Chapter 2
Ontological Analysis and Engineering Standards: an initial study of IFC

Stefano Borgo[1], Emilio M. Sanfilippo[1,2], Aleksandra Šojić[2], and Walter Terkaj[2]

Abstract There is an increasing interest in developing ontological versions of engineering standards. In general, this amounts to restating a given standard in some ontological language like OWL. We observe that without an ontological analysis of the standard, the conversion neither improves the clarity of the standard nor facilitates its coherent application. In this chapter we begin to study the Industry Foundation Classes (IFC), a standard providing an open vendor-independent file format and data model for data interoperability and exchange for Architecture/Engineering/Construction and Facility Management. We first look at IFC and at an existing OWL version of IFC; then we highlight the implicit assumptions and we apply ontological analysis to discuss how to best grasp the type/occurrence distinction in IFC. The goal is to show what has been done in IFC and the contribution of ontological analysis to help increasing the correct understanding of a standard. With this approach, we reach a deeper understanding, which can guide the translation from the original language to OWL with increased conceptual clarity while ensuring both logical coherence and ontological soundness.

Key words: IFC, Ontological analysis, Type, Occurrence, Class, Instance.

2.1 Introduction

Nowadays Information and Communication Technologies (ICT) play a central role in supporting various engineering tasks in the field of manufacturing, building and construction industry. Computer Aided technologies (CAx) are considered "one of

[1] Istituto di Scienze e Tecnologie della Cognizione (ISTC) CNR, Trento, Italy
stefano.borgo@loa.istc.cnr.it
[2] Istituto Tecnologie Industriali e Automazione (ITIA) CNR, Milano, Italy
{emilio.sanfilippo, aleksandra.sojic, walter.terkaj}@itia.cnr.it
phone: +39 0461 314873

the greatest technologies of the 20th century" [53]. Nevertheless, the use of heterogeneous application tools to support industrial activities, ranging from design to monitoring, and the lack of a common conceptualization of the terms used by various actors across different communities, as well as the lack of formal representations, threaten the quality of process and product modelling as well as the effective sharing of data between the stakeholders [53, 59]. In this context, information modelling standards and computational ontologies furnish innovative technologies and methodologies to overcome modelling and interoperability problems [44].

In this chapter, which is based on [11], we focus on the Industry Foundation Classes (IFC) [14], an information modelling standard which, according to the U.S. National Building Information Modeling Standard [41], is the most mature and widespread schema for the building industry. IFC is based on the data modeling language EXPRESS [47], which has no formal semantics and whose application scenario are not related to Semantic Web (SW) technologies. For this reasons, different communities have proposed translations of IFC in ontology-driven languages like the Web Ontology Language (OWL) [62]. Nevertheless, the development of these versions of IFC in some ontological format has not been supported by a methodology for ontological analysis. A systematic evaluation that would include a concise analysis of these various approaches to the specification of IFC is still missing. Actually, to the best of our knowledge no such analysis has been proposed so far. This problem is common across standards, for example see the case of BPMN [46], and there are events and initiatives aiming to improve this situation[1]. Indeed, we can say that today the research community is focusing more on *transcriptions* of standards into ontology languages than on their *translations* in ontology. A transcription focuses on rewriting the constraints into a new language while a translation goes beyond, it aims to characterize the intended interpretations of the theory obtained with the transcription. Ontology aims at making explicit, and possibly formalizing, the implicit commitments and conceptualizations of reality that justify a certain terminology or information system. In this way, the captured information can be reliably distributed for automated information sharing and exploitation. Without a careful ontological analysis of the source, this goal is hard to achieve and the notions on which a standard relies cannot be reliably aligned for automated information sharing and exploitation.

In the following sections we focus on existing techniques to construct an IFC-driven ontological system and we apply ontological analysis to understand and possibly clarify important aspects of IFC, namely the *type/occurrence* distinction and the notion of *property*. Although these elements are just a fragment of IFC, they drive the whole structure of the standard. For this reason it is particularly important to evaluate how to best model these parts in OWL and how to correctly understand them. In Section 2.2, we introduce basic notions of applied ontology. IFC and its general structure is presented in Section 2.3; Section 2.4 reviews the state of the art about building OWL versions of IFC and discusses the basic conversion patterns. Section 2.5 presents an ontological analysis of the IFC type/occurrence distinction,

[1] `http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologyBasedStandards`.

and Section 2.6 continues with an analysis of the different types of properties that are used in IFC. We conclude with a few remarks on other aspects of the use of IFC in which our results can contribute, and highlight some advantages of using a well-structured OWL version of the standard.

2.2 Ontology and ontological analysis

Ontologies, in the sense of artefacts that capture a conceptualization of the world, have attracted increasing interest since the 1990s when they started to be developed as tools to aid modeling and to facilitate information sharing and integration. During the same period, the search for sound techniques to build ontologies and to foster interaction across different ontologies, led to the study of a series of methodologies that are collected under the label *ontological analysis*, and which help to properly study the domain of interest in order to coherently develop its models, and to reliably link different representation frameworks. The research area that comprises the development of ontologies, and the methodologies for their construction and application is called *applied ontology*.

Today, applied ontology provides the tools to overcome the limitations of traditional approaches in knowledge representation and reasoning as provided by the graphical-based languages (like BPMN[2]) and expert systems [20]. To foster flexibility and reusability of existing models, applied ontology can guide the analysis of the underlying frameworks, e.g. see [51, 46], help to select the suitable representation language [6], and structure the reconstruction of the systems [34, 9]. An ontologized version of an existing system is generally clearer than the original, unambiguous, ideally even self-explicative for humans and machines, and suitable for automated deduction, i.e., for the exploitation via software systems.

Since ontological analysis can be complex, sometimes a community develops ontologized versions of their models via the transcription of the framework constraints into an ontological language without a systematic study of its ontological assumptions about the domain. Ontological languages give the advantage to enforce and verify the consistent use of important distinctions like class vs. instance. The former refers to collections of entities sharing the same set of attributes, while the latter denotes the specific entity, aka token or particular, which does not classify itself but might be classified by one or more classes: see for instance the T-Box and A-Box distinction in Description Logics (DL) [1]. By modelling a domain of discourse in terms of classes and instances, the ontologized system can be turned into a modern *Knowledge Base* that keeps apart the vocabulary of an application domain on the one side, and assertions about named individuals on the other side [1]. While the process of building the transcription gives the possibility to discover missing constraints or implicit general patterns, and allows exploiting the system via software tools, it does not *per se* ensure conceptual coherence nor semantic clarity. Ambiguities might go

[2] `http://www.bpmn.org/`

undetected and implicit assumptions on the domain remain undisclosed unless one focuses on the very viewpoint that justifies that modeling system [52]. In short, we cannot take full advantage of ontological modeling without going through ontological analysis with the goal of isolating and clarifying the basic concepts and relations on which the information system at stake relies.

Nowadays, there are different types of ontologies, so that we need to be clear about what we consider important in this context and in our analysis of IFC. We are interested in the ontologies that are called *formal* and *foundational*. Roughly speaking, an ontology is formal if it is expressed in a logic language endowed with formal semantics, for instance in model-theoretic terms à la Tarski as for first-order predicate logic [27] and OWL. Formal semantics gives two crucial advantages: it ensures semantic transparency, thus clarity and lack of ambiguity, and enables the use of automatic deduction. By foundational ontologies we mean those ontological systems that focus on very general concepts like object, event, quality, and basic relations such as participation, dependence, parthood. Often the term *formal ontology* is used to cover both of the above requirements. In this specific meaning, formal ontology is the study of the interconnections between entities, properties, parts, wholes and collectives which are considered to be "formal" because they can be exemplified by objects in all domains of reality [49]. To restate this position from another perspective, one can say that formal ontology is the study of logical systems which are: *general*, since they include the most usable and widely applicable concepts; *reliable*, as they are logical theories with clear semantics, rich axiomatizations and carefully analysed formal consequences (theorems); and *well organized*, because they are based on philosophical principles, the choice of which is explicitly motivated and remains independent from particular domains.

Thus, from the ontological viewpoint this chapter aims to show that the transcription in an ontology language of an engineering standard can be improved via the application of ontological analysis, and that by performing this operation within a suitable foundational ontology we further secure interoperability.

2.2.1 The DOLCE *foundational ontology*

DOLCE – the *Descriptive Ontology for Linguistic and Cognitive Engineering* [39, 10] – is the foundational ontology that we use in the analysis of the IFC engineering standard. DOLCE was introduced in 2002 and since then it has been used in areas like enterprise modeling, financial transactions, medicine, geographic information systems, and the semantic web. Here we provide a brief introduction to the elements of the ontology that are of interest in this work, the reader can find in [39, 10] motivations and technical aspects.

The DOLCE taxonomy of categories is depicted in Figure 2.1.

The DOLCE categories are related to each other via existence and dependence conditions, and the categorical structure is based on philosophical principles whose choice is explicitly motivated. DOLCE was intentionally built to be independent from

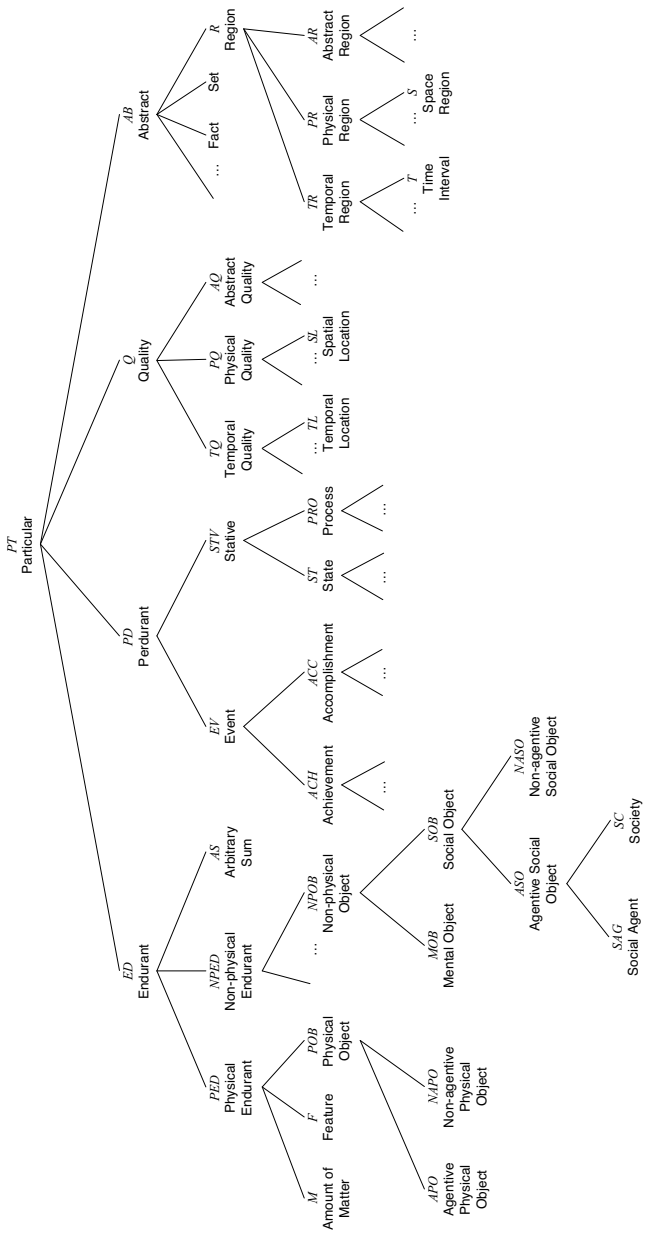

Figure 2.1 The taxonomy of the DOLCE foundational ontology.

applications but the viewpoint it embraces is especially apt for those domains that take a commonsense perspective; this makes it suitable for areas like engineering where we rely on the standard notion of object, classical physics and traditional views on causality.

The DOLCE ontology concentrates on the characterization of particulars as opposed to universals. Roughly speaking, a *universal* is an entity that is instantiated or concreted by other entities, like "being human", "being a car", and "being an operation". A *particular* is an entity that is not instantiated by other entities, like the Eiffel Tower in Paris. Your pen is a particular as opposed to the model of your pen, provided that the latter is interpreted as a type being instantiated by a number of entities among which is your pen. Particulars comprise physical and abstract objects as well as perdurants, like events, processes and the like, and qualities. The ontology provides a framework that is fairly natural in engineering. It adopts the distinction between objects like products and events like operations. It includes a differentiation among individual qualities (e.g., the weight of a specific material item), quality types (weight, color, and the like), quality spaces (spaces for weights, for colors, etc., where the individual qualities are compared and classified) and quality positions or qualia (informally, locations in quality spaces). These, together with measure spaces (where the quality positions get associated to a measure system and, thus, to numbers), are important to describe and compare products. Furthermore, DOLCE has been extended in order to ontologically characterize engineering functions, products and artefacts in general [7, 12, 13].

Regarding the main categories[3], ENDURANT (ED) collects objects, like a *person*, a *building* and a *machine*, and amounts of matter, like *a given amount of plastic*, while the category PERDURANT has as elements events, like *making a hole* or *assembling a product*, that is, things that happen in time. The term 'object' is used in the ontology to capture a notion of unity as suggested by the partition of the category PHYSICAL ENDURANT (formally, PED) into categories AMOUNT OF MATTER (M), FEATURE (F), that includes entities like *this bump of the road* and *the front of my house*, and PHYSICAL OBJECTS (POB). Both endurants and perdurants are associated with a group of individual qualities (elements of the category QUALITY). The exact list of qualities may depend on the entity. *Shape* and *weight* are usually taken as qualities of endurants, called physical qualities in DOLCE and denoted by means of the PQ predicate. *Duration* and *direction* are instead examples of qualities of perdurants. An individual quality is a quality associated with *one and only one* particular; it can be understood as the special way in which that entity instantiates the general property. For example, the weight of the car you are driving is an individual quality of that unique car and it instantiates the property "having weight" relative to that car.

Overall, DOLCE is a quite complex system and since in this chapter it furnishes just the background ontological framework, we can limit the presentation to these main categories; they will suffice for our purposes in the following sections.

[3] Foundational ontology is quite careful in its use of the terminology; the notion of category comes from the philosophical tradition and has the notion of class as its logical counterpart.

2.3 Industry Foundation Classes

The Industry Foundation Classes (IFC) [14] is a standard providing an open vendor-independent file format and data model for data interoperability and exchange for Architecture, Engineering, Construction and Facility Management (AEC/FM). It is released by buildingSMART International (formerly the International Alliance for Interoperability, IAI) and the current release IFC 4 is registered as ISO 16739. IFC supports interoperability across different applications used to design, construct and operate construction facilities by capturing information about all aspects of a building throughout its lifecycle [33, 59, 3, 44]. It is supported by most of the major CAD vendors [48] and its evolution is held by hundreds of companies [33, 5].

The development of IFC was motivated at its early stages by the slow and unresponsive evolution of the *Automation systems and integration - Product data representation and exchange* (ISO 10303) standard - informally known as STEP, *Standard for the Exchange of Product Model Data* [30] - to match the information modelling needs of the AEC/FM industry [37]. Although part of the team that developed STEP participated in the definition of the IFC data model [33, 37], there is no precise analysis on the relation between the two standards. IFC constitutes the core of the buildingSMART initiatives for interoperability, which include also the Information Delivery Manuals (IDM) and the International Framework for Dictionaries (IFD), as shown in Figure 2.2. The former is standardised as ISO 29481-1:2010 and provides a notation for the homogeneous representation of process information relative to building constructions, and the latter is a vocabulary of terms to be shared among IFC-driven building information models.

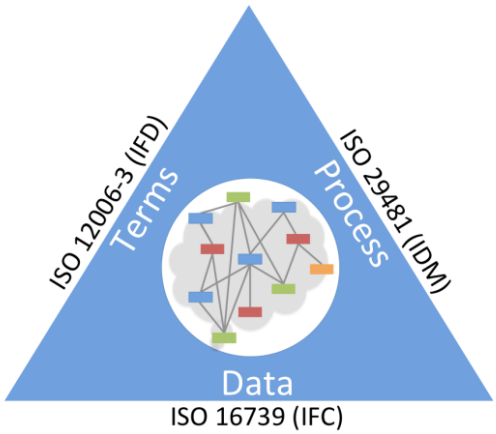

Figure 2.2 buildingSMART standards, from [37]

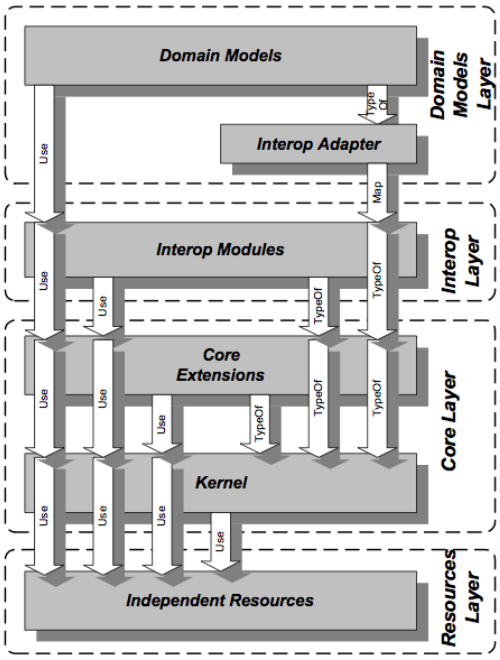

Figure 2.3 IFC conceptual layers, from [3]

The current IFC release is built on a modular structure that distinguishes among four conceptual layers, shown in Figure 2.3, which are tailored in turn in different schemas:

- Resource layer: it specifies classes that do not stand in taxonomical relationships with classes in the other layers; rather they can be linked by means of specific relationships. For instance, in the resource layer we have the schema *IfcGeometryResource* that contains classes needed to define geometric representations like `IfcSurface`, `IfcCartesianPoint`, and `IfcPlacement`. In this way, a material product –defined in the Core Layer by the class `IfcProduct` (see also below)– can be characterised as having a specific placement by the `ObjectPlacement` attribute that points to `IfcPlacement`;
- Core layer: it contains the most general classes of the IFC data model. Its purpose is to provide the most relevant structural classes and relationships of the IFC data model. It thus supports interoperability among the IFC layers and compatibility with the various IFC releases. The Core layer comprises two main schemas:
  1. *IfcKernel*, which collects the most general classes of the standard like, e.g., `IfcRoot`, `IfcObjectDefinition`, `IfcProcess`, `IfcContext`;
  2. *IfcCoreExtension*, which is further subdivided into three modules: *IfcControlExtension*, *IfcProcessExtension* and *IfcProductExtension*. These specialise the *IfcKernel* with AEC/FM concepts. In particular, the *IfcControlExtension*

contains classes for control objects like `IfcPerformanceHistory` and `IfcControl`; the *IfcProcessExtension* specifies classes for the representation of process-like entities, e.g. `IfcEvent`, `IfcProcedure`, `IfcTask`; the *IfcProductExtension* contributes to the specialisation of classes for product modelling like `IfcElement`, `IfcElementAssembly`, `IfcGrid`;

- Interoperability layer: it contains classes, defined in the Domain layer (see below), shared by multiple domains and used for inter-domain exchange and sharing of construction information. Amongst its schemas, it includes the *IfcSharedBuildingElements* and the *IfcSharedFacilitiesElements*. The former specialises the *IfcProductExtension* (Core) schema by classes used for representing building structures. Amongst its classes, it includes `IfcChimney`, `IfcDoor`, `IfcRamp`. The latter provides a set of classes concerning facilities management. Some of them specialise the *IfcProductExtension* schema (e.g. `IfcFurniture`, `IfcFurnitureType`), while others are attached directly under the *IfcKernel* (e.g. `IfcInventory`, `IfcOccupant`);
- Domain layer: it contains the most specific classes of IFC. The Domain layer organises classes according to industry disciplines and amongst its schemas it includes the *IfcArchitectureDomain* and the *IfcBuildingControlsDomain*. The former defines classes used in architecture including, e.g., `IfcDoorStyle`, `IfcWindowStyle`, `IfcWindowPanelProperties`. The latter supports the modelling of building automation, control, instrumentation and alarm. It includes classes like `IfcActuator`, `IfcAlarm` and `IfcSensor`.

The modular architecture operates on a so-called *gravity principle*, in the sense that at any layer, a class may refer to classes at the same or lower layer but cannot refer to classes from higher layers [44]. For instance, classes at the Core layer may refer to other Core classes, as well as to Resource layer classes, but cannot refer to classes within the Interoperability or Domain layers. The same principle applies also within the Core layer, in the sense that classes in schema *IfcKernel* cannot refer to those in *IfcCoreExtensions*, while the reverse is allowed.

The IFC data model is specified by means of EXPRESS [47], the dedicated formal language developed within STEP. EXPRESS is platform-independent and allows the taxonomical classification, via classes, of the domain entities that share certain attributes, thus distinguishing between classes and their members. The language, however, lacks a set-theoretical approach to semantics [5] and thus is expressively much weaker than modelling languages enriched with formal semantics (in the Tarskian sense) like OWL. In the EXPRESS language, an IFC concept is introduced via the **ENTITY** construct (the most important data type of EXPRESS) and classified with respect to other classes via the supertype/subtype partial ordering relation (**SUBTYPE OF/SUPERTYPE OF**). For instance, the class `IfcObjectDefinition` is the supertype of the classes `IfcObject`, `IfcTypeObject` and `IfcContext`. Some classes, e.g. `IfcObject`, are declared to be *abstract*: this means that they can only be instantiated through their subtypes. Inheritance is allowed along the hierarchy, subtypes inherit the attributes of their supertypes. Figure 2.4 shows an example of the taxonomical constraints in

EXPRESS. It shows the EXPRESS specification of the class `IfcObject` introduced in [14] as follows:

> "An IfcObject is the generalization of any semantically treated thing or process. Objects are things as they appear - i.e. occurrences. Examples of IfcObject include physically tangible items such as wall, beam or covering, physically existing items such as spaces, or conceptual items such as grids or virtual boundaries. It also stands for processes such as work tasks, for controls such as cost items, or for actors such as persons involved in the design process."

Looking at the specification, this class is the abstract supertype of `IfcActor`, `IfcControl`, `IfcGroup`, `IfcProcess`, `IfcProduct`, and `IfcResource`. These subclasses are mutually disjoint due to the use of the **ONEOF** construct. The relationship **SUBTYPE OF** states that `IfcObject` is subsumed under class `IfcObjectDefinition`. *ObjectType*, *IsDeclaredBy*, *Declares*, *IsTypedBy*, and *IsDefined* are attributes. Finally, in the definition the **SET OF ... FOR** construct specifies the **INVERSE** attributes by giving a collection of elements, possibly with indication of their minimum and maximum number, and the related attribute.

Following this brief overview of IFC and the introduction of ontology and some categories in Section 2.2, we now sketch basic rules for the translation of IFC into an ontological system and later analyze the ontological coherence of the result.

```
ENTITY IfcObject
  ABSTRACT SUPERTYPE OF(ONEOF(IfcActor, IfcControl, IfcGroup, IfcProcess, IfcProduct, IfcResource))
  SUBTYPE OF IfcObjectDefinition;
    ObjectType                    : OPTIONAL IfcLabel;
  INVERSE
    IsDeclaredBy                  : SET [0:1] OF IfcRelDefinesByObject FOR RelatedObjects;
    Declares                      : SET OF IfcRelDefinesByObject FOR RelatingObject;
    IsTypedBy                     : SET [0:1] OF IfcRelDefinesByType FOR RelatedObjects;
    IsDefinedBy                   : SET OF IfcRelDefinesByProperties FOR RelatedObjects;
END_ENTITY;
```

Figure 2.4 IfcObject specification in EXPRESS, from [14]

2.4 Ontologizing IFC

2.4.1 State of the art

Different groups have studied the formalization of IFC in OWL with different motivations which in some cases go beyond the scope of this paper. Technically, these formalizations cannot be classified as transcriptions nor as translations since the resulting systems can be semantically richer than IFC and yet lack the depth of a translation, at least in the sense discussed in Sections 2.1 and 2.2, as the formalization does not provide a full characterization of the class of models of IFC. In the following, we will use the neutral term *conversion* to address these formalizations.

Beetz and colleagues [4, 5], as well as Krima et al. [36, 2], draw attention to the EXPRESS's lack of formal semantics and claim that a logic-based language such as OWL is preferable for the capacity of axiomatic theories to better support knowledge representation and semantic data sharing. Additionally, Beetz et al. [5] emphasize that, apart from the STEP initiative, the popularity of EXPRESS among the engineering community is very limited and that the reuse of existing ontologies or tools for interoperability is often inhibited, especially those related to the Semantic Web initiative. In the paper, the authors explore a semiautomatic method for lifting EXPRESS schemas to OWL files. Barbau and colleagues developed the OntoSTEP model [2]. They specified the rules they use for the automated conversion from EXPRESS to OWL and implemented the system as a plug-in for Protégé [35]. Another conversion tool was presented by Pauwels and Deursen [43]. Schevers et al. [48], Barbau et. al [2], as well as Zhang et al. [63], argue that the conversion of IFC into OWL, beside enabling the exploitation of Semantic Web technologies for building information models, even facilitates the linkage between different IFC models and databases. In addition, Pauwels et al. [42] exploit the possibility of adding Semantic Web Rule Language (SWRL) [29] rules to enrich an OWL version of IFC and enable the use of reasoning engines.

Katranuschkov and colleagues [32] develop an ontology framework that includes domain-independent as well as domain-specific schemas, aimed at supporting data modelling and data sharing in civil engineering by reusing the IFC data model. However, differently from the other approaches, their library of ontologies is developed as an XML Schema (XSD).

Terkaj et al. [54] propose a modular OWL ontology for factory modelling and data sharing between heterogeneous and geographically distributed software tools. The main structure of the ontology, called Virtual Factory Data Model (VFDM), is based on IFC (mainly Resource and Core Layer, see Section 2.3) and its conversion into OWL was inspired by the methodology in [4, 5]; therefore, a relevant part of the VFDM can be applied also to other domains that are not directly related to factory and manufacturing. The ontology modules focused on manufacturing aim at modelling the products to be manufactured, the manufacturing processes, the manufacturing resources, and their inter-relations, as already addressed in previous works presenting a conceptual model based on UML class diagram [15] and a relational

database [16] to support the design of factories composed of Flexible Manufacturing Systems (FMS) [21, 24] and/or Focused Flexibility Manufacturing Systems (FFMS) [55, 56, 22, 23]. The VFDM can be exploited to enable interoperability between different software tools by developing VFDM-based plugins for I/O data exchange applicable both to commercial (e.g. Arena by Rockwell Automation [57], Plant Simulation by Siemens PLM [31]) and non-commercial (e.g. GIOVE Virtual Factory [60], OntoGUI [58]) applications. However, the software tools can effectively interoperate to support the design and management of a factory [17, 18] only if a proper software platform is realized to provide storage [40] and communication services [45, 19].

Our analysis of the literature shows that the ontological formalizations of IFC beyond EXPRESS are highly motivated, and to a large extent oriented by implementation goals. In this effort, significant attention is given to the conversion of the standard into OWL. We now discuss in more detail the underlying ideas of the proposed conversions looking at both practical and ontological aspects.

2.4.2 From EXPRESS to OWL

The conversion of EXPRESS schemas into OWL ontologies is largely documented in the literature [4, 5, 36, 2]. In this section, we introduce the main strategy for converting EXPRESS entities and their attributes, which will be the focus of our ontological analysis of IFC in Section 2.5. Other EXPRESS constructs, like data types and enumerations, will be considered in future works.

Research groups dealing with EXPRESS schemas and OWL ontologies agree to convert **ENTITY** into **owl:Class**, i.e., the OWL construct that specifies classes in general, including classes of particulars. The EXPRESS taxonomical relationship **SUPERTYPE OF/SUBTYPE OF**, holding between concepts classified by **ENTITY**, has its OWL equivalent in **rdfs:subClassOf**, which holds between **owl:Class** concepts. Differently from EXPRESS, OWL can reason with the stated constraints: if one declares a class C_1 to be the *subclass of* another class C_2, the inverse relationship (supertype) is automatically derived.

The **ABSTRACT** characterization in EXPRESS rules out direct instantiation of certain classes (see `IfcObject` in Section 2.3). This constraint does not have a counterpart in OWL where the corresponding structural pattern is obtained by declaring that the subclasses form a partition, i.e., that the OWL subclasses of the given class cover the extension of the concept. Analogously, the **ONEOF** construct is obtained by declaring that the OWL subclasses of that class are disjoint.

In EXPRESS the properties of the domain entities are modeled via attributes. By default, an attribute listed in the class definition is mandatory for each element of that class. When this is not so, the attribute is labeled **OPTIONAL**: if a class C_1 has optional attribute A_1, then it is a matter of choice whether A_1 should be instantiated by a particular value. EXPRESS allows also to declare **INVERSE** attributes (Figure 2.4).

EXPRESS attributes are converted into OWL class restrictions by means of data properties or object properties, the latter if it is necessary to link individuals to individuals. The **OPTIONAL** construct is converted via a universal quantification (**owl:AllValuesFrom**), the mandatory attribute with an existential quantification (**owl:SomeValuesFrom**). The cardinality of the EXPRESS attributes can be converted to OWL by making use of explicit assertions on cardinalities and/or by defining specific characteristics of the data/object properties (e.g. functional, inverse functional).

According to the above discussion, the conversion of `IfcObject` (see Section 2.3) into OWL is given by the conjunction of the following statements, from (1) to (12), given in the Manchester Syntax [28] (the symbols *or* and *only* stand for the logical disjunction and the universal quantifier, respectively).

DisjointClasses: `IfcActor, IfcControl, IfcGroup, IfcProcess, IfcProduct, IfcResource` (1)

Class: `IfcObject` (2)
EquivalentTo:
`IfcActor` **or** `IfcControl` **or** `IfcGroup` **or** `IfcProcess` **or** `IfcProduct` **or** `IfcResource` (3)
SubClass Of:
`IfcObjectDefinition` (4)
`hasObjectType` **only** `IfcLabel` (5)
`isDeclaredBy` **only** `IfcRelDefinesByObject` (6)
`declares` **only** `IfcRelDefinesByObject` (7)
`isTypedBy` **only** `IfcRelDefinesByType` (8)
`isDefinedBy` **only** `IfcRelDefinesByProperties` (9)

ObjectProperty: `hasObjectType, isDeclaredBy, isTypedBy` (10)
Characteristics: Functional (11)

ObjectProperty: `declares, isDefinedBy` (12)

Note that by declaring `IfcObject` equivalent to the union of disjoint classes in statement (3), every instance of `IfcObject` is an instance of one and only one of those classes; in other terms, multiple inheritance is not allowed. For example, it cannot be the case that a particular object (e.g. *Object#101*) instantiates both a subclass of `IfcGroup` and a subclass of `IfcProcess`. Furthermore, the same OWL statement captures the EXPRESS abstract construct since it states that every instance of `IfcObject` has to be an instance of one of the classes declared in the equivalence. It follows that any `IfcObject` instance must be an instance of one of its subclasses.

The OWL statement (8) involves the class `IfcRelDefinesByType` standing for the IFC objectified relationship entity that is formally treated as an object. Ac-

cording to IFC, the objectified relationship `IfcRelDefinesByType` is defined as follows:

> "The objectified relationship IfcRelDefinesByType defines the relationship between an object type (see IfcTypeObject) and object occurrences (see IfcObject). The IfcRelDefinesByType is a 1-to-N relationship, as it allows for the assignment of one type information to a single or to many objects. Those objects then share the same object type, and the property sets and properties assigned to the object type."

where `IfcTypeObject` is in turn defined as follows:

> "The object type defines the specific information about a type, being common to all occurrences of this type. It refers to the specific level of the well recognized generic - specific - occurrance modeling paradigm. The IfcTypeObject gets assigned to the individual object instances (the occurrences) via the IfcRelDefinesByType relationship. The object type is represented by a set of property set definitions. The attached property sets describe the available alpha-numeric information about the object type. and are used to define all common properties that apply to all object occurrences of that type."

Practically, objectified relationships are used in IFC to separate the properties specific to relationships from the object attributes. This allows the development of a separate subtype tree for the relationship semantics [38]. Figure 2.5 visualises the OWL restrictions linking `IfcObject`, `IfcRelDefinesByType` and `IfcTypeObject`, which are depicted by dashed lines. This conversion of the IFC EXPRESS specification to OWL is taken from [54] and covers also the **INVERSE** attributes which were not addressed by the OntoSTEP plug-in [2].

Figure 2.5 OWL restrictions linking IfcTypeObject, IfcRelDefinesByType, and IfcObject.

According to this conversion pattern, the IFC classes and object properties are formalized in the statements (13)-(27). In particular, statements (13)-(18) define the class `IfcTypeObject`, whereas statements (19)-(24) define the class `IfcRelDefinesByType`. Finally, statements (25)-(27) define the object properties that are needed to formulate the restrictions.

DisjointClasses: `IfcTypeProcess`, `IfcTypeProduct`, `IfcTypeResource` (13)

Class: `IfcTypeObject` (14)
EquivalentTo:
`IfcTypeProcess` **or** `IfcTypeProduct` **or** `IfcTypeResource` (15)
SubClass Of:

`IfcObjectDefinition` (16)
`types` **only** `IfcRelDefinesByType` (17)
`hasPropertySets` **only** `IfcPropertySetDefinition` (18)

Class: `IfcRelDefinesByType` (19)
SubClass Of:
`IfcRelDefines` (20)
`hasRelatingType` **only** `IfcTypeObject` (21)
`hasRelatingType` **some** `IfcTypeObject` (22)
`hasRelatedObjects` **only** `IfcObject` (23)
`hasRelatedObjects` **some** `IfcObject` (24)

ObjectProperty: `types, hasRelatingType` (25)
Characteristics: Functional (26)

ObjectProperty: `hasPropertySets, hasRelatedObjects` (27)

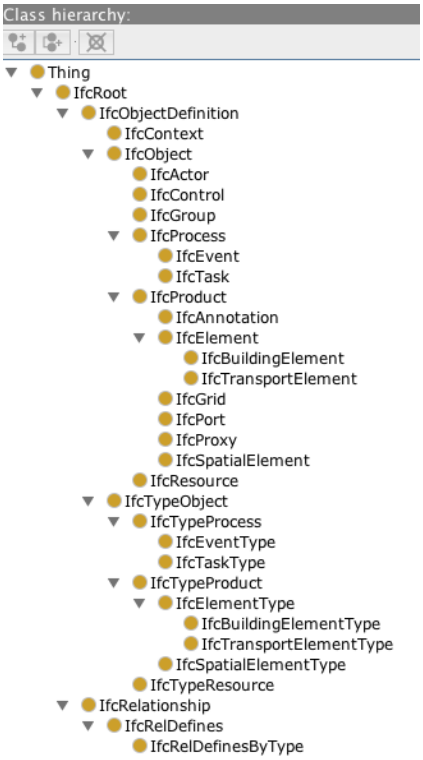

Figure 2.6 Fragment of IFC class hierarchy as visualised in Protégé [35].

Figure 2.6 presents a fragment of the IFC standard converted to OWL and visualised in Protégé. This fragment consists mainly of the IFC class hierarchy related to the previous examples, specifically including subclasses of `IfcObject` and `IfcTypeObject`. It must be noted that all the classes shown in Figure 2.6 are abstract except for `IfcBuildingElement`, `IfcTransportElement`, `IfcTask`, `IfcTaskType`, `IfcEventType`, `IfcBuildingElementType`, `IfcTransportElementType`, `IfcRelDefinesByType`, and `IfcEvent`. For example, this means that the conversion of `IfcTransportElement` cannot include a statement like (3) since this class can be directly instantiated, whereas such constraint can be included in the conversions of `IfcProduct` and `IfcElement`.

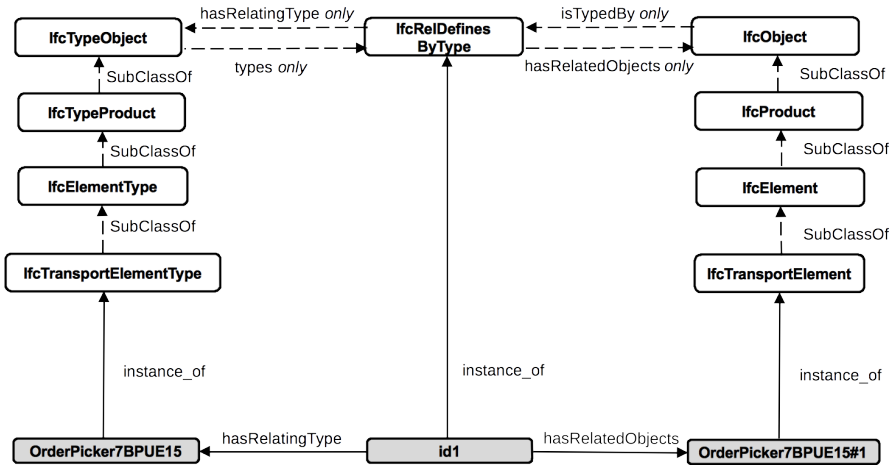

Figure 2.7 Fragment of IFC class hierarchy with the objectified relation and instances of non-abstract classes.

Figure 2.7 is an OWL based graph that represents the IFC class hierarchy with the objectified relation and the instances of non-abstract classes. The individual `OrderPicker7BPUE15` is an instance of `IfcTransportElementType`, `OrderPicker7BPUE15#1` is an instance of `IfcTransportElement`, and `id1` is an instance of the objectified relation `IfcRelDefinesByType`. Moreover, a typing relationship is defined by linking the individual `id1` to individuals `OrderPicker7BPUE15` and `OrderPicker7BPUE15#1` by means of object properties `hasRelatingType` and `hasRelatedObjects`.

This section showed how a basic conversion of EXPRESS to OWL can be carried out. The further ontological analysis can help to shed light into other aspects of the standard as well. On the one hand, it helps to understand classes and relationships, raising the awareness of the modeler toward a coherent use of these elements. On the other hand, it provides guidelines to choose the appropriate representation schema depending on the information to be modeled. It remains to verify whether this fur-

ther analysis leads to improve the conversion results here discussed as suggested by previous experiences in applied ontology, e.g. [26, 25, 61, 7].

2.5 Types and occurrences in IFC: an ontological analysis

As it stands today, IFC is a data model that relies on human interpretation, in the sense that the meaning of its classes is not constrained by means of formal semantics. Although one can use an EXPRESS-version of the standard, such an implementation relies on procedural and functional specifications, e.g. where-rules, whose applications and consequences cannot be verified semantically. As discussed in the previous sections, this situation leads to looking for a formalization of IFC in modern ontological languages like OWL (cf. Section 2.4). If the change of language helps to improve the system as we have argued, it does not *per se* lead to the clarification of the IFC conceptualization and ontological coherence. For this reason, the use of ontological analysis can help to highlight possible drawbacks in IFC and in the conversion patterns used to build IFC-driven ontological systems.

There are different ways to analyze a standard from the ontological viewpoint. Our plan comprises four steps: first we aim to verify whether the main distinctions on which IFC relies are well defined and understood. The second step looks at which properties and relations are introduced in the system, how they help to constrain the classes and leads to identify relationships which could be needed but are not in the system, if any. Third, an analysis is carried out to verify which classes are explicitly defined, the role they have, how they are characterized and related. Finally, the general perspective of the standard is extracted, and its overall structure is assessed in terms of ontological coherence. In this and the next section, we present material that falls within the first and second steps, respectively.

The conceptual schema of IFC is grounded on the distinction between modelling constructs for type classes like `IfcTypeObject` and for occurrence classes like `IfcObject`, that have been discussed in section 2.4.2. The type-occurrence distinction is central as one can see looking at IFC modelling constructs like `IfcProduct` and `IfcTypeProduct`, `IfcEvent` and `IfcEventType` as well as `IfcProcedure` and `IfcProcedureType`, to name a few. We thus begin with the investigation of the type-occurrence distinction aiming at clarifying what it amounts to in ontological terms. This investigation reduces the risk of an erroneous implementation of IFC by clarifying the notion and how it should be used. More specifically, we want to understand what kind of entities are interested by the distinction, how they are classified, and what kind of relationships hold between them.

The type-ocurrence dichotomy can be discussed by referring to Figure 2.7. Recall that IFC, being formalized in the EXPRESS language, explicitly models the upper side of the schema only; the boxes in the lower side, `OrderPicker7BPUE15` and `OrderPicker7BPUE15#1`, are here included to help interpreting the system. Also, from Section 2.4.2 we know that all elements represented in white boxes are

classes; that IfcRelDefinesByType stands for the IFC objectified relationship holding between IfcTypeObject and IfcObject and their subclasses; and that OrderPicker7BPUE15 and OrderPicker7BPUE15#1 refer to instances that an engineer aims to model.

The terms *instance* and *class* are commonly used with a variety of meanings in the literature and we need to make their interpretation precise. The main distinction between a class and an instance is that the former is a collection of entities (its members), while the latter is not:[4] a class is said to *have instances*, while an instance itself cannot instantiate. In the language of our foundational ontology DOLCE (Section 2.2.1), classes are a form of categories and instances are particulars. For example, the particular OrderPicker7BPUE15#1 is an instance of the class IfcTransportElement, and the distinction between this class and this instance relies on the way they are related to the expression *being a TransportElement*: we say that any instance of the class satisfies the expression and that the class is characterised by such an expression. More specifically, *being a TransportElement* is a property used to *talk about* instances: some instances are transport elements, while others, e.g. a product manufactured by Microsoft as well as the Everest Mountain, are not. Properties help to discriminate entities because they allow stating which entities are distinct and why. To be an instance of the class IfcTransportElement is thus equivalent to satisfying the property *being a TransportElement*, which is a shortcut for a conjunction of several more specific properties regarding e.g. the engine, the electric circuit, the mechanical components and so forth. This complex property is known, in logical and ontological terms, as the *intension* of the class IfcTransportElement, while the collection of the entities satisfying the property is called the *extension* of the class. The difference between intensionality and extensionality plays a relevant role in ontological engineering, because it allows understanding whether a class is bound or not to the particulars it talks about [26]. In Figure 2.7, a specific transport element corresponds to the box at bottom-right. This object, OrderPicker7BPUE15#1, is characterized by a conjunction of properties (like *engine model*, *load capacity*, *max speed*, *wheel configuration*) with their qualifications (values and ranges like engine model #*A*22, load capacity 1350 kg, max speed 12 km/h and wheel configuration 4). The conjunction of these qualified properties provides the identity criterion for belonging to the class IfcTransportElement which, depending on the purpose of the model, in Figure 2.7 could be the class IfcTransportElement itself or a subclass of it.

Object OrderPicker7BPUE15#1 and class IfcTransportElement are connected by the *instance_of* relation. From this perspective, occurrences stand for extensional classes, whose instances are particular elements satisfying certain properties. It is however a matter of ambiguity whether an occurrence instance represents a physical entity (e.g. the transport element owned by a particular company), or it stands for a virtual representation in an information system. In our experience we

[4] We do not distiguish classes from sets, see [50] for a philosophical discussion of these notions in ontology. Also, we distinguish an instance from a generic element of the class and will call instances only members of a class which are not classes themselves.

note that IFC practitioners adopt both readings depending on their application tasks. Nevertheless, physical entities on the one side, and virtual entities on the other side have different ontological properties: the former has e.g. a spatiotemporal location and in DOLCE is classified as a *physical object*; the latter lacks spatial location and is classified as an *abstract* when lacking temporal location also; or as a *concept* when existing in time, for example in the form of a description.[5] Since physical, abstract and conceptual elements are distinguished by incompatible properties, ontology is required to clearly separate between them. Note also that the identification of classes at the virtual (abstract) and the physical levels play a relevant role in data sharing, communication and verification since it determines which data are affected by time and which are not. Consider a scenario in which a practitioner develops an IFC-driven ontology instance model (that is, an A-Box) with an occurrence o_1 which is a virtual entity. When a user accesses the ontology, she would not be able to properly interpret o_1 as a virtual entity unless this is somehow modeled in the system: this leads to lack of trust on the data by the user when the source has not been visited for some time or is temporarily not accessible, and in a waste of the user's resources when the source is continuously verified on this occurrence.

Let us assume that the class `IfcTransportElement` is associated to a subclass of the class `IfcElementType` of Figure 2.7. This relationship is at the intensional level: it relies on the properties characterizing `IfcTransportElement` and does not depend on its instances, whether they exist or else. We have seen some examples of these properties: *engine model*, *load capacity* and *wheel configuration* including their values. The subclass of `IfcElementType` associated with `IfcTransportElement`, call it `IfcTransportElementType`, is the collection of all the properties characterizing the class `IfcTransportElement` but, generally speaking, not of their values. For example, it contains *having engine model*, *having max speed* and so on. We call the relationship between the type and occurrence elements, i.e. the classes on the top of Figure 2.7, *typization* since it allows associating to each homogeneous collection of instances in the right hand-side of the diagram, a single general description in terms of which properties it is meaningful to consider for their comparison[6]

Finally, an instance of `IfcTransportElementType`, that is, an element in the bottom-left of Figure 2.7, is associated with a collection of these very properties with specified values and ranges: in our example, *engine model* #A22, *load capacity* 1350 *kg*, and the like. We call this object `OrderPicker7BPUE15`. In ontological terms,
`OrderPicker7BPUE15` is an information object associated with a specific de-

[5] Note that the occurrent as a description is meant to refer to the content of the description independently of its coding on a piece of paper, a written text or a memory support. The latter being physical objects with representation functionalities.

[6] Ontologically speaking, we could say that IFC is a contextual modeling framework since the notion of 'homogeneous instances' depends on the user or the application task and not necessarily on the type of property one is considering. This observation is crucial for a suitable choice of the classes to consider in the diagram but also ontologically problematic since it suggests that modeling guidelines needs to include the purpose of the model.

scription. Ontologically, the description of `OrderPicker7BPUE15` can be specific or generic, depending on whether the property values are given and, when so, if specified as precise values or ranges. We dub the relationship between the property collection `OrderPicker7BPUE15` and its instance `OrderPicker7BPUE15#1` that satisfies them a *realization* since the object `OrderPicker7BPUE15#1` manifests ("realizes") all the properties given by `OrderPicker7BPUE15` with the requested qualifications. Note that a realization does not need to be a physical entity; it may very well be a virtual element (abstract or conceptual in the DOLCE terminology). As observed before, an ontological rendering of IFC must address this aspect since each class of the system should be classified as belonging to the physical, abstract or conceptual branch of the taxonomy. However, there are different ways (and levels of constraints) to set this classification. An overall assessment can be done only after a more comprehensive evaluation of IFC and may turn out to be influenced by contextual issues. We conclude this analysis observing that realization differs from typization, i.e., the relationship between the class `IfcTransportElementType` and the class `IfcTransportElement`: realization holds between instances and says that an entity satisfies a set of properties with given values; typization is between classes and associates the class corresponding to a set of properties (a type) to the occurrence class whose members satisfy those properties for some (admissible) value.

2.6 Properties in IFC ontologies

The ontological analysis performed in the previous section explains the IFC type-occurrence modelling pattern in terms of the difference between descriptions within the type hierarchy, and their realizations within the occurrence hierarchy, where descriptions and realizations may very well be both virtual elements in the system.

In this section we focus on properties. We have seen that IFC types can be associated with sets of properties that characterize occurrences. Properties are an interesting topic in ontology because their ontological nature leads to precise structures that are not fully exploited today. Our goal in this part is to verify whether distinguishing different types of properties can deepen our understanding of, and even improve, the conceptual structure of IFC-driven ontologies. For the purpose of this analysis, we use the modelling constructs `IfcTransportElementType` and `IfcTransportElement` of the previous section to exemplify our observations.

The class `IfcTransportElementType` defines different properties among which `isDecomposedBy`, `hasPropertySet` and `IfcElementType`. The first is used to specify that instances of `IfcTransportElementType` (and of `IfcTransportElement`) are composed of other entities by means of the part-whole (objectified) relationship `IfcRelDecomposes` (and its subclasses). The second refers to the property sets associated to `IfcTransportElementType` that are common to all associated instances of the class `IfcTransportElement`. As the term suggests, properties sets are understood as sets of properties, where

properties are things like volume, pressure, temperature and so on.[7] The third property is meant to provide the parent class of class `IfcTransportElementType` within the IFC taxonomy.

Assume an IFC-driven ontology is given. We aim to extend it with the properties *hasWeight* and *hasWheel*[8] because these are relevant to model transport elements. First, we verify whether these properties can be managed under existing IFC modelling constructs. From an ontological viewpoint, the differences between *hasWeight* and *hasWheel* can be understood in terms of the differences between structural properties and qualities. In particular, if we analyze the domain of these relations we find something like the following:

(a) $\forall x,y(hasWeight(x,y) \rightarrow TransportElement(x) \wedge Weight(y))$
(b) $\forall x,y(hasWheel(x,y) \rightarrow TransportElement(x) \wedge Wheel(y))$

that is, *hasWeight* applies to a pair in which the first argument is a transport element and the second a weight, *hasWheel* to pair formed by a transport element and a wheel. Since weight and wheel are different types of elements, *hasWeight* and *hasWheel* are ontologically distinct as they are committed to the existence of different things. In (a) *Weight* can be understood as a DOLCE *(individual) quality*, i.e. an entity that *inheres* in another one (the so called *bearer* of the quality) and cannot exist if the inheriting entity fails to exist. In (b) *Wheel* can be conceptualized as a physical entity that, at a specific time point, is *part of* another entity as one of its components, thus it specifies the structural relationship existing between two objects for some time. We can thus re-write (a) and (b) more properly as follows (note the temporal parameter in the second expression):

(a.1) $\forall x,y(hasWeight(x,y) \rightarrow TransportElement(x) \wedge Weight(y) \wedge inheres_{in}(y,x))$
(b.1) $\forall x,y,t(hasWheel(x,y,t) \rightarrow TransportElement(x) \wedge Wheel(y) \wedge part_{of}(y,x,t))$

Note that (b.1) only states that *Wheel* is part of *TransportElement*, while in our conceptualization, we want to specifically model that, differently from other parts, *Wheel* refers to a transport element's *component*. The concept of part is general enough to apply to many different things: for example, one could say that a small splinter of the transport element's chassis was a part of the chassis at a certain time point, while not acknowledging it as a component. In our understanding, a component c is an entity that is part of an object o at a certain time t and it satisfies a certain unity criteria u in t, i.e. a property (e.g. topological, morphological or functional) by which it can be considered as a *whole* object. Thus, we can reformulate (b.1) as:

(b.2) $\forall x,y,t(hasWheel(x,y,t) \rightarrow TransportElement(x) \wedge Wheel(y) \wedge$
$Component(y,x,t)) \wedge \forall x,y,t(Component(y,x,t) \rightarrow part_{of}(y,x,t) \wedge U(y,t))$

[7] Cf. the examples provided in the documentation available at: http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/.

[8] The latter property is strictly related to property "wheel configuration" mentioned in Section 2.5. For the sake of the presentation it is clearer to use "hasWheel" in the following discussion.

where, U is a predicate specifying the unity property of components and *Component* is a ternary predicate stating that y is component of x at t.

Additionally, we can also specify that all transport elements modelled in our ontology are vehicle, that is:

(c) $\forall x(TransportElement(x) \rightarrow Vehicle(x))$

where the property *Vehicle* identifies an *ontological type*. Ontological types refer to *rigid* properties that supply their own *identity criterion*. In ontological engineering methodologies like OntoClean [26] a property is said to be rigid if it is essential to all its possible instances, i.e. if it is context-independent. This means that if at some time or in some context p is an instance of the rigid property P, p cannot stop being an instance of P in any other time or context while remaining the same entity. In an engineering context, one can assume that the term 'vehicle' refers to a rigid property, because if v is a vehicle at time t_1, it will be a vehicle also at a different time t_2 and, in principle, if it stops being a vehicle at some point, it won't become something different but simply will disappear from the model. The classification of a vehicle can be contrasted with that of a resource, i.e., something that is available to perform a task, such as a machine or a person. If r is a resource at t_1, r might not be as such at t_2, because, for example, it might be under repair or busy at that time. In particular, a vehicle can be a resource as well during some time but, while it will always be a vehicle, it may cease to be a resource in some periods of time. An identity criterion (IC) tell us what conditions need to be satisfied by objects referred to by general terms [26]. From a logical viewpoint, IC is usually represented as follows:

$$\forall x, y((\phi(x) \wedge \phi(y)) \rightarrow (x = y \leftrightarrow Rxy))$$

where ϕ is a general term and R is an equivalence relation between x and y. For example, one could say that two transport elements are numerically the same one if they have the same chassis' number, where R is relation *having the same chassis's number* and ϕ is the property *being a transport element*. Type properties play a fundamental role in taxonomically organizing the classes of an ontology and, in particular, they help in constituting the so-called *backbone taxonomy*, that is, the most relevant and stable properties being represented in the ontology [26].

Recognizing the differences between properties related to components, qualities and ontological types in ontology and data models is relevant to sharply describe the domain elements we are interested in. The IFC data model has a quite rich set of constructs for properties and is already equipped with constructs that are suitable to model the ontological differences we just presented. While the consistency of the IFC guidelines in using these constructs can be assessed only by looking at the overall framework, here we can still evaluate whether IFC lacks some needed formal restrictions to properly model these distinctions.

One could classify *hasWheel* by means of `IfcRelDecomposes` and, in particular, of `IfcRelAggregates`, which is the modelling construct to represent the part-whole relation holding between the subclasses of `IfcObjectDefinition`.

`IfcRelAggregates` can thus specify structural properties. According to the informal semantics of `IfcRelDecomposes` (and thus of `IfcRelAggregates`), this relation imposes a mutual existential dependency between a whole and its parts: the whole explicitly depends on the parts, and the parts on the whole. It follows that if a transport element is disassembled it still exists but the loss of a single component (e.g. wheel or even a screw that breaks in two parts so that it ceases to be a screw) threatens the existence of the transport element itself. When a wheel or screw of a transport element breaks and needs to be substituted, you do not expect that the transport element stop to exist and perhaps later it resume existence just because one of its wheel has been (temporarily) removed. Considering the ontological modeling of artifacts [8], this modeling feature is too strong. Moreover, one would still recognise a transport element spare wheel as a wheel even though it is not installed in a transport element. These ambiguities might be solved by introducing a finer grained concept of *component* and a temporal account of the part/whole relation in IFC along the line of (b.2). According to (b.2), indeed, c is a component of object o only within a certain time period t, i.e., when c stands in a parthood relationship with o and satisfies a unity property U. This means that c's existence is independent of its parthood relationship with o; instead c's role as a component of o depends on this very relationship: a new wheel becomes a component of a transport element only once it is installed. Secondly, the existence of the $TransportElement\ o$ in (b.2) does not rely on the existence of all its components. That formalization leaves room for distinguishing components with different dependence relationships: one can assume that composed objects like transport elements have components as a matter of necessity, while avoiding constraining the transport element's existence to components like wheels, but binding it to things like engine and chassis. A distinction that in ontology is discussed in terms of essential vs non-essential parts or components.

The fact that `IfcRelDefinesByProperties` associates the subclasses of the class `IfcObjectDefinition` to their qualities (which are subclasses of `IfcPropertySetDefinitionSelect`) resembles the $inheres_{in}$ relationship introduced in (a.1) and, as consequence, *hasWeight* can be classified by means of `IfcRelDefinesByProperties`. This construct does not existentially bind qualities to their bearers, a constraint that is however widely used in foundational ontologies and recall the common sense idea that qualities exist only attached to other objects. For example, it might be odd to talk about the weight or the color of a transport element, if the latter does not exist[9].

2.7 Conclusion and further discussion

We looked at different strategies to produce an IFC-driven ontology based on patterns for generating OWL ontologies from IFC models. Later, we applied ontolog-

[9] Existence is not a matter of physical actuality in this case. It also refers to conceptual existence, as e.g. when you discuss about the max speed of the transport element you want to buy next year, while it has not been produced by the reseller yet

ical arguments to investigate how to interpret IFC from an ontological viewpoint looking for a sound and unifying perspective on some basic constructs, namely the *Type/Occurrence pattern* and the different ways to model properties.

While the advantages of an OWL version of IFC have been discussed at length in the literature, we recall here a few important consequences of making available a well structured IFC-driven OWL system: (i) IFC can be reliably integrated with other data models that cover different industrial domains like STEP-NC for machining operations and BPMN for business processes; (ii) a good OWL version allows to efficiently model and manage distributed data. The integration of fragments of IFC documents (formalized as STEP-file according to ISO 10303-21 or as XML documents according to ISO 10303-28) is difficult to achieve because intra-document references are supported, but inter-document references (i.e. cross-references) are poorly modeled, thus jeopardizing data and knowledge reuse; (iii) OWL has a rigorous formal semantics; and (iv) via OWL one can use general-purpose reasoning tools at the taxonomic and instance level without developing specific systems for each data model.

As the further steps towards the establishment of the ontology-based IFC standard that will be robust in terms of its logical coherence and flexible in terms of its capability to deal with the real world-scenarios we foresee the analysis of the specific relationships between `IfcObject` and `IfcTypeObject` on the level of their subclasses. In particular, we see the need for proposal of the semantically sound and explicit specification of the connection between subclasses of `IfcObject` and `IfcTypeObject`. Without a robust (onto)logical definition of constraints that should hold between the IFC subclasses (e.g. `IfcTransportElement` and `IfcTransportElementType`), the IFC leaves open space for an erroneous interpretation of the typing relation, allowing the constructs that declare an arbitrary typing relation (e.g. connecting an instance of `IfcTransportElement` with an instance of a subclass of `IfcProcess`, i.e. `IfcTask`). We aim at dealing with this problem in the future by means of (i) complex class expressions that can be used to distinguishing accurate from inaccurate typing relationships; (ii) distinguish mandatory (i.e. essential) properties of a type and optional (i.e. contingent) properties that are defining a type (see Sect. 2.6).

A proper modeling of the *Type/Occurrence pattern* in the OWL version of IFC can lead to practical advantages by allowing (i) to dynamically group individuals of an *occurrence* class to avoid excessive hierarchy revisions and to control subclass proliferation (in particular when these are hard to formalize) by focusing on the set of properties relevant to the user or application. Ideally, given the task and the properties, one should be able to a-priori set the depth of the taxonomy which is optimal for the given domain; (ii) to correctly define an individual of an *occurrence class* by inheriting property values from the related individuals in the associated *type class*. Moreover, updating property values of the individual in a *type class* is simpler and less error prone; eventually, this update automatically spreads to all the related individuals; and (iii) to organize the system so that the user can focus on populating the A-box without worrying about changing the T-box. However, it is still needed to explore the relation between the intensional and extensional

definition of a class, where the former is understood as the conceptual specification of a *type class* and the latter as the specification via the instances of a *type class* (understood as a prototype). The ontological analysis would help to explain the concept of resemblance between a type and its prototype, by means of distinguishing the typical mandatory properties of a type. In that way we can avoid a multiplication of instances of types, while providing a supervised method that can be used in the ontologically sound creation and classification of types.

Acknowledgements This research has been partially funded by MIUR under the Italian flagship project "Fabbrica del Futuro", Subproject 2, research project "Product and Process Co-Evolution Management via Modular Pallet configuration" (PRO2EVO).

References

[1] F. Bader, D. Calvanese, D.L. McGuinnes, D. Nardi, P.F. Patel-Schneider (Eds.), *The Description Logic Handbook. Theory, Implementation and Applications*, Second Edition, Cambridge University Press, 2007

[2] R. Barbau, S. Krima, S. Rachuri, A. Narayanan, X. Fiorentini, S. Foufou, and R.D. Sriram, "OntoSTEP: Enriching product model data using ontologies". in *Computer-Aided Design*, 44:6, pp. 575–590, Elsevier, 2012.

[3] V. Bazjanac and D.B. Crawley, "The Implementation of Industry Foundation Classes in Simulation Tools for the Building Industry," Lawrence Berkeley National Laboratory, Tech. Rep., 1997.

[4] J.Beetz, J. van Leeuwen, and B. de Vries, "An Ontology Web Language Notation of the Industry Foundation Classes", in *22nd CIB W78 Conference on Information Technology in Construction*, 2005.

[5] J.Beetz, J. Leeuwen, and B. Vries, "IfcOwl: A case of Transforming EXPRESS Schemas into Ontologies", in *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 23: 89–101, 2009.

[6] S. Borgo, How Formal Ontology can help Civil Engineers. In Jacques Teller, John Lee, and Catherine Roussey, editors, *Ontologies for Urban Development*, Studies in Computational Intelligence, pp. 37–45. Springer, 2007.

[7] S. Borgo, M. Carrara, P. Garbacz, and P. E. Vermaas, "A Formalization of Functions as Operations on Flows", in *Journal of Computing and Information Science in Engineering*, 11(3):031007 1–14, 2011.

[8] S. Borgo, M. Franssen, P. Garbacz, Y. Kitamura, R. Mizoguchi and P. E. Vermaas Technical Artifact: An Integrated Perspective, In Formal Ontologies Meet Industry, FAIA 229, IOS Press, pp. 3-15, 2011

[9] S. Borgo and P. Leitão. Foundations for a Core Ontology of Manufacturing. In Sharman R. Kishore R., Ramesh R. (eds) *Ontologies: A Handbook of Principles, Concepts and Applications in Information Systems*, *Integrated Series in Information Systems*, Vol. 14, pp. 751–776. Springer, 2007.

[10] S. Borgo and C. Masolo. Foundational Choices in DOLCE. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, International handbooks on information systems, pages 361–381. Springer Verlag, 2nd edition, 2009.

[11] S. Borgo, E. Sanfilippo, A. Šojić, and W. Terkaj. Towards an Ontological Grounding of IFC. In Formal Ontology meets Industry (FOMI), to appear, 2014.

[12] S. Borgo and L. Vieu. From physical artifacts to products. In *Proceedings of the second FOMI Workshop*, Trento (Italy), 2006.

[13] S. Borgo and L. Vieu. Artifacts in Formal Ontology. In Anthonie Meijers, editor, *Handbook of the Philosophy of the Technological Sciences, Technology and Engineering Sciences*, Vol. 9, pp. 273–307. Elsevier, 2009.

[14] buildingSMART International, "Industry Foundation Classes. IFC Release Candidate 4," 1999-2012. Available on: `http://www.buildingsmart-tech.org/ifc/IFC2x4/rc4/html/`. Last access: June 2014.

[15] M. Colledani, W. Terkaj, T. Tolio, M. Tomasella, "Development of a Conceptual Reference Framework to manage manufacturing knowledge related to Products, Processes and Production Systems," in *Methods and Tools for Effective Knowledge Life-Cycle-Management* (A. Bernard, S. Tichkiewitch, eds.), pp. 259–284, Springer-Verlag, 2008.

[16] M. Colledani, W. Terkaj, T. Tolio, "Product-Process-System Information Formalization," in *Design of Flexible Production Systems* (T. Tolio, ed.), pp. 63–86, Springer-Verlag, 2009.

[17] M. Colledani, G. Pedrelli, W. Terkaj, M. Urgo, "Integrated Virtual Platform for Manufacturing Systems Design", in *Procedia CIRP* 7:425–430, 2013.

[18] S. Gagliardo, F. Giannini, M. Monti, G. Pedrielli, W. Terkaj, M. Sacco, M. Ghellere, F. Salamone, "An Ontology-based Framework for Sustainable Factories", in *Computer-Aided Design & Applications*, vol. 12(2), pp. 1–10, 2015.

[19] G. Ghielmini, P. Pedrazzoli, D. Rovere, W. Terkaj, C.R. Bor, G. Dal Maso, F. Milella, M. Sacco, "Virtual Factory Manager for semantic data handling", in *CIRP Journal of Manufacturing Science and Technology*, vol. 6(4), pp. 281–291, 2013.

[20] J. C. Giarratano and G. Riley. *Expert Systems*. PWS Publishing Co., Boston, MA, USA, 3rd edition, 1998.

[21] A. Gola, J. Montusiewicz, A. Świć, "Computer Aided FMS machine tools subsystem selection using the Evolutionary System of Multicriteria Analysis", in *Applied Computer Science*, vol. 7(1), pp. 18–29, 2011.

[22] A. Gola, A. Świć, "Computer-Aided Machine Tool Selection for Focused Flexibility Manufacturing Systems Using Economical Criteria", in *Actual Problems of Economics*, vol. 124(10), pp. 383–389, 2011.

[23] A. Gola, A. Świć, V. Kramar, "A multiple-criteria approach to machine-tool selection for focused flexibility manufacturing systems", in *Management and Production Engineering Review*, vol. 2(4), pp. 21–32, 2011.

[24] A. Gola, A. Świć, "Design of storage subsystem of flexible manufacturing system using the computer simulation method", in *Actual Problems of Economics*, vol. 142(4), pp. 312–318, 2014.

[25] M. Grueninger, "Using the PSL Ontology," in S. Staab, R. Studer (Eds.), *Handbook on Ontologies*, pp. 423–443, Springer-Verlag Berlin Heidelberg, 2009

[26] N.Guarino, C.Welty, "An Overview of OntoClean," in S.Staab, R.Studer (Eds.), *Handbook on Ontologies*, pp. 201–220, Springer-Verlag Berlin Heidelberg, 2009

[27] W. Hodges. Elementary predicate logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume I, pages 1–131. Dordrecht: Reidel, 1983.

[28] M. Horridge, N. Drummond, J.Goodwin, A.L.Rector, R.Stevens, H. Wang, "The Manchester OWL Syntax", in *OWLed*, 216, 2006

[29] I.Horrocks, P.F.Patel-Schneider, H.Boley, S.Tabet,B.Grosof, M.Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML W3C Member Submission, 2004

[30] International Organization for Standardization (ISO), "Industrial Automation Systems and Integration - Product Data Representation and Exchange. Part 1: Overview and Fundamental Principles", ISO 10303-1:1994(e) Ed., 1994.

[31] B. Kadar, W. Terkaj, and M. Sacco, "Semantic Virtual Factory Supporting Interoperable Modelling and Evaluation of Production Systems," *CIRP Annals - Manufacturing Technology*, 62:443–446, 2013.

[32] P. Katranuschkov, A. Gehre, and R. Scherer, "An Ontology Framework to Access IFC Model Data," *ITcon*, 8:413–437, 2003.

[33] L. Khemlani, "The IFC Building Model: A Look under the Hood," AECbytes Feature 2004. Available on: `http://www.aecbytes.com/feature/2004/IFCmodel.html`. Last access: June 2014.

[34] Y. Kitamura, M. Kashiwase, M. Fuse, and R. Mizoguchi. Deployment of an ontological framework of functional design knowledge. *Advanced Engineering Informatics*, 18(2):115–127, 2004.

[35] H. Knublauch, R.W.Fergerson, N.F.Noy, M.A.Musen, "The Protégé OWL plugin: An open development environment for semantic web applications" in *The Semantic Web–ISWC 2004*, pp. 229–243, Springer, 2004

[36] S. Krima, R. Barbau, X. Fiorentini, R. Sudarsan, and R.D. Sriram, "OntoStep: OWL-DL Ontology for STEP," U.S. National Institute of Standards and Technology (NIST),Tech. rep., 2009.

[37] M. Laakso, A. Kiviniemi, "The IFC Standard - A Review of History, Development, and Standardization" in *Journal of Information Technology in Construction (ITcon)*, 17:134-161, 2012

[38] C. Lima, B. Fiès, C. Silva, S. Barresi, "Setting up the Open Semantic Infrastructure for the Construction Sector in Europe–the FUNSIEC Project." in *5th European Conference on Product and Process Modelling in the Building and Construction Industry, ECPPM 2004*, pp.540

[39] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider. The wonderweb library of foundational ontologies. Deliverable 17, EU WonderWeb Project, http://wonderweb.man.ac.uk/deliverables.shtml, 2002.

[40] G. Modoni, M. Sacco, W. Terkaj, "A survey of RDF store solutions", in *roceedings of 2014 International ICE Conference on Engineering, Technology and Innovation (ICE), Bergano, Italy*, 2014.

[41] National Institute of Building Sciences (NIBS), "United States National Building Information Modeling Standard" Version 1 – Part 1: Overview, principles and methodologies, NIBS, Tech. rep., 2007.

[42] P. Pauwels and D. Van Deursen and R. Verstraeten and J. De Roo and R. De Meyer and R. Van de Walle and J. Van Campenhout, "A semantic rule checking environment for building performance checking" in *Automation in Construction*, vol. 20:5, pp. 506–518, 2011.

[43] P. Pauwels and D. Van Deursen ,"IFC/RDF: Adaptation, Aggregation and Enrichment", in *First International Workshop on Linked Data in Architecture and Construction (LDAC 2012)*, 2012.

[44] L.C. Pouchard and A.F. Cutting-Decelle, "Ontologies and Standard-based Approaches to Interoperability for Concurrent Engineering" in C.J.Anumba, J.M.Kamara, A.-F. Cutting-Decelle (Eds.), *Concurrent Engineering in Construction Projects*, pp.118-160, Taylor & Francis, London-New York, 2007

[45] M. Sacco, G. Dal Maso, F. Milella, P. Pedrazzoli, D. Rovere, W. Terkaj, "Virtual Factory Manager" in *Virtual and Mixed Reality - Systems and Applications, Lecture Notes in Computer Science* (R. Shumaker , ed.), vol. 6774, pp. 397–406,Springer Berlin Heidelberg, 2011.

[46] E. Sanfilippo, S. Borgo, and C. Masolo. States, events, activities: Is there an ontology behind BPMN? In *International Conference on Formal Ontology in Information Systems (FOIS 2014)*, FAIA, IOS Press, 2014 (to appear).

[47] D. Schenck and P.R. Wilson, *Information Modeling the EXPRESS Way*, Oxford University Press, 1994.

[48] H. Schevers, R. Drogenmuller, "Converting the Industry Foundation Classes to the Web Ontology Language", in *First International Conference on Semantics, Knowledge and Grid (SKG)*, 2006.

[49] B. Smith. Basic concepts of formal ontology. In N. Guarino, editor, *Proceedings of the First International Conference FOIS 1998*, pages 19–28. IOS Press, 1998.

[50] B. Smith, "Against fantology," *Experiecen and Analysis* (J.C.Marek and M.Reicher, eds.), pp. 153–170, HPT and OBV, 2005.

[51] B. Smith, W. Ceusters, B. Klagges, J. Köhler, A. Kumar, J. Lomax, C. Mungall, F. Neuhaus, A. Rector, and C. Rosse. Relations in biomedical ontologies. *Genome Biology*, 6(5):R46, 2005.

[52] B. Smith. Against idiosyncrasy in ontology development. *Frontiers in Artificial Intelligence and Applications*, Vol. 150, pp. 15–26, 2006.

[53] S. Szykman, R.D. Sriram, W.C. Regli, ' "The Role of Knowledge in Next-Generation Product Development Systems," in *Journal of Computing and Information Science in Engineering*, 1:3-11, 2001.

[54] W. Terkaj, G. Pedrelli, M. Sacco, "Virtual Factory Data Model," in *Proceedings of the Workshop on Ontology and Semantic Web for Manufacturing, Graz, Austria, 24-27 July, 2012*, pp. 29–43, 2012.

[55] W. Terkaj, T. Tolio, A. Valente, "Design of Focused Flexibility Manufacturing Systems (FFMSs)," in *Design of Flexible Production Systems* (T. Tolio, ed.), pp. 137–190, Springer-Verlag, 2009.

[56] W. Terkaj, T. Tolio, A. Valente, "A Stochastic Programming Approach to support the Machine Tool Builder in Designing Focused Flexibility Manufacturing Systems (FFMSs)", in *International Journal of Manufacturing Research*, vol. 5(2), pp. 199–229, 2010.

[57] W. Terkaj, M. Urgo, "Virtual Factory Data Model to Support Performance Evaluation of Production Systems," in *Proceedings of the Workshop on Ontology and Semantic Web for Manufacturing, Graz, Austria, 24-27 July, 2012*, pp. 44–58, 2012.

[58] W. Terkaj, M. Urgo, "Ontology-based Modeling of Production Systems for Design and Performance Evaluation", in *12th IEEE International Conference on Industrial Informatics, Porto Alegre, 2014.*

[59] V. Thein, "Industry Foundation Classes (IFC). BIM Interoperability through a Vendor-independent File Format. A Bentley White paper", Bentley Sustaining Infrastructure, Tech. rep., 2011.

[60] G.P. Vigano, L. Greci, S. Mottura, and M. Sacco, "GIOVE Virtual Factory: A new viewer for a more immersive role of the user during factory design," in *Digital Factory for Human-oriented Production Systems. The Integration of International Research Projects*, pp. 201–216, Springer London, 2011.

[61] M. West, *Developing High Quality Data Models*. Morgan Kaufmann, 2011.

[62] W3C OWL Working Group, "OWL 2 Web Ontology Language Document Overview (second edition)". Available on: `http://www.w3.org/TR/owl2-overview/`. Last access: June 2014.

[63] L. Zhang, R. R. Issa, "Development of Ifc-based Construction Industry Ontology for Information Retrieval from IFC Models" in *EG-ICE Workshop, University of Twente, The Netherlands, July, 2011.*