TUM OpenInfraPlatform: The Open-Source BIM Visualisation Software

Helge Hecht[1] and Štefan Jaud[1]

[1]Chair for Computational Modeling and Simulation · Technische Universität München · Arcisstraße 21 · 80333 München · Germany · E-Mail: helge.hecht@tum.de & ga24yuk@mytum.de

The backbone of building information modeling (BIM) processes and exchanges are digital models which contain the relevant design and production data. Thus, the quality of models being exchanged is of utmost importance to ensure transparency and avoid ambiguities. There are many BIM model viewers available on the market, being proprietary or open, free or costly. Nevertheless, the current BIM software landscape lacks open-source solutions providing user and developer friendly, versatile application programming interfaces (APIs) for management of BIM data up-to-date. TUM OpenInfraPlatform (OIP) is a open-source C++ software which provides format independent data visualisation and inspection tools. OIP supports the whole programming process – from early-binding class generation to complex computations during geometry generation, visualisation and import/export functionality for various data formats. This paper presents the architecture of the framework and more importantly the rationale behind it. It should serve as a reference to everybody interested to join the cause.

Keywords: BIM, visualisation, Open Source, C++

1 Introduction

In the current developments in the architecture, engineering and construction (AEC) industry, building information modeling (BIM) is steadily replacing and enriching traditional computer aided design (CAD) practises (Borrmann et al., 2017). BIM methods promise to remove the need for blue-prints altogether, since digital representations of designed assets will become the end delivery of projects. These BIM models contain the relevant design and production data needed for a successful completion of a construction project. As such, they represent the backbone of BIM processes and provide a single source of truth to all stakeholders.

Following these developments, it is clear that the quality of models being exchanged is of utmost importance. Clear definitions and standards are crucial in order to validate the models being exchanged between project partners. The digital models are encoded in many different ways and one of the most prominent open standards currently available are the industry foundation classes (IFC) (ISO, 2013). Visualisation of these digital models would allow for easy error detection and verification. Moreover, the responsible actor checking these models needs to maintain independent work flows from other stakeholders and proprietary software solutions. Supporting various data formats eases communication and collaboration of different parties.

Various contributors at the Chair of Computational Modeling and Simulation (CMS) of the Technische Universität München (TUM) have been developing an open-source software suite

named TUM OpenInfraPlatform (OIP) for half-a-decade now (Amann et al., 2019) [1]. It is royalty-free, vendor-neutral software that fulfils the requirements stated above. It bases its internal data model on the IFC standard and provides visual representation of the BIM model. This paper provides an update on its architecture and functionality from the recent rework of its whole code-base.

The paper is structured as follows. In this section a short introduction to the topic of BIM visualisation is presented. Next section provides an overview over similar software suites available on the market and lists their different advantages and disadvantages over OIP. In section 3 the architecture of OIP with its different components and functions is described.

2 Related Works

There are some similar tools available on the market which are listed below with their web-page and responsible party.

CodeSynthesis XSD by Code Synthesis Tolls CC. URL: `https://www.codesynthesis.com/products/xsd/`.

FreeCAD by opensourceBIM. URL: `https://www.freecadweb.org/`.

FZK Viewer by Karlsruhe Institute of Technology (KIT). URL: `https://www.iai.kit.edu/1302.php`.

IFC Engine by RDF. URL: `http://rdf.bg/`

IfcKit by buildingSMART International (bSI). URL: `https://github.com/buildingSMART/IfcDoc/tree/master/IfcKit`.

IfcOpenShell by opensourceBIM. URL: `http://ifcopenshell.org/`.

IFC Query by Bauhaus University Weimar. URL: `http://ifcquery.com/`

IFC Quick Browser by GEM Team Solutions. URL: `http://www.team-solutions.de/2003/07/25/ifc-quick-browser/`

Solibri Model Viewer by Solibri. URL: `https://www.solibri.com/`.

Tekla BIMsight by Trimble. URL: `https://www.tekla.com/tekla-bimsight/`.

xBIM by Lockley et al. (2017). URL: `http://docs.xbim.net/`.

We list the meta-data of these products and projects in Table 1 together with OIP for comparison. Is the software or library open-source, does it provide a viewer of the geometry, does it provide an application programming interface (API) and in which programming language, can it be used cross platform (Windows, Unix), is it an early-binding approach and if so, to which IFC versions is it applicable. This list is by no means exhaustive, however all important open-source projects which passed the prototype stage are listed. It is trivial that all open-source software libraries provide an API.

[1] All code is published on Bitbucket and is versioned with HG: `https://bitbucket.org/tumcms/openinfraplatform/src/default/`

Table 1: An overview of the software suites similar to OIP.

Software	Open-source	Geometry Viewer	API	Language	Cross Platform	Early bind	IFC version
Code Synthesis	✓	✗	✓	C++	✓	✓	any
FreeCAD	✓	✓	✓	C++, Python	✓	✓	2x3, 4
FZK Viewer	✗	✓	✗	–	✗	✓	2x3 – 4x2
IfcEngine	✗	✗	✓	C++, C#	✓	✗	any
IfcKit	✓	✗	✓	C#	✗	✓	2x3, 4
IfcOpenShell	✓	✗	✓	java, python	✓	✓	2x3 – 4x1
IFC Query	✓	✓	✓	C++	✓	✓	2x3 – 4
IFC Quick Browser	✗	✗	✗	–	✗	✗	any
Solibri Model Viewer	✗	✓	✓	java	✗	✓	2x3 – 4x1
Tekla BIMsight	✗	✓	✓	java	✓	✓	2x3, 4
XBim	✓	✓	✓	C++,C#	✗	✓	2x3 – 4x1
OIP	✓	✓	✓	C++	✓	✓	any

3 TUM OpenInfraPlatform

OIP relies solely on IFC as its internal data model and other formats are converted to IFC since it is a powerful and global open standard in the AEC industry. To visualise the data, the possibly complex and abstract geometric representations used in IFC are converted to simple surfaces by the geometry converter. These meshes are then displayed in an interactive 3D viewport within the graphical user interface (GUI). The underlying rendering engine is based on the BlueFramework, an open-source library also developed by TUM [2].

The software uses CMake as build system to provide user friendly dependency management and a modular framework architecture. Settings required for the compilation process are specified in CMake during project configuration to ease collaboration and code portability for different platforms or system configurations. OIP is divided into the following parts: (i) the early-binding generator, (ii) the GUI, (iii) the core components, and (iv) the additional plugin modules. These are explained in detail in the following sections and an overview is presented in Figure 1.

The core components include the internal data representation as well as the converter for import and export operations for different data formats such as Objekt Katalog Straße (OKSTRA) or LandXML. The internal data representation is implemented in an early-binding approach, modelling all objects specified in IFC in a C++ library. This mapping is automated by the early-binding generator since implementing all classes manually is time consuming and merely a replication of performed work since all required information is contained within the

[2] All code is published on Bitbucket and is versioned with HG: `https://bitbucket.org/tumcms/blueframework/src/default/`

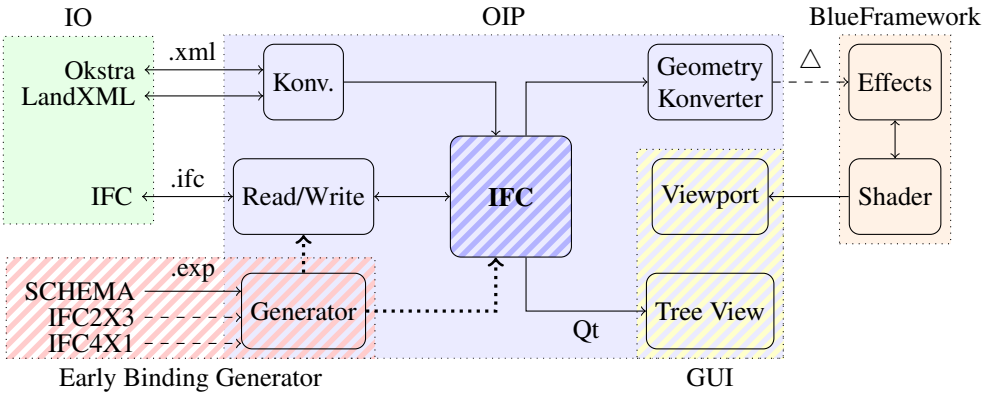

Figure 1: Overview of the different components of OIP.

schema specification. While the early-binding generator could also be used as a standalone tool, it is integrated into the framework.

3.1 Early-Binding Generator

The early-binding generator parses an EXPRESS schema specification and translates it into C++ classes which mirror the classes specified in the EXPRESS language. The schema file is processed using a Yacc and Bison parser to generate a corresponding meta-model representation in C++. This in-data model is then used to generate the early-binding IFC C++ classes, creating a bijective mapping between EXPRESS and C++.

Since both languages use different concepts and structures, expressing a specification modelled in EXPRESS within the domain of C++ can be quite challenging. To remain as close to the original formulation as possible while following an early-binding approach, we introduced an intermediate layer written in C++ which models the language rules of EXPRESS. The main reasoning behind this is the discrepancy between the basic types of both languages and the need for a common set of base functionality for all classes to ease usage as well as reading from and writing to files. These classes are the backbone for the final model implementation. The generator also produces the classes for reading and writing IFC files.

While C++ only distinguishes very basic built-in types (`char`, `int`, `float`, `double` and `bool`) and user-defined types, EXPRESS includes more abstract built-in types like lists and strings and has two mutually exclusive categories of user-defined classes (ISO, 2004). The C++ standard library (STL) provides some similar higher level functionality which extends the built-in types. Despite being a library of user-defined classes and functions they're considered part of the language standard and are supported by all compilers and platforms.

A central aspect of EXPRESS is the distinction between types and entities. They make up the user-defined classes in EXPRESS. Types can merely exist as attributes of entities while entities are *standalone* objects which are referenced by their ID in the scope of the model.

The attributes of entities can be types and references to entities. Since types have no ID, they are *composed* in the attributes while entities are *aggregated* - they are not owned but referred to by the respective entity (Sebesta, 1996). All entity attributes can be *optional* (ISO, 2004). This concept of optionals and references can be modelled in C++ using `std::optional` and `std::weak_ptr` respectively. A user defined type can be (i) a built-in type, (ii) a restricted type, (iii) a container-type, (iv) a select, (v) or an enumeration. The first category comprises simple wrappers like `IfcInteger` which could be expressed by a `typedef` or `using` directive. Restricted-types are types having `WHERE` clauses mapping the object to a condition which has to evaluate to `true`. An example would be the `IfcPositiveInteger`, which is an `IfcInteger` restricted to positive numbers. Note that EXPRESS lacks the concept of unsigned arithmetic types. A container-type is a set, list, array or bag of types. It has to be kept in mind that a container-type is different from a built-in container of types – compare `IfcCartesianPointList` and `IfcCompoundPlaneAngleMeasure`. The enumeration type is – similar to its C++ equivalent – a named value out of a fixed size collection.

Besides the `WHERE` clauses, all of these concepts have corresponding elements in C++ or the STL. Most user-defined types in EXPRESS are merely wrappers of built-in types. Implicitly converting these wrappers to their underlying types makes additional interface layers to other APIs redundant and gives access to the C++ STL functionality. We therefore implemented this concept by defining a template class `ValueType<T>` which has an attribute of type `T` – the underlying type – and defines implicit conversion and assignment operators as well as reading and writing. In order to provide string as well as IO operations, the enumeration type is implemented as a template class `EnumType<enum>` derived from `ValueType<enum>`, inheriting all implicit operators and defining some additional functionality. The built-in container types are implemented as sub-classes of the C++ STL containers with additional functionality to model the concept of cardinality restriction from EXPRESS. Container-types are derived from these and implement the common `TYPE` interface for string and IO operations.

While these concepts are relatively easy to express in C++ because of their simple and well defined form, select types are more complex. A `SELECT` as specified in EXPRESS is a `TYPE` in the sense of not having attributes or an ID while being one out of a set of types and/or entities. The reasoning behind this becomes more clear considering the aspect that the `SELECT` is a `TYPE` that can hold a type or a reference to an entity, not the entity itself. Selects are used quite often to combine variants of a specification with multiple spatial dimensions, i.e `IfcGeometricSetSelect` is a `IfcPoint`, `IfcCurve` or `IfcSurface` (ISO, 2013). They can be modelled in various ways and have no clear counterpart in C++. It strongly resembles the idea of unions, which provide different ways of accessing the same physical address in memory through members with different names and types. A more sophisticated version of unions called *variant* is provided by the boost library (David Abrahams, 2004). It was included in the C++ standard with C++17. The `variant` is a template class with variable sized template arguments where the actual contained value can be queried and accessed at runtime. The `std::variant` class makes error handling and type checks possible which isn't the case for unions. We implemented the select as specialisation of `ValueType` holding a `boost::variant` to inherit the common `TYPE` interface, to make use of the functionality provided by the boost library and to remain in the C++14 language standard (David Abrahams, 2004).

All classes described in an IFC schema specification inherit from their representative intermediate layer class and from their base classes as described in the specification. To allow for runtime type inspection (RTTI), an additional library called visit_struct is used. It is based on macros which inject RTTI functionality into non trivial classes and provides a uniform interface for accessing the types and names of attributes and functions.

3.2 Geometry Generation

The IFC specification includes descriptions for object geometries which can be used to visualise the model (ISO, 2013). The complexity of geometry descriptions ranges from simple Boundary representation (BRep) object to complex mathematical operations like extrusions along spline curves. These representations have to be converted to a simple triangle mesh in order to be visualised with a rendering engine. The implementation in OIP of this conversion process is based on carve, a geometry kernel like OpenCASCADE [3]. It implements a unified interface for all IFC specifications since the geometric entitities and types aren't subject to large changes in different versions of IFC.

3.3 Visualization

The GUI of OIP is implemented with Qt. It is platform independent, comes with a visual programming tool to design widgets in .ui files and includes a wide range of preset user interface (UI) elements which can be connected using a signals and slots system. Each element emits signals upon interaction which can be bound to C++ functions prefixed with the `Q_SLOT` macro as callbacks. The UI includes a tree structured view window to inspect IFC models in a textual manner. It mimics the behaviour of the IfcQuickBrowser software. The textual representation is useful to display non geometric entities within the model such as properties and relations. The tree based layout makes navigating through the file structure and the dependencies between entities intuitive.

The triangulated geometric representations contained in the model are displayed in an interactive viewport in the application's main window. The viewport is implemented using the QViewport class provided by Qt to embed the continuously rendered image in the application. Interactions like mouse movement or clicks can be captured to trigger callbacks, similar to the signals and slots system. Even though Qt provides a rendering engine using OpenGL, we developed a custom library, the BlueFramework.

BlueFramework is a 3D graphics rendering engine API written in C++. The main concept is the implementation of the high level `IEffect` interface. Custom rendering effects are created by sub-classing `IEffect` and implementing the interface functions for initialisation and drawing. All required operations such as data transfer are implemented in the base class. It provides intuitive functions to supply the index and vertex buffers as well as other resources like textures. Rendering simple shapes like lines, points or triangles without complex shading can be done easily using the `VertexCache` effect classes.

The `IEffect` interface API is implemented on top of multiple rendering backends which

[3] `https://www.opencascade.com/`

can be switched during runtime. No changes to the code implementing the visualisation are necessary to switch from DirectX to OpenGL as long as the shaders used by the effect are supplied for both backends.

3.4 Additional Modules

Additional functionality for processing data from other domains is provided through a plugin module system. Each module encapsulates it's own dependencies and interfaces to the OIP core API and user interface. The plugins which should be included can be selected during project generation in the CMake GUI. Import and Export modules for standards other than IFC are implemented manually and therefore only cover specific parts of the whole standard. Creating a tool for automatic generation of conversion functions for different standards

Point Clouds are large collections of 3D point data and are used for construction or infrastructure monitoring and maintenance. A single cloud can contain multiple millions of points including additional information like colour or surface normals. OIP uses the CloudCompare open-source project to represent and handle point cloud data while providing additional functionality built on top of the base API (Girardeau-Montaut, 2012). It supports reading and writing the native CloudCompare binary data format as well as the open LAS format. Visualisation of point clouds and other related objects like octrees is also implemented using the BlueFramework.

LandXML is a standard used to describe infrastructure models. They can be imported and are converted into a corresponding representation in the internal IFC based data model. LandXML uses similar structures to describe horizontal and vertical alignments as IFC enabling loss less conversion between both formats (Amann and Borrmann, 2015). The OKSTRA standard follows a similar pattern as LandXML. Due to the large scope of OKSTRA, only a small subset related to alignments is converted into IFC and visualised (Amann et al., 2014). Both IFC and OKSTRA have linked data specifications of the standards using the web ontology language (OWL) (Jakob Beetz, 2018; Pauwels et al., 2017; Beetz and Borrmann, 2018). OIP supports writing IfcOWL for Ifc4x1 and parts of OkstraOWL in turtle and rdf format.

4 Conclusions

This paper presents the OIP software developed at the TUM. We provide an overview over currently available free software tools which support IFC and compare them to the OIP. We explained the rationale behind our early binding implementation and the mapping between EXPRESS and C++. A lot can still be done in the area of continuous integration. This includes a better modular architecture and dependency management. The software toolkit has large potential for being developed as a platform independent application due to utilising Qt and BlueFramework for visualisation and boost to extend the C++ STL. Current developments are dedicated towards implementing an automated generation of a super-set of multiple IFC specifications to create a unified interface for conversion of geometric representations.

References

Amann, J. and Borrmann, A. (2015), Creating a 3d-bim-compliant road design based on ifc alignment originating from an okstra-accordant 2d road design using the tum open infra platform and the okstra class library, Technical report, Technische Universität München.

Amann, J., Flurl, M., Jubierre, J. R. and Borrmann, A. (2014), An alignment meta-model for the comparison of alignment product models, *in* 'Proc. of the 10th European Conference on Product & Process Modelling', Vienna, Austria.

Amann, J., Schöttl, F., Singer, D., Kern, M., Widner, A., Geisler, P., Below, D., Hecht, H., Coetzee, C., Duckerschein, L., Schill, R., Werner, M., Böckle, M., Gupta, N., Mustafa, A., Markič, Š. and Borrmann, A. (2019), 'Tum open infra platform 2019'.
URL: *https://www.cms.bgu.tum.de/oip*

Beetz, J. and Borrmann, A. (2018), Benefits and limitations of linked data approaches for road modeling and data exchange, *in* I. F. C. Smith and B. Domer, eds, 'Advanced Computing Strategies for Engineering', Springer International Publishing, Cham, pp. 245–261.

Borrmann, A., König, M., Koch, C. and Beetz, J. (2017), *Building Information Modeling – Technology Foundations and Industry Practise*, Springer International.
URL: *https://link.springer.com/book/10.1007/978-3-319-92862-3*

David Abrahams, A. G. (2004), *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*, Addison-Wesley Professional.

Girardeau-Montaut, D. (2012), *CloudCompare: User's Manual for version 2.1*.
URL: *www.cloudcompare.net*

ISO (2004), Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual, Standard, International Organization for Standardization, Geneva, CH.

ISO (2013), Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries, Standard, International Organization for Standardization, Geneva, CH.

Jakob Beetz, André Borrmann, J. A. (2018), Linked data: Analyse von einsatzmöglichkeiten von verbundenen informationen (linked data) und ontologien und damit befassten technologien (semantic web) im bereich des straßenwesens, Technical report, Technische Universität München, Rheinisch-Westfälische Technische Hochschule Aachen.

Lockley, S., Benghi, C. and Černý, M. (2017), 'Xbim.Essentials: A library for interoperable building information applications', **2**(20), 473.
URL: *http://joss.theoj.org/papers/b23bed93a0377b4f4317d0583b4d2c5e*

Pauwels, P., Krijnen, T., Terkaj, W. and Beetz, J. (2017), 'Enhancing the ifcowl ontology with an alternative representation for geometric data', *Automation in Construction* **80**, 77 – 94.
URL: *http://www.sciencedirect.com/science/article/pii/S0926580517301826*

Sebesta, R. W. (1996), *Concepts of Programming Languages (3rd Ed.)*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.