Automatic BIM filtering using Model View Definitions

Ken Baumgärtel, ken.baumgaertel@tu-dresden.de
Technische Universität Dresden, Germany

Stephan Pirnbaum, stephan.pirnbaum@tu-dresden.de
Technische Universität Dresden, Germany

Hervé Pruvost, herve.pruvost@tu-dresden.de
Technische Universität Dresden, Germany

Raimar J. Scherer, raimar.scherer@tu-dresden.de
Technische Universität Dresden, Germany

Abstract
Interoperability between multiple software applications used in the construction domain and based on Building Information Modeling (BIM) should support common common stable technical interfaces and standardized data formats, like for example the Industry Foundation Classes (IFC). Moreover, interoperability can be enhanced by using so called model view definitions (MVDs) for exchanging building information in several BIM domains between different participants. While model views are currently mainly used for documentation and checking of building exchange data the need of transferring partial building information models between participants is raising. For example, in the context of building energy performance analysis an energy point-of-view on a building is needed. Therefore, this information expressed through MVDs must be present and level of details can help to optimize building simulations. In this paper, we present a concept how MVDs can be used to check and respectively reduce building information.

Keywords: BIM, IFC, MVD, mvdXML, Validation, Filtering, ifcQL

1 Introduction

A model view is an excerpt of a specific data set in a predefined structure and it expresses data requirements in a building information model. For that purpose, buildingSMART introduced model view definitions and the related data format mvdXML which is based on XML schema. They allow the declaration of particular IFC types, attributes and data types required by a software application. For example, with the help of a MVD it can be defined that a specific property like the thermal transmittance has to be linked to every wall, or that space boundaries between walls and rooms must exist.

Software applications use these MVDs to check if required information is provided in the building information model. For creating MVDs there exists a tool called ifcDoc (buildingSMART International Ltd. 2016) from buildingSMART which allows domain users to specify MVDs, to generate subsequent mvdXML documents and to validate a given IFC file based on that. These requirement checks are an essential task especially because building information increases considerably in time and information gaps too. Besides the different design and construction domains need very specific data which are not relevant for every BIM participant. For this reason, this paper presents an innovative concept which allows checking BIM data, but also filtering with model views on the basis of mvdXML.

The concept extends the validation of a building information model with the possibility of reducing provided information to a new partial building information model and consists of

three steps: (1) transformation of a mvdXML file to complex SQL-like queries, (2) check of the original BIM with the converted query commands and (3) reuse of the commands for extracting requested information from the original IFC file and export of a reduced IFC file.

This paper addresses the conversion steps and the implemented prototypes with their algorithms for handling IFC and mvdXML files. First, we take a look on state of the art in BIM filtering and validation. After that we introduce the IFC Query Language (ifcQL) which enables a SQL-like syntax for selecting, updating, creating and deleting information in IFC files. Then the mvdXML conversion process is presented and some examples are shown. Finally, we summarize the paper and highlight the advantages of our approach.

2 Filtering and Validation in BIM

With the IFC, all the domains in the construction industry are addressed. While this is good for interoperability reasons it also leads to some problems regarding level of details in each domain. Pauwels et al. (2011) describe that the schema is too general for most of the purposes. Because there exist many engineering domains, which influence partially each other, emerge necessarily optional data types that represent important information in a domain while in another domain they are not needed. For example, a valid BIM for structural analyses doesn't need the declaration of building storeys or rooms, but load-bearing walls and slabs should be marked. This leads to redundancy in the data model, because it provides too many options instead of including constraints, which are needed in software development to implement stable application and interfaces. Therefore, many software developer are interpreting the schema in a wrong way and the data exchange is often erroneous (Sacks et al. 2010). For some tasks, such as thermal energy building calculations, the IFC scheme have limited semantics to define how information is declared and exchanged in the right way (Venugopal et al. 2012). Many optional attributes and the possibility to declare object properties with *IfcPropertySet* complicate a uniform expression and the preservation of model quality in the schema.

In the context of filtering, i.e. the reduction of information, of building information models raises the question which information is needed (information requirements). Katranuschkov et al. (2010) classified different information requirements with following types of model views: a) Domain model view, b) Ad-hoc model view and c) Multimodel view. The domain model view corresponds to the approach of buildingSMART by using Model View Definitions (Hietanen 2006), which define the information requirements of a specific exchange scenario based on Information Delivery Manuals (IDM) (Wix & Karlshoej 2010). For such reasons, Zhang et al. developed the mvdXMLChecker prototype to parse MVD on the basis of mvdXML and to check IFC models (Zhang et al. 2014; Zhang et al. 2015). When one or more checks fail, a report is generated and lists all potential problems. The Ad-hoc model view refers to information of a concrete model. The extracted information is specified on-the-fly on the basis of the provided BIM. In the Multimodel view concrete information requirements are specified, like in the Ad-hoc model view, but they refers to additional information from different domain models, e.g. cost model, which are connected to the architectural model (for example linking costs to walls).

The development and usage of model views as described above depend on time and model detailing aspects. The definition of (partial) data sets is often done preliminary, i.e. it is uncoupled with the actual data reduction of a concrete model, but it can be also carried out on the basis on the information in a concrete model (Windisch et al. 2012). For realizing the data reduction with model views, an implementation of different types of filter or filter layers is required. Windisch et al. (2012) presented four filter types: (1) filtering on schema level, (2) filtering on class level, (3) filtering on object level and (4) filtering on reasoning level. The possible detail level in Ad-hoc model views is higher than in domain model views, because of the knowledge of concrete instances and model-specific aspects (for example dimensions of building elements or used materials).

3 The Query Language ifcQL

For the definitions of MVDs and their application on IFC files a relational algebra is used with an own syntax. It is similar to SQL and is named ifcQL. The primary purpose is not only (1) validation of building information data, but also (2) extraction of a specified building information set and the creation of a new version of the building in IFC. In this sense, a reduced building information model can be created which can be exchanged with one or more software applications. The ifcQL complies with the CRUD principle:

- **Create** Creating a new data set.
- **Read** Reading an existing data set.
- **Update** Modifying an existing data set.
- **Delete** Deleting an existing data set.

3.1 Structure of ifcQL

An MVD defined with ifcQL can be specified in one or more files in which one command per line is stated and refers to one data set. It can be determined if one command is applied on the previous one or creates a new data set statement from the full building information. The ifcQL provides many functions to check, reduce, extend or modify a building information model. In ifcQL following commands can be used:

- **Filter** to reduce a building information model.
- **Check** to validate a building information model.
- **Add** to add new entities to a building information model.
- **Insert** to add new properties to existing entities in a building information model.
- **Update** to modify existing entities of a building information model.
- **Delete** to delete entities or properties in a building information model.
- **Load** to import existing MVDs as part of a new MVD.

The commands are defined in a custom syntax and each begins with the character "**~**". The declaration of concrete entity types, attributes as well as constraints are each specified with a "-" on the line. The syntax diagram of the initial command declaration is presented in Figure 1. In a similar manner to SQL, building information data set can be queried using the commands SELECT, FROM and WHERE. The FROM refers to the set of IFC entities by declaring types and identifiers. The SELECT is one of the commands above and the WHERE refers to object attribute values, for example the "GlobalId" of *IfcRoot* objects and the target value.

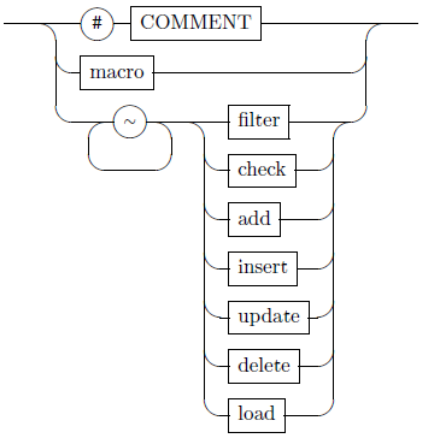

Figure 1 ifcQL commands

The most important commands are *check* and *filter*. They are shown in Figure 2 and Figure 3. In the validation phase, objects and their specific conditions (object semantics) on the one hand and relations between objects (relation semantics) to the other hand can be checked. This allows not only the declaration of attribute values, but also the classification of

objects. For example, with the *relationcheck* it is possible to check the assignment of rooms to a specific building storey. The outcome is a Log file which contains all checks and their results written in a report.

The result of the filtering task is instead a partial building information model whose information content correlates to the assigned sub schema defined in the MVD. Filtering on class level reduces, like the filtering on schema level, building information models on the basis of conditions defined in term of types and their relations. These conditions are applied during the filtering task on concrete models (e.g. filter all building elements of a specific type). The *filter* command can be followed by the declaration of a type ("-type") to search and extract objects of that type, or by a specific "GlobalId" to search an exact object ("-guid") or to exclude it from the partial set of the result. With the command "-recursive" it is additionally possible to include entities to the result set which are directly linked to the queried entity. The ifcQL provides more possibilities to work with building information, because it is based on the Java-based framework BIMfit which supports any STEP data format (any IFC version) and provides several functions to access BIM data (Wülfing et al. 2012; Technische Universität Dresden 2016).

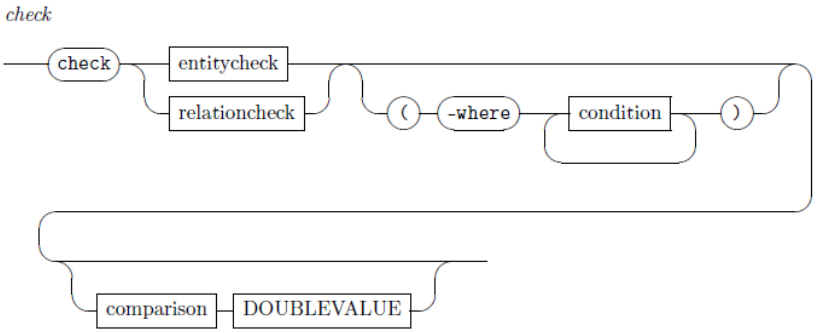

Figure 2 Check command in ifcQL

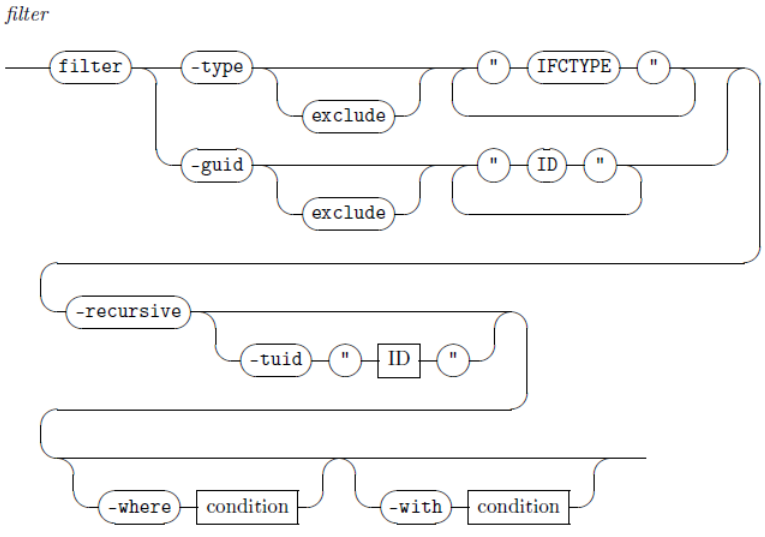

Figure 3 Filter command in ifcQL

4 Filtering and Validation using mvdXML

In the definition phase of MVDs there still exists in principle no building information model. The exchange requirements are created through declaring classes and attributes. For doing so, the official software tool for MVDs from buildingSMART ifcDoc is used (buildingSMART International Ltd. 2016). With ifcDoc the MVDs can be exported in the mvdXML file format and can be used to validate any BIM by loading it into the tool. Another software application is mvdXMLChecker which was implemented for the BIMserver (de Laat 2016;

opensourcebim.org 2016). It can also be used in other tools to check building semantics against requirements. Nevertheless, filtering is currently not possible with any of these tools. Because of that we developed the tools **mvdXML converter** and **ifcQL processor** to enable validations as well as filtering based on MVDs.

4.1 Mapping Process

Through a two-step process an mvdXML file can be used to extract partial building data from a BIM. Figure 4 shows the tool chain:

(1) Usage of mvdXML converter to map an mvdXML file into the corresponding ifcQL syntax. During this step, it can be specified at application start-up if a validation or a filtering query should be made.

(2) Executing the ifcQL processor with import of an IFC file and the created ifcQL file from step (1) that contain all the commands. The output is either a LOG file with the summary of checks (validation) or a reduced IFC file (filtering).

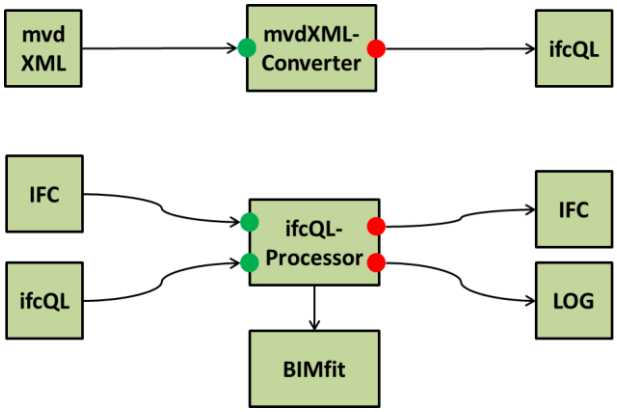

Figure 4 Process chain in mapping process

The second step was introduced in detail in section 3. Looking deeper in the first step, following tasks are performed:

1. **Parsing of mvdXML** - The mvdXML converter parses the XML content of the given MVD.
2. **Instantiation of MVD objects based on their content** - The XML elements are evaluated, mapped to corresponding Java classes and instantiated. The mvdXML structure Version 1.1 (Chipman et al. 2015) is used to create a Java object graph of exchange requirements.
3. **MVD objects are converted to ifcQL commands** - After the initialisation of the object graph with exchange requirements, every node is transformed to the correct ifcQL command step by step.
4. **Export of ifcQL commands** - Finally, the ifcQL commands are collected in the right order and exported to an ifcQL file which can be used to validate (check) or filter a building information model.

The element <Templates> includes all rules which can be applied in validating or filtering a BIM. It can consist of several <ConceptTemplate> elements which can be separated in multiple <SubTemplates> to improve the overall structure. For each <ConceptTemplate> an ifcQL command is created (with command *filter* or *check* depending on the use case).

Listing 1 shows an example of retrieving building façade elements in the model view “FacadeModelView”. The structuring of <Templates> and <ConceptTemplate> is presented in mvdXML. In particular the XML attribute *applicableEntity* is from importance. It declares the IFC type on which the query refers. Furthermore, the declaration of <Rules> is presented. The elements in it are used to create the query path starting from *applicableEntity*. The element <AttributeRule> consists of *AttributeName* which marks the corresponding IFC attribute. The attribute *Cardinality* is also processed and can be 0, 0:1, 1

or 1:n. The representations are shown in Table 1. Like in Listing 1 an <AttributeRule> element can consist of multiple <EntityRule> elements which again consist of several <AttributeRule> elements. A <EntityRule> have the attributes *EntityName* and *Cardinality* to mark the IFC type(s).

Table 1 Cardinality representations in mvdXML and ifcQL

Cardinality	mvdXML	ifcQL
1:n	_asSchema	ONETOMANY
0	Zero	ZERO
0:1	ZeroToOne	ZEROTOONE
1	One	ONE
1:n	OneToMany	ONETOMANY

For a better structuring it is possible to use <Views> elements with the specified <Templates>. Each <ModelView> element can include multiple <ExchangeRequirement> elements. For each combination of those elements the mvdXML converter creates one ifcQL file which is named like the model view. Figure 5 presents the relationships between <Concept> elements and <ModelView> elements. To each <Concept> a *name*, an IFC type (*applicableRootEntity*) and a <Template> have to be assigned. The attribute *applicableRootEntity* can be used to overwrite *applicableEntity* of a <ConceptTemplate> element.

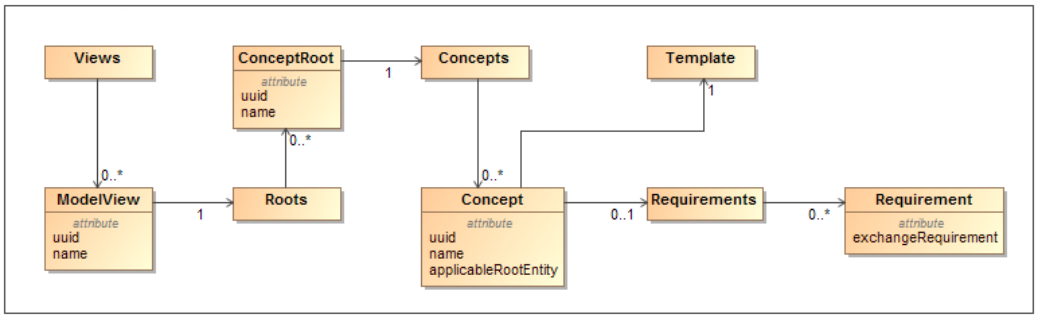

Figure 5 relation between Model Views and Exchange Requirements

4.2 Use Case

The simple MVD in Listing 1 can be used to check or filter façade elements. A path beginning from IfcBuildingElement with the IFC inverse attribute “ProvidesBoundaries” is declared to reach the connected space boundaries. The value for “InternalOrExternalBoundary” is set to “EXTERNAL” which means value “2” in the enum. This <ConceptTemplate> is assigned to the exchange requirement “FacadeElements” and attached to a model view called “FacadeModelView”. The transformation in the appropriate *filter* (left) and *check* (right) commands is presented in Listing 2. Together with the ifcQL processor arbitrary building information models can be loaded and reduced respectively checked. Figure 6 shows an example building before and after the filtering task based on the MVD. Only the façade elements, rooms, storeys and the building entity are exported in a new IFC file.

Listing 1 example of a mvdXML for checking or filtering building façade elements

```
1   <?xml version="1.0"?>
2   <mvdXML xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      uuid="6080" status="sample" xsi:schemaLocation="http://buildingsmart-tech.org/mvd/XML/1.1 http://www.buildingsmart-
    tech.org/mvd/XML/1.1/mvdXML_V1.1_add1.xsd"
    xmlns="http://buildingsmart-tech.org/mvd/XML/1.1">
3     <Templates>
4       <ConceptTemplate uuid="6a0d" name="FacadeTemplate" applicableSchema="IFC4" applicableEntity="IfcBuildingElement">
          <Rules>
5           <AttributeRule AttributeName="ProvidesBoundaries">
```

```
            <EntityRules>
              <EntityRule EntityName="IfcRelSpaceBoundary">
                <AttributeRules>
                  <AttributeRule AttributeName="InternalOrExternalBoundary">
                    <EntityRules>
                      <EntityRule EntityName="IfcInternalOrExternalEnum">
                        <Constraints>
                          <Constraint Expression="Value='2'" />
                        </Constraints>
      <!-- ...Closing Tags... -->
        <ConceptTemplate uuid="fea4" name="RoomTemplate" applicableSchema="IFC4" applicableEntity="IfcSpace" />
        <ConceptTemplate uuid="5e8a" name="StoreyTemplate" applicableSchema="IFC4" applicableEntity="IfcBuildingStorey" />
        <ConceptTemplate uuid="0efe" name="BuildingTemplate" applicableSchema="IFC4" applicableEntity="IfcBuilding" />
      </Templates>
      <Views>
        <ModelView uuid="2fe8" name="FacadeModelView" applicableSchema="IFC4">
          <ExchangeRequirements>
            <ExchangeRequirement uuid="efc8" name="FacadeElements" applicability="both" />
          </ExchangeRequirements>
          <Roots>
            <ConceptRoot uuid="c7ec" applicableRootEntity="IfcBuildingElement">
              <Concepts>
                <Concept uuid="367e" name="FacadeTemplate" override="false">
                  <Template ref="6a0d" />
                  <Requirements />
                  <TemplateRules operator="and" />
                </Concept>
              </Concepts>
            </ConceptRoot>
            <!-- ...more references and closing Tags... -->
    </mvdXML>
```

Listing 2 Filter (left) and Check (right) command in ifcQL of the mvdXML example above

```
#FacadeTemplate(367e)
~filter -type "Ifcbuildingelement" -with path
ONETOMANY
"ProvidesBoundaries"("Ifcrelspaceboundary")
:((((("InternalOrExternalBoundary" EQUALS "2"))))
#RoomTemplate(f8ca)
~filter -type "Ifcspace"
#BuildingTemplate(3f77)
~filter -type "Ifcbuilding"
#StoreyTemplate(7283)
~filter -type "Ifcbuildingstorey"
```

```
#FacadeTemplate(367e)
~check entity -type "Ifcbuildingelement" having path ONETOMANY
"ProvidesBoundaries"("Ifcrelspaceboundary"):((((("InternalOrExternalBoundary"
EQUALS "2"))))
#RoomTemplate(f8ca)
~check entity -type "Ifcspace"
#BuildingTemplate(3f77)
~check entity -type "Ifcbuilding"
#StoreyTemplate(7283)
~check entity -type "Ifcbuildingstorey"
```

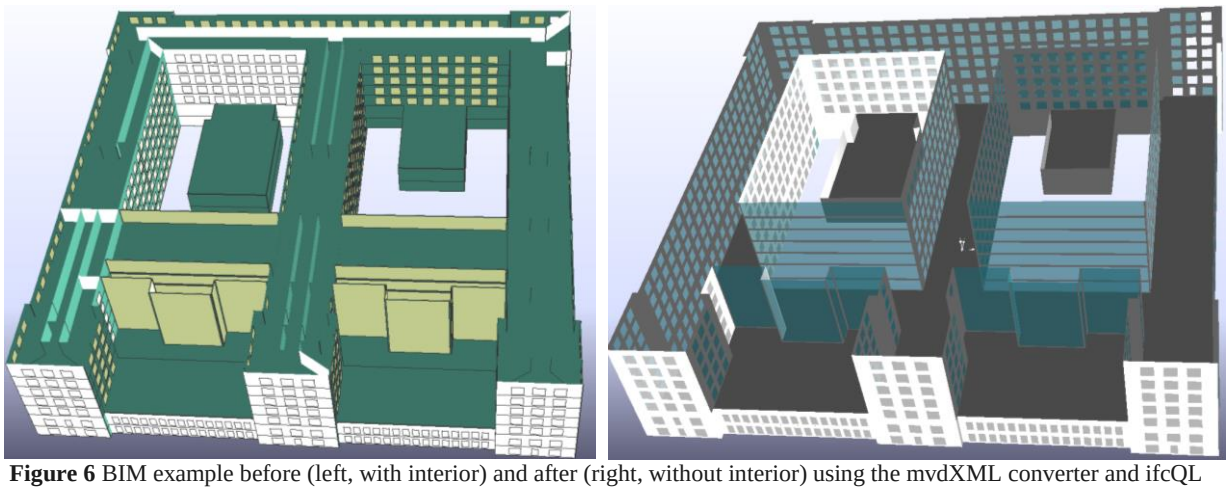

Figure 6 BIM example before (left, with interior) and after (right, without interior) using the mvdXML converter and ifcQL processor – the roof is visually hidden to provide a better insight into the building

5 Summary

The importance of MVDs is continuously increasing, while the complexity and amount of information grows in BIM permanently. The domain experts own different perceptions on building information and a homogeneous way to exchange different building data in a BIM project should be supported. In this paper, we introduced the concept of using model view definitions based on mvdXML for validation and for filtering of building information models. The validation with mvdXML was already presented by Zhang et al. (Zhang et al. 2014;

Zhang et al. 2015). But the filtering with mvdXML files was not provided up to now. Beside checking and providing required data to a specific software application, it is conceivable that MVD based filtering could in the future become interesting for certification purpose. Following the intention of buildingSMART to develop MVDs for certification purpose, more than checking compliance of IFC models against certain standard MVDs, such MVDs or specific new ones could be used for producing an IFC file that complies with a specific quality standard. In view of that, different simulation tools could for example use filtering for importing one reduced IFC model that is conform to a simulation model view recognised for best practice. More specifically a MVD could be produced as a certification standard that guaranty a simulation expert to use a filtered IFC model of high quality, thus reducing simulation errors due to building information inconsistency.

IFC is suitable for exchanging partial building information models. Therefore it is possible to transfer only required building data sets between domain experts to fulfil their tasks. This minimizes data transfers and simplifies partially the building information which is a big advantage in the context of building simulations. Hence, particular building storeys or rooms can be passed over to thermal energy simulation tools without sending the whole IFC data. For the implementation of this process we developed a two-step conversion. First, the mvdXML converter imports an mvdXML file and translates the contents to corresponding ifcQL commands. With the SQL-like syntax of ifcQL it is very easy to create, read, update or delete BIM data. Hereby, the user has to specify if the result should be used for validation or filtering. In a second step the application of the MVD on a building with the tool ifcQL processor is carried out. The output is a validation report or a reduced building information model in IFC (currently IFC2x3 and IFC4 possible). In an example we show the reduced BIM which can be used in any other BIM-based software application. The Java programs ifcQL processor and mvdXML converter can be integrated in any software application. The documentation and the prototypes itself can be found on *https://openeebim.bau.tu-dresden.de/bimfit.html*.

Acknowledgements

We kindly acknowledge the support of the European Commission to the projects HOLISTEEC (Grant Agreement No. 609138, http://www.holisteecproject.eu) for the development of the mvdXML converter, Design4Energy (Grant Agreement No. 609380, http://www.design4energy.eu) for the development of ifcQL and ifcQL processor and eeEmbedded (Grant Agreement No. 609349, http://eeembedded.eu) for the extension of the developments.

References

buildingSMART International Ltd., 2016. ifcDoc. *online*. Available at: http://www.buildingsmart-tech.org/specifications/specification-tools/ifcdoc-tool/ifcdoc-download-page [Accessed July 15, 2016].

Chipman, T., Liebich, T. & Weise, M., 2015. *mvdXML - Specification of a standardized format to define and exchange Model View Definitions with Exchange Requirements and Validation Rules*,

Hietanen, J., 2006. *IAI Official IFC Model View Definition Format*,

de Laat, R., 2016. mvdXMLChecker. *online*. Available at: https://github.com/opensourceBIM/mvdXMLChecker [Accessed July 15, 2016].

opensourcebim.org, 2016. BIMserver.org. *online*. Available at: http://bimserver.org [Accessed July 15, 2016].

Sacks, R. et al., 2010. The Rosewood experiment - Building information modeling and interoperability for architectural precast facades. *Automation in Construction*, 19(4), pp.419–432. Available at: http://dx.doi.org/10.1016/j.autcon.2009.11.012.

Technische Universität Dresden, I. of C.I., 2016. BIMfit - BIM filtering and integration toolbox.

online. Available at: https://openeebim.bau.tu-dresden.de/bimfit.html [Accessed July 19, 2016].

Venugopal, M. et al., 2012. Semantics of model views for information exchanges using the industry foundation class schema. *Advanced Engineering Informatics*, 26(2), pp.411–428. Available at: http://dx.doi.org/10.1016/j.aei.2012.01.005.

Windisch, R., Wülfing, A. & Scherer, R., 2012. A generic filter concept for the generation of BIM-based domain- and system-oriented model views. In G. Gudnason & R. Scherer, eds. *eWork and eBusiness in Architecture, Engineering and Construction*. Reykjavik, Iceland: CRC Press, pp. 311–319. Available at: http://books.google.com/books?hl=en&lr=&id=z1z83fcH4zMC&oi=fnd&pg=PA311&dq=A+Generic+Filter+Concept+for+the+Generation+of+BIM-based+Domain-+and+System-oriented+Model+Views&ots=T3N-1hEf3S&sig=BIp2DESSr1N-LDnhNrWsJDxzpiY [Accessed December 8, 2014].

Wix, J. & Karlshoej, J., 2010. *Information Delivery Manual Guide to Components and Development Methods*,

Wülfing, A., Windisch, R. & Baumgärtel, K., 2012. BIMfit – Ein modulares Softwarewerkzeug zur Abfrage und Filterung von Gebäudeinformationsmodellen. In *Proceedings des Forum Bauinformatik 2012*. Bochum, Deutschland.

Zhang, C., Beetz, J. & Weise, M., 2015. Interoperable Validation for IFC Building Models Using Open Standards. *Journal of Information Technology in Construction*, 20(2015), pp.24–39.

Zhang, C., Beetz, J. & Weise, M., 2014. Model view checking: automated validation for IFC building models. In Mahdavi, ed. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014*. Vienna, Austria, p. 123. Available at: http://books.google.com/books?hl=en&lr=&id=tw7NBQAAQBAJ&oi=fnd&pg=PA123&dq=“models+are+the+pre-condition+for”+“supports+a+full+range+of+data+exchanges”+“dominant+citizens,+and+the+model+instances”+“needed+for+these+processes+is+contained+in”+&ots=1edn5mhe.