Textual Editing of Partial IFC Model

Nicolai Dahl Blicher-Petersen, Christian Harrington, Thomas Hallier Didriksen, Sune Alkærsig, and Anders Høst Kjærgaard
{ndbl, cnha, thdi, sual, ahkj}@itu.dk

IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark

Abstract. Industry Foundation Classes (IFC) is an ISO standard used for modelling various physical and functional aspects of a building. IFC models are often large and complex, so a key challenge is: how can work be carried out on e.g. plumbing, heating, or electrical wiring independently, without introducing inconsistencies. This paper describes a prototype solution that eases cross-disciplinary work, by allowing model evolution and resolution of inconsistencies on a small subset of a large IFC model. The model-driven methodology is used to achieve a generally applicable and flexible solution, usable in various domains. In this paper we demonstrate an open source prototype handling a common industry problem that specifically makes it possible to edit pipes, walls, and openings in walls.

Keywords: IFC, BIM, EMF, DSL, model-driven development, model transformation, Xtext, Xtend, BIMServer

1 Introduction

Building Information Modeling (BIM) is the process of modelling various physical and functional aspects of a building. Currently, many BIM software products exist. These allow users to model and analyse the many interactions between different parts of a building. Many of these software products use common file formats for modelling buildings, such as Industry Foundation Classes (IFC). This specification improves interoperability between different software packages in different domains, and is integrated into a variety of commercial applications.

In a real-life scenario, engineers usually work on separate parts of a building model using advanced, proprietary, and visual tools such as Revit.[1] Due to the complexity of most BIM projects, it is desirable to be able to work with aspects of a model such as heating, electrical wiring, or plumbing independently. In this paper, we present a prototype that demonstrates that such editing on a subset of an IFC model is feasible using modern modelling tools from the Eclipse Modeling Framework (EMF).

[1]An overview of Revit products can be found at `http://usa.autodesk.com/revit/`

Problem In particular, we focus on solving a common problem of combining a construction model with a plumbing model in a manageable way. When modelling buildings, the structure of the building and its plumbing installations are modelled separately [4, pp. 19–20]. The construction model describes which walls are load-bearing and which walls have openings for installations. Because of this separate modelling, it is possible for plumbing installations to be inconsistent with the construction model. For example, a pipe could intersect a solid wall, without an opening for it to go through. This could result in a situation at the construction site where a planned installation is not possible in reality. This problem has been defined by Kaj A. Jørgensen [5], associate professor at Aalborg University, who has substantial domain knowledge, through his work with IT methodologies for building modelling.[2]

Solution Due to the difficulties mentioned above, it would be desirable if the construction engineer had an easy way to find these collisions and solve them by making an opening or moving the installation [5]. To facilitate this, we will. . .

- Produce an analysis of the subset of IFC used to define pipe installations, and develop a simple, textual domain specific language (DSL), called Pipes, to specify these installations.
- Develop an Eclipse editor that eases the writing of files with the DSL, including syntax highlighting and autocompletion.
- Create an update mechanism between the DSL and the IFC model being manipulated, so changes made in the DSL are applied correctly to the model.
- Interface with a central server for storing, merging and (future) collaboration support. We use BIMServer[3], which is open source.
- Present ideas for future BIM projects in relation to the ITU research initiative Energy Futures, based on this project.[4]

Workflow The workflow revolves around a server. We save a construction model and a plumbing model on the BIMServer, which then merges the two. This merged IFC model is retrieved by our client, and the relevant parts are extracted and transformed to the Pipes DSL syntax. An editor for the Pipes DSL is launched, and the user can edit and verify the consistency of its data and structure. Finally, these changes are updated in the merged model and stored server-side. The prototype solution is extensible and constitutes a good starting point for future automation of the workflow and visualisation of the domain.

This paper is structured as follows. In Section 2 we provide a background of the problem space, and in Section 3 we analyse the problem and list prototype requirements. Section 4 describes the solution, which we evaluate in Section 5. In Section 6 we suggest ideas for future projects. Section 7 lists related work, and we conclude on our findings in Section 8.

[2]Kaj A. Jørgensen at Aalborg University: `http://www.kaj.person.aau.dk/`
[3]BIMServer Documentation can be found at `http://bimserver.org/documentation/`
[4]For further information, see `http://energyfutures.itu.dk/`

2 Background

2.1 Building Information Modeling

As previously described, BIM is a process that models physical constructions by generating digital representations. The resulting model is a shared knowledge base that can be used in all stages of a building project. BIM extends traditional building design from two-dimensional drawings and three-dimensional physical models, to include aspects such as time, cost, staffing, and design process. This makes BIM very attractive for modern building construction, as it not only eases the communication between the different stakeholders, but also allows for early planning in terms of time and energy consumption.

To support the flexibility required in BIM, this shared knowledge base has to be represented in a format that enables interoperability. And since building projects involve many different contributors from different domains, it has to be adoptable by many different software applications [11]. Consequently, an interoperability standard is needed.

2.2 Industry Foundation Classes

IFC is widely used for BIM, and is mandatory for new public buildings in Denmark.[5] Its goal is to facilitate interoperability between different software platforms [12] and can reduce time spent on building design processes [2]. It does this through an object-based data model, represented in two file formats. The first is a EXPRESS-based format, IFC-EXPRESS (ISO-10303-21). The other is an XML format, called ifcXML (ISO-10303-28). The EXPRESS-based format is the most widely used due to its relative compact size, while still being readable.

One of the main challenges when working with IFC is that the models often are very large and complex. The meta model contains more than six hundred entities organised in an object-based inheritance hierarchy. Entities can be both tangible elements, such as an IfcWall, but also abstract entities like IfcAxis2Placement3D, describing the location and orientation of another IFC entity. On the highest level of abstraction, IFC defines two categories of elements: rooted and unrooted elements respectively. Unrooted elements do not have an identity and only exist if referred to by other elements. Rooted elements have a unique global identifier (globalId).[6]

2.3 Eclipse Modeling Framework and Ecore

The Eclipse Modeling Framework is a modelling framework and code generation facility for building tools and other applications, based on a structured data

[5]Regulations for buildings built by the Danish state: `https://www.retsinformation.dk/forms/R0710.aspx?id=134884`

[6]IFC2x3 Final Documentation: `http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm`

model.[7] EMF is capable of producing a Java object graph from a model instance described in XMI. For meta modelling, EMF features Ecore. Ecore is a meta modelling language allowing the user to describe and build meta models for their domain. Furthermore, several transformation tools, such as Xtend and Xpand, are associated with EMF. They support work with model transformations. Xtend is a programming language that compiles into Java. It is very well integrated into EMF, and has several language features that are well suited for model-to-model transformations. Xtend is implemented with Xtext, a tool allowing easy creation of DSLs. Xpand is a template language used in model-to-text transformations. EMF also features a workflow system, Modeling Workflow Engine 2 (MWE2), that makes it easy to chain these tools together.

3 Problem Analysis and Requirements

In the following section we will analyse the problem, and discuss a set of requirements for a solution that is capable of editing a partial IFC model within a specific domain. This list should be thought of as a guideline for how one should ideally approach this problem.

3.1 Problem Analysis

The separation of the construction and plumbing models makes it difficult to work on objects that are separated but interrelated. We focus on the case of IfcFlowSegments (i.e. a pipe), IfcWallStandardCases (a regular wall) and an IfcOpeningElement (an opening in a regular wall). Jørgensen mentions that a possible solution to the problem is to allow the building service engineer, responsible for the plumbing installations, to create a message with precise information about required openings for the pipes [5]. This message could then be handed to the construction engineer, so that he can verify that these are properly placed. As such, the primary goal of the solution is to enable the user to work on a subset of the IFC model involving pipes, openings, and walls. In Figure 1, a graphical representation of this subset is presented.

3.2 Requirements

R1. Working with a Partial Model With the aforementioned complexities and challenges of IFC in mind, a primary focus is to be able to extract a well-specified subset of an IFC model. It is desirable to have an architecture that separates the extraction of a partial model from the rest of the workflow. This component will only extract the IFC elements that we are interested in. This should make the component easier to reuse, and make it easier to verify that the extraction is performed correctly.

[7]The EMF project page can be found at `http://www.eclipse.org/modeling/emf/` and leads to MWE2, Xtext, Xtend, and Xpand resources.

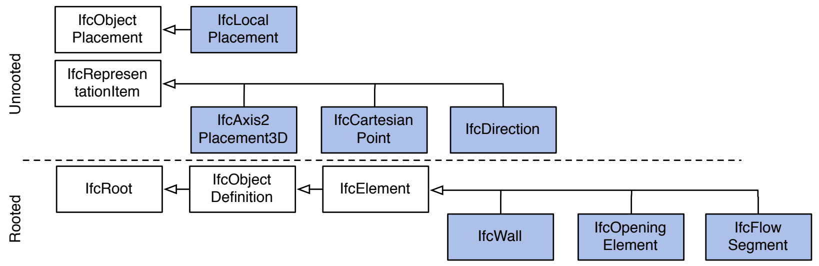


Fig. 1. A graphical representation of the subset used, excluding relational objects for the sake of simplicity. The concrete classes of the chosen subdomain are highlighted in blue.

Furthermore, a clear domain definition is needed to implement and verify the extraction of the partial model. Achieving this in a concise but generic way is not trivial. An evolving standard for doing this is via Model View Definitions (MVD) [10], which allow fine-grained definitions of IFC subsets using XML. The non-profit organisation, buildingSMART, that supports open source BIM software, propagates MVD as their standard.[8] Nour discusses this and other challenges when working with partial editing on IFC [10]. Unfortunately, it was not possible to obtain the product of Nour's project. Even though MVD seems to be a promising approach for extracting IFC subsets, it is outside the scope of this project to develop this functionality ourselves. We found that simply defining a partial model with MVD is in itself a somewhat complex task. Therefore, for the purposes of designing a single experimental DSL with only a few IFC classes, we argue that a more informal definition of the domain is sufficient and will simplify the development process.

R2. Correct Meta Model It is vital for the solution that the IFC meta model is in fact correct. This point may seem obvious at first, but in our experience, finding a correct EMF meta model that reflects the actual IFC standard is difficult. In particular, finding a meta model with a proper serialiser and deserialiser that converts from either IFC-EXPRESS or ifcXML to the corresponding Ecore instance can be difficult. Further, IFC models are not treated consistently across existing BIM software. Thus, one must be aware of inconsistencies [11, p. 4].

R3. Valid Model Transformations An ideal solution would feature model transformations that are verifiable and correct. By verifiable, we mean being able to trace or test that transformations actually occur in the way that we expect. By correct, we mean not corrupting any model structure during a transformation. However, with the complexity of IFC in mind, we allow that some constraints are broken in the IFC model at the end of the transformation. A possible scenario

[8]buildingSMART, MVD, `http://buildingsmart.com/standards/mvd`

is that in an IFC model, all pipe objects could be referenced by some central element, e.g. for calculation of maintenance costs for these. It would be difficult to require and assert that no such constraints are broken in the general case, and is a problem outside the scope of this project.

R4. A Simple DSL The simple DSL will demonstrate that the editing of partial models is feasible for our own, as well as other, similar subdomains of IFC. The implementation will show how a small, but significant domain can be managed separately from the full IFC model. To support extensibility and reusability, it should be built with widely used tools, like the ones in EMF. One could imagine that a future solution would have a visual syntax instead of a textual syntax.

R5. Structural Editing It must be possible to use the DSL to add and remove elements. This is required to resemble a real-world scenario, where an element could be missing. The solution must be able to do this for flow segments and openings, without corrupting the IFC model. Structural editing of walls is not needed for the workflow described above, so it may be omitted.

This concludes the list of desirable features, but please note that it is only a core selection, and that many extensions could be added to it. Some of these will be discussed in Section 6 as ideas for a future projects.

4 Solution

Our prototype combines several open source technologies to attain an extensible solution, with two reusable MWE2 client side workflows at its core. The setup allows the user to edit a subset of IFC in an auto-generated Eclipse editor in between the two workflows. The two building models are stored on a BIMServer for possible merging, version control, and collaboration through use of the client-server design pattern, as this section will explain further. The prototype serves as an open source alternative to similar, proprietary BIM software.

4.1 IFC Meta Model

The prototype uses an IFC meta model that is auto-generated from the XML Schema Definition (XSD) of ifcXML.[9] This allows utilisation of the XML serialiser and deserialiser provided by EMF, to convert ifcXML to and from its Java representation. One of the advantages of using these tools is that we do not have to develop any serialisers ourselves. However, generating the meta model from an XSD also means that the client application of the prototype only supports

[9]ifcXML XSD: `http://buildingsmart-tech.org/ifcXML/IFC2x3/FINAL/IFC2X3.xsd`

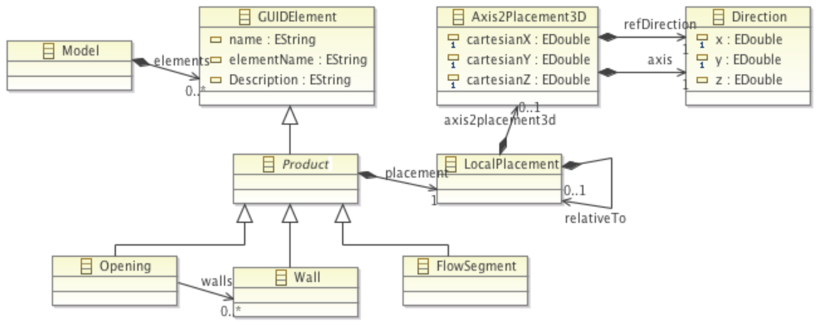


Fig. 2. Pipes DSL Ecore meta model.

ifcXML, and not the more widely used IFC-EXPRESS format. Because BIMServer accepts and generates both formats, this is a minor concern. But it means that the client side of the prototype relies on BIMServer for IFC-EXPRESS handling.

Although we chose not to use it, BIMServer also provides a meta model for IFC. This meta model comes with a custom serialiser and deserialiser for IFC-EXPRESS. We found that using this meta model and the provided serialisation tools was undesirable, as it would mean that the client application of the prototype would be completely dependent on the BIMServer. An advantage of using this meta model would be that it resembles the documented IFC specification more closely than the auto-generated meta model (even though both are equally correct). Specifically, the BIMServer meta model includes inverse relationships, which are not represented in the generated meta model, as they are not modelled in ifcXML [9, p. 24].

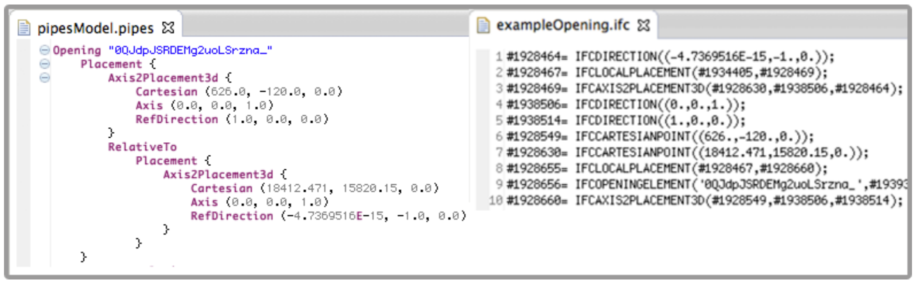
A third meta model for IFC has been created by Jim Steel, Lecturer at the University of Queensland.[10,11] This meta model seems to model all relationships better than the two aforementioned models, but does not provide a deserialiser to any IFC format. As it is not within the scope of this project to create a deserialiser for such a meta model, it was not used. However, using this meta model could be considered as a future improvement.

4.2 Pipes DSL

The simple Pipes DSL serves as the front-end of the prototype. Figure 2 shows the Ecore meta model that the DSL is built upon. Compared to Figure 1, all the critical elements of the domain are still present, but the inheritance hierarchy has been greatly simplified. As shown, Opening, Wall, and FlowSegment are top

[10] Jim Steel at the University of Queensland: `http://itee.uq.edu.au/~uqjstee8/`

[11] Jim Steel's model can be found at `http://www.emn.fr/z-info/atlanmod/index.php/Ecore#ifc2x3_0.1`



```
Opening "0QJdpJSRDEMg2uoLSrzna_"
    Placement {
        Axis2Placement3d {
            Cartesian (626.0, -120.0, 0.0)
            Axis (0.0, 0.0, 1.0)
            RefDirection (1.0, 0.0, 0.0)
        }
        RelativeTo
            Placement {
                Axis2Placement3d {
                    Cartesian (18412.471, 15820.15, 0.0)
                    Axis (0.0, 0.0, 1.0)
                    RefDirection (-4.7369516E-15, -1.0, 0.0)
                }
            }
    }
```

```
#1928464= IFCDIRECTION((-4.7369516E-15,-1.,0.));
#1928467= IFCLOCALPLACEMENT(#1934405,#1928469);
#1928469= IFCAXIS2PLACEMENT3D(#1928630,#1938506,#1928464);
#1938506= IFCDIRECTION((0.,0.,1.));
#1938514= IFCDIRECTION((1.,0.,0.));
#1928549= IFCCARTESIANPOINT((626.,-120.,0.));
#1928630= IFCCARTESIANPOINT((18412.471,15820.15,0.));
#1928655= IFCLOCALPLACEMENT(#1928467,#1928660);
#1928656= IFCOPENINGELEMENT('0QJdpJSRDEMg2uoLSrzna_',#19393
#1928660= IFCAXIS2PLACEMENT3D(#1928549,#1938506,#1938514);
```

Fig. 3. A comparison of an opening specification in the Pipes DSL (left) and the IFC-EXPRESS format (right) with placement references and metadata.

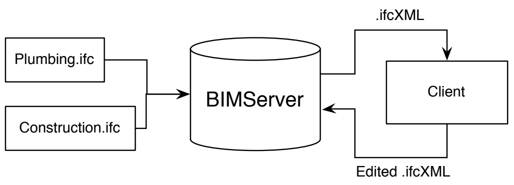


Fig. 4. Overall product workflow starting with a merge of two models. The user can edit a subset of the model through the client-side application.

level elements, each with a physical location specified by the LocalPlacement reference. Figure 3 depicts an example of an opening as it is defined in the Pipes DSL in an uncluttered and manageable way, compared to the IFC-EXPRESS format. The textual syntax has been created from the Pipes DSL meta model using Xtext, which also provides an editor with syntax highlighting and auto-completion. The editor is launched as an Eclipse instance.

4.3 BIMServer

The overall workflow of the final product is shown in Figure 4. As described in Jørgensen's workflow document [5], a construction model and a plumbing model are combined into a single model that needs to be verified. Openings need to be present where the plumbing model defines pipes are to be installed. The user merges these models on BIMServer as the first step of the workflow. When the models have been merged, the client application can retrieve the merged model as ifcXML. The user is now allowed to edit and add elements to the subset of the model on the client side, before sending the building model back to the BIMServer as ifcXML.

Note that the prototype solution does not take concurrent editing by multiple clients into account, although BIMServer does support version control and change notifications for this exact purpose.

4.4 Client Application of the Prototype

The client side of the prototype is implemented as two MWE2 workflows and a textual editor for the Pipes DSL. The workflows, called IFC2Pipes and Pipes2IFC, invoke a series of workflow components, implemented in Xtend. The IFC2Pipes workflow executes the transformations from IFC to the Pipes DSL, after which the textual editor can be used to modify the Pipes DSL model. When the editing is complete, the Pipes2IFC workflow is run to update the IFC model, based on the changes in the Pipes DSL model.

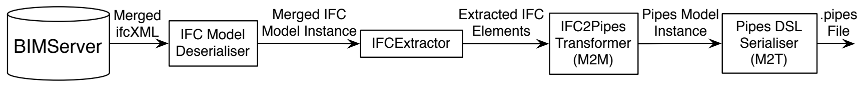


Fig. 5. IFC2Pipes workflow. Processes ifcXML from BIMServer and outputs a .pipes file, loadable in the Pipes DSL editor.

IFC2Pipes Figure 5 illustrates the first workflow. The newest revision of the IFC model is downloaded from BIMServer as ifcXML, and cached on disk. This is deserialised by the deserialiser. After the model has been loaded into memory, the relevant elements (see Section 3.2) are extracted by a workflow component, IFCExtractor. These elements are passed to the IFC2PipesTransformer component, which creates a Pipes DSL model based on the extracted elements. This is done by creating a Pipes DSL model equivalent for each extracted element, preserving the global ID. Finally, using an Xpand template, text conforming to the concrete syntax for Pipes DSL is generated, and serialised to a file on disk.

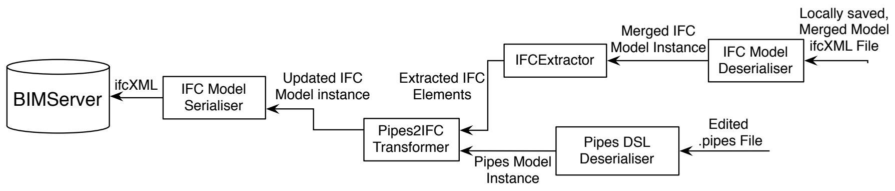


Fig. 6. Pipes2IFC workflow. Updates the main IFC model with the changes written in the Pipes DSL editor, then sends the IFC model back to BIMServer.

Pipes2IFC Figure 6 shows the second workflow. As MWE2 workflows run in their own processes, it is necessary to deserialise the IFC model again, this time using the cached ifcXML. Consequently, the IFCExtractor must also be invoked again to get an instance of the extracted model. The Pipes2IFCTransformer will then compare the extracted model with an instance of the edited Pipes DSL, and update the IFC model with the result. To determine if an element has been added, updated, or removed, Pipes2IFCTransformer will check global IDs in the following way:

- If a global ID appears on an element in the Pipes DSL model, which does not exist in the IFC model, the element is transformed to its IFC equivalent, and added to the IFC model.
- If a global ID exists in both models, but the elements have different metadata, the IFC element is updated.
- If a global ID exists in the IFC model, but not in the Pipes DSL model, it is removed from the IFC model.

When the IFC model has been appropriately updated, a simple cleanup routine removes any orphaned elements or references, and the model is serialised as ifcXML. The ifcXML is then sent to BIMServer as a new revision.

5 Evaluation

In this section we evaluate the solution, with reference to each of the requirements in Section 3.2. Furthermore, we discuss the methods used to verify the correctness of the prototype.

5.1 Requirements Evaluation

R1. Working with a Partial Model We are capable of extracting partial IFC models, and subsequently updating these by using the Pipes DSL editor. In Section 3.1 we presented the informal definition of the IFC subset that we extract, and in Figure 2 we depicted the corresponding Pipes DSL meta model. We focus on a simple partial model, but it would be possible to extend this if other IFC objects were to be added. We have ensured modularity by implementing the extraction process as a modular workflow component. Hence, the extraction process is verifiable as confirmed by the automated tests described in Section 5.2.

R2. Correct Meta Model As explained in Section 4.1, the prototype uses the official ifcXML XSD to generate an Ecore meta model that is both correct and replaceable, in theory. However, there are certain serious drawbacks to this solution.

EMF does not seem to handle the large IFC meta model very well. The IFC meta model generated from the ifcXML XSD, was incomplete, with certain methods not being generated (eUnset methods for fields in some entity classes). Also, EMF generated methods that exceeded the maximum size allowed by the Java Development Kit (64kB)[12]. While it is possible to work around such problems, it is not viable that the generated code has to be corrected.

[12]JDK bug 4262078: `http://bugs.sun.com/view_bug.do?bug_id=4262078`

One of the desirable features provided by using the generated meta model, was the availability of serialisers in EMF. However, these turned out to be too slow for working with realistic IFC models. A real-world model can easily result in load and save times of more than ten minutes. The EMF site lists some performance tips that should solve these problems[13,14], but the improvement was minimal, if any. Thus, all of our tests have been carried out using smaller subsets of IFC models.

R3. Valid Model Transformations To validate our transformations, both automated white box tests and black box test are used. The testing methods are described in detail in Section 5.2, and the results can be found in Appendices A and B.

R4. A Simple DSL Section 4.2 describes the benefits of the DSL implementation. It serves as a proof of concept that other subdomains of IFC can be edited in the same way. Being a modular component of the entire prototype, the textual Pipes DSL editor could be substituted with a visual one.

R5. Structural Editing The prototype supports the addition and removal of both flow segments and openings, and thus the requirement is fulfilled.

5.2 Verification Methods

We have implemented automated test workflows for verifying that our transformation components work as desired. The IFCExtractor, IFC2PipesTransformer, and Pipes2IFCTransformer components (see Figure 6) are the most interesting to validate, as these are the central components containing most of the transformation logic. Furthermore, black box tests of the entire prototype have been conducted with four input test buildings. These tests check that both the IFC model and the Pipes model is left in a valid state after running each workflow component. The results of the tests can be found in Appendices A and B.

5.3 Threats to validity

Internal Validity When a prototype is primarily black box tested, it is natural to question the correctness of all the individual components. However, we argue that it is highly likely that any internal errors, undiscoverable by the black box tests, will either be discovered by the automated tests or result in program crashes, as the workflow components are highly modular with well-defined interfaces raising error flags if invalid input is provided.

[13]EMF Performance Tips: `http://www.eclipse.org/modeling/emf/docs/performance/EMFPerformanceTips.html`

[14]Performance and Extensibility with EMF: `http://www.slideshare.net/kenn.hussey/performance-and-extensibility-with-emf`

Using EMF we have encountered a substantial amount of technical difficulties related only to the Eclipse IDE and its modelling tools. We identify these tools as an internal threat to the validity of the prototype, as our trust in them has been decreasing over the course of the project.

External Validity Retrieving building models for testing has been challenging. Given the fact that most modern visual BIM tools are very complex, we were unable to produce meaningful construction and plumbing models for testing, and had to rely on external contacts for providing test models (see Section 8). That means four example models were used for testing, which is considered sufficient, but not optimal. More tests with more example models must be performed to solidify the claim of general applicability of the prototype to all building models.

6 Ideas for Future Projects

An overall goal of this pilot study is to increase the BIM knowledge of the environmentally concerned ITU research initiative, Energy Futures. With this project as a point of departure, we have developed the following ideas for future follow-up projects.

6.1 Tools for Custom Open Source IFC Meta Model

At the core of the prototype described in this paper, is the IFC meta model. As described in Section 5.1, our meta model generated from the ifcXML XSD has several problems, such as the lack of inverse relationships. Most of these problems could be solved by using a better, customised Ecore meta model, such as one made by Steel. However, using this meta model would require writing serialisers, which is not a trivial task. With Steel's Ecore meta model as a starting point, one could imagine an open source project in which a full-fledged serialiser and deserialiser for the IFC-EXPRESS format is developed, along with a revision of the IFC meta model. This would enable developers to benefit from an independent, open source toolset for working with IFC in EMF, making them less dependent on proprietary tools.

6.2 Synchronisation

If BIM client software is to be useful in a real world scenario when working on a partial model, it should be able to synchronise with the central model. As it was outside the scope of this project to require this kind of synchronisation for concurrent editing scenarios, our prototype is rather static compared to what is desirable for a final product. To be able to approach the synchronisation problem, we believe it is possible to expand on the current transformations implemented with Xtend. The addition of tracing would make it possible to track which transformation rules are applied to which elements in the source and target models, thus making it easier to verify the correctness of the transformations.

For this purpose, Xtend offers a tracing package.[15] A project on synchronisation could investigate how a simple domain like Pipes DSL can handle proper synchronisation with a source model, possibly using BIMServer.

6.3 Structural Changes

Handling structural changes in the source model is challenging. With Pipes DSL, we simply insert new elements without taking changes in the source model into account, which is too naïve to work in real-world scenarios. For example, when inserting a new pipe, an existing cost or energy object in the building model might need to reference this newly inserted object. We have allowed this type of constraint to be broken by our prototype, as it is not clear how dependencies and constraints should be checked. We believe that domain specific software, capable of defining constraints for the IFC model and automatically validating it, would be useful. The project would require a clear definition of the target domain and a study of relevant real-world models.

7 Related Work

The problem of partial model editing and synchronisation is a common one. Here, we present a couple of closely related projects as well as other important alternative approaches and resources.

Nour argues that the exchange of partial models is one of the central challenges of widespread use of a central model for BIM, mainly IFC [10]. Although we do not adopt his idea of end-user filtering through the use of MVD, our approach does incorporate software filtering for defining a working subset of IFC.

Karola et al. describe an early solution to the partial model extraction and update problem implemented as middleware. The BSPro COM-Server bridges various existing tools, each requiring a specific subset of IFC, making interoperability possible [6].

An alternative to our approach, is projectional editing. As specified by Tomasetti et al. [13], projectional editors operate directly on the model, and thus no model synchronisation is involved. An example of a projectional editor is Jetbrains MPS.[16] This approach trades loss of notational flexibility for a guarantee of model consistency [14]. Consequently, in order to bring the notation as close to the domain as possible, a projectional editor was not desirable for our solution.

Another way to derive and edit subsets of an IFC model is through the use of the BIM Query Language [8]. This language provides access to IFC models via an SQL-like syntax. The queried model is stored on a BIMServer. While the BIM Query Language does provide much more flexibility in terms of extraction and editing of IFC data when compared to the prototype presented in this paper, it does not provide the same closeness to the chosen subdomain.

[15]Xtend tracing package: `http://download.eclipse.org/modeling/m2t/xpand/javadoc/1.2/org/eclipse/xtend/util/stdlib/tracing/package-summary.html`

[16]Jetbrains MPS can be found at `http://www.jetbrains.com/mps/`

Staworko et al. define and discuss the view-update problem for instances where both the view and the source are XML documents [1]. Any Ecore model, i.e. also the Pipes DSL model, can be converted to XMI, thus making the formalisations of both the view-update problem and its solutions are very relevant for the future work on our synchronisation logic.

The model transformations used in our prototype has mainly been realised through the direct manipulation approach described by Czarnecki et al. [3], facilitated by the dispatch method feature of Xtend and the clarity of the MWE2.

As the design of the concrete syntax for the presented DSL has been a minor aspect of our prototype, it is far from perfect. The guidelines for DSL design presented by Karsai et al. [7] has been used as the basis for the preliminary design of the DSL, and further work in this direction will most likely benefit from the guidelines.

8 Conclusion

As evaluated in Section 5, the requirements for a partial editor prototype set forward in Section 3.2, have been met. Furthermore, the correctness of the output has been verified with an account for threats to validity. Specifically, we have...

- Shown that it is feasible to define and edit a partial IFC model for a specific domain using a concrete syntax, the Pipes DSL.
- Developed an editor for the Pipes DSL with syntax highlighting and auto-completion using Xtext. A comparison between the Pipes DSL syntax and the IFC-EXPRESS syntax was shown in Figure 3.
- Changed metadata and edited the structure of a building model using the Pipes DSL. To the extent that all our tests run as expected, we believe that these changes happen correctly.
- Been able to interface with a BIMServer instance to store building models and merge a construction model and a plumbing model into a coherent IFC model used by the workflows shown in Figures 5 and 6.
- Presented ideas for projects that can improve upon the prototype and evolve the area of open source BIM software.

We believe that through the exploration of how model-driven development can be used with IFC, this pilot study has provided a valuable starting point for further research in this area, which the ITU research initiative Energy Futures can build upon. In conclusion, we find that editing of a partial IFC model defined by a concrete, real-world domain is feasible using the model-driven methodology.

Acknowledgements Special thanks to associate professor at Aalborg University Kaj A. Jørgensen for providing example IFC models as well as overall guidance in selecting a fitting IFC domain for our project. Furthermore, we thank construction management student at Copenhagen School of Design and Technology, Mathias Demant for providing example IFC models mainly used for testing.

References

[1] Sławek Staworko, Iovka Boneva, and Benoît Groz. The view update problem for xml. In *Updates in XML (EDBT Workshop Proceedings)*, 2010.

[2] Vladimir Bazjanac and Drury Crawley. Industry foundation classes and interoperable commercial software in support of design of energy-efficient buildings. *Proceedings of Building Simulation 99, Kyoto*, 1999.

[3] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal — Model-driven software development*, 2006.

[4] Kaj A. Jørgensen, Jørn Skauge, Per Christiansen, Kjeld Svidt, Kristian Birch Sørensen, and John Mitchell. Use of ifc model servers. 2010.

[5] Kaj Asbjørn Jørgensen. Collaboration regarding modelling of construction and building services. Unpublished manuscript available on request, 2012.

[6] Antti Karola, Hannu Lahtela, Reijo Hänninen, Rob Hitchcock, Qingyan Chen, Stephen Dajka, and Kim Hagström. Bspro com-server—interoperability between software tools using industrial foundation. *Energy and Buildings — ENERG BLDG*, 34(9):901–907, 2002.

[7] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. Design guidelines for domain specific languages. *9th OOPSLA Workshop on Domain-Specific Modeling*, 2009.

[8] Wiet Mazairac and Jakob Beetz. Towards a framework for a domain specific open query language for building information models. *Proceedings DDSS conference Eindhoven*, 2010.

[9] Nick Nisbet and Thomas Liebech. *ifcXML Implementation Guide*, 2007.

[10] Mohamed Nour. A graphical user interface for handling ifc partial model exchange. *http://www.inpro-project.eu/publications.asp*, 2008.

[11] James Steel, Robin Drogemuller, and Bianca Toth. Model interoperability in building information modelling. *Software and Systems Modeling*, 11(1):99–109, 2010.

[12] Jim Steel, Keith Duddy, and Robin Drogemuller. A transformation workbench for building information models. 2011.

[13] Federico Tomasetti and Marco Torchiano. Prede: a projectional editor for the eclipse modeling framework. *http://2011.eclipse-it.org/ProcEclipse-IT11/*, 2011.

[14] Markus Voelter. Embedded software development with projectional language workbenches. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, *MoDELS (2)*, volume 6395 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2010.

A Results of Black Box Tests

A.1 Input Models

For these tests, we have several input models: One of them is a subset of a real IFC model, provided by Kaj Jørgensen. The rest of the test models have been created by Mathias Demant for our tests, and do not represent any real buildings. The files can be found in Prototype/Model2ModelWorkflows/NanoLabBuilding/ on the attached medium. The results can be seen in Figures 7, 8, 9, and 10.

A.2 Tests

For each case, we will perform the following tests between the two MWE2 workflows:

Test 1: Modifying the metadata (name, description) for one of each of the three product types (wall, opening, flow segment).
Expected result: The output IFC model differs from the input IFC model in that it has altered metadata (name, description).

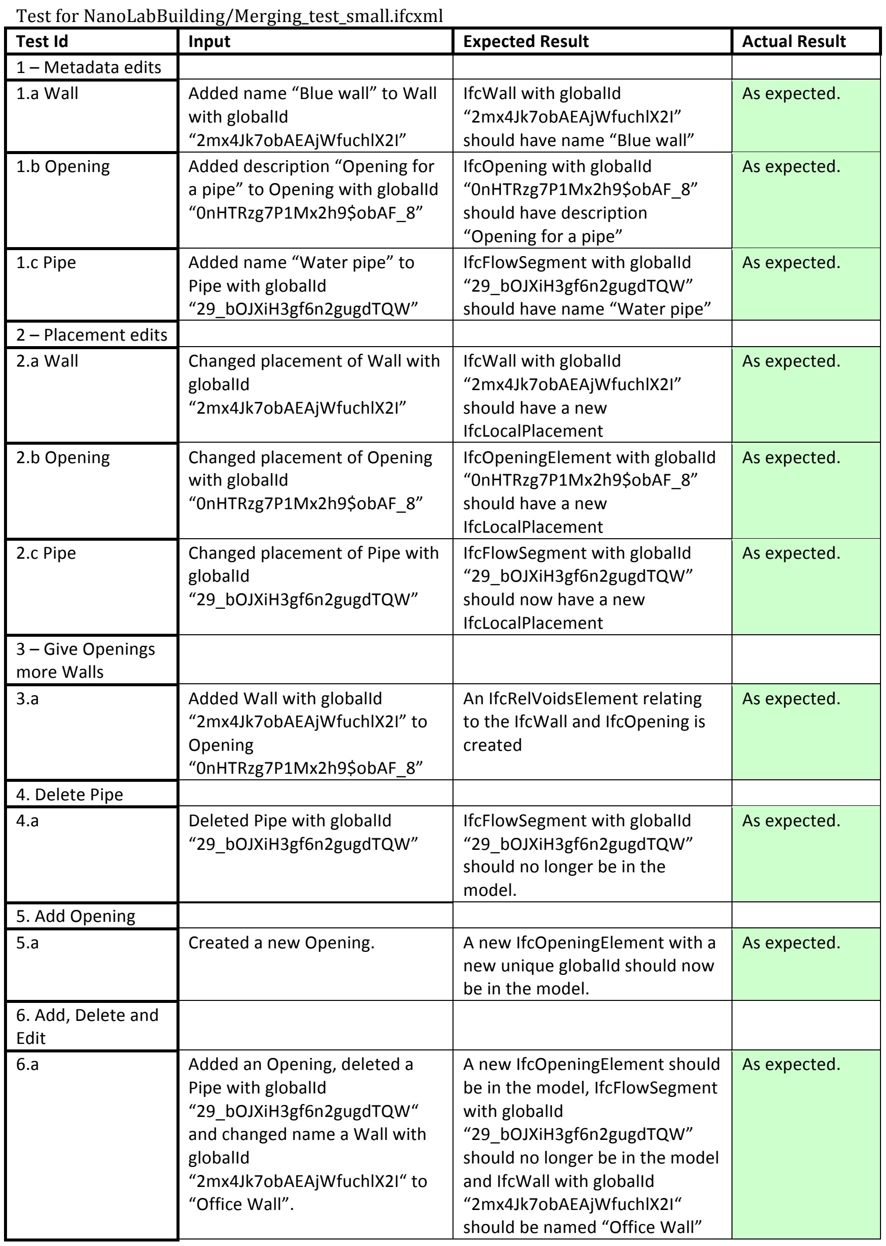
Test 2: Modifying the placement of one of each of the three products types.
Expected result: A new IFCLocalPlacement object now exists in the IFC output model with the modified metadata.

Test 3: Modifying the list of walls in a opening element.
Expected result: A new IFCRelVoidElement is present in the output IFC model, compared to the input mode, linking the wall and opening.

Test 4: Deleting a Pipe.
Expected result: The output IFC model no longer contains the deleted flow segment. Any created orphans should be garbage collected.

Test 5: Adding an Opening.
Expected result: The output IFC model should now contain the created opening, along with any new objects referenced by the opening (LocalPlacement, Axis2Placement3d)

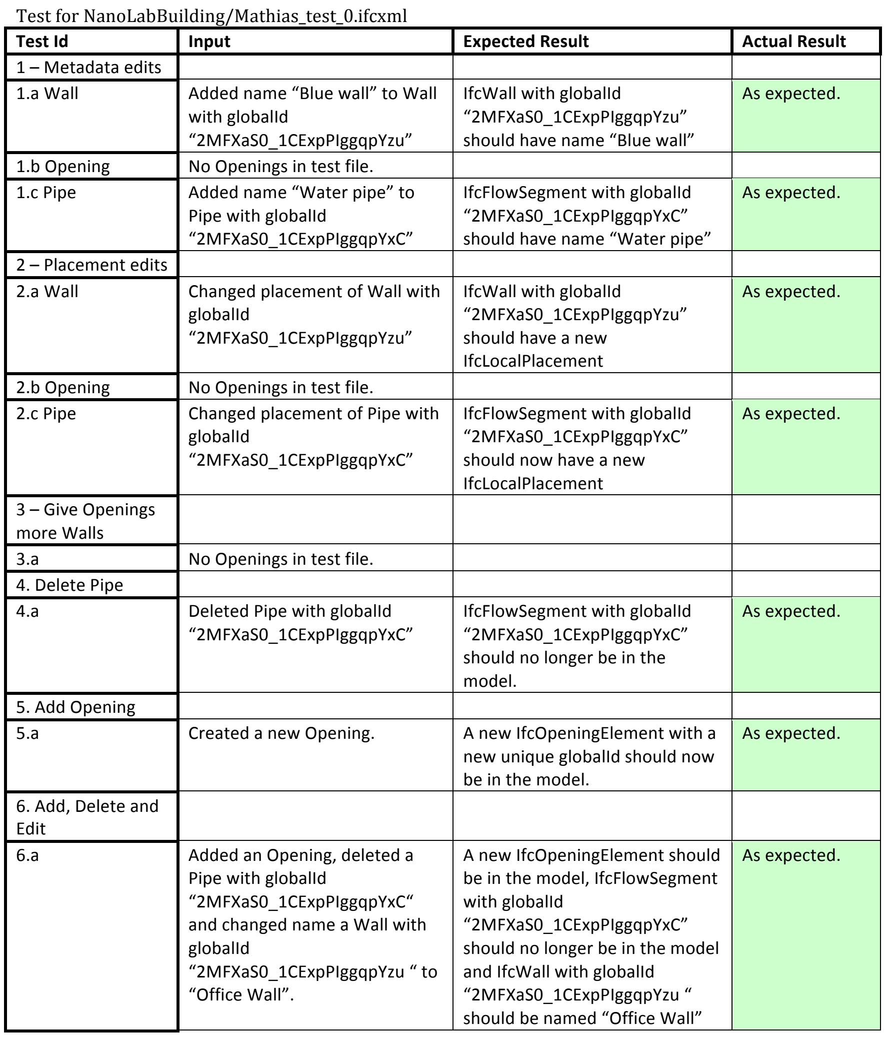
Test 6: Adding one opening, deleting one flow segment, and modifying metadata for one wall.
Expected result: The output IFC model should now contain the created opening (along with referenced objects), the flow segment should be deleted (along with orphans), and the metadata for the wall should be updated.



Test for NanoLabBuilding/Merging_test_small.ifcxml

Test Id	Input	Expected Result	Actual Result
1 – Metadata edits			
1.a Wall	Added name “Blue wall” to Wall with globalId “2mx4Jk7obAEAjWfuchlX2I”	IfcWall with globalId “2mx4Jk7obAEAjWfuchlX2I” should have name “Blue wall”	As expected.
1.b Opening	Added description “Opening for a pipe” to Opening with globalId “0nHTRzg7P1Mx2h9$obAF_8”	IfcOpening with globalId “0nHTRzg7P1Mx2h9$obAF_8” should have description “Opening for a pipe”	As expected.
1.c Pipe	Added name “Water pipe” to Pipe with globalId “29_bOJXiH3gf6n2gugdTQW”	IfcFlowSegment with globalId “29_bOJXiH3gf6n2gugdTQW” should have name “Water pipe”	As expected.
2 – Placement edits			
2.a Wall	Changed placement of Wall with globalId “2mx4Jk7obAEAjWfuchlX2I”	IfcWall with globalId “2mx4Jk7obAEAjWfuchlX2I” should have a new IfcLocalPlacement	As expected.
2.b Opening	Changed placement of Opening with globalId “0nHTRzg7P1Mx2h9$obAF_8”	IfcOpeningElement with globalId “0nHTRzg7P1Mx2h9$obAF_8” should have a new IfcLocalPlacement	As expected.
2.c Pipe	Changed placement of Pipe with globalId “29_bOJXiH3gf6n2gugdTQW”	IfcFlowSegment with globalId “29_bOJXiH3gf6n2gugdTQW” should now have a new IfcLocalPlacement	As expected.
3 – Give Openings more Walls			
3.a	Added Wall with globalId “2mx4Jk7obAEAjWfuchlX2I” to Opening “0nHTRzg7P1Mx2h9$obAF_8”	An IfcRelVoidsElement relating to the IfcWall and IfcOpening is created	As expected.
4. Delete Pipe			
4.a	Deleted Pipe with globalId “29_bOJXiH3gf6n2gugdTQW”	IfcFlowSegment with globalId “29_bOJXiH3gf6n2gugdTQW” should no longer be in the model.	As expected.
5. Add Opening			
5.a	Created a new Opening.	A new IfcOpeningElement with a new unique globalId should now be in the model.	As expected.
6. Add, Delete and Edit			
6.a	Added an Opening, deleted a Pipe with globalId “29_bOJXiH3gf6n2gugdTQW“ and changed name a Wall with globalId “2mx4Jk7obAEAjWfuchlX2I“ to “Office Wall”.	A new IfcOpeningElement should be in the model, IfcFlowSegment with globalId “29_bOJXiH3gf6n2gugdTQW” should no longer be in the model and IfcWall with globalId “2mx4Jk7obAEAjWfuchlX2I“ should be named “Office Wall”	As expected.

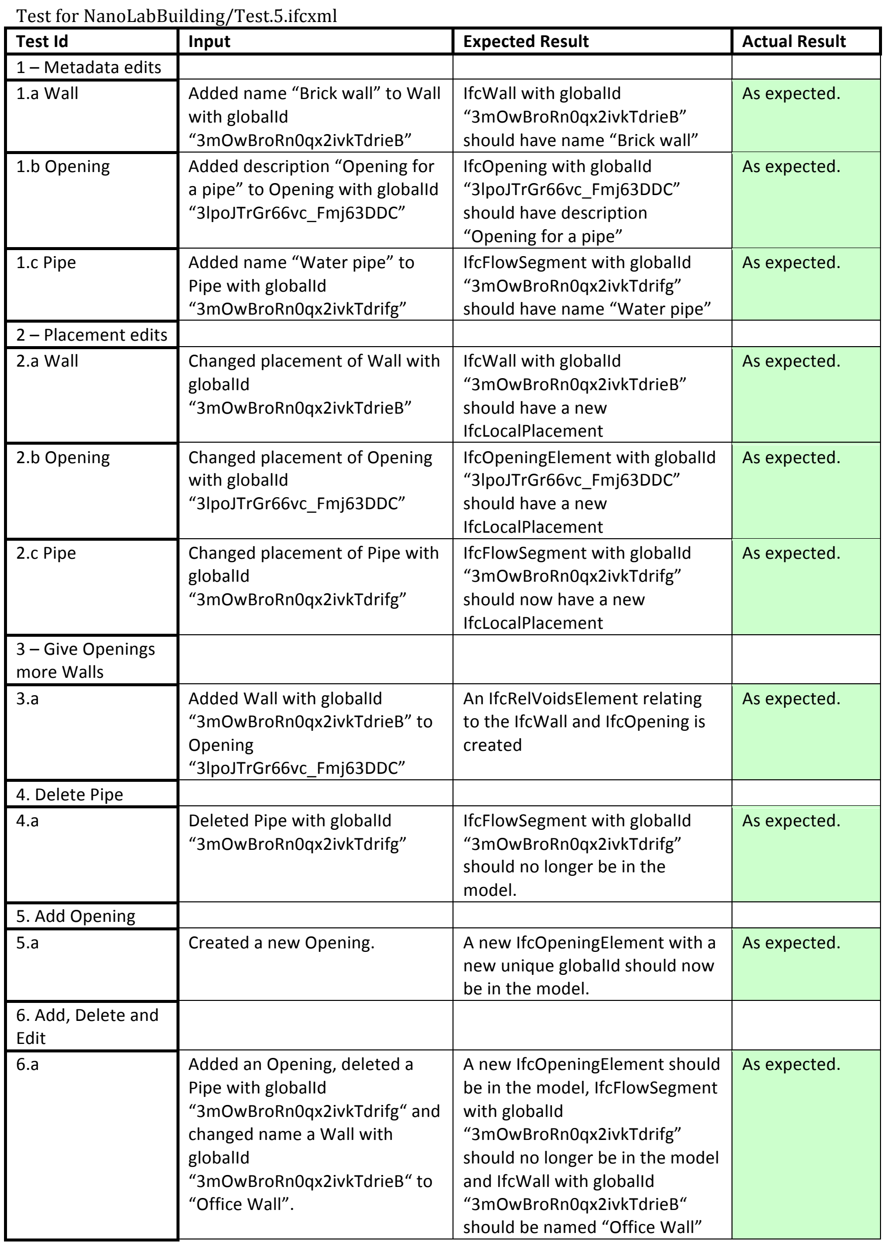
Fig. 7.



Test for NanoLabBuilding/Mathias_test_0.ifcxml

Test Id	Input	Expected Result	Actual Result
1 – Metadata edits			
1.a Wall	Added name “Blue wall” to Wall with globalId “2MFXaS0_1CExpPIggqpYzu”	IfcWall with globalId “2MFXaS0_1CExpPIggqpYzu” should have name “Blue wall”	As expected.
1.b Opening	No Openings in test file.		
1.c Pipe	Added name “Water pipe” to Pipe with globalId “2MFXaS0_1CExpPIggqpYxC”	IfcFlowSegment with globalId “2MFXaS0_1CExpPIggqpYxC” should have name “Water pipe”	As expected.
2 – Placement edits			
2.a Wall	Changed placement of Wall with globalId “2MFXaS0_1CExpPIggqpYzu”	IfcWall with globalId “2MFXaS0_1CExpPIggqpYzu” should have a new IfcLocalPlacement	As expected.
2.b Opening	No Openings in test file.		
2.c Pipe	Changed placement of Pipe with globalId “2MFXaS0_1CExpPIggqpYxC”	IfcFlowSegment with globalId “2MFXaS0_1CExpPIggqpYxC” should now have a new IfcLocalPlacement	As expected.
3 – Give Openings more Walls			
3.a	No Openings in test file.		
4. Delete Pipe			
4.a	Deleted Pipe with globalId “2MFXaS0_1CExpPIggqpYxC”	IfcFlowSegment with globalId “2MFXaS0_1CExpPIggqpYxC” should no longer be in the model.	As expected.
5. Add Opening			
5.a	Created a new Opening.	A new IfcOpeningElement with a new unique globalId should now be in the model.	As expected.
6. Add, Delete and Edit			
6.a	Added an Opening, deleted a Pipe with globalId “2MFXaS0_1CExpPIggqpYxC“ and changed name a Wall with globalId “2MFXaS0_1CExpPIggqpYzu “ to “Office Wall”.	A new IfcOpeningElement should be in the model, IfcFlowSegment with globalId “2MFXaS0_1CExpPIggqpYxC” should no longer be in the model and IfcWall with globalId “2MFXaS0_1CExpPIggqpYzu “ should be named “Office Wall”	As expected.

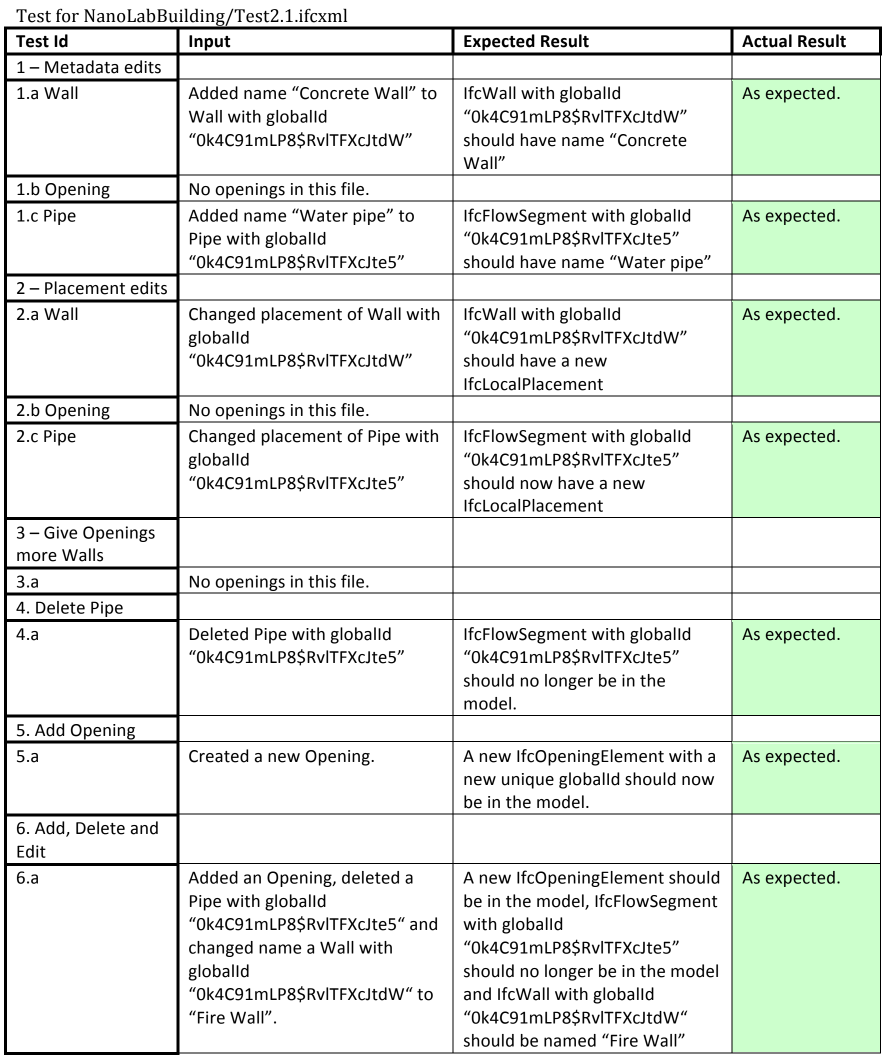
Fig. 8.



Test for NanoLabBuilding/Test.5.ifcxml

Test Id	Input	Expected Result	Actual Result
1 – Metadata edits			
1.a Wall	Added name “Brick wall” to Wall with globalId “3mOwBroRn0qx2ivkTdrieB”	IfcWall with globalId “3mOwBroRn0qx2ivkTdrieB” should have name “Brick wall”	As expected.
1.b Opening	Added description “Opening for a pipe” to Opening with globalId “3lpoJTrGr66vc_Fmj63DDC”	IfcOpening with globalId “3lpoJTrGr66vc_Fmj63DDC” should have description “Opening for a pipe”	As expected.
1.c Pipe	Added name “Water pipe” to Pipe with globalId “3mOwBroRn0qx2ivkTdrifg”	IfcFlowSegment with globalId “3mOwBroRn0qx2ivkTdrifg” should have name “Water pipe”	As expected.
2 – Placement edits			
2.a Wall	Changed placement of Wall with globalId “3mOwBroRn0qx2ivkTdrieB”	IfcWall with globalId “3mOwBroRn0qx2ivkTdrieB” should have a new IfcLocalPlacement	As expected.
2.b Opening	Changed placement of Opening with globalId “3lpoJTrGr66vc_Fmj63DDC”	IfcOpeningElement with globalId “3lpoJTrGr66vc_Fmj63DDC” should have a new IfcLocalPlacement	As expected.
2.c Pipe	Changed placement of Pipe with globalId “3mOwBroRn0qx2ivkTdrifg”	IfcFlowSegment with globalId “3mOwBroRn0qx2ivkTdrifg” should now have a new IfcLocalPlacement	As expected.
3 – Give Openings more Walls			
3.a	Added Wall with globalId “3mOwBroRn0qx2ivkTdrieB” to Opening “3lpoJTrGr66vc_Fmj63DDC”	An IfcRelVoidsElement relating to the IfcWall and IfcOpening is created	As expected.
4. Delete Pipe			
4.a	Deleted Pipe with globalId “3mOwBroRn0qx2ivkTdrifg”	IfcFlowSegment with globalId “3mOwBroRn0qx2ivkTdrifg” should no longer be in the model.	As expected.
5. Add Opening			
5.a	Created a new Opening.	A new IfcOpeningElement with a new unique globalId should now be in the model.	As expected.
6. Add, Delete and Edit			
6.a	Added an Opening, deleted a Pipe with globalId “3mOwBroRn0qx2ivkTdrifg“ and changed name a Wall with globalId “3mOwBroRn0qx2ivkTdrieB“ to “Office Wall”.	A new IfcOpeningElement should be in the model, IfcFlowSegment with globalId “3mOwBroRn0qx2ivkTdrifg” should no longer be in the model and IfcWall with globalId “3mOwBroRn0qx2ivkTdrieB“ should be named “Office Wall”	As expected.

Fig. 9.



Test for NanoLabBuilding/Test2.1.ifcxml

Test Id	Input	Expected Result	Actual Result
1 – Metadata edits			
1.a Wall	Added name “Concrete Wall” to Wall with globalId “0k4C91mLP8$RvlTFXcJtdW”	IfcWall with globalId “0k4C91mLP8$RvlTFXcJtdW” should have name “Concrete Wall”	As expected.
1.b Opening	No openings in this file.		
1.c Pipe	Added name “Water pipe” to Pipe with globalId “0k4C91mLP8$RvlTFXcJte5”	IfcFlowSegment with globalId “0k4C91mLP8$RvlTFXcJte5” should have name “Water pipe”	As expected.
2 – Placement edits			
2.a Wall	Changed placement of Wall with globalId “0k4C91mLP8$RvlTFXcJtdW”	IfcWall with globalId “0k4C91mLP8$RvlTFXcJtdW” should have a new IfcLocalPlacement	As expected.
2.b Opening	No openings in this file.		
2.c Pipe	Changed placement of Pipe with globalId “0k4C91mLP8$RvlTFXcJte5”	IfcFlowSegment with globalId “0k4C91mLP8$RvlTFXcJte5” should now have a new IfcLocalPlacement	As expected.
3 – Give Openings more Walls			
3.a	No openings in this file.		
4. Delete Pipe			
4.a	Deleted Pipe with globalId “0k4C91mLP8$RvlTFXcJte5”	IfcFlowSegment with globalId “0k4C91mLP8$RvlTFXcJte5” should no longer be in the model.	As expected.
5. Add Opening			
5.a	Created a new Opening.	A new IfcOpeningElement with a new unique globalId should now be in the model.	As expected.
6. Add, Delete and Edit			
6.a	Added an Opening, deleted a Pipe with globalId “0k4C91mLP8$RvlTFXcJte5“ and changed name a Wall with globalId “0k4C91mLP8$RvlTFXcJtdW“ to “Fire Wall”.	A new IfcOpeningElement should be in the model, IfcFlowSegment with globalId “0k4C91mLP8$RvlTFXcJte5” should no longer be in the model and IfcWall with globalId “0k4C91mLP8$RvlTFXcJtdW“ should be named “Fire Wall”	As expected.

Fig. 10.

B Automated Tests

The workflow components used for tests can be found in the Eclipse project under: Model2ModelWorkflows/src/tests. These components work as a part of two test workflows similar to the ones used in the actual prototype. The only difference is that test components have been inserted after extraction and transformation components in the test workflow. The IFC2Pipes.mwe2 workflow is thus tested by the IFC2PipesTest.mwe2 workflow and the Pipes2IFC.mwe2 workflow is tested by the Pipes2IFCTest.mwe2 workflow.

Testing Element Extraction The TestIFCExtractor workflow component verifies that the extraction process executed with a correct result. On any input building it checks that the number of extracted IfcProducts corresponds to the number of products that should be extracted by looking directly at the ifcXML. For each of the extracted product elements it tests for a valid globalId and Id. The tests have been run with the Merging_test_small model, only as this is the most realistic model. All assertions succeed in this test.

Testing IFC to Pipes Model Transformation The following consistency verification between elements in the IFC and Pipes models are carried out:

- IfcFlowSegment <-> FlowSegment
- IfcOpening <-> Opening
- IfcWall <-> Wall
- IfcLocalPlacement <-> LocalPlacement
- IfcAxis2Placement3d <-> Axis2Placement3D
- IfcDirection <-> Direction

By going through all extracted IFC products after the model to model transformation and comparing them to the elements of the Pipes model, the TestTransformers test component automates transformation testing of the subdomain. All consistency tests on Merging_test_small pass.

Testing Pipes-to-IFC Update Routine The TestTransformer test component is used for the purpose of testing the Pipes-to-IFC update routine as well. The same consistency checks are performed after the Pipes2IFCTransformer, which executes the update routine. The same element relations as above are tested. All consistency tests on Merging_test_small pass.