 ../mvdXML_V1.1.xsd">
<Templates>

The concept template "Property Sets for Objects and Types" defined the concept template structure about how to associate property sets and type objects with property sets to an occurrence object.

<?xml version="1.0" encoding="UTF-8"?>
<ConceptTemplate uuid="5c252c86-5bff-4372-9a27-b794069f9fbb" name="Property Sets for Objects and Types" applicableSchema="IFC4"
applicableEntity="IfcObject">
  <Rules>
    <AttributeRule RuleID="PredefinedType" AttributeName="PredefinedType"/>
    <AttributeRule AttributeName="IsDefinedBy">
      <EntityRules>
        <EntityRule EntityName="IfcRelDefinesByProperties">
          <AttributeRules>
            <AttributeRule AttributeName="RelatingPropertyDefinition">
              <EntityRules>
                <EntityRule EntityName="IfcPropertySet">
                  <References IdPrefix="O_">
                    <Template ref="7c4c45c5-7ba9-4e19-b473-3e97093b3e0d"/>
                  </References>
                </EntityRule>
              </EntityRules>
            </AttributeRule>
          </AttributeRules>
        </EntityRule>
      </EntityRules>
    </AttributeRule>
  </Rules>
</ConceptTemplate>
Here a partial concept template is referenced. In order to prevent a duplication of \texttt{@RuleID} names, which would otherwise occur, when the same partial template “Property Set” is referenced twice from the same main concept template, a \texttt{@IdPrefix} attribute is added. The \texttt{TemplateRule} Parameters at the \texttt{Concept} need to use these prefixes to unambiguously address the rule id.

\begin{verbatim}
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>

Here the partial concept template for property sets is referenced a second time. Therefore a different \texttt{@IdPrefix} is used.

\begin{verbatim}
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
  </EntityRule>
  </EntityRules>
  </AttributeRule>
  </AttributeRules>
\end{verbatim}
The following concept templates are partial concept templates used to allow for a more modular structure of concept template definitions. It includes the definition of property sets and the definition of a property with single value, which is referenced from the partial concept template for property sets. Hence partial concept templates can be nested.

```xml
<ConceptTemplate uuid="6655f6d0-29a8-47b8-8f3d-c9f6c9a620" name="Single Value" applicableSchema="IFC4"
applicableEntity="IfcPropertySingleValue" isPartial="true">
  <Rules>
    <AttributeRule RuleID="PName" AttributeName="Name">
      <EntityRules>
        <EntityRule EntityName="IfcIdentifier"/>
      </EntityRules>
    </AttributeRule>
    <AttributeRule AttributeName="Description">
      <EntityRules>
        <EntityRule EntityName="IfcText"/>
      </EntityRules>
    </AttributeRule>
    <AttributeRule RuleID="PSingleValue" AttributeName="NominalValue">
      <EntityRules>
        <EntityRule EntityName="IfcValue"/>
      </EntityRules>
    </AttributeRule>
  </Rules>
</ConceptTemplate>

<ConceptTemplate uuid="7c4c45c5-7ba9-4e19-b473-3e97093b3e0d" name="Property Sets" code="" applicableSchema="IFC4"
applicableEntity="IfcPropertySet" isPartial="true">
  <Rules>
    <AttributeRule RuleID="PsetName" AttributeName="Name">
      <EntityRules>
        <EntityRule EntityName="IfcLabel"/>
      </EntityRules>
    </AttributeRule>
    <AttributeRule AttributeName="Description">
      <EntityRules>
        <EntityRule EntityName="IfcText"/>
      </EntityRules>
    </AttributeRule>
  </Rules>
</ConceptTemplate>
```
<AttributeRule AttributeName="HasProperties">
  <EntityRules>
    <EntityRule EntityName="IfcPropertySingleValue">
      <References>
        <Template ref="665f6d0-29a8-47b8-8f3d-c9fce9c9a620"/>
      </References>
    </EntityRule>
  </EntityRules>
</AttributeRule>

This concept has a precondition that needs to be met before the template rules are executed. The Applicability imposes that only those instances of IfcWall are validated, that have the property set "Pset_WallCommon" and the two properties "IsExternal" and "LoadBearing" assigned:

- The value of "IsExternal" shall be "true" and the value of "LoadBearing" shall be "true".
In addition, it checks, whether the properties are provided at the occurrence or at the type, and if both are provided, that the override value from the occurrence is used for the check.

```xml
<Applicability>
  <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
  <!-- Applicability: check that IsExternal and LoadBearing property are both set to true (AND) -->
  <TemplateRules operator="and">
    <!-- two alternatives to provide the IsExternal property (as property on occurrence or type) (OR) -->
    <TemplateRules operator="or">
      <!-- check occurrence property -->
      <!-- for check type properties two criteria must be checked: 1) defined on type and 2) not redefined on occurrence -->
      <TemplateRules operator="and">
        <TemplateRules operator="not">
          <TemplateRule Parameters="O_PsetName[Value]="Pset_WallCommon" AND O_PName[Value]="IsExternal"/>
        </TemplateRules>
      </TemplateRules>
    </TemplateRules>
    <TemplateRules operator="or">
      <!-- for check type properties two criteria must be checked: 1) defined on type and 2) not redefined on occurrence -->
      <TemplateRules operator="and">
        <TemplateRules operator="not">
          <TemplateRule Parameters="O_PsetName[Value]="Pset_WallCommon" AND O_PName[Value]="LoadBearing"/>
        </TemplateRules>
      </TemplateRules>
    </TemplateRules>
  </TemplateRules>
</Applicability>
```

At the concepts the validation rules for the required provision of "FireRating", "ThermalTransmittance" and "PredefinedType" are defined. There are three individual concepts, each of them is validated separately.

In this example, they all refer to the same concept template "Property Sets for Objects and Types" via the IDREF link "5c252c86-5bff-4372-9a27-b794069f9fbb". The validation is enforced for the exchange requirement "design phase coordination" via the IDREF link "ae70f764-938b-4cf7-9814-c29a47f56b0e", the requirement is set to "mandatory", meaning, that an error is displayed, if the outermost template rule validates to "false".
<Concepts>

<!-- Test #1: check existence of FireRating property -->
<Concept uuid="983ddc5d-c0c8-47c9-8491-9add7677139" name="load bearing external walls required to have property 'FireRating'">
  <Definitions>
    <Definition>
      <Body lang="de"><![CDATA[For all load bearing external walls the property 'FireRating' shall be applied]]></Body>
    </Definition>
  </Definitions>
  <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
  <Requirements>
    <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
  </Requirements>
  <TemplateRules operator="or">
  </TemplateRules>
</Concept>

The first concept checks, that every instance of IfcWall, that passes the Applicability check (meaning that is an external and load bearing wall), has a property of name "FireRating" within a property set with name "Pset_WallCommon". Thereby it can either be the property set assigned to the occurrence, or the property set assigned to the type, or both. The variable "O_PsetName" refers to the @RuleID "PsetName" defined in the partial concept template that has been referenced by the main concept template “Property Sets for Objects and Types” with the @IdPrefix="O_" – i.e. to the property set assigned directly to the IfcWall occurrence. Similar the variable "T_PsetName" refers to the property set assigned to the associated IfcWallType.

<!-- Test #2: check existence of ThermalTransmittance property -->
<Concept uuid="e9941408-82a6-4c00-a397-11087e6c5df" name="load bearing external walls required to have property 'ThermalTransmittance'">
  <Definitions>
    <Definition>
      <Body lang="de"><![CDATA[For all load bearing external walls the property 'ThermalTransmittance' shall be applied]]></Body>
    </Definition>
  </Definitions>
  <Template ref="5c252c86-5bff-4372-9a27-b794069f9fbb"/>
  <Requirements>
    <Requirement applicability="export" exchangeRequirement="ae70f764-938b-4cf7-9814-c29a47f56b0e" requirement="mandatory"/>
  </Requirements>
  <TemplateRules operator="or">
  </TemplateRules>
</Concept>
The second concept checks the provision of the property “ThermalTransmittance” using the same method.

The third concept checks, whether the direct, optional attribute PredefinedType at IfcWall has a value associated. The metric “[Exists]” checks, that the value of PredefinedType is not NIL or false (in other words, that a value is provided for the optional attribute).
8  XSD Listing

The XSD can be downloaded from: http://buildingsmart-tech.org/mvd/XML/1.1/mvdXML_V1.1.xsd