Chapter 6
Industry Foundation Classes – A standardized data model for the vendor-neutral exchange of digital building models

André Borrmann, Jakob Beetz, Christian Koch and Thomas Liebich

Abstract The Industry Foundation Classes (IFC) provide a comprehensive, standardized data format for the vendor-neutral exchange of digital building models. Accordingly, it is an essential basis for the establishment of Big Open BIM. This chapter describes in detail the structure of the data model and its use for the semantic and geometric description of a building and its building elements. The chapter concludes with a discussion of the advantages and disadvantages of the IFC data model.

6.1 Background

The idea of Building Information Modeling is based on the consistent use of a comprehensive building model as a basis for all data exchange operations (see Chap. 1). This avoids the need to manually re-enter data or information already created, and reduces the accompanying risk of errors. In addition to the numerous data exchange

André Borrmann
Technical University of Munich, Chair of Computational Modeling and Simulation, Arcisstraße 21, 80333 Munich, Germany
e-mail: andre.borrmann@tum.de

Jakob Beetz
RWTH Aachen University, CAAD - Design Computation, Schinkelstr. 1, 52062 Aachen, Germany
e-mail: beetz@caad.arch.rwth-aachen.de

Christian Koch
Bauhaus-Universität Weimar, Chair of Intelligent Technical Design, Marienstraße 13a, 99423 Weimar, Germany
e-mail: c.koch@uni-weimar.de

Thomas Liebich
AEC3 Deutschland GmbH, Joseph-Wild-Str. 13, 81829 Munich, Germany
e-mail: tl@aec3.de

scenarios between participants in the planning process, this principle also enables building data to be transferred digitally to the contractors in the building phase, and on completion for the "handover" of building data to the client or operator of the building.

A wide range of software tools already exist for the numerous different tasks involved in planning buildings, for example for the geometric design of the building, for undertaking a range of analyses and simulations (structural design, heating requirement, costing, etc.), for operating the building (facility management) as well as other applications such as those detailed in Part IV of this book. These tools address different tasks and application areas, and for the most part serve their purpose well.

A problem, however, is that many of these tools are still *islands of automation* (Fig. 6.1), i.e. have no or only limited support for data exchange between the separate applications. Consequently, data and information that already exists in digital form needs to be re-entered manually, which is laborious and prone to introducing new errors.

To remedy this situation, a data exchange format is required that makes it possible to transport building data between software products with as little data loss as possible. Such a format must set out uniform, unequivocal descriptions of geometric information that are clear in their meaning and therefore not open to misinterpretation (see Chap. 2). A further important aspect is the detailed description of semantic

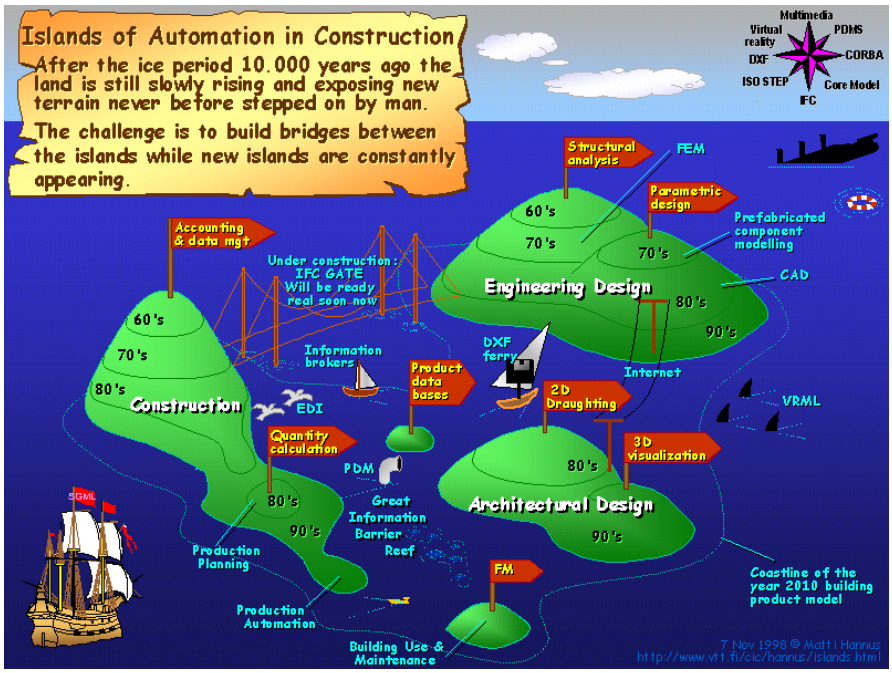


Fig. 6.1 Islands of automation in construction. The image was created in 1998 (©Matti Hannus, reprinted with permission)

information, including the classification of building components within a common hierarchy of types, the description of the relation-ships between them and the definition of their relevant properties (material, building times, etc. see Chap. 3).

This is where the term interoperability comes into play, which means the loss-free exchange of data between software products by different vendors. What differentiates the building sector from many other industries is the wide range of different products in use and number of software vendors in the market. In other industries (for example automobile and aircraft manufacturing), the main manufacturers stipulate which software products their suppliers must use. At the same time, large, global software manufacturers provide complete solutions for these industries that cover many parts of the design and engineering processes. It is more straightforward for these software manufacturers to ensure interoperability between their own software products, because they can design their own proprietary formats and methods without needing to go through lengthy and complex standardization procedures.

Compared with such stationary industrial applications, the building sector has several different boundary conditions that make it more difficult to achieve the goal of loss-free data exchange:

- A building's design and its construction are typically undertaken by different companies
- Building planning typically has several phases that are often undertaken by different planning offices
- Numerous different specialist planners are involved, each of which are separate companies.
- The building industry is very fragmented with numerous small and medium-sized companies. Statistics for Europe show that 93% of construction companies have fewer than 10 employees.
- Collaborations between different companies are typically ad-hoc partnerships for the duration of a project rather than long-term working relationships with well-defined processes and responsibilities.

In short, the building industry is characterized by a highly fragmented process with numerous different and independent participants. This means that lots of different tools are used and therefore uniform standards are difficult to enforce. At the same time, public authorities are required to be vendor-impartial, i.e. are not allowed to specify the use of certain software products when putting work out to tender. Likewise, public and private clients should not become too dependent on any one software producer to avoid vendor lock-in.

As a result, it has become common practice to specify widely-used proprietary formats for many typical data exchange scenarios in order to achieve a degree of predictable, i.e. pseudo-standardized interoperability. These formats are mostly used for 2D geometry formats augmented with a limited degree of semantic information, for example an agreed layer structure.

Proprietary formats are not conducive to realizing the BIG BIM goal of a consistent, high-quality digital building information model for the entire building process

(see Chap. 1). In most cases, proprietary data formats are tailored to specific application scenarios and not designed to cover the full range of different data exchange scenarios in the BIM context nor the necessary depth of information. As such, to achieve the goal of BIG BIM, it became clear that a vendor-neutral, open and standardized data exchange format was needed.

This approach, while undoubtedly better in the long term, is more difficult and more protracted to put into practice. The international organization buildingSMART has dedicated many years to the development of the Industry Foundation Classes (IFC) as an open, vendor-neutral data exchange format. This is a complex data model with which it is possible to represent both the geometry and semantic structure of a building model using an object-oriented approach. The building is broken down into its building components on the one hand and its spaces on the other, both of which are described in detail along with the interrelationships between them. Thanks to its comprehensive data structure, it can be used for almost any data exchange scenario in the life cycle of a building. The IFC data model is immensely important for implementing BIM concepts and is the basis of many standardization initiatives at an international, European and national level. It is described in detail in this chapter.

The process of establishing a neutral data exchange format is, however, lengthy. While the current Version 4 of the IFC can be considered largely mature and ready for use as a standard, it can only be used in practice once the different software vendors have implemented it as an import and export interface. The quality of the implementation of such interfaces is crucial for its take-up in the industry. In the past, errors in these import and export modules led to data errors or even data loss, impacting on the reputation and market acceptance of the IFC data format.

One reason for the inadequate implementation of the import and export functionality by software vendors is also the complexity of the IFC data model. For example, it is possible to represent 3D geometry in an IFC model in several different ways. For software vendors, this means that they need to support all geometric representation methods to offer full IFC compatibility, which is an immense implementation task.

To overcome this hurdle, buildingSMART introduced the concept of Model View Definitions (MVD) that define which parts of an IFC data model need to be implemented for a specific data exchange scenario. The underlying methods and concepts are discussed in more detail in Chap. 7. Consequently, the Model View Definitions form the basis for certifying the compatibility of software products with the IFC standard. The corresponding certification procedure is presented in Chap. 8.

6.2 History of the IFC data model

As far back as the late 1980s, researchers had already begun investigating ways to improve the exchange of data in the building and construction sector. The idea of "product modeling" as a means of digitally describing a product and its components,

both in terms of their geometry as well as semantic structure, stems from this time (Eastman, 1999).

Methods for exchanging data between different CAD systems first began to be developed in the 1970s to meet a need among major interest groups, such as the US Ministry of Defense and the German Association of the Automotive Industry (VDA), for a common interface for loss-free data exchange. These first data exchange formats, some of which are still in use today (such as the IGES Initial Graphics Exchange Specification), were mostly limited to the exchange of geometric data. Continuing efforts to improve standardization culminated in the 1980s in the development of STEP, a Standard for the Exchange of Product model data. Through Technical Committee 184, Sub-Committee 4 "Industrial Data" (ISO TC 184/SC 4) of the International Organization for Standardization (ISO), a series of different sub-norms were united in the ISO 10303 Standard, developed by a broad alliance of stakeholders from various industrial sectors. In addition to setting out an agreed framework for describing product data representation schemas ISO 10303-11, the family of 10303 standards included graphical notation methods, the definition of file formats for instances (serialization) in different syntactic variants, and uniform information processing interfaces. It also details semantic aspects of the individual product categories. Various industrial sectors grouped relevant product and data exchange scenarios into so-called Application Protocols (APs). Alongside object models for oil drilling platforms, airplane, automotive and ship components, a separate object model for buildings – Application Protocol for Building Elements Using Explicit Shape Representation – were developed.

For many stakeholders, however, the procedure for reaching a consensus on a common approach to modeling building data and its exchange was too long-winded: the bureaucratic framework for standardization under the auspices of the ISO was felt to be holding back developments in the then prospering construction industry. Spurred on by a series of (often EU-funded) research projects and the needs of the industry, a group of engineering offices, construction firms and software manufacturers, most notably Autodesk, decided to collaborate in the foundation of the International Alliance for Interoperability (IAI) in 1995 to speed up the process of standardization. The organization, which re-branded itself in 2005 as "buildingSMART", currently has 19 regional chapters, including the German "buildingSMART e.V.". More than 800 organizations, companies and institutes are now members of buildingSMART and promote the development of standards on behalf of the industry as a whole. A first version 1.0 of these standards was issued in 1997 as the "Industry Foundation Classes – IFC" and version 1.5.1 was the first to be implemented in dedicated construction software applications (Laasko and Kiviniemi, 2012).

Since the first version, numerous revisions and extensions have followed (see Fig. 6.2), which vendors then implemented shortly afterwards in their respective products. These standards were published independently of the ISO as a vendor-neutral standard and made freely available at no cost. Unlike other proprietary, vendor-specific object models, such as Autodesk's popular DWG and ARX formats, there are no licensing fees for using the IFC model. As a consequence, numerous

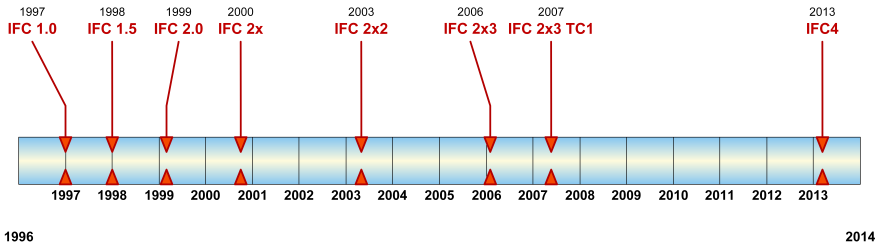


Fig. 6.2 Version history of the IFC format.

software products have since implemented the IFC model. Today there are more than 160 implementations of the standard in individual software products, with most widespread support for version 2x3, although this is gradually being replaced by IFC 4 (as of late 2017). The subsequent incorporation of the IFC in ISO Standard 16739 (2013) also fueled its adoption among public authorities, and in many countries it has now become an obligatory data exchange format for construction tendering and approval procedures.

In recent years, the IFC has become the definitive format for realizing Open BIM. It is already supported by numerous BIM software applications, ranging from BIM modeling tools to structural computation tools and thermal performance analysis tools to software for facility management.

Thanks to the open definition of the data structure and the neutrality of the IFC format, it has become the basis of almost all public sector initiatives that prescribe the use of BIM for public building projects. Pioneering initiatives have been made in Singapore, Finland, Norway, the USA and Great Britain.

The open data format means that data will continue to be legible many years into the future. This is especially important given the longevity of buildings, which typically spans several decades or more.

Currently the IFC data model focuses on the description of buildings. Extensions for describing other built structures, such as civil engineering infrastructure, are currently in development.

6.3 EXPRESS – A data modeling language for the IFC standard

Although the development of the IFC evolved independently of the ISO standardization body and the STEP procedure, it shares much of the same underlying technology, most notably the data modeling language EXPRESS which is defined in part 11 of the STEP standard (ISO 10303-11, 2004).

EXPRESS is a declarative language with which one can define object-oriented data models (Schenck and Wilson, 1993). That means it follows the object-oriented principles described in Chap. 3, such as the abstraction of objects in the real world

into classes (called entities in EXPRESS) which can have attributes and be related to other classes.

EXPRESS employs the construct of an entity type[1] as an equivalent to classes in object-oriented theory. For each entity type, attributes and relationships to other entity types can be defined. EXPRESS also implements the object-oriented concept of inheritance, enabling attributes and relationships to apply similarly to sub-types.

A relationship (association) between an object of Type A and an object of Type B is expressed by giving entity Type A an attribute from the type of Entity B. A special characteristic of the EXPRESS standard is the ability to explicitly define inverse relationships. In this case, no new information is modeled; just a relationship in the reverse direction.

A further special aspect is that aggregation datatypes – list, array, set and bag – are defined as part of the language, making it easier to define relationships with groups of objects. This construct of abstract datatypes makes it possible to define superclasses without these needing to be explicitly instantiated.

EXPRESS offers the possibility to define algorithmic conditions using an optional WHERE block as a means of describing rules for data consistency. The WHERE block contains Boolean expressions that have to evaluate as true for the respective instance to be deemed valid.

Figure 6.3 shows an excerpt of a data model from the IFC standard defined using EXPRESS.

The select type in EXPRESS offers an additional method, alongside inheritance hierarchies, for assigning several entity types to a higher-level construct. This can in turn serve as a placeholder, for example when defining the type of an attribute. An example from the IFC data model is the select type *IfcUnit*, which provides a choice between the types *IfcDerivedUnit*, *IfcNamedUnit* and *IfcMonetaryUnit*.

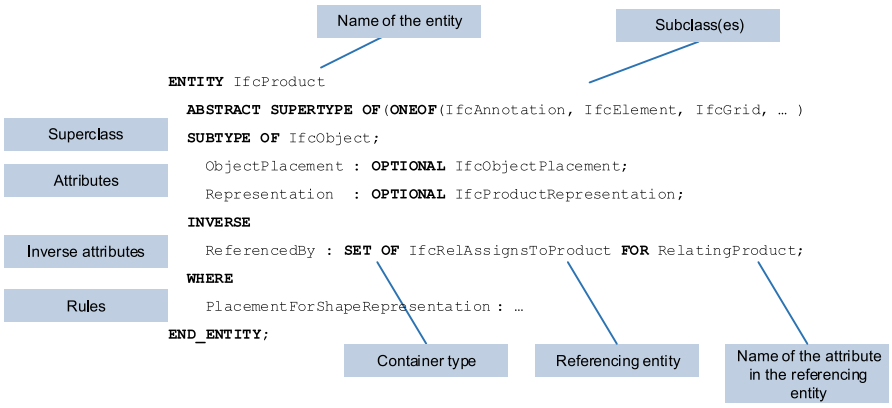


Fig. 6.3 Definition of an entity type using the data modeling language EXPRESS.

[1] In this chapter, the use of the term *class* is synonymous with *entity type*.

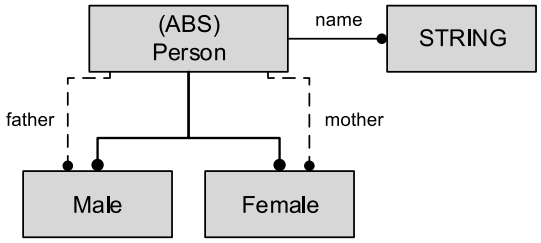


Fig. 6.4 Example of an EXPRESS-G diagram. The entity type *Person* is an abstract supertype for both entity types *Male* and *Female*. This is shown by the thick connecting lines. A circle at the end of a connecting lines denotes the direction of an inheritance relationship. A person has the attribute *name* of type *string* as well as two optional attributes: *father* and *mother* of type *Male* and *Female* respectively. The optional connection is denoted by the dashed connecting line.

Attributes that can only contain specific values from a selection of predefined strings are modeled in EXPRESS with the help of the Enumeration Type. For example, *IfcBooleanOperator* can be either UNION, INTERSECTION or DIFFERENCE.

In addition to this textual notation, EXPRESS also offers a means of modeling data graphically. The corresponding graphical notation language is called EXPRESS-G. Figure 6.4 shows an example of the elements of the graphical language.

It is important to remember that EXPRESS is designed for defining a data model (also known as a schema). It is not possible to describe concrete instances of the data model using EXPRESS. Various different methods can be used for this, of which a STEP Physical File (defined in STEP part 21) is most common. Other options include the use of XML instances or storing data in a database. More information on this is provided in Chap. 12.

6.4 Organization in layers

The IFC data model is both extensive and complex. To improve its maintainability and extensibility, it is therefore structured into several layers (Fig. 6.5). The general principle is that elements in the upper layers can reference elements in the layers below but not vice versa. This ensures that the core elements remain independent.

Core Layer

The Core Layer contains the most elementary classes of the data model. They can be referenced by all the layers above. These classes define the basic structures, key relationships and general concepts which can then be re-used and defined more precisely by classes in the upper layers.

The Kernel schema represents the core of the IFC data model and comprises basic abstract classes such as *IfcRoot*, *IfcObject*, *IfcActor*, *IfcProcess*, *IfcProduct*,

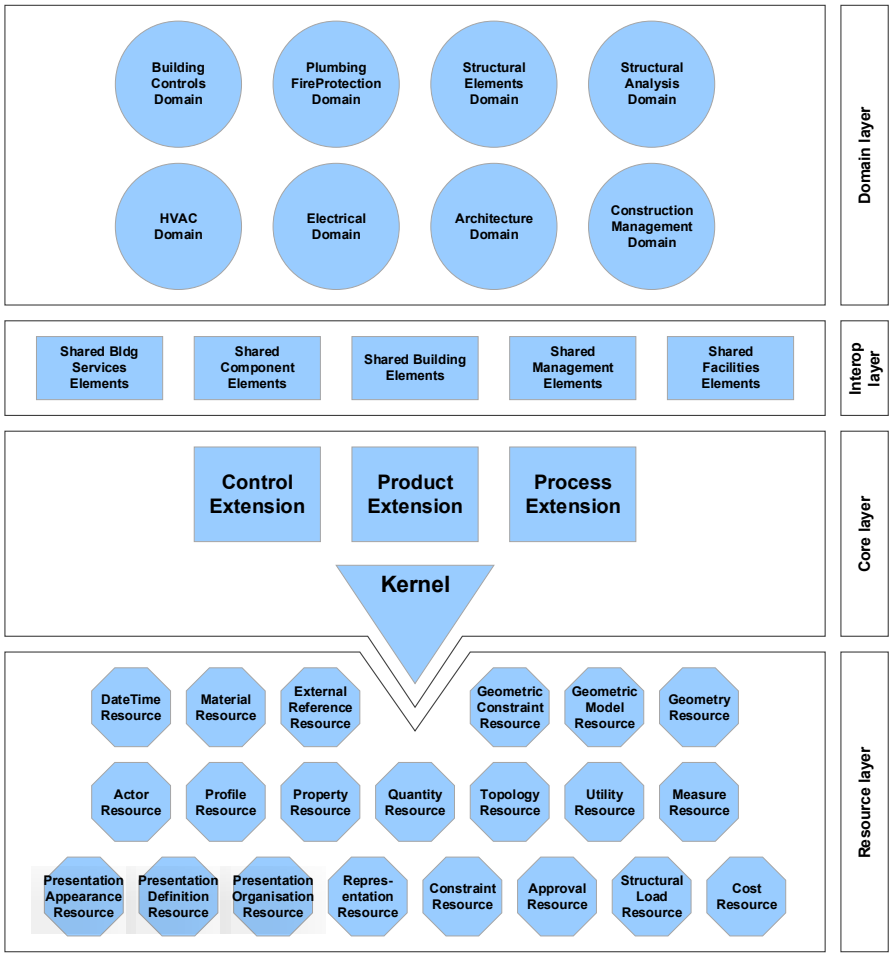


Fig. 6.5 The layers of the IFC data model. Source: IFC Documentation, ©buildingSMART

IfcProject, *IfcRelationship*. Based on these are three scheme extensions *Product Extension*, *Process Extension* and *Control Extension* which are also part of the Core Layer.

The *Product Extension* schema describes the physical and spatial objects of a building and their respective relationships. It comprises the subclasses of *IfcProduct* such as *IfcBuilding*, *IfcBuildingStorey*, *IfcSpace*, *IfcElement*, *IfcBuildingElement*, *IfcOpeningElement* as well as the relationships classes *IfcRelAssociatesMaterial*, *IfcRelFillsElement* and *IfcRelVoidsElement*.

The *Process Extension* schema comprises classes for describing processes and operations. It also provides a basic means for defining dependencies between process elements for linking them with resources.

The *Control Extension* defines the basic classes for control objects such as *IfcControl* and *IfcPerformanceHistory* as well as possibilities for allocating these objects to physical and spatial objects.

Interoperability Layer
The *Shared Layer* lies directly above the Core Layer and represents an interoperability layer between the basic core of the data model and the domain-specific schemes. Here classes are defined that are derived from classes in the Core Layer and can be used by a range of different application schemes, for example, important building element classes such as *IfcWall*, *IfcColumn*, *IfcBeam*, *IfcPlate*, *IfcWindow*.

Domain Layer
The domain-specific schemes contain highly specialized classes that only apply to a particular domain. They form the leaf nodes in the hierarchy of inheritance. The classes defined in this layer cannot be referenced by another layer or by another domain-specific schema.

The IFC4 defines domains for architecture, building control, construction management, electrical systems, heating, ventilation and air conditioning, plumbing and fire protection as well as structural elements (such as foundations, pylons, reinforcement, etc.) and structural analysis.

Resource Layer
At the lowest level, the *Resource Layer*, are schemes that detail basic data structures that can be used throughout the entire IFC data model.

The classes in this layer do not derive from *IfcRoot* and therefore have no identity of their own (see Sect. 6.5.1). Unlike entities in other layers, they cannot exist as independent objects in an IFC model but have to be referenced by an object that instantiates a subclass of *IfcRoot*.

Of these, the most important resource schemes include:

- Geometry Resource: contains basic geometric elements such as points, vectors, parametric curves, swept surfaces (see Sect. 6.7, see also Chap. 2).
- Topology Resource: contains all classes for representing the topology of a solid (see Sect. 6.7, see also Chap. 2).
- Geometric Model Resource: contains all classes for describing geometric models such as *IfcCsgSolid*, *IfcFacetBrep*, *IfcSweptAreaSolid* (see Sect. 6.7, see also Chap. 2).
- Material Resource: contains elements for describing materials (see Sect. 6.6.4).
- Utility Resource: provides elements for describing the ownership and version history (History) of IFC objects.

In addition to these, the resource layer also includes a whole series of further schemes, such as Cost, Measure, DateTime, Representation, etc. that we shall not deal with here.

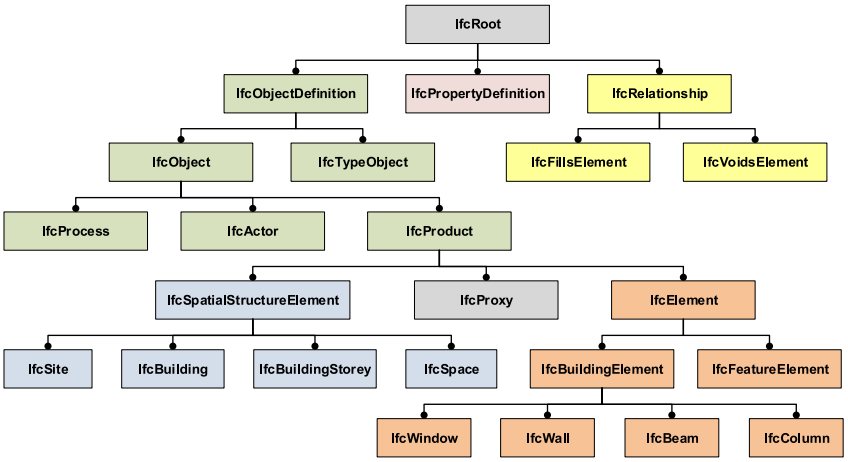


Fig. 6.6 Part of the IFC data model showing the most important entities in the upper layers of the inheritance hierarchy. © A. Borrmann

6.5 Inheritance hierarchy

As in any object-oriented data model, inheritance hierarchy plays a crucial role in the IFC. It defines specialization and generalization relationships and therefore which attributes of which classes can be inherited by other classes. Figure 6.6 shows part of the IFC inheritance hierarchy.

The inheritance hierarchy follows a semantic approach: the meaning of objects is the basis for modeling inheritance relationships. Here we shall concentrate on the most important classes of the IFC inheritance hierarchy.

6.5.1 IfcRoot and its direct subclasses

The starting point and root of the inheritance tree is the class *IfcRoot*. All entities, with the exception of those in the resource layer, must derive directly or indirectly from *IfcRoot*. This class provides basic functionality for uniquely identifying an object using a Globally Unique Identifier (GUID), for describing ownership and the origin of an object and to map the history of changes made to it (identity of the originator and other actors, its version history etc.). In addition, every object can be given a name and description.

Directly derived from *IfcRoot* are the classes *IfcObjectDefinition*, *IfcPropertyDefinition* and *IfcRelationship*, which represent the next level in the inheritance hierarchy.

The class *IfcObjectDefinition* is an abstract superclass for all classes that represent physical objects (e.g. building elements), spatial objects (e.g. openings and

spaces), or conceptual elements (e.g. processes, costs, etc.). It also includes definitions for describing those involved in the building project. The three subclasses of *IfcObjectDefinition* are *IfcObject* (individual objects in the building project), *IfcTypeObject* (object type) and *IfcContext* (general project information).

The class *IfcRelationship* and its subclasses describes objectified relationships. This decouples the semantic of a relationship from the object attributes so that relationship-specific properties can be saved directly with the related object. This concept is discussed in detail in Sect. 6.6.

The class *IfcPropertyDefinition* defines those properties of an object that are not already part of the IFC data model. This aspect is detailed in Sect. 6.8.

6.5.2 IfcObject and its direct subclasses

An *IfcObject* represents an individual object (a thing) as part of a building project. It is as an abstract superclass for six important classes of the IFC data model:

- *IfcProduct* – a physical (tangible) object or a spatial object. *IfcProduct* objects can be assigned a geometric shape representation and are positioned within the project coordinate system.
- *IfcProcess* – a process that occurs within a building project (planning, construction, operation). Processes have a temporal dimension.
- *IfcControl* – an object that controls or limits another object. Controls can be laws, guidelines, specifications, boundary conditions or other requirements that the object has to fulfill.
- *IfcResource* – describes the use of an object as part of a process.
- *IfcActor* – a human participant involved in the building project.
- *IfcGroup* – an arbitrary aggregation of objects.

This subdivision into the areas product, process, control element and resource corresponds to the principal approach to modeling business processes developed back in the 1980s by the IDEF initiative.

6.5.3 IfcProduct and its direct subclasses

IfcProduct is an abstract representation of all objects that relate to a geometric or spatial context. All classes used to describe a virtual building model are subclasses of *IfcProduct*. These can be used to describe both physical objects as well as spatial objects. *IfcProduct* objects can be assigned a geometric shape representation and a location (see Sect. 6.7).

The subclass *IfcElement* is the superclass for a whole series of important basic classes including *IfcBuildingElement*, which is the superclass for all building elements such as *IfcWall*, *IfcColumn*, *IfcWindow* etc.

The *IfcSpatialElement* class, by comparison, is used to describe non-physical spatial objects. Its respective subclasses include *IfcSite*, *IfcBuilding*, *IfcBuilding-Storey* and *IfcSpace*. The organisation of a corresponding relationship structure between these elements is described in Sect. 6.6.2.

The *IfcProduct* subclass *IfcProxy* serves as a placeholder for representation objects that do not correspond to any of the semantic types so that they can still be defined in the IFC model, and if necessary be assigned a geometric representation. *IfcProduct* has further subclasses for describing objects that are embedded within a spatial context, for example *IfcAnnotation*, *IfcGrid* and *IfcPort*.

6.6 Object relationships

6.6.1 General concept

Object relationships are an important part of the IFC data model. In fact, the IFCs powerful functions for detailing relationships between objects can be seen as one of its key qualities. The ability to describe relationships, along with the semantic classification of objects, is a fundamental aspect of an "intelligent" building information model that not only records building elements as isolated bodies but highlights their function and interaction with other objects. Typical relationships can be whole/part relationships (Meronymy, "the south wing is part of the overall building"), connections ("the floor slab is connected to the column") or type definitions ("Beam with an HE-A 140 profile").

The IFC data model follows the principle of objectified relationships (see also Sect. 3.3.2). That means that semantically relevant relationships between objects are not formed by direct association but instead with the help of a special intermediary object that represents the relationship itself (see Fig. 6.7). An important principle of data modeling that has been implemented in the IFC is that the forward relationship (the defined attribute) is always made from the relationship object and points to the related object. The corresponding attributes of the relationship object always have names according to the schema *related...Element* and *relating...Element*. The reverse path from the related objects to the relating objects can be navigated using corresponding inverse attributes.

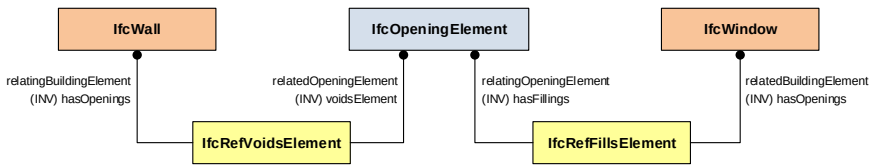


Fig. 6.7 The principle of objectified relationships illustrated using the example of a wall, opening and window.

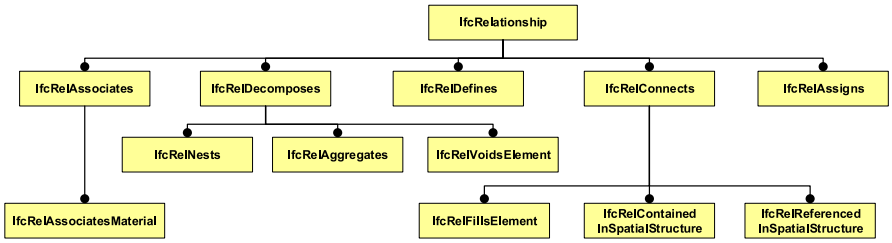


Fig. 6.8 The inheritance hierarchy of relationship classes in the IFC data model.

Relationship objects are always instances of a subclass of *IfcRelationship*. The inheritance tree of the object relationships is shown in Fig. 6.8. The element *IfcRelationship* is the root and every relationship can have an informal description that details the precise purpose for using this relationship.

The following six relationship types serve specific basic functions in the IFC data model:

- IfcRelAssociates – serves to relate an external source of information (such as classifications, libraries or documents) to an object or its properties.
- IfcRelDecomposes – serves as a means of representing concepts of composed objects. The decomposition relationship denotes a whole/part hierarchy with the ability to navigate from the whole to the parts and vice versa. Its subclasses include *IfcRelNests* (the nested parts have an order) and *IfcRelAggregates* (the aggregated parts have no order) and *IfcVoidsElement* (opening relationship).
- IfcRelDefines – links an object instance with a Property Set Definition (Sect. 6.8) or a Type Definition (Sect. 6.9)
- IfcRelConnects – describes a connection between two objects.
- IfcRelDeclares – represents the link between an object, its defined properties and the respective context.
- IfcRelAssigns – represents a generalization of “link” relationships between object instances.

The purpose and application of the individual relationship types will be discussed in the following sub-sections.

6.6.2 Spatial aggregation hierarchy

An important underlying concept for the description of buildings using IFC is the representation of aggregation relationships between spatial objects on the different hierarchical levels. All classes with spatial semantics inherit attributes and properties from the class *IfcSpatialStructureElement*. These are *IfcSite* which describes the building site, *IfcBuilding* to represent the building, *IfcBuildingStorey* for representing a particular story and *IfcSpace* for the individual rooms and corridors. *Ifc-*

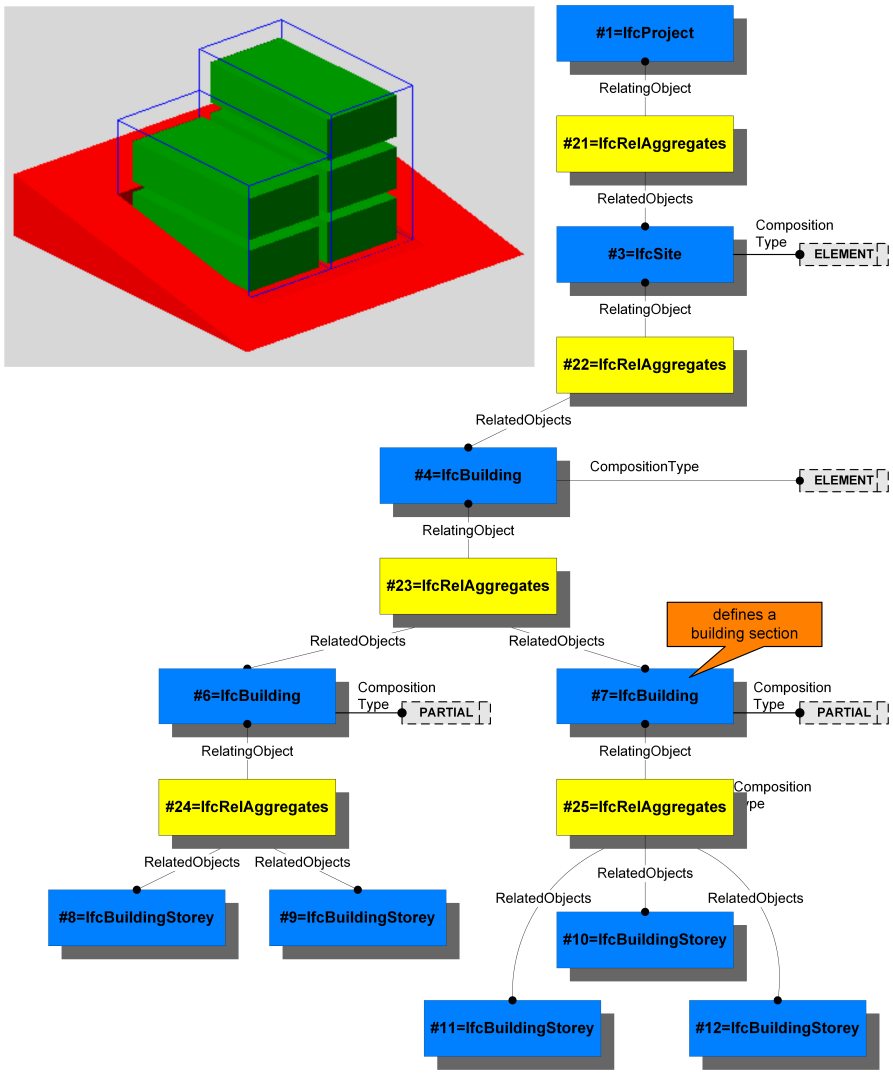


Fig. 6.9 Example of the structure of a hierarchical aggregation relationship between spatial objects in the IFC model (instance diagram). Source: IFC Documentation. ©buildingSMART

SpatialZone introduces a further method for representing general spatial zones that does not correspond to the default building structure taking into account a functional consideration. Instances of these classes are related to one another via relationship objects of the type *IfcRelAggregates*.

Figure 6.9 shows an example of how spatial hierarchy can be represented in an IFC model. At the top of the hierarchy is the *IfcProject* object that describes the context within which the information about the project as a whole is represented.

Important in this context is the use of the attribute *CompositionType* on the aggregated *IfcSpatialStructureElement* which is used to define whether the element is part of a whole (PARTIAL) or simply an embedded element (ELEMENT). For example, sections of buildings are generally modeled as *IfcBuilding* with the *CompositionType* attribute set to PARTIAL.

The data model itself does not define which hierarchy levels may be linked to which other hierarchy levels via aggregation relationships. However, some informal rules do apply, for example that the resulting graph must be acyclic and that elements on a lower level cannot encompass objects from a higher level. The correctness and consistency of the information stored is the responsibility of the respective software program.

To model which building elements lie in which spatial objects, instances of the relationship class *IfcRelContainedInSpatialStructure* are used (see Fig. 6.10). In most cases building elements are linked to stories. However, one must be careful to observe that one building element can only be assigned to a single spatial object per *IfcRelContainedInSpatialStructure* at any one time. Should a building element be linked to several stories (for example a multistory facade element), it should be linked to all other instances via the relationship *IfcReferencedInSpatialStructure* (see Fig. 6.10).

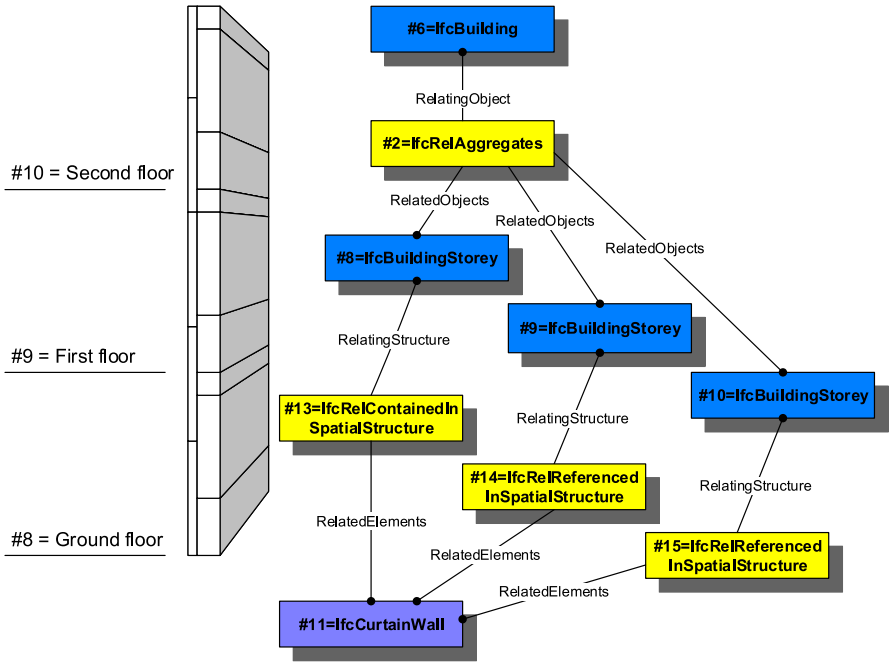


Fig. 6.10 Example of the use of the relationship *IfcRelContainedInSpatialStructure* and *IfcReferencedInSpatialStructure* to describe spatial relationships to a multistory wall element. Source: IFC Documentation. ©buildingSMART

6.6.3 Relationships between spaces and their bounding elements

Numerous applications in the BIM context require a link between a spatial object and the objects that bound the space, such as walls, floor and ceiling. For example, programs for calculating quantities (see Chap. 20) or for computing the energy demand (see Chap. 18). To model such relationships, the IFC data model includes the relationship class *IfcRelSpaceBoundary* (Weise et al., 2009). The attribute *RelatingSpace* refers to the spatial object while *RelatedBuildingElement* refers to the respective bounding element (see Fig. 6.11).

In addition, it is possible to link a relationship object to an actual object using the class *IfcConnectionGeometry* which describes the surface where the space meets the building element. This can be invaluable for certain calculations and simulations.

Space Boundaries are always described from the perspective of the spatial object. One differentiates between two key levels of Space Boundaries (Fig. 6.12):

- Level 1 Space Boundary: boundaries of a space disregarding any changes in building elements or spaces on the other side.

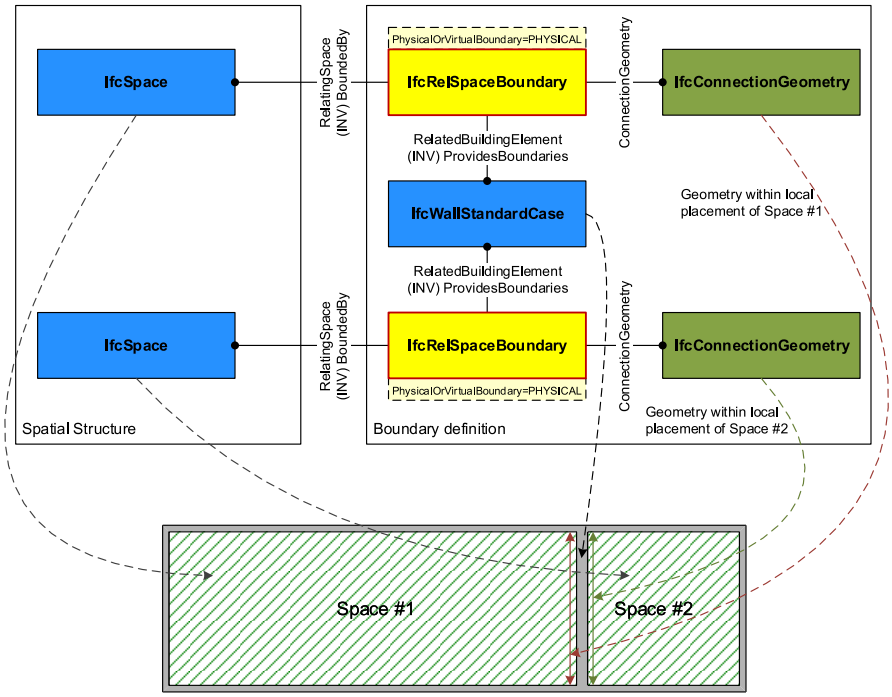


Fig. 6.11 The relationships between a spatial object and the bounding elements are represented using instances of the relationship class *IfcRelSpaceBoundary*. Source: IFC Documentation. ©buildingSMART

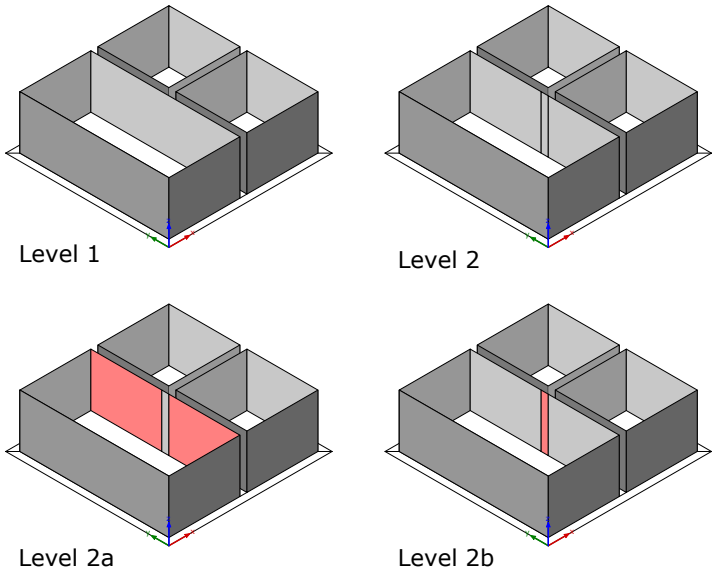


Fig. 6.12 Differences between the space boundaries on Level 1, Level 2a and Level 2b. On Level 1, the boundary of the spaces is modeled without taking into account changes in building elements or spaces on the other side of the boundary. Level 2 takes these into account and subdivides the surfaces accordingly. Level 2 Type A shows all surfaces with a space on the other side, Level 2 Type B all surfaces with a building element on the other side. Source: IFC Documentation. ©buildingSMART

- Level 2 Space Boundary: boundaries of a space taking into account changes in building elements or spaces on the other side:
  - Level 2, Type A: On the other side of the boundary is a space.
  - Level 2, Type B: On the other side of the boundary is a building element.

A more precise definition of space boundaries can be made using application-specific model view definitions as the requirements for the respective situations vary considerably (see Liebich, 2009).

6.6.4 Specifying materials

The specification of materials is an important part of a digital building model. Without information on the materials of each building element it would not be possible to automatically calculate quantities of materials required. Calculations and simulations such as for structural analyses or energy demand calculations likewise require information about the materials used along with their respective parameters. A further important aspect of the IFC model is the ability to represent building elements

comprised of several materials. A typical example is a wall with several layers of different materials.

Materials are specified using the relationship class *IfcRelAssociatesMaterial* linked to a building element (an arbitrary subclass of *IfcElement*). The attribute *RelatingMaterial* typically refers to an object of the class *IfcMaterialDefinition*, which can have several subclasses, the most important of which are described here:

- *IfcMaterial*: the basic entity for describing a material.
- *IfcMaterialConstituent*: describes the material as part of a building element. The material attribute itself refers to an *IfcMaterial* object. The attribute Name is used to unequivocally attribute the material to the respective building element, more precisely to the respective part of the element via *IfcShapeAspect*.
- *IfcMaterialConstituentSet*: describes a set of *IfcMaterialConstituent* objects. Each of these objects is assigned to a part of the building element. For example: a window is comprised of the glazing and the frame. The window is modeled as a building element and associated with an *IfcMaterialConstituentSet* which in turn contains two *IfcMaterialConstituent* objects, one for the frame and the other for the glazing.
- *IfcMaterialLayer*: describes the material of a layer of a multilayer building element. The attribute *LayerThickness* denotes the thickness of the layer, while the attribute Material refers to an *IfcMaterial* object. The attribute *IsVentilated* is set to true if the layer is a ventilated cavity.
- *IfcMaterialLayerSet*: describes a set of *IfcMaterialLayer* objects. Instances of this class are associated with a multilayer building element (see Fig. 6.13).

Composite materials are modeled using the relationship class *IfcMaterialRelationship*, with which it is possible to represent aggregation relationships. The attribute *RelatedMaterials* refers to the individual components while the attribute *RelatingMaterial* refers to the composite material.

The class *IfcMaterial* includes the attribute *Name* as a means of specifying a unique name and can also accommodate classifying materials according to an external classification system by linking it with *IfcExternalReferenceRelationship*.

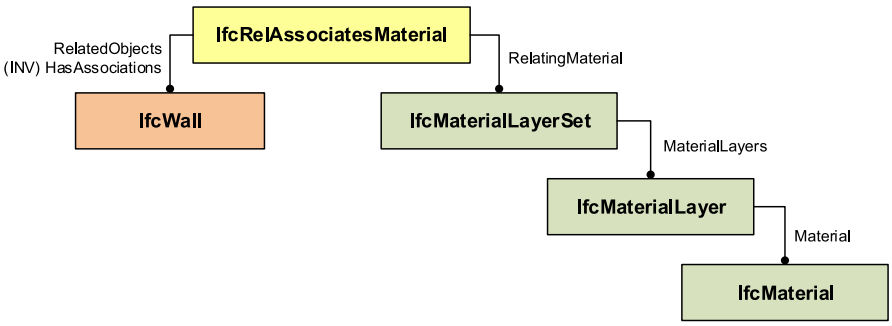


Fig. 6.13 Example for linking a building element comprised of multiple layers with its materials using the relationship class *IfcRelAssociatesMaterial*.

In addition, material parameters can also be linked to one or more objects of the type *IfcMaterialProperties*, which can be referenced via the inverse attribute *HasProperties*. The class *IfcMaterialProperties* describes a set of material properties in the form of a name-value list (see Sect. 6.8). A series of predefined property sets already exist, for example for mechanical properties (*Pset_MaterialMechanical*), optical properties (*Pset_MaterialOptical*), thermal properties (*Pset_MaterialThermal*), and parameters for energy demand calculations (*Pset_MaterialEnergy*). Specific parameter sets have been developed for common materials such as concrete, steel and wood.

Using the *IfcMaterial*, presentation information can also be associated with a building element. The inverse attribute *HasRepresentation* is applied to an object of type *IfcMaterialDefinitionRepresentation*, which defines the line type and thickness as well as hatching (for 2D drawings) or information necessary for rendering the surface of the material (for 3D presentations).

6.7 Geometric representations

6.7.1 Division between semantic description and geometric representation

The IFC data model makes a strict division between the semantic description and its geometric representation. The semantic representation is the defining aspect: all objects within a building project are initially described as a semantic identity and can then be linked with one or more geometric representations (Fig. 6.14). The concept of identity is therefore linked only to a semantic object, and not its geometric representation.

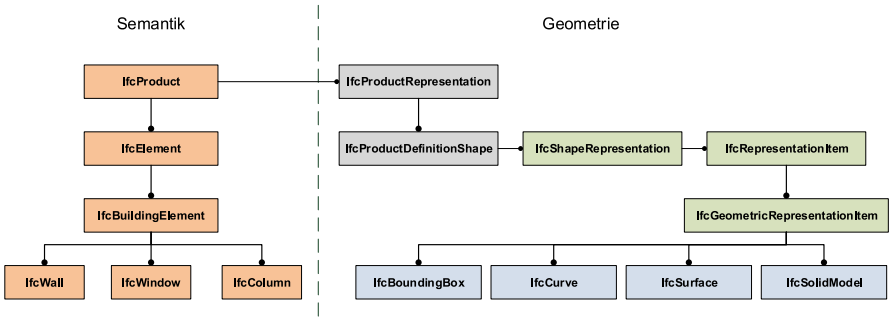


Fig. 6.14 The IFC data model makes a strict division between the semantic structure and geometric description. This affords the flexibility to link one or more geometric representations with a semantic object.

The ability to link distinct geometric representations with an object addresses the need for distinct geometric representations for distinct application scenarios. For example, visualization programs usually only need a simple triangulated geometric description while BIM modeling tools need good quality Brep (boundary representation) or CSG (constructive solid geometry) descriptions in order to be able to make changes to the model. It is also possible to link a 2D representation with a semantic object so that drawings can be stored within an IFC model to be compliant with standards.

The problem of maintaining consistency between the distinct representations must be dealt with by the modeling programs as the IFC data model does not include such functionality.

6.7.2 Forms of geometric description

The IFC model implements a broad range of the geometric models presented in Chap. 2. This section focuses on the most important geometric representations.

All the classes needed for geometric modeling belong to one of the three schemes *Geometric Model Resource*, *Geometry Resource*, or *Topology Resource*. In the majority of cases, the definitions and data structures correspond exactly to those set out in part 42 of the STEP standard, and in the case of indexed geometry descriptions from the X3D standard (ISO/IEC 19775-1, 2004).

All geometry classes inherit from the abstract superclass *IfcGeometricRepresentationItem*. Its subclasses can be grouped into classes for representing curves (*IfcCurve* and its subclasses), classes for describing surfaces in space (*IfcSurface* and its subclasses) and classes for representing solids (*IfcSolidModel* and its subclasses). The dimensions are specified using the *Dim* attribute in the class *IfcGeometricRepresentationItem*.

Points, Vectors, Directions
The entity types *IfcCartesianPoint*, *IfcCartesianPointList*, *IfcVector* and *IfcDirection* are used to define points, vectors and directions.

Curves in 2D and 3D
To model line objects, the entity type *IfcCurve* and its subclasses *IfcBoundedCurve*, *IfcConic*, and *IfcLine* are used. Freeform curves can also be modeled using the class *IfcBSplineCurve* (see also Chap. 2). The *IfcCompositeCurve* can be used to model complex curves comprised of several curved sections.

In addition to 3D geometric representation, the IFC data model explicitly supports the storage of 2D representations for plan drawings. In such cases, the dimensionality of the respective *IfcCurve* objects must be set to 2. This approach can be used to model profiles for extrusion and other similar operations.

Bounding Box
The Bounding Box is a highly simplified geometric representation for three-dimen-

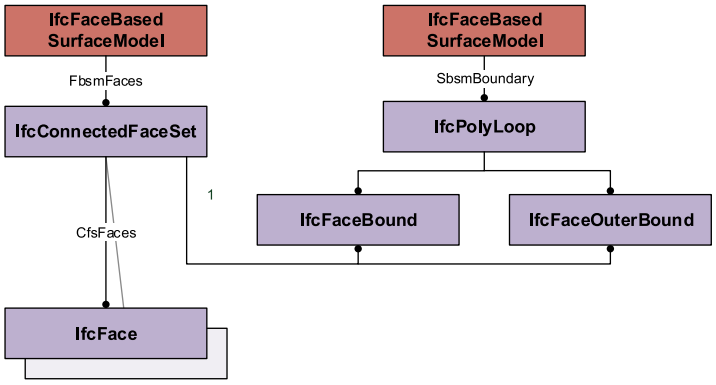


Fig. 6.15 Entities used to describe surface models.

sional objects that is commonly used only as a placeholder, or in combination with a more detailed description. Using the class *IfcBoundingBox* one can define a corner point and three edge lengths for the respective dimensions of the box.

Surface Model

Surface models offer a means of describing composite surfaces comprised of several sub-surfaces. They are used to describe broad surfaces (such as a terrain) or very flat surfaces (such as metal sheeting). 3D solids can also be described via their surfaces. An advantage of this method over Brep modeling is its simpler data structure; a disadvantage is the limited ability to verify the correctness of the modeled solid, for example incorrect intersections (e.g. gaps or overlaps) between faces.

The IFC data model supports two different variants of surface models (Fig. 6.15). The *IfcFaceBasedSurfaceModel* makes it possible to model simple bodies without holes or cavities while the *IfcShellBasedSurfaceModel* can be used to model solids with cavities or holes through the use of any number of *IfcShell* objects. These shell objects can be either open shells (*IfcOpenShell*) or closed shells (*IfcClosedShell*).

Triangulated surface descriptions/Tessellation

A widely used method for describing geometric forms is the use of triangulated nets. This very general and simple form of geometric representation can be interpreted by nearly all visualization software applications. Its main limitations are that curved surfaces are not represented precisely but approximated into triangular facets, that they are data intensive and that many applications offer only limited support for editing them. As such, this geometric representation is not always the most suitable form for building geometries. One area where triangulated surfaces excel is for the description of digital terrain models (DTMs).

For such uses, the IFC data model provides the class *IfcTriangulatedFaceSet*. This is derived from the class *IfcTessellatedFaceSet* that represents the general principle of tessellated surfaces, i.e. polygons with an arbitrary number of edges. *IfcTessellatedFaceSet* is not derived from *IfcSolidModel* but instead inherits from *IfcTesselatedItem*.

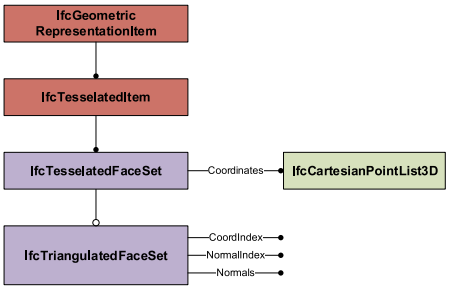


Fig. 6.16 Data structure for the representation of triangulated surfaces.

The IFC model implements the *Indexed Face Set* approach described in Chap. 2. The class *IfcTriangulatedFaceSet* refers via the Coordinates attribute to an object of type *IfcCartesianPointList3D* which describes a list of points (Fig. 6.16). A further attribute, *CoordIndex*, describes the index of the three vertices for each triangle. The normals for each triangle can be optionally specified using the *Normals* attribute. In addition, it is possible to link color values or textures with the index values.

Solid Modeling

The IFC data model supports a number of different ways of modeling 3D solids. These are represented by the abstract superclass *IfcSolidModel* and its subclasses *IfcCsgSolid*, *IfcManifoldSolidBRep*, *IfcSweptAreaSolid* and *IfcSweptDiskSolid* (see Fig. 6.17). This section describes each of these approaches in detail.

Boundary Representation

The most powerful and flexible approach to modeling geometric solids is through Boundary Representation (Brep). The two subclasses of *IfcManifoldSolidBrep*, *IfcFacetedBrep* and *IfcAdvancedBrep* implement the typical Brep data structure, as de-

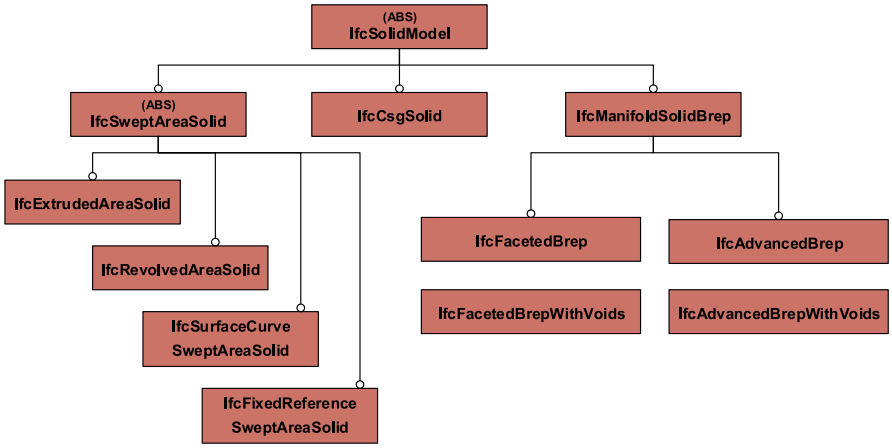


Fig. 6.17 The IFC data model provides a number of different ways to model volumetric bodies (solids).

scribed in Chap. 2. *IfcFacetedBrep* is limited to flat surfaces while *IfcAdvancedBrep* can model surfaces with curved edges.

Both Brep types are, however, limited to the description of shells, making them unsuitable for modeling geometric objects with cavities and holes. To model these kinds of objects, the corresponding subclasses *IfcFactedBrepWithVoids* or *IfcAdvancedBrepWithVoids* should be used which extend their respective superclasses by providing the ability to specify several closed shell objects (Fig. 6.17).

For the modeling of solids with flat surfaces, the class *IfcFacetedBrep* is used (Fig. 6.18). The basis of this is an *IfcFacetedBrep* object, the *Outer* attribute of which references an object of type *IfcClosedShell* which in turn references a series of *IfcFace* objects. Each of these *IfcFace* objects can have any number of bounding surfaces modeled using *IfcFaceBound*. Each *IfcFaceBound* object refers to an *IfcLoop* object which describes a list of points (the vertices of the solid). It is important that each object (point, edge) is not instantiated more than once but merely referenced several times as required.

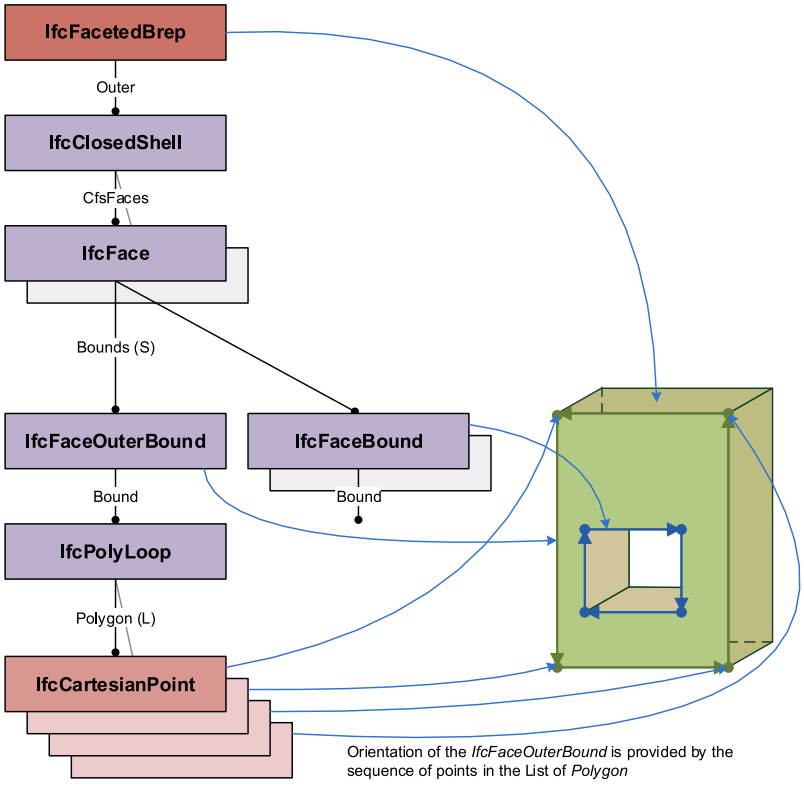


Fig. 6.18 Data structure for representing solids with flat surfaces and straight edges. Source: IFC Documentation. ©buildingSMART

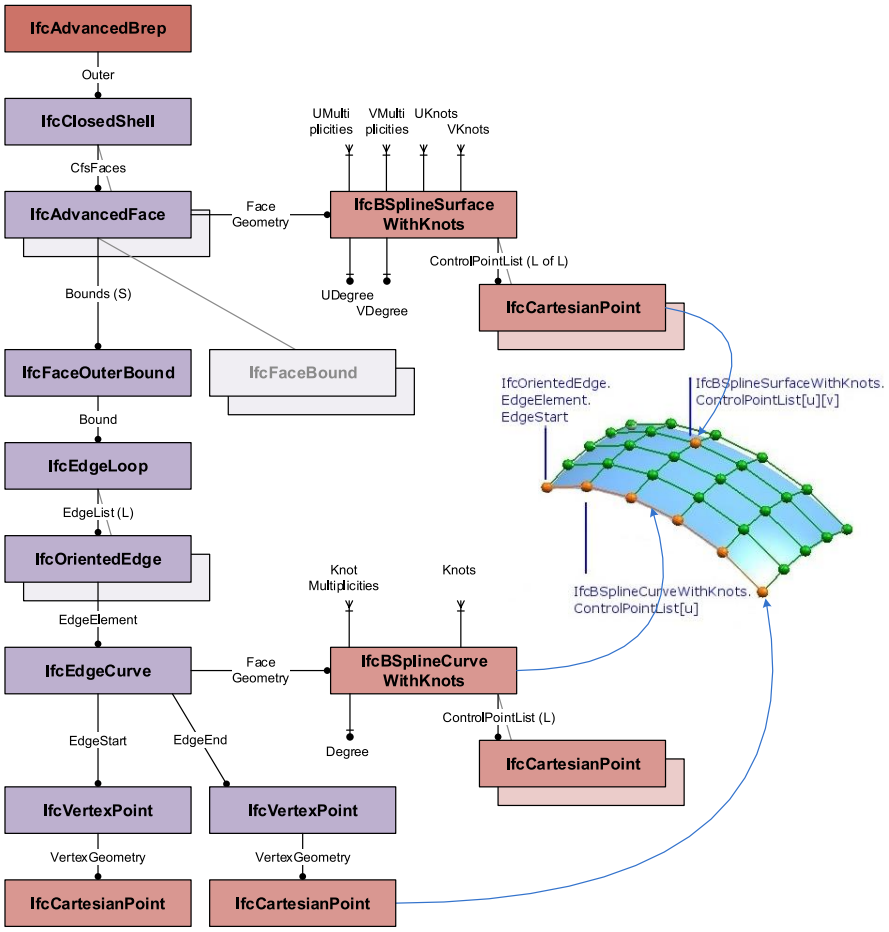


Fig. 6.19 Data structure for representing solids with curved surfaces and edges. Source: IFC Documentation. ©buildingSMART

The data structure for describing solids with curved surfaces extends this basic topological data structure by providing elements for modeling the geometric progression of surfaces and edges (Fig. 6.19). The basis for this is the class *IfcAdvancedBrep*. As above, this is linked to an *IfcClosedShell* object which in turn refers to surface objects of type *IfcAdvancedFace*. Unlike the *IfcFace* objects described above, these include an explicit geometric description. This can be a NURBS surface modeled as an *IfcBSplineSurface*. Objects with this class refer to the corresponding control points and must specify all the necessary parameters to describe a NURBS surface (see Chap. 2). To model the (curved) progression of edges, these can be linked to *IfcBSplineCurve* objects, which in turn reference the corresponding control points and parameters.

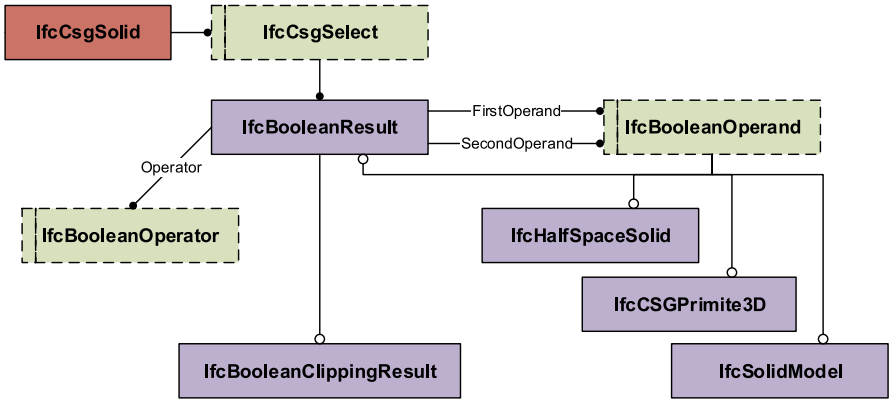


Fig. 6.20 Data structure for describing solids using the CSG approach.

Constructive Solid Geometry

As described in Chap. 2, the Constructive Solid Geometry (CSG) approach models solids by combining predefined basic solid objects (primitives) using Boolean operations such as union, intersection and difference. The IFC data model provides the class *IfcCsgPrimitive3D* with its subclasses *IfcBlock*, *IfcRectangularPyramid*, *IfcRightCircularCone*, *IfcRightCircularCylinder* and *IfcSphere*.

The class *IfcBooleanResult* is used to model the results of the combination operations (Fig. 6.20). This class provides an *Operator* attribute which can have one of three values – UNION, INTERSECTION, or DIFFERENCE – along with the attributes *FirstOperand* and *SecondOperand* which refer to the two operands. The operands can be of type *IfcSolidModel*, *IfcHalfSpaceSolid*, *IfcCsgPrimitive3D* or *IfcBooleanResult*. CSG models are described exclusively by the latter two classes. The class *IfcBooleanResult* can be used recursively to define a tree-like structure. What makes this data structure especially powerful is the ability to use instances of any subclass of *IfcSolidModel* as an operand, for example solids that have been defined elsewhere by extrusion.

Clipping

Clipping can be used to model solids that are cut off by a plane. Clipping is implemented as a special variant of the CSG approach. The first operand is always a volumetric solid (*IfcSolidModel*) and second operand is a so-called half-space solid (*IfcHalfSpaceSolid*), that is defined along a plane and in one direction. The operator is always DIFFERENCE. Clippings can occur anywhere as a node in a CSG tree, and are used, for example, to model the slanted tops of walls that meet diagonal surfaces (see Fig. 6.21).

Rotation, Extrusion and Swept Solids

The IFC data model also provides a means for modeling 3D solids as the result of the rotation or extrusion of a 2D profile (Fig. 6.22) through the class *IfcSweptAreaSolid* and its subclasses *IfcExtrudedAreaSolid*, *IfcRevolvedAreaSolid*, *IfcFixedRef-*

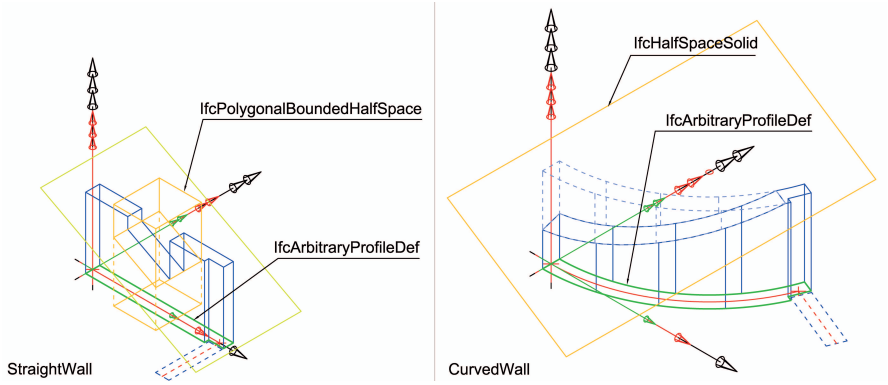


Fig. 6.21 Clippings are often used to model the slanted tops of walls that meet diagonal surfaces. Source: IFC Documentation. ©buildingSMART

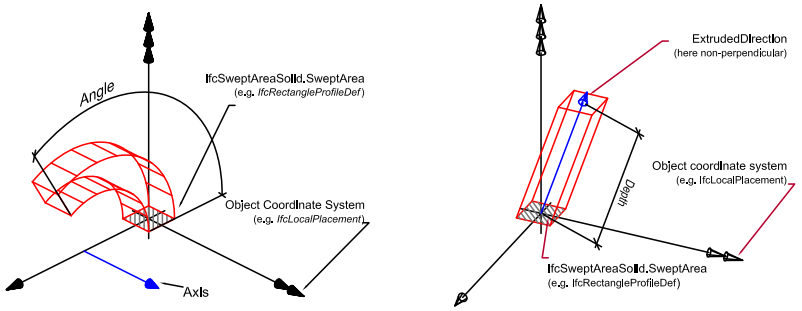


Fig. 6.22 The geometric representations *IfcRevolvedAreaSolid* and *IfcExtrudedAreaSolid*. Source: IFC Documentation. ©buildingSMART

erenceSweptAreaSolid, and *IfcSurfaceCurveSweptAreaSolid*. In addition, there is also the class *IfcSweptDiskSolid*, which inherits directly from *IfcSolidModel*.

The basis for each operation is the definition of a profile in the form of an *IfcProfileDef* object referenced by the *SweptArea* attribute. The most common subclass of *IfcProfileDef* is *IfcArbitraryClosedProfileDef*, which defines a closed profile by referencing an arbitrary *IfcCurve* object.

Through the use of the class *IfcExtrudedAreaSolid*, this profile can be used as a basis for an extrusion operation along a given direction (*ExtrudedDirection* attribute) for a specified distance (*Depth* attribute). Using the class *IfcRevolvedAreaSolid*, the profile is instead rotated around a given axis (*Axis* attribute) for a specified angle (*Angle* attribute).

The class *IfcFixedReferenceSweptAreaSolid* can be used to model an object as the result of the sweeping of a profile along a given curve in space (*Directrix* attribute). A key characteristic of this representation is that the profile does not twist during the sweep but remains orientated on reference to the fixed reference vector (*FixedReference* attribute).

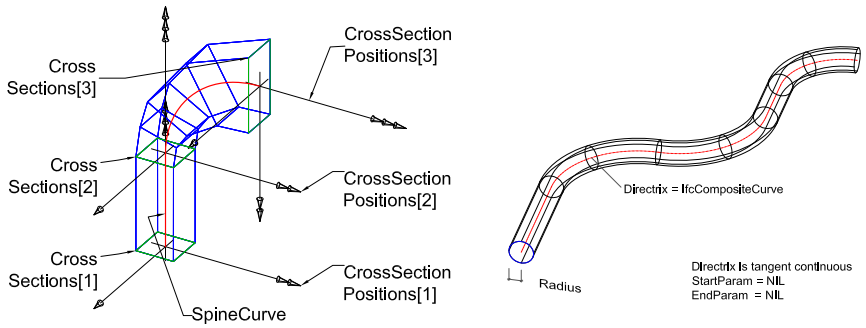


Fig. 6.23 The geometric representations *IfcSectionedSpine* and *IfcSweptDiskSolid*.

Using instances of the class *IfcSectionedSpine* it is possible to describe objects that result from the linear interpolation between a series of successive profiles (Fig. 6.23 left). The attribute refers to an *IfcCompositeCurve* object which describes the path as a composite curve, the segments of which lie between two profiles. The *CrossSections* attribute refers to a list of profiles whose positions are defined by the attribute *CrossSectionPositions*.

The class *IfcSweptDiskSolid* is not derived from *IfcSweptArea* but directly from *IfcSolidModel*. The underlying profile is always a circular disc which follows the path of a curve through space (*Directrix* attribute) as shown in Fig. 6.23 (right). Unlike *IfcFixedReferenceSweptAreaSolid* the circular profile does not maintain a fixed orientation but turns with the path of the sweep so that it is always perpendicular to the path of the curve.

6.7.3 Relative positioning

Geometric modeling in the IFC data model is strongly oriented around the use of a local coordinate system. As such the corners of a wall object, for example, are not specified globally but in relation to the coordinate system of the respective story. The story's coordinates are, in turn, modeled in relation to the coordinate system of the building, and so on. This hierarchical organization of the coordinate system affords greater flexibility should changes occur. For example, if the height of an individual object in the building needs to be modified, only one value needs to be changed, and all relative coordinates remain unchanged.

In the IFC data model, this concept is known as *Local Placement*. The IFC includes a series of classes for this purpose that all inherit from *IfcObjectPlacement* (Fig. 6.24). The class *IfcLocalPlacement* is derived from *IfcObjectPlacement* and provides two attributes: the optional attribute *PlacementRelTo* refers to the *IfcObjectPlacement* that the parent coordinate system provides. If this is not set, the respective object is positioned absolutely within the global coordinate system. The

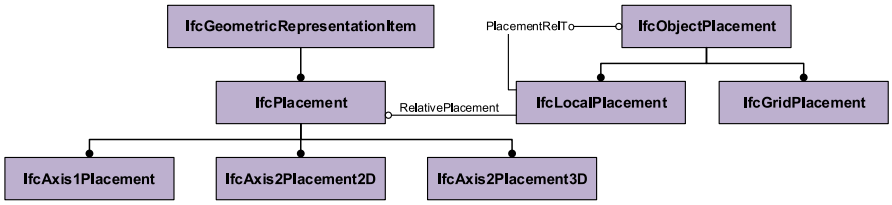
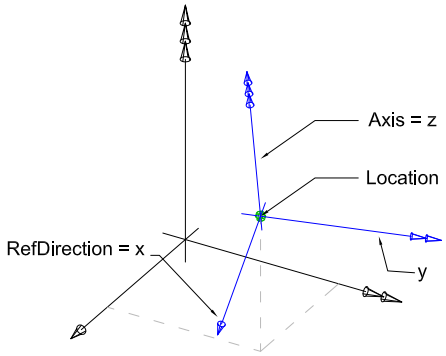


Fig. 6.24 The inheritance hierarchy of entities for describing location relationships.

Fig. 6.25 The functioning of relative positioning using an *IfCAxis2Placement3D*.



attribute *RelativePlacement* refers to an *IfcAxis2Placement* object that defines the transformation between the parent coordinate system and the embedded local coordinate system. This transformation can be either in 2D (*IfcAxis2Placement2D*) or 3D (*IfcAxis2Placement3D*).

Figure 6.25 shows how *IfcAxis2Placement3D* works. The location of the origin of the local coordinate system in relation to the parent coordinate system is defined using the *Location* attribute. Any rotation of the local coordinate system is specified by two vectors: the *Axis* vector defines the direction of the local z-axis while the *RefDirection* vector defines the direction of the local x-axis. Both vectors must be perpendicular to each other. The class *IfcAxis2Placement2D* works the same way but for 2D coordinate systems. Here only one rotation needs to be given, namely the attribute *RefDirection*.

There is a close relationship between the hierarchy of the *IfcLocalPlacement* objects and the aggregation hierarchy of the spatial objects (see also Sect. 6.6.2). The convention is that only the *IfcSite* object is positioned within the global coordinate system. All elements beneath this in the spatial hierarchy are positioned as a local placement with respect to the respective parent object (Fig. 6.26).

In addition to the aforementioned method of local placement, there is also the ability to use a grid as a basis for aligning objects. The class *IfcGrid* provides a very flexible means of defining grids. The predefined grid types include rectangular, radial and triangular grid layouts (Fig. 6.27) but entirely irregular grids can also be defined. The actual placement is undertaken using the class *IfcGridPlacement*

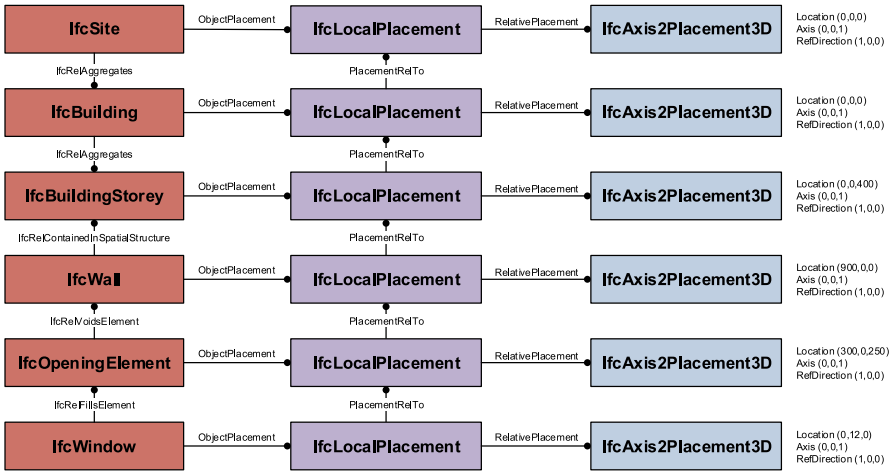


Fig. 6.26 Relationship between LocalPlacement and aggregation hierarchy of the building object.

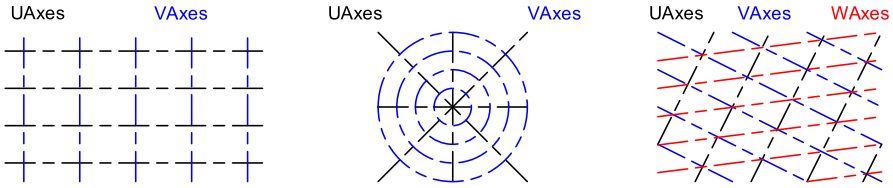


Fig. 6.27 Different forms of grids on which building elements can be placed.

and its attribute *PlacementLocation* which refers to a node in the underlying grid (*IfcVirtualGridIntersection*).

6.8 Extension mechanisms: Property sets and proxies

Several key characteristics of objects, for example of door and wall elements, can be defined directly within a schema of the IFC model with the help of attributes in an entity definition. For standard doors, these might the absolute height and width of the door, which can be specified by the attributes *OverallWidth* and *OverallHeight* when instantiating a door object. The many other important and desirable characteristics of doors (fire safety class, security, thermal performance, etc.) would make the already extensive schema unnecessarily bloated and slow its implementation. Similarly, it would not be possible to include all the unforeseen or international standardized characteristics needed by various users without making changes to the schema. To address this problem, the IFC model takes a two-pronged approach to defining characteristics: static attributes that are defined within the schema along with dynamically created properties. Such properties can be defined with the help of

the subclasses of *IfcProperty* (typically *IfcPropertySingleValue*) and added freely as required to the instance model. There is no limit to the number of properties that can be added. The definition of a new object property is defined via a simple name-value datatype-unit tuple, for example: “Name: ’FireRating’; Value: ’F30’; Datatype: ’IfcLabel”’. Individual *IfcProperty* definitions are grouped into an *IfcPropertySet* and assigned to an object (*IfcRelDefinesByProperties*). A schematic overview of these two primary mechanisms for defining properties is shown in Fig. 6.28.

Software vendors need only implement the basic entity of the properties, for example *IfcPropertySingleValue* with the attributes ’Name’, ’NominalValue’ ’Type’ and ’Unit’ in order to provide a generally applicable template mechanism in their application. This extension mechanism for property definitions is supplemented by the placeholder entity *IfcProxy* which makes it possible to also define the semantic meaning of a class dynamically (i.e. “in run-time”). This provides the IFC with a meta-model that permits numerous semantic extensions, making it possible to cover a wide range of application scenarios independently of the implementation. This flexibility is desirable for many scenarios where special objects and properties are not defined in the schema, for example because they have limited general applicability. German building codes, acoustics simulations or vendor-specific product properties are not general enough to warrant their definition in a globally applicable data schema, but can nevertheless be created within a model as needed in a standard-compliant form that can be transported and read by software. If the recipient/receiving software is unable to interpret a property (for example the value “FireRating” for the attribute instance “Name”) in its respective context, it can simply leave it as is.

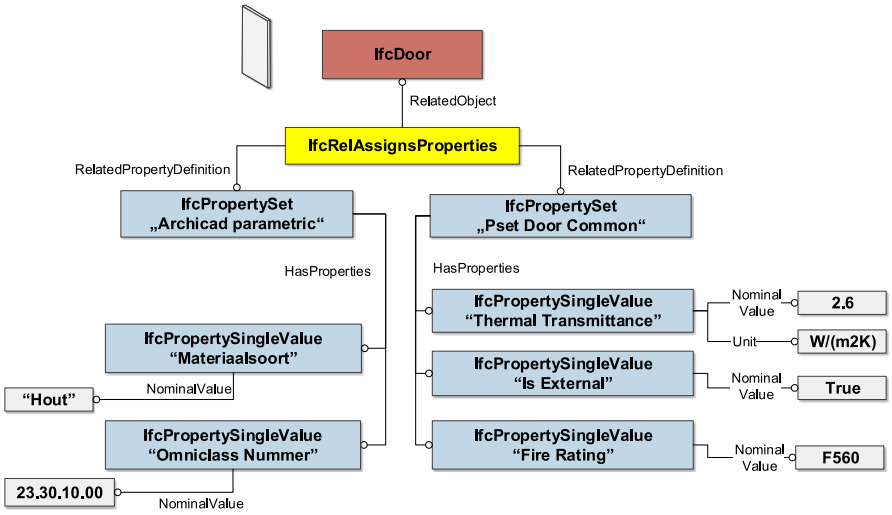


Fig. 6.28 Example use of properties. Left: Ad hoc properties assigned at the level of the instance, Right: Properties from a standardized *PropertySet*.

The disadvantage of this dynamic approach and the external definition of semantics is the potential for the creation of large numbers of arbitrary objects and properties by different parties for the same purpose: what one user defines as "FireRating" may be "FireResistanceClass" for another. To help minimize multiple occurrences (which was the original dilemma that the IFC tries to address in order to improve interoperability) users and developers have jointly attempted to voluntarily standardize the most common properties.

Instead of anchoring these within the schema, they are made available as separate files, embedded within the model documentation, on the website of the buildingSMART organization. These *PropertySet* definitions are saved as straightforward XML-format files with the naming scheme "Pset_*.xml", for example "Pset_DoorCommon.xml". Many object classes such as typical building elements (roof, wall, column, etc.) have extensive collections of such standardized properties. The door classes *IfcDoor* and *IfcDoorType*, for example, have, in addition to the *Pset_DoorCommon* collection with 16 properties (e.g. "AcousticRating" and "FireExit"), further property sets including *Pset_DoorWindowGlazingType* and *Pset_DoorWindowShadingType* covering door glazing and shading properties.

Together with the door-specific properties for the door frame, door case and door leaf and the general properties that apply to all building elements (environmental aspects, guarantee and service properties, vendor-specific information, etc.), more than 135 further properties are available for describing doors.

As the administration and upkeep of the growing amount of additional information in individual files has become increasingly ineffective, the buildingSMART organization began with version 2x3 to incorporate the standardized *PropertySet*-Definitions into the database of the buildingSMART Data Dictionary (bSDD, see also Chap. 9) for better administration. Alongside the master definitions in English, many properties are now also available in other languages such as German, French, Japanese and Chinese.

A further means of extending the IFC model is by making direct references to properties in external classification and product libraries such as the bSDD. This approach is described in a section of its own in Chap. 9.

Future developments, for example in the field of the "Semantic Web", will introduce further means of dynamic property generation and more flexible extension possibilities.

6.9 Typification of building elements

To describe building elements that occur frequently within a project (beams with a certain profile, internal doors, light fittings, etc.) more efficiently, the IFC model supports the concepts of reusable types. To begin with, a "template" of an element is defined which can then be instantiated and adapted accordingly. As a result, only the data that is different needs to be adapted, for example the spatial location of the object or its relationship to a neighboring building element ("Door in a wall",

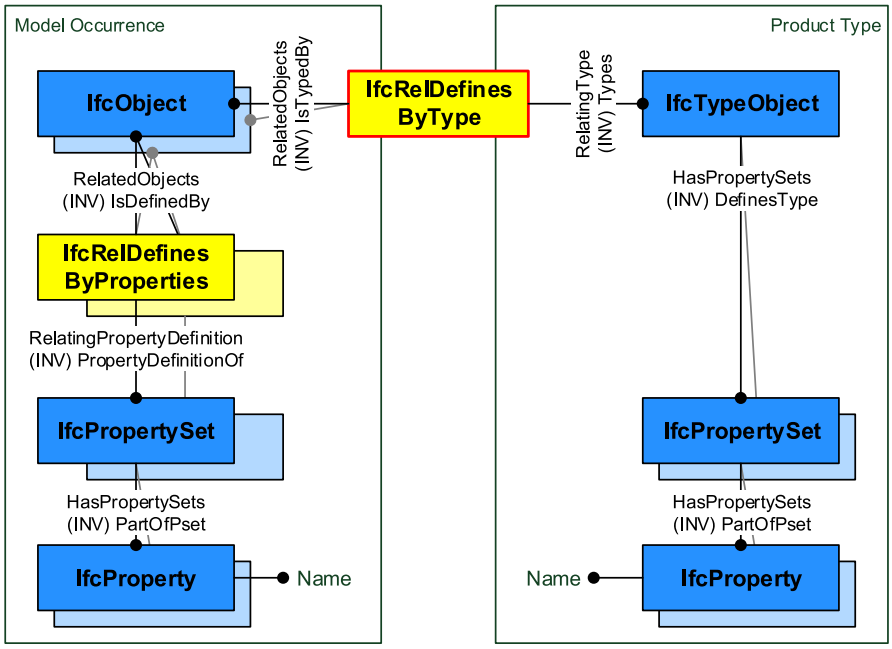


Fig. 6.29 Semantic typification of an object. Source: IFC Documentation. ©buildingSMART

"Beam resting on a column") while the other basic parameters remain unchanged. The IFC model supports typification in two different places:

Semantic typification: An *IfcTypeObject* is assigned to an object using the *IfcRelDefinesByType* relationship. Before a concrete object is instantiated, a collection of properties is defined and grouped in *IfcPropertySets* (see Sect. 6.8) and then applied via the attribute *HasPropertySets* to the type that will be valid for all object instances of that type, such as the fire rating of a door. All concrete instances of an *IfcDoor* object that are assigned via *IfcRelDefinesByType* to this *IfcTypeObject* will then have the same fire rating class. This mechanism is shown schematically in Fig. 6.29. The type properties can, however, be adapted for each instance of an object. A door, that has been assigned the property "FireRating" is "F30" through a door type, can be assigned the same "FireRating" with a higher rating "F60" at the level of the instance. This value that applies to the individual instance overrides or replaces the original "F30" value of the type object.

Geometric typification, i.e. the recurrence of a geometric representation of an object can be modeled in the IFC model using the concept of *IfcMappedItems* (see Fig. 6.30). In a manner similar to the block concept of most CAD programs, a geometric representation of the form (*IfcShapeRepresentation*) is first created and stored together with a local coordinate system in an *IfcRepresentationMap* object. This is then, as with the semantic *IfcPropertySets*, assigned to an *IfcTypeObject*,

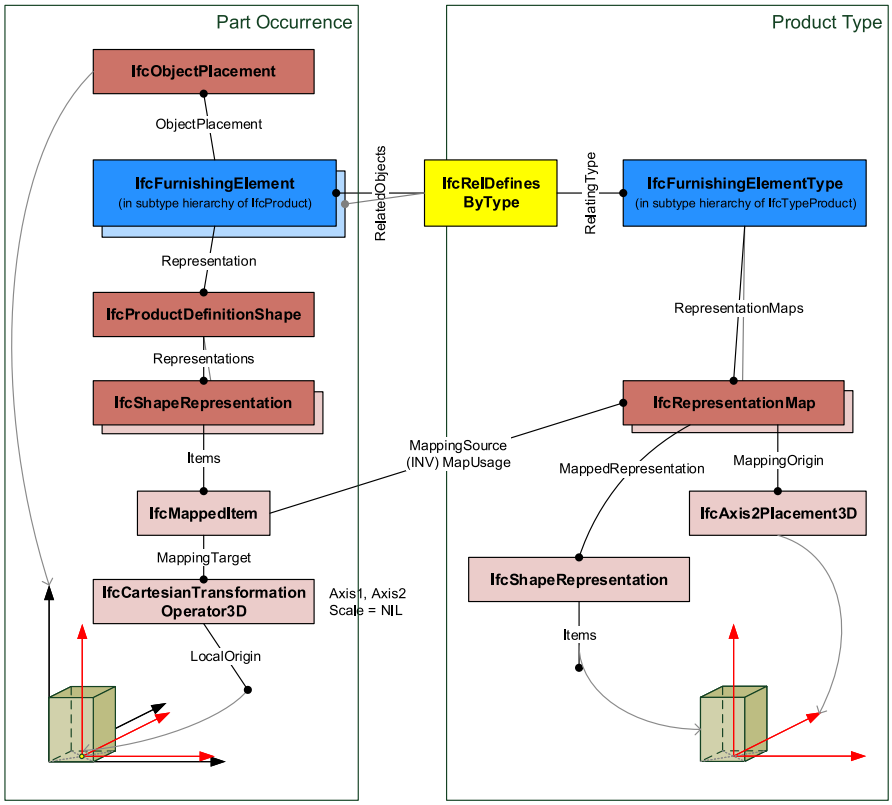


Fig. 6.30 Example for the use of object types in IFC – an instance object is associated with a type object containing a geometric representation, which is mapped to the instance object using the *MappedItem* concept. Source: IFC Documentation. ©buildingSMART

for example a door type. When a new door instance is created, the *IfcRepresentationMap* is then referenced. The spatial position of the element instance is then determined using a local transformation (*IfcCartesianTransformationOperator*). With the help of this transformation it is also possible to change not only the position and rotation of an instance but also its scale. In practice, however, this is rarely undertaken as it can easily lead to inconsistencies and simple changes in scale are not parametric, i.e. increasing the width of a window also increases the size of the profiles and the window handle rather than maintaining their size and repositioning them accordingly as would happen with a true parametric object.

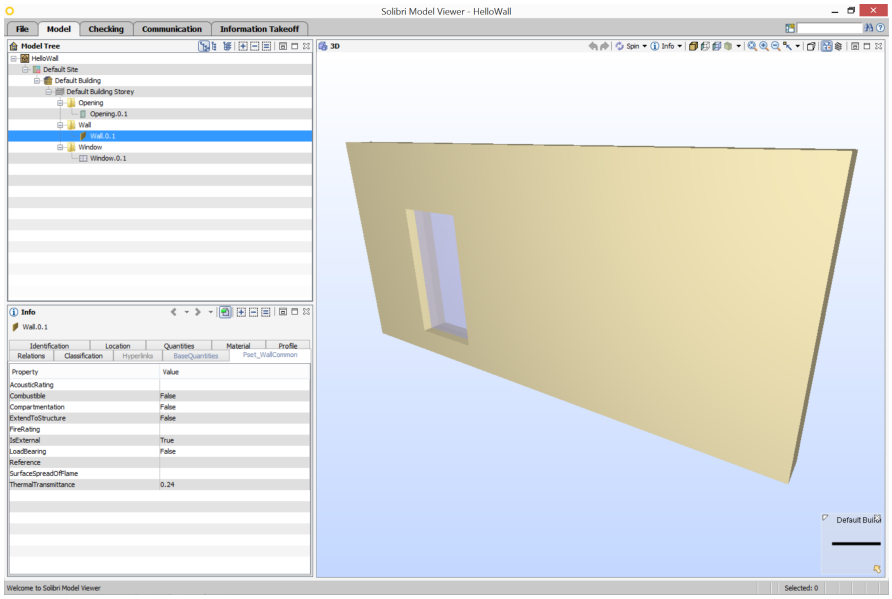


Fig. 6.31 Example model HelloWall.ifc

6.10 Example: HelloWall.ifc

The following section uses a simple example of a wall with a window to show how a building is modeled using the IFC and saved in the file *HelloWall.ifc*[2]. Figure 6.31 shows the example model in an IFC viewer. The IFC file is saved in the alphanumeric file format defined in part 21 of the STEP standard ISO 10303-21. An IFC file is structured in two sections: (1) a HEADER section with information about the file, and (2) a DATA section with the project information. Internal file object identifiers are denoted in the STEP21 file format by a natural number prefixed by a #-sign.

The first line denotes that the physical file adheres to the format defined in STEP-Standard ISO 10303 Part 21. The HEADER section follows immediately thereafter. The file description (FILE_DESCRIPTION) indicates the model view definition to which the IFC file complies (see also Chap. 7), in this case the Coordination View with additional elements according to the Quantity Take-off view. The entry FILE_NAME specifies the file name, the creation time of the file, the file creator and the organization to which the creator belongs, the name of the application, and the name of the authorizing user. Finally, the version of the IFC schema is specified, in this case Version IFC 2x3.

```
ISO-10303-21;
HEADER;
```

[2] The example is available online from: http://www.buildingsmart-tech.org/implementation/get-started/hello-world/example-1

```
FILE_DESCRIPTION (('ViewDefinition [CoordinationView,
QuantityTakeOffAddOnView]'), '2;1');
FILE_NAME ('HelloWall.ifc', '2014-10-20T17:02:56',
    ('Architect'), ('Building Designer Office'), 'My IFC tool',
    'My IFC tool', 'Simon Sample);
FILE_SCHEMA (('IFC2X3'));
ENDSEC;
```

The file data section follows the header and contains information about the project. To begin with the IFC project (#1) is given a globally unique identifier (0YvctVUKr0kugbFTf53O9L) as the root element in an IFC exchange file for the coordination view. In addition, details on the past user history (#2), the basic units (#7 - #19) and the geometric representation contexts for the shape representations in the file (#20 - #22) are given, including precision factor (0.00001) and the relative placement point (0,0,0).

```
DATA;
#1 = IFCPROJECT('0YvctVUKr0kugbFTf53O9L', #2, 'Default Project',
     'Description of Default Project', $, $, $, (#20), #7);
#2 = IFCOWNERHISTORY(#3, #6, $, .ADDED., $, $, $, 1217620436);
#3 = IFCPERSONANDORGANIZATION(#4, #5, $);
#4 = IFCPERSON('ID001', 'Sample', 'Simon', $, $, $, $, $);
#5 = IFCORGANIZATION($, 'MF', 'Testco', $, $);
#6 = IFCAPPLICATION(#5, '0.10', 'My IFC tool', 'TA 1001');
#7 = IFCUNITASSIGNMENT((#8, #9, #10, #11, #15, #16, #17, #18,
     #19));
...
#11 = IFCCONVERSIONBASEDUNIT(#12, .PLANEANGLEUNIT., 'DEGREE',
      #13);
#12 = IFCDIMENSIONALEXPONENTS(0, 0, 0, 0, 0, 0, 0);
#13 = IFCMEASUREWITHUNIT(IFCPLANEANGLEMEASURE(1.745E-2), #14);
...
#20 = IFCGEOMETRICREPRESENTATIONCONTEXT($, 'Model', 3, 1.000E-5,
      #21, $);
#21 = IFCAXIS2PLACEMENT3D(#22, $, $);
#22 = %IFCCARTESIANPOINT((0., 0., 0.));
```

The next section defines the project structure. This example shows a three level project structure, created by exactly one site (#23), one building (#29), and one building story (#35). The position of the building site is given within the global coordinate system located at 24° 28′ 0″ north, 54° 25′ 0″ west. The local coordinate system of the building site is positioned at the origin (0.0, 0.0, 0.0) with no rotation (#24 - #28). Both the building and the building story are positioned relative to the building site with no rotation or offset (#30 - #34 and #36 - #40).

```
#23 = IFCSITE('3rNg_N55v4CRBpQVbZJoHB', #2, 'Default Site',
      'Description of Default Site', $, #24, $, $, .ELEMENT.,
      (24, 28, 0), (54, 25, 0), $, $, $);
#24 = IFCLOCALPLACEMENT($, #25);
#25 = IFCAXIS2PLACEMENT3D(#26, #27, #28);
#26 = IFCCARTESIANPOINT((0., 0., 0.));
#27 = IFCDIRECTION((0., 0., 1.));
#28 = IFCDIRECTION((1., 0., 0.));
```

```
#29 = IFCBUILDING('0yf_M5JZv9QQXly4dq_zvI', #2,
      'Default Building', 'Description of Default Building',
      $, #30, $, $, .ELEMENT., $, $, $);
#30 = IFCLOCALPLACEMENT(#24, #31);
#31 = IFCAXIS2PLACEMENT3D(#32, #33, #34);
#32 = IFCCARTESIANPOINT((0., 0., 0.));
#33 = IFCDIRECTION((0., 0., 1.));
#34 = IFCDIRECTION((1., 0., 0.));
#35 = IFCBUILDINGSTOREY('0C87kaqBXF$xpGmTZ7zxN$', #2,
      'Default Building Storey',
      'Description of Default Building Storey', $, #36, $, $,
      .ELEMENT., 0.);
#36 = IFCLOCALPLACEMENT(#30, #37);
#37 = IFCAXIS2PLACEMENT3D(#38, #39, #40);
#38 = IFCCARTESIANPOINT((0., 0., 0.));
#39 = IFCDIRECTION((0., 0., 1.));
#40 = IFCDIRECTION((1., 0., 0.));
```

The section that follows defines the project structure hierarchy of the project levels defined above by placing them in an aggregation relationship (see also Chap. 3). The building (#29) comprises one building story (#35), one building (#29), the building site (#23) and the project (#1). A hierarchy of spatial relationships is likewise defined (#44) within which the wall (#45) and window (#124) are assigned to the building story (#35), in this case the only one in this model.

```
#41 = IFCRELAGGREGATES('2168U9nPH5xB3UpDx_uK11', #2,
      'BuildingContainer', 'Container for BuildingStories',
      #29, (#35));
#42 = IFCRELAGGREGATES('3JuhmQJDj9xPnAnWoNb94X', #2,
      'SiteContainer', 'Container for Buildings', #23, (#29));
#43 = IFCRELAGGREGATES('1Nl_BIjGLBke9u_6U3IWlW', #2,
      'ProjectContainer', 'Container for Sites', #1, (#23));
#44 = IFCRELCONTAINEDINSPATIALSTRUCTURE('2O_dMuDnr1Ahv28oR6ZVpr',
      #2, 'Default Building', 'Contents of Building Storey',
      (#45, #124), #35);
```

The section that follows defines the creation of the actual wall object of type *IfcWallStandardCase* (#45) positioned relative to the building story (#46 points to #36). Two different geometric representations are defined for the wall (#51). The wall axis is defined as a two-dimensional curve (#79) comprised of a polyline (#80) from (0.0, 0.15) to (5.0, 0.15), and the three-dimensional volumetric solid is defined as a "SweptSolid" (#83, #84). This solid is the product of the extrusion of the footprint (#85) described by a closed polyline (#86). The extrusion is in the vertical direction (0.0, 0.0, 1.0) (#96) with a height of 2.30 meters (#84).

```
#45 = IFCWALLSTANDARDCASE('3vB2YO$MX4xv5uCqZZG05x', #2,
      'Wall xyz', 'Description of Wall', $, #46, #51, $);
#46 = IFCLOCALPLACEMENT(#36, #47);
#47 = IFCAXIS2PLACEMENT3D(#48, #49, #50);
#48 = IFCCARTESIANPOINT((0., 0., 0.));
#49 = IFCDIRECTION((0., 0., 1.));
#50 = IFCDIRECTION((1., 0., 0.));
#51 = IFCPRODUCTDEFINITIONSHAPE($, $, (#79, #83));
```

```
#79 = IFCSHAPEREPRESENTATION(#20, 'Axis', 'Curve2D', (#80));
#80 = IFCPOLYLINE((#81, #82));
#81 = IFCCARTESIANPOINT((0., 1.500E-1));
#82 = IFCCARTESIANPOINT((5., 1.500E-1));
#83 = IFCSHAPEREPRESENTATION(#20, 'Body', 'SweptSolid', (#84));
#84 = IFCEXTRUDEDAREASOLID(#85, #92, #96, 2.300);
#85 = IFCARBITRARYCLOSEDPROFILEDEF(.AREA., $, #86);
#86 = IFCPOLYLINE((#87, #88, #89, #90, #91));
#87 = IFCCARTESIANPOINT((0., 0.));
#88 = IFCCARTESIANPOINT((0., 3.000E-1));
#89 = IFCCARTESIANPOINT((5., 3.000E-1));
#90 = IFCCARTESIANPOINT((5., 0.));
#91 = IFCCARTESIANPOINT((0., 0.));
#92 = IFCAXIS2PLACEMENT3D(#93, #94, #95);
#93 = IFCCARTESIANPOINT((0., 0., 0.));
#94 = IFCDIRECTION((0., 0., 1.));
#95 = IFCDIRECTION((1., 0., 0.));
#96 = IFCDIRECTION((0., 0., 1.));
```

The next section defines the wall layers and their materials. The wall in the example has a single layer (#77) of thickness 0.3 meters made of the material "Reinforced concrete C30/37". This material layer is placed in the middle of the wall axis (#79) as expressed by the negative offset of -0.15 meters (#75).

```
#74 = IFCRELASSOCIATESMATERIAL('2zeggBjk9A5wcc3k9CYqdL', #2, $,
      $, (#45), #75);
#75 = IFCMATERIALLAYERSETUSAGE(#76, .AXIS2., .POSITIVE.,
      -1.500E-1);
#76 = IFCMATERIALLAYERSET((#77), $);
#77 = IFCMATERIALLAYER(#78, 3.000E-1, $);
#78 = IFCMATERIAL('Reinforced concrete C30/37');
```

The definition of alphanumeric properties such as dimensions and quantity information follows. For these a property set (*IfcPropertySet*, #52) and an element quantity set (*IfcElementQuantity*, #64) are defined and attached to the wall by means of relationship objects (*IfcRelDefinesByProperties*, #63 and #73). Lines #53 to #63 define properties such as "ThermalTransmittance" (#58) while lines #65 to #72 specify values for measurements and quantities such as the gross volume (#69). The names "Pset_WallCommon" and "BaseQuantities" indicates that these properties and quantities information are defined as part of the IFC specification.

```
#52 = IFCPROPERTYSET('18RtPv6efDwuUOMduCZ7rH', #2,
      'Pset_WallCommon', $, (#53, #54, #55, #56, #57, #58,
      #59, #60, #61, #62));
...
#58 = IFCPROPERTYSINGLEVALUE('ThermalTransmittance',
      'ThermalTransmittance', IFCREAL(2.400E-1), $);
...
#61 = IFCPROPERTYSINGLEVALUE('LoadBearing', 'LoadBearing',
      IFCBOOLEAN(.F.), $);
...
#63 = IFCRELDEFINESBYPROPERTIES('3IxFuNHRvBDfMT6_FiWPEz', #2, $,
      $, (#45), #52);
```

```
#64 = IFCELEMENTQUANTITY('10m6qcXSj0Iu4RVOK1omPJ', #2,
      'BaseQuantities', $, $,
      (#65, #66, #67, #68, #69, #70, #71, #72));
#65 = IFCQUANTITYLENGTH('Width', 'Width', $, 3.000E-1);
#66 = IFCQUANTITYLENGTH('Length', 'Length', $, 5.);
...
#69 = IFCQUANTITYVOLUME('GrossVolume', 'GrossVolume', $, 3.450);
...
#73 = IFCRELDEFINESBYPROPERTIES('0cpLgxVi9Ew8B08wF2Ql1w', #2, $,
      $, (#45), #64);
```

The next section defines the creation of an opening object of type *IfcOpeningElement* (#97) relative to the local coordinate system of the wall (#98 points to #46). A geometric representation (#103) is defined for the opening object as a three-dimensional "SweptSolid" (#110, #111) and the opening object (#97) is related via *IfcRelVoidsElement* (#109) to the wall (#45), indicating that the opening is to be subtracted from the wall.

```
#97 = IFCOPENINGELEMENT('2LcE70iQb51PEZynawyvuT', #2,
      'Opening Element xyz', 'Description of Opening', $,
      #98, #103, $);
#98 = IFCLOCALPLACEMENT(#46, #99);
#99 = IFCAXIS2PLACEMENT3D(#100, #101, #102);
#100 = IFCCARTESIANPOINT((9.000E-1, 0., 2.500E-1));
#101 = IFCDIRECTION((0., 0., 1.));
#102 = IFCDIRECTION((1., 0., 0.));
#103 = IFCPRODUCTDEFINITIONSHAPE($, $, (#110));
#109 = IFCRELVOIDSELEMENT('3lR5koIT51Kwudkm5eIoTu', #2, $, $,
       #45, #97);
#110 = IFCSHAPEREPRESENTATION(#20, 'Body', 'SweptSolid',
       (#111));
#111 = IFCEXTRUDEDAREASOLID(#112, #119, #123, 1.400);
#112 = IFCARBITRARYCLOSEDPROFILEDEF(.AREA., $, #113);
...
```

Here too a set of measurements and quantities is defined (#104) and associated with the opening object (#97) by means of a relation #108.

```
#104 = IFCELEMENTQUANTITY('2yDPSWYWf319fWaWWvPxwA', #2,
       'BaseQuantities', $, $, (#105, #106, #107));
#105 = IFCQUANTITYLENGTH('Depth', 'Depth', $, 3.000E-1);
#106 = IFCQUANTITYLENGTH('Height', 'Height', $, 1.400);
#107 = IFCQUANTITYLENGTH('Width', 'Width', $, 7.500E-1);
#108 = IFCRELDEFINESBYPROPERTIES('2UEO1blXL9sPmb1AMeW7Ax', #2,
       $, $, (#97), #104);
```

Finally, the creation of the window object of type *IfcWindow* (#124) is defined and positioned relative to the local coordinate system of the opening (#125 points to #98). A three-dimensional geometric representation (#130) is defined for the object as a "SweptSolid" (#150, #151). This solid is created by extruding the footprint (#152) described as a closed polyline (#153). The window object (#124) is given a *IfcRelFillsElement* (#131) relationship to the opening (#97), indicating that the opening is to be filled with the window.

```
#124 = IFCWINDOW('0LV8Pid0X3IA3jJLVDPidY', #2, 'Window xyz',
       'Description of Window', $, #125, #130, $, 1.400,
       7.500E-1);
#125 = IFCLOCALPLACEMENT(#98, #126);
#126 = IFCAXIS2PLACEMENT3D(#127, #128, #129);
#127 = IFCCARTESIANPOINT((0., 1.000E-1, 0.));
#128 = IFCDIRECTION((0., 0., 1.));
#129 = IFCDIRECTION((1., 0., 0.));
#130 = IFCPRODUCTDEFINITIONSHAPE($, $, (#150));
#131 = IFCRELFILLSELEMENT('1CDlLMVMv1qw1giUXpQgxI', #2, $, $,
       #97, #124);
#150 = IFCSHAPEREPRESENTATION(#20, 'Body', 'SweptSolid',
       (#151));
#151 = IFCEXTRUDEDAREASOLID(#152, #159, #163, 1.400);
#152 = IFCARBITRARYCLOSEDPROFILEDEF(.AREA., $, #153);
#153 = IFCPOLYLINE((#154, #155, #156, #157, #158));
...
```

As with the wall above, alphanumeric properties (#132) and quantities and measurements (#146) are defined for the window and related to it via the relationship objects (*IfcRelDefinesByProperties*, #145 and #149). Lines #133 to #144 specify properties and values, such as "ThermalTransmittance" (#139) while lines #147 and #148 define measurement values, in this case the height (#147) and breadth (#148) of the window.

The penultimate line marks the end of the project data section (DATA) of the IFC file and the final line the end of the entire ISO standard file.

```
#132 = IFCPROPERTYSET('0fhz_bHU54xB$tXHjHPUZl', #2,
       'Pset_WindowCommon', $, (#133, #134, #135, #136, #137,
       #138, #139, #140, #141, #142, #143, #144));
...
#139 = IFCPROPERTYSINGLEVALUE('ThermalTransmittance',
       'ThermalTransmittance', IFCREAL(2.400E-1), $);
...
#145 = IFCRELDEFINESBYPROPERTIES('2fHMxamlj5DvGvEKfCk8nj', #2,
       $, $, (#124), #132);
#146 = IFCELEMENTQUANTITY('0bB_7AP5v5OBZ90TDvo0Fo', #2,
      'BaseQuantities', $, $, (#147, #148));
#147 = IFCQUANTITYLENGTH('Height', 'Height', $, 1.400);
#148 = IFCQUANTITYLENGTH('Width', 'Width', $, 7.500E-1);
#149 = IFCRELDEFINESBYPROPERTIES('0FmgI0DRX49OXL_$Wa2P1E', #2,
       $, $, (#124), #146);$
ENDSEC;
END-ISO-10303-21;
```

6.11 ifcXML

The descriptive language of the IFC schema is EXPRESS (ISO 10303-11, 2004), a data modeling language specially developed for product modeling. As mentioned

earlier, the accompanying exchange format for model instances is defined in part 21 of the STEP specification. When the IFC was first developed, the XML format (developed by W3C, 2015) which is now very popular, was not available.

From the early 2000s onwards the eXtensible Markup Language XML, a simpler and optimized version of the SGML standard, began to gain popularity. Many development tools were introduced and XML became a mainstream language for formally describing structured data.

As a consequence, buildingSMART were asked to also provide IFC data in XML format. From 2001 onwards, a number of different approaches to translating the EXPRESS schema into an XML compatible form were developed as a means of creating valid IFC XML documents:

- 2001 – the first version of an XML translation of the IFC 2.0 schema as an XDR (XML Data Reduced) schema definition. The translation rule from EXPRESS to XDR was a private development.
- 2002 – the first version of an XML translation of the IFC 2.0 schema as an XSD (XML Schema Definition). Here too the translation rule from EXPRESS to XSD was a private development that was later adopted as a proposal by the ISO Group ISO/TC 184/SC 4 for a general standard for mapping EXPRESS to XSD.
- 2005 – a new method for XML translation of the IFX2x2 schema according to the developmental stage (working draft) of the ISO 10303:28-ed2 standard which was developed for the standard-compliant translation of EXPRESS to XSD. A default configuration was chosen which, however, led to very large XML data files.
- 2007 – the same methodology, this time for the IFC2x3 schema.
- 2013 – a newly developed version of ifcXML was developed as part of the development of IFC4 in which the transfer from IFC EXPRESS to XSD is compliant to the final version of ISO 10303-28:ed2 using an optimized configuration of the mapping rules. XSD definitions were given alongside the EXPRESS definition in the official IFC4 documentation. The new configuration of the ISO 10303-28:ed2 rules is much more efficient and this method is often known as Simple ifcXML.

As a rule, the XML serialization of IFC data has exactly the same depth of information as the Part-21 serialization. IFC XML data can be validated against the online ifcXML XSD schema. Only detailed validation against the validation rules available in EXPRESS is not possible as the scope of the XSD language is not sufficient to translate these rules. Another limitation is that inverse attributes are not included in the ifcXML schema.

Due to the additional XML syntax, ifcXML files are significantly larger than a regular ifc file for the same information content. In the earlier ifcXML conversions (up to IFC2x3), ifcXML files were typically 6-8 times larger than an ifc file, but with the newer simple ifcXML convention in IFC4 they are now approximately 2-3 times larger.

6.12 Summary

The IFC data model is an open, mature and internationally standardized data model. It permits the exchange of digital building models beyond the limits of functionality of individual applications and between various software vendors and supports a diverse range of application scenarios.

With the IFC data model it is possible to model buildings digitally in great detail including the comprehensive semantic description of a building, the modeling of all building elements and spaces as well as the reciprocal relationships between them. Each semantic building object can have one or more geometric representations associated with it, making it possible to cater for the different needs for presenting building information geometrically.

The IFC data model is an extremely powerful and also very complex data model. That has the advantage that buildings can be described very completely and in different ways. But it also has disadvantages. For example, different planning stages may require different geometric representations, for example a surface model or a finite element net, each of which can be modeled differently. A typical stumbling block is the modeling of a continuous external wall as opposed to story-wise in individual sections. Both variants are possible, and even sensible for different application scenarios, but they can rarely be derived from one another or described parallel to one another.

This complexity requires a considerable effort for software vendors that wish to make their products compatible with the IFC standard. Many software vendors therefore only offer partial support for the data model in their import and export modules. To avoid incompatibilities as a result of this, buildingSMART introduced the concept of Model View Definitions (MVD, described in Chap. 7), with which it is possible to specify which parts of the IFC data model must be implemented for specific data exchange scenarios. Accordingly, MVDs are also the basis for certifying IFC compatibility: software products are not certified for the entire data schema but only for specifically defined sections.

Despite the formal mechanisms of the data scheme and the MVDs, the model's flexibility is still so complex that further agreements are necessary to achieve homogeneous and compatible implementations. These so-called "Implementers' Agreements" can contain extensive sets of agreements, but are increasingly being described in semi-automated test procedures and therefore becoming part of the certification of software products. This is expected to lead to further improvements in the quality and reliability of IFC data, as described in detail in Chap. 8.

Despite the complexity of the data model and the problems this brings with it, the IFC data format plays a key role in the path towards Big Open BIM. On the one hand, a neutral, open format is the only way to ensure vendor-neutrality and true, sustainable data continuity. And on the other hand, rules governing the provision of BIM models must specify an open format in order to avoid skewing the competition in favor of specific software vendors. Last but not least, the usefulness of the long-term archiving of digital data from the monitoring of a building's operation can only be reliably guaranteed if this information is available in an open, well-documented

format that is not dependent on an individual manufacturer's specific format. Similar attempts to make data available in the long term in a vendor-neutral format can be observed in other industrial sectors such as the automotive industry and in aerospace technology. As a consequence, some national organizations have decided to specify the use of the IFC format for public building projects, including public authorities in Singapore, the Netherlands and Finland. Similar developments are expected to follow in the near future in the USA, Great Britain and in the Scandinavian countries. In the long term, therefore, one can expect to see the IFC standard play an important role in the search for a legally-binding digital equivalent to paper-based, rubber-stamped and hand-signed planning documents at a national and European level.

The standardization organization buildingSMART offers all its members, whether individuals, companies or public authorities and organizations, extensive opportunities to participate and contribute to the IFC, and with it numerous opportunities to influence the quality and future development of the standards.

Among these future developments, outlined in part in the Technical Committee's "Roadmap 2020", are ways of integrating complementary standards and models such as the IDM/MVD (see Chap. 7), bSDD (see Chap. 9) and BCF (see Chap. 14) as well as improvements to the quality of implementation through more stringent certification procedures (see Chap. 8). Further developments are also underway in the field of extending geometric representations, for example through the support of point clouds, the improved support of model servers and the dynamization of semantic extensions and distributed instance models using Semantic Web Technologies such as the Resource Description Framework (RDF). To improve model consistency, it would be desirable in the long term to parametrize objects to remedy the currently lacking connection between an attribute (such as the "OverallWidth" of a door) and its geometric representation. Similarly, links to existing standards from the field of Geo-information (CityGML, LandXML, etc.) as well as model extensions for infrastructure objects such as bridges and tunnels or streets and railway tracks are currently being actively developed.

References

BuildingSMART (2013). *Industry Foundation Classes, Version 4, Documentation*. Retrieved from http://www.buildingsmart-tech.org/ifc/IFC4/final/html/ (Accessed on 13.01.2015)

BuildingSMART (2013a). *IFC and related data model standards documentation*. Retrieved from http://www.buildingsmart-tech.org/ (Accessed on 26.09.2017)

Eastman, C. (1999). *Building Product Models: Computer Environments Supporting Design and Construction*. CRC Press.

Laakso, M., & Kiviniemi, A. (2012). The IFC Standard – A Review of History, Development and Standardization. *ITcon Journal of Information Technology in Construction, 17*, pp 134-161. Retrieved from http://www.itcon.org/data/works/att/2012_9.content.01913.pdf (Accessed on 13.01.2015)

Liebich, T. (2009). *IFC 2x Edition 3 Model Implementation Guide*. Retrieved from http://www.buildingsmart-tech.org/implementation/ifc-implementation/ifc-impl-guide (Accessed on

14.01.2015)
Schenck, D. A., & Wilson, P. R. (1993). *Information Modeling the EXPRESS Way*, Oxford University Press.
ISO 10303-11 (2004). *Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual*. International Organization for Standardization, Geneva, Switzerland.
ISO 10303-21 (2002). *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure, International Organization for Standardization*, Geneva, Switzerland.
ISO 16739 (2013). *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*. International Organization for Standardization, Geneva, Switzerland.
ISO/IEC 19775-1 (2004). *Information technology – Computer graphics and image processing – Extensible 3D (X3D)*. International Organization for Standardization, Geneva, Switzerland.
W3C (2015). *XML Standard, Word Wide Web Consortium (W3C)*. Retrieved from http://www.w3.org/standards/xml/ (Accessed on 22.01.2015)
Weise, M., Liebich, T., See, R., Bazjanac, V., & Laine, T. (2009). *IFC Implementation Guide: Space Boundaries for Thermal Analysis*. BuildingSMART. Retrieved from http://www.buildingsmart-tech.org/downloads/accompanying-documents/agreements (Accessed on 11.11.2014)

Index