Future of the Industry Foundation Classes: towards IFC 5

Berlo, Léon van; Krijnen, Thomas; Tauscher, Helga; Liebich, Thomas; Kranenburg, Arie van; Paasiala, Pasi

Publication date
2021

Document Version
Final published version

Published in
Proceedings of the 38th International Conference of CIB W78

Citation (APA)
Berlo, L. V., Krijnen, T., Tauscher, H., Liebich, T., Kranenburg, A. V., & Paasiala, P. (2021). Future of the Industry Foundation Classes: towards IFC 5. In *Proceedings of the 38th International Conference of CIB W78* (pp. 123-137). (CIB W78 conference series). https://itc.scix.net/paper/w78-2021-paper-013

Future of the Industry Foundation Classes: towards IFC 5

Léon van Berlo, leon.vanberlo@buildingsmart.org
buildingSMART International, London, United Kingdom

Thomas Krijnen, t.f.krijnen@tudelft.nl
Delft University of Technology, Delft, The Netherlands

Helga Tauscher, helga.tauscher@htw-dresden.de
HTW Dresden - University of Applied Sciences, Dresden, Germany

Thomas Liebich, tl@aec3.de
AEC3, Munich, Germany

Arie van Kranenburg, management@arkey.nl
Arkey Systems, Houten, The Netherlands

Pasi Paasiala, Pasi.Paasiala@solibri.com
Solibri, Helsinki, Finland

Abstract
The buildingSMART Technical Roadmap, published in April 2020, was the start of multiple modernization efforts for buildingSMART Solutions and Standards. The modernization, modularization, and normalization of the Industry Foundation Classes (IFC) is one of the priorities. A taskforce has been working on restructuring the core of IFC for the IFC 5 developments. The following topics are discussed and researched (a) modularization of IFC (b) normalization of the IFC object trees and relations (c) language independency of the base data structure (d) modernization of the deployment tools and procedures for maintaining IFC. This paper reports progress on all these topics.

The normalization of the object tree is an integrated effort that involves changes in the use of objectified relations, property sets and predefined types. The modularization provides interoperability between domains and a solution to easily support incremental updates in software implementations. This so called 'late binding' approach and the consequences on the IFC schema have been researched and reported in this paper.

The paper is a progress report with an overview of considerations and work done so far. It will end with a discussion chapter about consequences of the modernization work and a call for participation in further developments.

Keywords: IFC, openBIM, BIM, Industry Foundation Classes

1 Introduction

In April 2020 buildingSMART International published the 'Technical Roadmap' (Berlo, 2020) subtitled 'Getting ready for the future'. The future of the Industry Foundation Classes (IFC) was a major topic of the roadmap. After publication of the roadmap, a small taskforce team has started laying the groundwork of the new IFC version 5 standard. This taskforce has extensive experience with IFC and consists of a mix of implementors, data modelers and academic influences. The

authors of this paper are all part of the IFC 5 taskforce. The taskforce met every two weeks to integrate and discuss ideas and solutions for IFC 5. In between they worked on their own on certain topics.

This paper is authored a year after publication of the roadmap and conception of the IFC 5 taskforce. It is intended as a status report of work in progress. It should be seen as a discussion paper with the intent to broaden the participation in the development of IFC 5.

1.1 Background and Motivation

The 'buildingSMART Technical roadmap' published in April 2020 covered a wide range of technical topics, including standardization of APIs and additional services like the buildingSMART Data Dictionary (bSDD). It is an integrated approach to modernization and integration of different standards and solutions from buildingSMART. The Industry Foundation Classes (IFCs) are the flagship standard of buildingSMART. It is the most mature standard that has a long history and many implementations in software. Currently, the conceptual model of IFC is closely associated with the EXPRESS modelling language. This makes it hard to represent IFC in different formats than STEP. Using STEP was probably the best choice to model the IFC schema when it was targeting file-based exchange.

With new concepts like smart buildings, smart cities, and digital twins just around the corner, there is an increased expectation for future-proof standards and solutions. The increased demand for partial updates of BIM data, filtering high data volumes, low latency in exchange, applications of Artificial Intelligence and Machine Learning cannot be met with file-based information silos.

The IFC 5 taskforce was facing a massive challenge when starting to work since the topics of language independency, modularization, normalization, and new use-cases are interconnected. Topics of discussion are interwoven and there is a high threshold to the integrated view. After a year many of the core topics have been discussed and this paper provides an overview of the current status. Next steps will be to further develop IFC 5 in different sub teams focused on the product tree, relations, geometry, and other resources.

The IFC 5 initiative focusses on modernization and normalization to support the new use-cases. It does not intent to add new domains or other content. That means the upcoming IFC 4.3 version and IFC 5 will have the same semantic scope. This way the IFC 5 release can focus on technological agreements and is not limited by long consensus-forming processes on semantic definitions.

1.2 History and formative principles of IFC

The origins of IFC date back to 1995 when the development started. Many of the now common frameworks for describing data models and data exchange serializations, such as UML, XML or JSON did not exist at that time, or they were still in early development stages. The most mature framework to define a data schema and its serialization at that time was STEP (the series of ISO 10303 standards). Therefore, IFC had been developed based on EXPRESS for data schema and STEP physical file, SPF, for file-based exchanges.

Now, 25 years later, IFC has become the most recognized open exchange format for Building Information Modeling (a term, that was only coined after IFC development had begun). Hundreds of software applications are IFC-compatible by allowing export and/or import of IFC files, and millions of IFC files have been generated, still based on the original technology stack of EXPRESS and SPF.

On the other hand, new challenges lie ahead, and new technology frameworks are available. UML has become the most commonly used language for defining object diagrams, XSD/XML has become the most used technology to define and exchange semantic data followed by Json Schema and Json, and semantic web technologies such as OWL are around. Most developers are fluent in these technologies but have little to no experience with EXPRESS, the same applies to the available tooling.

Therefore, the time is ripe to reconsider the underlying framework on which IFC is based. A similar challenge is coming from the ever-growing IFC data schema itself. There is the constantly increasing subtyping tree of elements, often only representing a particular domain-specific

classification. This becomes particularly evident when the infrastructure extensions require addition of new element classifications for the new domains. The latest head count had about 880 entities (or classes) for the upcoming IFC 4.3 release. Within the geometry resources, the various ways to express shapes add to the complexity.

Over the last 20 years many extension projects that were carried out by buildingSMART (or IAI, as it was previously known) led to additions to the schema, not all of which had ever been supported by a broader range of applications. In practice there is a subset of the schema that is well-supported among applications, namely the IFC 2x3 Coordination View and the current IFC 4 Reference View. At the same time, other parts of the schema are not yet validated by intense use.

A very early powerful decision in IFC development was to exclude most domain properties (with some partially questionable exceptions) from the IFC EXPRESS schema. These are instead treated as reference data - the so called IFC property sets. Even though this may not be precise language, they are referred to as a "late bound" extension to IFC. They are not part of the schema, but still part of the overall IFC specification. To illustrate this, consider *IfcProduct*, the abstract supertype in the IFC schema that introduces the concept of representation and placement, in other words, the supertype of most elements that are visible to a user. In the IFC 4.3 rc3 schema there are 221 (transitive) subtypes of this class. The median number of attributes introduced at these specializations is 1 (mean: 0.987, max: 9 for *IfcReinforcingMesh*). In most cases[1] this is the attribute named PredefinedType, an attribute that points to a specific enumeration for an entity that (by means of an attribute) establishes a more granular subtyping hierarchy. When ignoring the PredefinedType attribute, the median number of attributes is 0, the mean 0.34. To put this into perspective, there are 2661 property and 266 quantity definitions distributed over 462 property and 96 quantity sets (although note that these are not necessarily unique as currently there is no mechanism for inheritance among property sets).

1.3 New use-cases

The current IFC is optimized for file-based exchange. To facilitate current use-cases like working with connected CDEs, and new business concepts like Digital Twins, connections to (streams of) sensor information, automated (micro)services, and future Smart cities require an object-based use of IFC data. IFC needs to become capable of transactional exchanges, allowing smaller discrete exchanges.

Transforming IFC to be capable of being used in a transactional environment is a huge task, and a drastic shift from the file-based optimization modelling techniques that are used up until now. Transactional capable IFCs can still be exchanged as files but also accessed, maintained, and exchanged using an 'Application Programming Interface' (API). Partial (transactional) file exchange, or exchange of 'changes' (sometimes called 'deltas') is an industry need already. CDEs and Digital Twins are strongly based on the interoperability of different systems via APIs. Thus, API standardization is an activity buildingSMART should focus on in the coming years. Both use IFCs through an API and partial file exchange are difficult with the current structure of the files.

Changing the optimization objective for IFC from file-based exchange to use in a transactional environment is a big cultural change. It means the tech community of buildingSMART needs to use other key performance indicators for the development of IFC than they have been used to for the last 15 years.

1.4 Scope

The normalization of the IFC Geometry core and the IFC Resource layer have not been discussed in the IFC 5 taskforce yet and therefore this paper cannot report any progress. The integrated modernization and normalization of the *IfcProduct* tree got priority. Changes to the *IfcProduct* tree are likely to influence the content in the resource layer. Standardized conformance levels will certainly influence the publication of concept templates and the purpose of mvdXML. And the changes in the specialization tree will most likely change the way material layers are defined and modelled.

[1] 163 subtypes have at least one attribute, 37 subtypes have at least one attribute when disregarding PredefinedType.

In parallel to the IFC 5 taskforce, work on IFC 4.3 alignment has also inspired the direction for IFC 5. The split between geometry and semantics for *IfcAlignment* makes the schema more predictable. This concept is likely to be used for other placement entities like *IfcGrid* as well.

1.5 Context and related initiatives

Amor (2015) presents an analysis of how the IFC schemas has evolved over time. He found significant increases in number of entities and attributes for the transition from IFC 2x FINAL to IFC 2x2 FINAL and from IFC 2x3 to IFC 4. He notes that the number of optional attributes has increased in IFC 4 (for example owner history and the position placement on sweeps) and that little of the schema semantics are expressed in formal rules.

Especially in the Semantic Web domain a lot of energy has been invested in transforming, modularizing, and simplifying the IFC schema (or ontology; by the typical idiom in that field). Where this started as direct transformation of the IFC EXPRESS schema and modifications to result in a more idiomatic ontology (Beetz et al. 2009) and analyses to introduce modularity (Terkaj and Pauwels 2017). In later years effort has shifted more towards novel independent ontologies such as presented in Pauwels et al. 2017 and Rasmussen et al. (2019). Not all this work can be easily embedded back into the core IFC schema development. Some of the changes recommended in this domain are specific to work around limitations in their encodings such as inefficiencies in ordered sequences or require the extensibility and flexibility of subject-predicate-object information representation mechanism. The aims are also slightly different, with a central notion of empowering engineers to make ad-hoc links (Linked Data) as opposed to the standardization of workflows and usage patterns for implementation by a wide set of software vendors.

While IFC is developed outside of the ISO 10303 community, it shares the main technologic foundations with the STEP standards: the IFC schema is predominantly defined in EXPRESS (part 11), the most prevalent encoding of instance models is the Step Physical File (part 21) and the majority of geometry definitions in the IFC schema are derived from part 42. The committee behind STEP, ISO/TC 184/SC 4, is in the process of adopting SysML for parts of their schema management, although it is not entirely clear to the authors to what extent, as there is little information published publicly. From conversations with people in this community the authors have heard that one of the main drivers behind this transition is to be able to apply model-based engineering to the schema development itself. In this approach, every schema component (entity, attribute, rule, …) is an object in a model that has a unique identifier and can be annotated and formally checked. This is different from the primarily text-based EXPRESS modelling language. Since ISO 10303 is a network of standards with complex relationships between the parts and application protocols a model-based approach is deemed necessary.

Model-driven architectures (MDE) and engineering (MDE) have advanced to mature technologies with wide-spread application during the last decades. With the Meta-Object Facility (MOF) the OMG has lifted the Unified Modelling Language (UML) from a visual language for diagrammatic representation of object-oriented software systems to a full-grown modelling language. This forms the base for early and recent works to apply MDA/MDE to STEP or IFC and leverage its potential for schema and data integration (for example Combemale 2017, Jetlund 2020, Tauscher 2020).

2 Modularization

IFC originally focused on the standardization of data definitions in the building industry. In recent years it expanded into other domains and covers the whole built environment. IFC 'Model View Definitions' (MVDs) are a layer on top of IFC to define additional restrictions. Interoperability is only guaranteed within a single MVD, not between different MVDs.

To solve these issues, the IFC schema needs to become modular. Modularization of the schema makes it easier to separate responsibilities, distribute the maintenance of the entities, and possibly even have separate release cycles per module. The three functionalities that are currently provided by the MVDs need to be further developed in separate, coordinated initiatives. Modularization will facilitate this by creating a shared (interoperability) layer in the schema as a

base for the modules. The specialization structure of IFC allows for dynamic ('late binding') modules that extend the base layer.

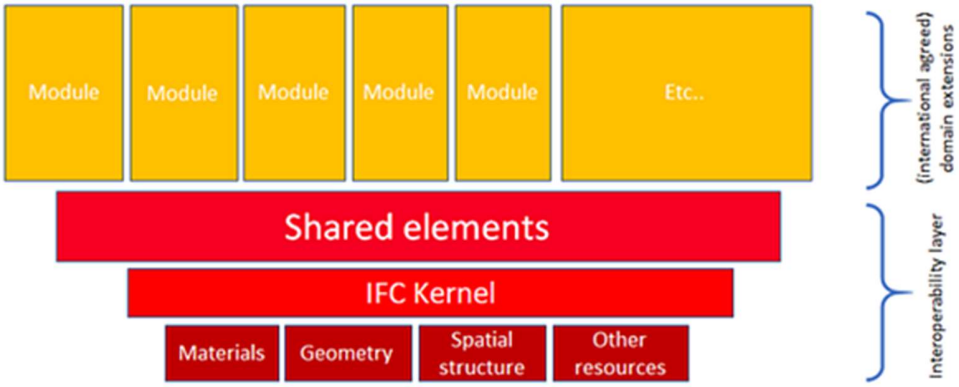

Figure 1: Modules (yellow) on a shared base (red layers) for interoperability between domains.

This will make the implementation in software more predictable. A conceptual representation of modularization is provided in Figure 1. It shows the Interoperability layer, comprised of the three red layers in the picture. The dark red layer represents the IFC resource layer.

When the shared base is implemented, the modules are extensions to define additional classification and properties on top of the shared layer. The split could be done on all specializations from *IfcBuiltElement*, or possibly on *IfcProduct* when the implications are not too drastic. Other branches like the tree below *IfcControl* and *IfcProcess* need to be reconsidered as well. When the red interoperability layers are implemented in full, no matter what module the software supports, interoperability between domains will be guaranteed. Strict implementation of this shared base is crucial. Therefore, this part of the schema needs to be revised to create a base that is straight-forward, with minimal modelling complexity to make it stable and predictable to implement. When the shared base is implemented in software according to a strictly defined conformance level, every export from any software will provide a dataset that can be imported in any other software tool with the same conformance level. In other words, a 'one star' export from 'module A' can always be imported in any other software tool that implemented 'one star' IFC, no matter what module is supported or certified. This creates the predictability for IFC to be the base for Digital Twins, and the ability to support use cases where data is exchanged through APIs instead of files.

Depending on how software vendors implement the modules (extensions with classifications and properties), new modules or updates could potentially be supported instantly. Even when this is not immediate, the time between the publication of a new version of an IFC module (i.e., extension), and the availability in software would decrease drastically.

2.1 Current situation: Static schema with PSets

In IFC 4.x the definitions are published in the form of a static schema defined in EXPRESS and additional definitions stored in property set definitions (PSD). From the viewpoint of management of the standard, the distinction is that, on the one hand, there are resources in the schema that have an unambiguous clear meaning: e.g., the dimensions of a rectangular profile govern the exact geometric form, but the semantic distinction between major element categories such as walls and columns are less precise, their properties often do not affect processing in software but are mostly for human inspection or filtering.

Currently, the separation between these two major forces in the IFC schema development, well-established foundational resources and domain specific taxonomies and properties, is not clearly articulated. As a result, the support of new domain-specific use cases is hindered by slower moving forces of the schema development, software implementation and release cycles. On top of that, the semantic link between property sets and elements is weak, it consists of ad-hoc XML files of which the exact semantics are underspecified. These things considered, one of the main objectives outlined in this document is to unify these principles of fast and slow evolution into a

well-understood meta-modelling approach. This will enable model-based engineering and development in a cohesive modeling environment so that the implication of changes on all layers is clear. This paper will not introduce a new approach into the IFC specification, but rather rectify the distinction between schema and standardized meta-data to also include the taxonomy in the latter.

2.2 Late-binding implementations and the "late-bound portion" of IFC

The following paragraphs provide a brief informal introduction to the idea of late-binding implementations. We then highlight the use cases that benefit from a full or partial late-binding implementation and show how the approach has been adopted in IFC as so-called "late-bound" portion of the schema. In Section 2.3, we describe and discuss the planned changes and improvements for IFC 5.

Binding, in particular name-binding, denotes the resolution of names that point to data or program execution instructions in computer memory. In a program, for example, a variable name needs to be resolved to the memory address of the assigned value when reading the respective variable or a method name needs to be resolved to the memory address of instructions derived from the method's code when calling the respective method.

With regard to standardized conceptual models such as IFC, the notion of name-binding has been extended from how names in a computer program are resolved to how names in the conceptual model are resolved. Early-binding implementations consider a particular conceptual model or schema at implementation/compile time by creating particular data structures (e.g., classes in object-oriented programming) following the standardized model or schema. They are called early-binding, because the binding happens at compile time and cannot be changed at runtime. Late-binding implementations, on the other hand, consider a particular conceptual model or schema at runtime only and keep implementation and compilation based on names as unbound string values.

While a late-binding implementation treats instances generically only based on the serialization data structure and thus can handle unknown model elements, an early-binding implementation is better able to actually interpret model elements and implement operations according to model semantics. However, not all use cases need full domain model semantic support. The following use cases can be distinguished in the AEC context with regard to the necessity of interpreting of domain model semantics:

- *Plain viewers*: There are no native concepts that correspond to domain concepts and no interpretation is necessary.
- *Cross-discipline import for reference in an authoring tool*: This is similar to the case of plain viewers.
- *Import for processing in same-discipline authoring tool*: Only domain concepts corresponding to native concepts of the importing software need interpretation, others may only need to be retained for export or for generic display.
- *Analysis and model checking*: This use case is similar to authoring tools, but no export involved.

None of the use cases requires interpretation and thus direct early-binding implementation of the entire schema. All cases, however, require a way to ignore unknown semantics or to handle it in a generic way – for all or large parts of the domain-specific part of the schema. Yet, the core and in some cases a particular use-case relevant portion of the domain module would benefit from more rigid treatment as in early-binding implementations. The question arises how to achieve forward-compatibility, partial and reference-only coverage in otherwise early-binding implementations and thus how to bring some of the flexibilities of late-binding implementations into early-binding implementations. In other words: How to combine the best of both worlds?

The IFC-answer to this question was to embed an approach into the schema that is inspired by late-binding implementations. Instead of instantiating domain-level entities and attributes as per the mechanisms of the implementation method (for example SPF, XML, or source code),

domain-level classes and attributes are referenced by name with dedicated String attributes. Only generic entities and attributes from the core are instantiated properly, while entities and attributes from the core are only marked as instances of a particular schema concept via reference which can be resolved dynamically in applications. The essential IFC concepts to achieve these name-based references are the attributes *IfcObject.PredefinedType* to represent class names[2] and *IfcProperty.Name* to represent attribute names.[3] This way most attributes and many classes have already been factored out of the core schema. With IFC 5 we attempt to strengthen and extend this approach and apply it consistently.

2.3 A meta-model for the late-bound part

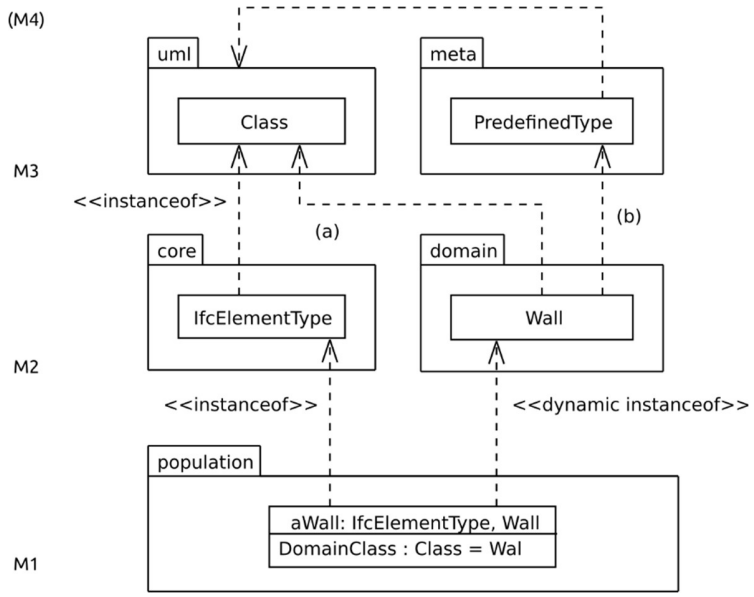

Figure 2. MOF metamodel levels and dynamic instantiation ("late binding"), core, domain, and meta packages. Each package contains one model element for illustration purposes. Different options to represent the "late bound" domain package is labelled (a) and (b), see also Figure 3.

First, in IFC 5, both the core and domain parts of the schema shall be specified using the UML as can be seen in Figure 2. The Figure shows a metamodel architecture with three metamodel layers according to the Meta Object Facility (MOF)4 defined by the OMG. Level M1 comprises an IFC population with a particular instance. Level M2 contains the IFC schema, divided into a core and domain package. Entities in M1 are instances of the concepts in M2, with a dynamic (late-bound) instancing mechanism for the domain module. Level M3 contains the metamodels used to specify schemas in M2, the UML is used for both the core and domain package (a). In parallel, the domain package is described using the property set meta concepts (b) introduced in IFC 4 as *IfcPropertyTemplateDefinition* and subclasses. We are extending and harmonizing this part of the schema into a meta package that corresponds well with a subset of the UML. Figure 3 shows excerpts of the UML and PSet meta models and how these relate to each other. Elements with the same horizontal alignment (in one row) correspond and can be easily mapped.

[2] This claim may seem counter intuitive because PredefinedType is not a "free" string attribute, but an enumeration defined in the schema. However, keep in mind that an enumeration is basically just a constraint on the possible values of a string attribute and all PredefinedType enumerations contain a value "USERDEFINED" that allows to circumvent this constraint.

[3] To be complete, we also must mention *IfcPropertySet.Name* to represent names for structured datatypes consisting of multiple attributes.

[4] Object Management Group (2019). Meta Object Facility (MOF) Core Specification 2.5.1. https://www.omg.org/spec/MOF/

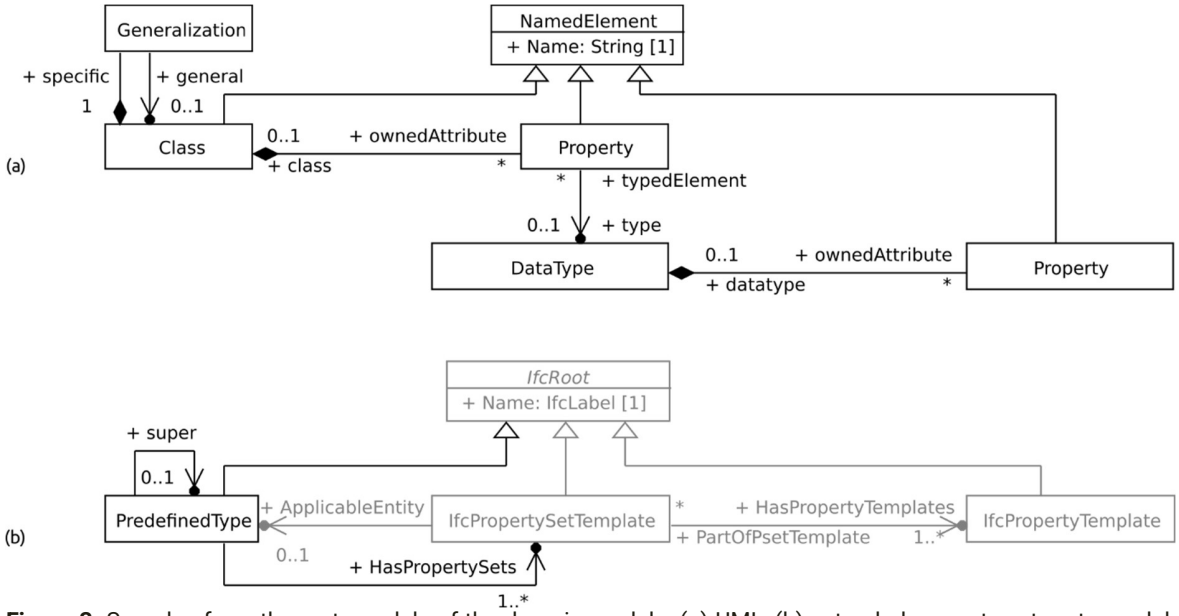

Figure 3. Samples from the metamodels of the domain module: (a) UML, (b) extended property set meta model. Existing classes and associations are shown in grey, new classes/relations in black. Names provisional.

Further changes in IFC 5 include the relocation of more elements from the core into the late-bound portion of the schema, in particular predefined types and potentially some of the leaf and low level *IfcObject* subtypes. With larger part of the taxonomy moved to the late-bound part, the domain metamodel will have to include concepts to express subclassing. Remaining direct attributes in the core, such as *IfcDoor.OverallHeight* are moved to late-bound part. There is still an ongoing discussion as to whether relationships must be included in the late-bound part.

The late-bound portion of the IFC schema will ease implementation of software implementation of reading IFC, since it can be applied runtime, as opposed to the early binding part that is typically applied compile time. An instance in an IFC file will reference a late-bound concept by name and importing software can use that reference as a key to read more information about the component, such as its display name in a local language. This way, the software does not need to be aware of the late bound concepts and still its users will be presented with meaningful information. For concepts that a software has native support for, the reference to the late-bound concept can be used as a key to retrieve information from the schema such as allowed attribute names and types and process those instances in more detail. With the more formal definition of the late-bound portion (continued from the IFC 4 attempts), this will foster schema validation of instances.

It must be noted that the so-called late-bound approach comes at a price: 1. Population sizes increase for generic instances, for example because schema implementation methods with positional assignment of attributes turn into named assignment and implementation methods with named assignment introduce redundant name attributes. 2. Mixing different meta-levels and modelling languages increases complexity and may make it more difficult to understand the specification. There are, for example, two different ways of how instances relate to the defining classes for the core and domain extension modules. 3. There is less flexibility in choosing early- or late-binding approaches on the implementation side, as the scopes are prescribed by schema. Even though this is done in the most reasonable way with existing and future software implementations in mind, implementation of only parts of the late-bound schema in early-binding fashion may be attempted and will be more difficult. For interpretation of the semantics of concepts in the late-bound part, instances must be bound twice, at compile time for the core schema, and at runtime for the domain schema. 4. Domain package may get large and need further internal differentiation, e.g., to distinguish essential (mandatory) from standardized and user-defined domain-specific attributes or properties.

3 IFC Product tree normalization

3.1 New hierarchy

One of the core parts of the IFC schema is the taxonomy of "rooted" elements, a single-inheritance hierarchy of all entities outside of the resource layers, shown in Figure 4. In this taxonomy, each level follows a unique discriminator to iteratively refine the classification of entities. Level 1 contains the root of this hierarchy, conveniently named *IfcRoot*. It has a mechanism for a stable instance identity (an attribute called *GlobalId*, optional at this level), a mechanism for tracking changes and status (made optional in IFC 4, named OwnerHistory) and a textual name and description. Level 2 provides the main differentiation between elements, their attributes, and relations with three subtypes of *IfcRoot*: *IfcObjectDefinition* for things and processes, *IfcPropertyDefinition* (renamed) for characteristics, and *IfcRelationshipDefinition* for relationships. The attribute *GlobalId* becomes mandatory for object definitions. These two levels are in the main unchanged from the current schema. Level 3 introduces occurrence and type for object and property definitions. Changes for this level are discussed in Section 11. Level 4 adds classification regarding shape and location.

To prevent unnecessary incompatibilities between IFC versions, the taskforce has been careful applying changes to this hierarchy. One of the main changes, that is necessitated by the shift to a partial late-bound representation of this hierarchy is a clear demarcation for the subtypes of *IfcProduct* in Level 5 to distinguish between all things physical (*IfcElement*), all elements that pertain to the spatial subdivision structure (*IfcSpatial*[*Element*]) and all constructs that affect appearance of other elements (*IfcFeature*[*Element*]). The practical advantage of this is that implementations of older versions of the early-bound schema can function well on newer versions of the late-bound schema. For example, most viewers hide *IfcSpace* elements when the model is initially loaded, and, depending on model view, need to subtract opening elements from their hosts. In this late-bound subtypes for spaces and openings can be introduced inheriting their implied semantics from early-bound types.

The placement and specification of *IfcOwnerHistory* and *IfcContext* are still subject of discussion.

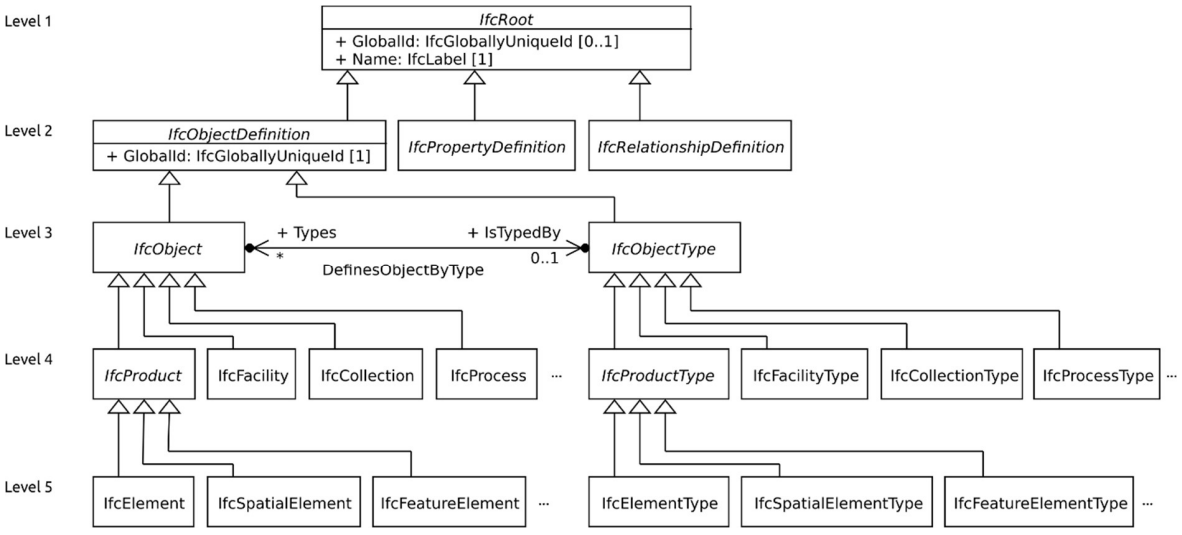

Figure 4. Overview of the new taxonomy. Note that the subtypes in this overview are not exhaustive.

3.2 Occurrence and type

In IFC nearly all elements can be of a type. Elements can reuse information stored at the element type. Every property that is not specific for a particular instance can be stored in an element type, including the geometry. Even spatial elements can have a type, but in the current IFC version, this is only true for *IfcSpace*. The *IfcSite*, *IfcBuilding* and *IfcBuildingStorey* do not have a corresponding

type, which can be seen as an omission. The relation between an element and its type is currently defined with the objectified relation *IfcRelDefinesByType*, which is a 1: N relation between a single element type and multiple elements.

Elements can be of a type. There can be elements without a corresponding type. As a result, in current schemas there are not only the entities *IfcElement* and *IfcElementType*, but also an *IfcDoor* and an *IfcDoorType*. The full specialization of *IfcElement* is mirrored at the side of *IfcElementType*. These two taxonomies are not always synchronized, neither at the entity level, nor on the attribute level. For example, there is an *IfcFeatureElement*, but no *IfcFeatureElementType*. IfcDoor has attributes *OverallHeight* and *OverallWidth*, but they are missing for *IfcDoorType*, where they would belong instead.

Both *IfcElement* and *IfcElementType* carry an attribute *PredefinedType*, which contains for every kind of element a value out of a domain of subtypes of that kind. The current IFC schema already contains a constraint for element is of a type to have their *PredefinedType* attribute set on the type side. For example, for an *IfcWindow* the values of *PredefinedType* can be *Window*, *Skylight* or *Lightdome*. The value *Window* can be seen as "usual" or "normal" window. Instead of the usage of the attribute PredefinedType there could be the possibility to define real subtypes as *IfcSkylight* and *IfcSkylightType*. But that will lead to an explosion of entities. And changing the domain of an enumeration is far easier than introducing new entities. We propose to make the element type mandatory, such that the specialization of *IfcElement* is no longer necessary. A door will become an *IfcElement* of an *IfcDoorType*. As a result, more than hundred entities can be removed from the schema. But also, the attributes *ObjectType* and *PredefinedType* can be removed from the remaining *IfcElement*. Until now it is possible, but invalid, that a window is of a door type. This check is now made in the receiving application but is not supported by the schema. Therefore, is it not certain if this situation is recognized by the software. It is possible that one application says interprets it is a window and another as a door.

Applications which will export elements without the corresponding element type have to introduce the usage of element types, even if there is no reuse of the type. And if there are subtypes of *IfcElement* in the schema without a corresponding type, that type must be added to the schema. Example of this is the already mentioned *IfcFeatureElementType*. When combined with moving all the subtypes out of the core, the advantage becomes even more evident. An *IfcElement* will be of the type *IfcElementType*. The *IfcElementType* will have a single attribute which have a value that defines that the element type is "door", or one of its subtypes. Even the user defined subtypes will move to the "late binding". Figure 5 shows an example with an entity instance (ET2) of class *IfcElementType* and dynamic domain class *SolidWall*. As a result, the current "double tree" at the "early binding" will change into a "single tree" in the "late binding". The attribute *PredefinedType* can disappear, because it will be replaced by real subtyping at the "late-bound" side.

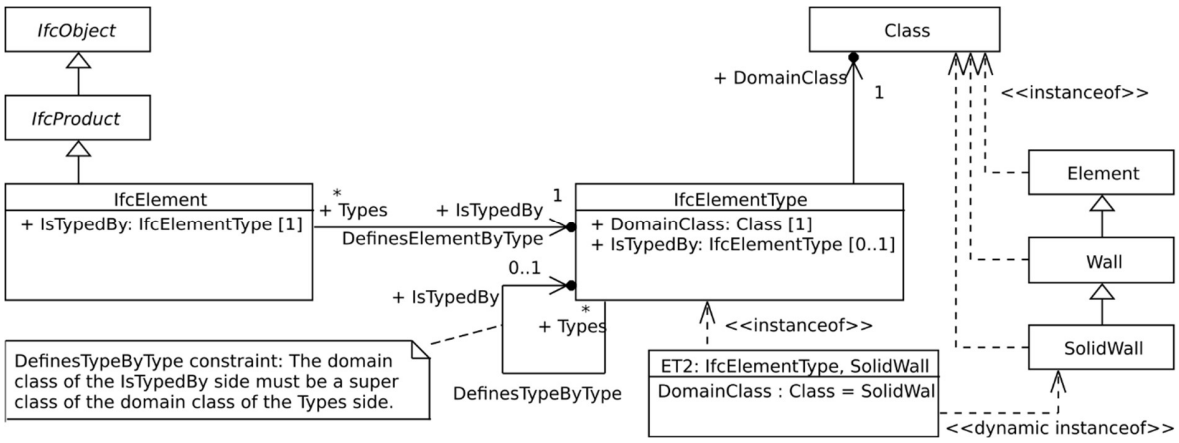

Figure 5. Provisional UML diagram showing excerpts of the core (left) and domain (right) taxonomies, classes, and associations for the occurrence-object type relationship (*DefinesElementByType*) as well as for instance-level "inheritance" (*DefinesTypeByType*). Besides elements from the core and domain, the diagram contains a metamodel element (Class) and a population element (*ET2*).

In the current version of IFC, element types themselves cannot have types. If there are two different product type instances that only have a different color (as a property) and because of that, a different article number, then there will be two different element types. There will be two different property sets as well, only differing in values for the *Color* and *ArticleNumber* property, and all other property values duplicated in both property sets.

In Figure 5, this kind of instance-level "inheritance" is shown as association *DefinesTypesByType*. With the inheritance of an element type, all shared properties can be collected at the higher-level element type. Then there will be two subtypes of that element type, an A version, and a B version. They have the specific properties for that specific type, all others will be connected to the supertype. An element instance should be of one of the specific types.

There is no consensus yet as to whether the object-level inheritance of values can freely follow class-level inheritance or there are stronger restrictions to either the most specific or resemblance of the full class-level taxonomy.

4 Relations

Relations between entities make up substantial part of the schema. There are various suggestions to simplify relationships, which can be discussed and analyzed along the following independent aspects: Semantics, Reification, Cardinality, Tree, Navigability.

The taskforce analyzed relationships in the IFC 4.3 RC2 and RC3 schemas thoroughly to answer the corresponding questions. For every objectified relationship type, the entity types and cardinalities of the relationship ends have been listed together with the number of additional explicit attributes (excluding binary relationship ends[5]), further subdivided into number of direct and inverse, mandatory and optional, simple (defined) and entity type attributes.

4.1 Semantics: Can we remove the semantics and merge this relationship with others?

Some relationship classes are identical in terms of additional attributes, do not carry any individual semantics and are only differentiated by the type of relation ends. As an example, this applies to three subclasses of *IfcRelAssigns*, namely *IfcRelAssignsToProduct*, *IfcRelAssignsToResource*, and *IfcRelAssignsToControl*. They do not have any additional attributes and do not carry any individual semantics beyond the type of one association end.

We are going through all existing relationships individually and question first the dynamic semantics (interpretation at runtime). If the relationship does not establish a necessary semantic distinction from other relationships, it can be removed. If it does, it could still be removed or merged as long as the semantics are uniquely defined by the types of the relation endpoints (like in the example above). This restriction is necessary to avoid potential shadowing issues.

There are cases where dynamic semantics seem similar at first glance, but on closer observation appear to have subtle differences in static semantics (characteristics defined in the schema, such as additional attributes). For example, the subclasses *IfcRelAssignsToActor*, *IfcRelAssignsToGroupByFactor*, and *IfcRelAssignsToProcess* of *IfcRelAssigns* each carry different additional attributes as opposed to the subclasses mentioned earlier. Those they cannot simply be merged. Likewise, relationships with different cardinalities cannot be merged. In these cases, we consider to first harmonize the static semantics where appropriate before attempting merge with other relationship classes. Some relationships do not need this conditioning, because they fall under transformations described in the next sections anyhow.

4.2 Reification: Can we turn this relationship from an objectified relation into a direct attribute?

In IFC, references between entities of the core and higher layers are modelled as objectified relationships - independent identifiable entities (subclasses of *IfcRelationship*) referring to the relationship ends instead of direct attributes of a relating entity referencing the related entities. Direct relationships are confined to the resource layer, which also contains a few objectified relationships (subtypes of *IfcResourceLevelRelationship*).

[5] These are the ones that are mostly called *Relating* and *Related*.

There is consensus that where possible, objectified relationships should be replaced with direct attributes. However, there are restrictions to this effort. First, arity: Only binary relationships can be turned into direct attributes. If the relationship is non-binary (meaning, there are additional attributes beyond the relating and related ends) it cannot be de-objectified. This applies to many of the *IfcRelConnect* subclasses, including the most prominent representative *IfcRelSpaceBoundary.* In this case, either the additional attributes can be removed via harmonization of static semantics, or we consider changing the class from a relationship class (subclass of *IfcRelationship*) into an object in its own right (subclass of *IfcObject*). Depending on whether the additional attributes are of simple or complex type, and on the decision regarding relationships in the late-bound IFC part, the attribute can then be moved to the domain-specific module. For simple attributes, it would be integrated into an attribute or property set.

It must be noted that conversion to an *IfcObject* subclass goes hand in hand with object-specific attributes and inclusion in the typing mechanism described in Sections 11 and 12. There is also an impact on modularity and extensibility, which should not be underestimated and needs thorough consideration.

4.3 Cardinality: Which cardinality class does this relationship belong to?

We have subjected cardinality ranges in IFC 4 to a thorough analysis. The results are shown in Table 1. Cardinalities of objectified relationships consist of 4 cardinality constraints, taken from the EXPRESS schema, the cardinality constraints of the *Relating* and *Related* attribute of the *IfcRelationship* entity class as well as their respective inverses (columns 1-4)[6]. From those we can derive overall cardinalities for the *Relating* and *Related* side and finally for the relationship as such (columns 5-7).

Table 1. Cardinalities of objectified relationships in IFC 4: EXPRESS constraints in columns 1-4, derivation of overall cardinality in column 5-7, number of relationships in IFC 4 in column 8

Relating	Relating inverse	Related inverse	Related	Relating overall	Related overall	Result	Number
1..1	0/1..*	0/1..*	1..*	1:N	N:M	N:M	23
1..1	0..*	0/1..*	1..1	1:N	N:1	N:M	12
1..1	0..*	0..1	1..*	1:N	1:N	1:N	8
1..1	0..*	0/1..1	1..1	1:N	1:1	1:N	4
1..1	0..1	0..*	1..*	1:1	N:M	N:M	3
1..1	0..1	0..*	1..1	1:1	N:1	N:1	1
1..1	0..1	0..1	1..*	1:1	1:N	1:N	2
1..1	0..1	0..1	1..1	1:1	1:1	1:1	1

The table shows that most of the relationships (38) have a cardinality of N:M, a good deal (14) has 1: N and only a single relationship is a 1:1 relationship. There is one anomaly with seemingly inverted *Related* and *Relating* side, but this entity is already deprecated. The lower bounds of inverse cardinality ranges indicate whether the relationship in a particular direction is mandatory or optional. Most are optional with a few exceptions.

Given the variety of how 1: N and N:M relationships are modelled in current EXPRESS, we are investigating to which extent these reflect semantic distinctions and need to be kept or can be simplified to less variants.

4.4 Tree: Can/should this relationship be part of an overarching tree structure?

Trees are an intuitive and popular way to structure information hierarchically. As such trees are used in many software applications that handle IFC data to provide an outline and navigation capabilities. Even though the IFC conceptual model is not structured in a hierarchical fashion, it contains hierarchical relationships, for example, spatial aggregation and containment which are

[6] The order of columns corresponds to the order the cardinality ranges would appear in a UML diagram. Readers familiar with EXPRESS-G are warned that this differs from the EXPRESS-G convention.

predominantly used for outline and navigation. The tree generated this way, is made up of different relationship classes such as *IfcRelAggregates* and *IfcRelContainedInSpatialStructure.* It has been discussed whether other domains like infrastructure may use other relationships to form this overarching tree structure. We are investigating whether this structure should be represented conceptually as a dedicated and uniform relationship and if so, which current relationship classes are candidates for such a structure. They would have to be of cardinality 1: N and have a constraint to not contain any (undirected) circuits to form a tree. With suitable class or interface structure and de-objectified attributes owned at the child side, this constraint could be modelled without additional constraints.

4.5 Navigability: Can/should this relation be navigated from one (which) or both ends?

A further question to be analyzed for every remaining relationship is: Does this relationship have a main navigation direction and if so, which? If a relationship is not navigable in neither direction, it should likely not be a relationship, but an object class. For analysis of navigability from either side, we must look at the inverses of the binary ends (*Related* and *Relating*) of objectified relations.

5 Maintenance and quality control

The maintenance of IFC has been a challenge in recent years. Custom made tools that have high costs and risks do not perform as expected, and traceability and transparency of changes is lacking. Some projects have been experimenting with modeling IFC as a UML class diagram. In parallel, the STEP community has also shifted to a UML/SysML based maintenance setup.

IFC 5 will also be modelled as a UML Class diagram and published on a GitHub repository. Relating documentation will be stored as Markdown pages in the same repository. Changes will be done using pull requests and a custom buildingSMART Workflow for quality control and validation.

For IFC 4.3 a similar ecosystem has been set up that gets input from UML as XMI, Markdown and mvdXML to generate an EXPRESS Schema, Property set XML files, UML Diagram PNG pictures and the HTML documentation package. This is done after every change (upload, accepted pull request) and triggers automatic publication of the IFC entities and properties in the buildingSMART Data Dictionary, update of the Translation Framework content and performs automated quality controls.

With the normalization of IFC 5 it will be even more easy to model IFC as a class diagram in UML. Experiments have been performed to transform the current IFC 4.3 Product Tree to the intended IFC 5 product tree using scripts. This has also proven to work and could potentially be used to document the transformation between IFC 4.3 and IFC 5.

It is the intent for IFC 5 to unlock the potential of community inputs for the improvements of the IFC schema and documentation. The different IFC modules will be published as separate documents on GitHub. Change suggestions (pull requests in GitHub) will trigger an automated review workflow using GitHub actions. The required domains experts will review the suggestions and the automatically generated quality checks. After acceptance it will be integrated in the latest version on GitHub. When declined, there needs to be proper motivation of why the suggestion is declined. This will also be part of the repository to build a traceable and transparent process for IFC developments.

For updates or changes to the documentation, the process could be similar, or different depending on the type of documentation. Additional clarifications could have a lighter review process, compared to changes or fixes to semantic definitions.

After every change, GitHub triggers the execution of custom-built Python scripts for quality control and consistency checks. After every accepted change, the deployment scripts generate new output like an updated EXPRESS schema, HTML package, diagrams, etc.

6 Related and Future work

The described changes for IFC 5 will drive the ability to create object-based incremental updates. To further facilitate this, some other elements are needed.

Development of a query language depends on the changes planned for IFC 5. There is a strong relation between an IFC Query language and object-based API development. For an object-based API there needs to be a stable foundation and established file-based API first. Therefore, the query language development is probably out of scope for the next 2 to 3 years but mentioned as the objective for the development of the overall API strategy, and the transformation of IFC.

Since IFC only holds agreements that have found global international consensus, it will never be a complete set of required agreements in everyday practice. Additional agreements, classification systems or user defined property sets need to work seamlessly together with IFC and IFC supporting software implementations. To facilitate this buildingSMART is providing the buildingSMART Data Dictionary (bSDD). The bSDD was rebuild in 2020 and scheduled to launch in the Summer of 2021. It supports the new ISO 12006-3 and ISO 23386. The bSDD hosts and links agreements that users need in their projects or regions. Every class and property in the bSDD have a unique URI that can be used by external tools. Data in the bSDD are published though a JSON API, a GraphQL API and as Linked Data (RDF, possibly TTL in the future).

The data requirements in day-to-day projects can be dynamic and very specific per use-case. To support the definition of such 'Information Delivery Specifications' (IDS), buildingSMART has developed an XML based IDS standard to define information requirements and how they should be exchanged with IFC. The IDS standard is an integral effort to combine IFC with regional and use-case specific agreements. It can link to URIs of classes and properties inside and outside the bSDD. The IDS structure and content is compatible with Product Data Templates (PDTs).

The ifcJSON project has been developing a JSON serialization for IFC 4.3 and have also experimented with a STEP independent JSON serialization of IFC. The lessons learned from this work have been used as input into the IFC 5 development discussions. In parallel an experiment has been conducted to publish the full IFC content as JSON-LD context and taxonomy. This helps digital twin developments to better use semantic agreements that are already available in IFC but are out of sight because they are not part of a common use-case. Explorations have also been done on representing IFC in an indexed binary format to reduce file size. The work of Krijnen & Beetz (2017 and 2020) on HDF5 exchange of IFC and point cloud data has been used as a base.

All these topics will continue to be discussed and explored in separate subgroups that work on parts of the IFC 5 development.

7 Conclusion and discussion

The results so far have proven that the ideas and concepts published in the 'Technical Roadmap' in April 2020 are feasible and executable. After a year of intense collaboration, it can also be concluded that the changes to IFC are serious but needed for future use and reliability of IFC. The changes to IFC will keep the expressiveness and the related topics from the roadmap strengthen the integral proposition of openBIM. There are still a couple of open issues that have not been addressed. Moving the border between the IFC schema and the dynamic part has not been tested in actual implementations. The proposed conformance levels in the Technical Roadmap might require domain specific adjustments, which would undermine the principle of interoperability between software implementations and domains. These and other topics will be further researched, and this paper is an open call for collaboration with the academic community.

The work from the IFC 5 Taskforce has proven that intense collaboration between Software Vendors and academia can deliver high quality and sustainable results.

Acknowledgements

BuildingSMART International would like to thank the authors and their organizations for their commitment to contribute to this important work.

BuildingSMART and the authors would like to thank all the experts that have donated their time and expertise to the progress of the IFC 5 work. We would also like to thank the openBIM community for inputs on presentations, the buildingSMART forum and on GitHub.

References

Amor, R. (2015). Analysis of the evolving IFC schema. In 32nd CIB W78 Information Technology for Construction Conference (CIB W78 2015), Eindhoven, Netherlands.

Beetz, J., Leeuwen, J. van, Vries, B. de. (2009): IfcOWL: A case of transforming EXPRESS schemas into ontologies. AI EDAM. 23. 89-101. 10.1017/S0890060409000122.

Berlo, L.A.H.M. van (2020). BuildingSMART Technical Roadmap. Published on buildingsmart.org. Version 30 April 2020.

Combemale, B., France R., Jézéquel J.-M., Rumpe B., Steel J., Vojtisek D. (2017): Engineering Modeling Languages. Turning Domain Knowledge into Tools. Chapman; Hall/CRC, New York.

Jetlund, K., Onstein, E., Huang, L. (2021): IFC Schemas in ISO/TC 211Compliant UML for Improved Interoperability Between BIM and GIS. ISPRS International Journal of Geo-Information, Vol. 9, No. 4, p. 278.

Krijnen, T., Beetz, J. (2017): An IFC schema extension and binary serialization format to efficiently integrate point cloud data into building models. Advanced Engineering Informatics. 33. 10.1016/j.aei.2017.03.008.

Krijnen, T., Beetz, J. (2020). An efficient binary storage format for IFC building models using HDF5 hierarchical data format. Automation in Construction. 113. 103134. 10.1016/j.autcon.2020.103134.

Pauwels, P., Krijnen, T., Terkaj, W., & Beetz, J. (2017). Enhancing the ifcOWL ontology with an alternative representation for geometric data. Automation in Construction, 80, 77-94.

Rasmussen, M. H., Lefrançois, M., Schneider, G. F., & Pauwels, P. (2019). BOT: the building topology ontology of the W3C linked building data group. Semantic Web, (Preprint), 1-19.

Tauscher, H. (2020): Towards a Generic Mapping for IFC-CityGML Data Integration. In: Proceedings of the 3rd BIM/GIS Integration Workshop and the 15th 3D GeoInfo Conference 2020, pp. 151–158, London, UK (online), 2020.

Terkaj, W., & Pauwels, P. (2017). A method to generate a modular ifcOWL ontology. In 8th International Workshop on Formal Ontologies meet Industry (Vol. 2050).