Automation in Construction 128 (2021) 103689

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Leveraging integration facades for model-based tool interoperability

Galina Paskaleva [a,*,1], Alexandra Mazak-Huemer [a,2], Manuel Wimmer [b,3], Thomas Bednar [c,4]

[a] Subsurface Engineering - Digital Transformation in Tunnelling, Montanuniversität Leoben, Erzherzog-Johann Strasse 3, 8700 Leoben, Austria
[b] Institute of Business Informatics - Software Engineering, Johannes Kepler Universität Linz, Altenberger Straße 69, Science Park 3, 4040 Linz, Austria
[c] Research Unit of Building Physics, Institute of Material Technology, Building Physics and Building Ecology, Faculty of Civil Engineering, TU Wien, Karlsplatz 13/207/2, 1040 Vienna, Austria

ARTICLE INFO

Keywords:
MDE
Data exchange
Big Open BIM
Semantic integration
Pragmatic integration
Heterogeneity

ABSTRACT

Data exchange and management methods are of paramount importance in areas as complex as the Architecture, Engineering and Construction industries and Facility Management. For example, Big Open BIM requires seamless information flow among an arbitrary number of applications. The backbone of such information flow is a robust integration, whose tasks include overcoming technological as well as semantic and pragmatic gaps and conflicts both within and between data models. In this work, we introduce a method for integrating the pragmatics at design-time and the semantics of independent applications at run-time into so-called "integration facades". We utilize Model-driven Engineering for the automatic discovery of functionalities and data models, and for finding a user-guided consensus. We present a case study involving the domains of architecture, building physics and structural engineering for evaluating our approach in object-oriented as well as data-oriented programming environments. The results produce, for each scenario, a single integration facade that acts as a single source of truth in the data exchange process.

1. Introduction

Today, the integration of distributed, autonomous and heterogeneous data sources across application boundaries is gaining importance due to the increasing networking of organizations and companies in many application fields. This leads to the situation that the information flow goes hand in hand with the translation among different data models with different syntax and semantics. A current example for this evolution is the domain of Architecture, Engineering and Construction (AEC) as well as Facility Management (FM) industries. In these industries' daily business, the volume of data to be exchanged among various stakeholders (e.g., building developers, energy suppliers, architects, structural engineers, building physicists, etc.) is rapidly increasing. Each of these stakeholders possesses specific domain knowledge and has a specific view of the project. These different perspectives may cause different forms of heterogeneity in the definition and handling of data, which hinders an interference-free communication within a building project.

This heterogeneity can be syntactic, semantic, pragmatic or a difference in the level of detail, i.e., granularity. *Syntactic heterogeneity* is caused by working with different data models in different tools. One example of *semantic heterogeneity* is the concept of homogeneous unit in the description of the building ground in different norms and regulations. The Austrian guideline[5] defines it on the basis of homogeneous substrate, the Swiss norm[6] - on the basis of similar structural behavior. *Pragmatic heterogeneity* occurs, for instance, when in the absence of a clear regulation, one architect considers slabs with inclination of more than 15% as walls and another with inclination of more than 25%. An example of *heterogeneity based on diverging levels of detail* can be found in the following scenario. The architecture domain relies on geometry not

* Corresponding author.
E-mail addresses: galina.paskaleva@tuwien.ac.at (G. Paskaleva), alexandra.mazak-huemer@unileoben.ac.at (A. Mazak-Huemer), manuel.wimmer@jku.at (M. Wimmer), thomas.bednar@tuwien.ac.at (T. Bednar).
[1] www.bph.tuwien.ac.at/no_cache/team/assistentinnen/staffmembers/staff/detail/paskaleva/.
[2] www.subsurface.at/.
[3] www.se.jku.at/manuel-wimmer/.
[4] www.bph.tuwien.ac.at/team/professorinnen/staff/show/person/bednar/.
[5] https://www.oegg.at/upload/download/downloads/geotech_rili_08_2.1.pdf (last accessed 2021-02-26).
[6] https://www.beuth.de/de/norm/sn-531199/246688888 (last accessed 2021-02-26).

https://doi.org/10.1016/j.autcon.2021.103689
Received 21 January 2020; Received in revised form 28 February 2021; Accepted 26 March 2021
Available online 5 May 2021

only as a representation of a building project, but also as an information carrier, whereas the building physics domain is concerned with thermal and hygric flux through the construction of a building. For this reason, building physicists consider points of interest (e.g., the joints between walls) in much greater detail than architects do.

In addition to data heterogeneities, the data exchange within or across different phases of a project is challenging, since there are different exchange standards used as well as various requirements that must be considered.

To overcome these obstacles, *Building Information Modeling (BIM)* has become more and more established in recent years. The main motivation behind BIM is to accommodate the heterogeneous nature of the AEC industries and involved domains, and to provide seamless data flows within any building or infrastructure project. To put it in a nutshell, BIM aims to support the generation and management of digital representations of physical and functional characteristics of a built structure throughout all phases. The ultimate goal of this development is *Big Open BIM*, a method for loss- and distortion-free, possibly real-time, data exchange across technological spaces and domains [7].

Preliminaries to BIM: The realization of BIM is not limited to a single data exchange standard. In fact, there have been multiple attempts to develop a suitable standard for it. For example, the proprietary Drawing Exchange Format (DXF)[7] standard was widely used in the 1980s and 1990s, and is currently still in use. Furthermore, the open Standard for the Exchange of Product model data (STEP)[8] was developed in the 1980s and, subsequently, became an ISO standard (ISO 10303).[9] The Collaborative Design Activity (COLLADA)[10] standard includes the exchange of geometric constraints and animations and is used not just in the AEC industries, but also in the automated production industry as part of the industry standard Automation ML.[11] The Construction Operations Building Information Exchange (COBie) [11] was developed by the US military in 2007 and aims to store construction related data during all phases of a building's life cycle. However, the currently most widely used BIM data exchange standard is the Industry Foundation Classes (IFC) standard [6]. The aim of IFC is to support Big Open BIM by providing a seamless communication among all stakeholders within the AEC domain. Therefore, it offers very abstract (e.g., *IfcObjectDefinition*) as well as very specific (e.g., *IfcBoiler*) concepts to describe a domain of interest.

A deeper look at IFC: Currently, IFC provides semantic types for the following domain groups: Building Controls, Plumbing and Fire Protection, Structural Elements, Structural Analysis, Heating Ventilation Air-Conditioning (HVAC), Electrical, Architecture, and Construction Management [6]. Building physics, which we mentioned above, is not regarded as a domain, but as a *resource*. It is partially covered by module *8.10 Material Resource*. Other sub-domains of building physics, e.g., acoustics, have not been integrated yet. Attempts to extend the standard or to define appropriate property and quantity sets for energy simulation tools have already been made, for example, by Chen et al. in [8] and by Bracht et al. in [3]. Similarly, domains involving underground facilities, such as tunnels, are yet to be fully developed [2,29]. Even the domains with good coverage cannot be regarded as semantically complete, because new technologies and methods enter the industry at a pace the IFC development cycle cannot keep up with. For example, the HVAC semantic types do not contain elements for thermal mass activation as a method of heating, e.g., for floor heating. These examples outline only some of the major challenges even a widely adopted and rigorously developed BIM standard, such as IFC, has to face on the road to Big Open BIM.

BIM Tools: There are software tools developed specifically for a particular domain (e.g., C.A.T.S.,[12] DDS-CAD,[13] Autodesk AutoCAD Mechanical[14] and Electrical[15] for the building services engineering field, Dlubal RFEM,[16] Tekla Structures[17] and AXISVM[18] for the structural engineering field), or for multiple domains (e.g., Autodesk REVIT,[19] Neme-tschek AllPlan[20] and ArchiCAD[21]). Each of these listed tools has its own, in most cases, closed data model. It is to be noted that these are only some of the most widely used software tools. In addition to them, there are many more, most of them dedicated to performing only a small subset of tasks within a single domain or project. However, in order to be "BIM-ready", each and every software tool needs to implement some form of BIM.

1.1. Problem statement

Due to its very active development, wide usage and continuing efforts to integrate more and more domains, IFC presents a particularly suitable example for exploring the still existing technical challenges and missing links towards a realization of Big Open BIM via data exchange standards that honor the multiple types of heterogeneity we mentioned above. In the following, we outline three technical challenges and some practical examples as an illustration of their practical implications (see also Table 1):

Table 1
Problem statement outline: types of heterogeneity that require integration.

No	Heterogeneity	Features	Issues	Integration implications
1	Semantic	Semantic discrepancy between domains or regulations	Can contain explicit and implicit, formal and informal specifications	The implicit and informal parts can lead to errors
2	Pragmatic	Implicit assumptions	Cannot be included in an integration specification	Lead to errors
3	Syntactic	Heterogeneity between standards, seldom within the same standard	Need for consistency checks	Easiest to automate, least potential to produce errors
4	Semantic modeling	Interlocking of syntax and semantics due to deficits in the tools for semantic modeling	The semantics becomes syntax-dependent	A change of syntax during data exchange can lead to a hidden change in semantics

[7] https://www.autodesk.com/techpubs/autocad/acad2000/dxf/ (last accessed 2020-11-13).
[8] https://www.steptools.com/stds/step/ (last accessed 2020-11-13).
[9] https://www.iso.org/standard/55257.html (last accessed 2020-11-13).
[10] https://www.khronos.org/collada/ (last accessed 2020-11-13).
[11] https://www.automationml.org/o.red.c/home.html (last accessed 2020-11-13).
[12] http://www.cats-software.com (last accessed 2020-11-13).
[13] https://www.dds-cad.net/ (last accessed 2020-11-13).
[14] https://www.autodesk.com/products/autocad/included-toolsets/autocad-mechanical (last accessed 2020-11-13).
[15] https://www.autodesk.com/products/autocad/included-toolsets/autocad-electrical (last accessed 2020-11-13).
[16] https://www.dlubal.com/en/products/rfem-fea-software/what-is-rfem (last accessed 2020-11-13).
[17] https://www.tekla.com/products/tekla-structures (last accessed 2020-11-13).
[18] https://axisvm.eu/index.html (last accessed 2020-11-13).
[19] https://www.autodesk.com/products/revit/overview (last accessed 2020-11-13).
[20] https://www.allplan.com/en/ (last accessed 2020-11-13).
[21] https://www.graphisoft.com/ (last accessed 2020-11-13).

(1) ***Explicit heterogeneity:*** Standards serving multiple domains are complex and rich, both syntactically and semantically. Both the syntax and the semantics are formally specified and, therefore, explicit. Inconsistencies in the syntax within the same standard are easily detected by various programming and validating tools. However, the presence of more than one syntax, as in IFC (the EXPRESS modeling language and XSD) can produce syntactic heterogeneity as well. The semantic specification, on the other hand, can contain, in addition to the formal definitions, informal ones in a natural language. For example, the specification of *IfcWallStandardCase* in the IFC specification [6] contains both a formal definition in the EXPRESS modeling language and an informal definition of the requirements for its geometric representation. Such definitions give room for interpretation, which can produce implicit heterogeneity between separate implementations of the same standard.

(2) ***Implicit heterogeneity:*** Even when working with a consistently implemented data exchange standard, there are still communication difficulties among domain experts due to inconsistencies in the modeling styles, or to different interpretations of the same concept within the standard [16,20,25,27]. This is the pragmatic heterogeneity. It is, by definition, implicit, i.e., without formal specification as it occurs in the mind of the user, and cannot be addressed in a purely automated manner.

(3) ***Interlocking of concerns:*** A feature common to both large and small standards is the interlocking of the syntax and the semantics. This involves the re-purposing of syntactic constructs to express semantics. For example, dynamic typing is often implemented as an object-type pattern that employs a syntactic referencing relationship to express semantic instantiating. This makes the semantics syntax-dependent and can be a hidden error source. We will take an in-depth look into this phenomenon in Section 3.3.

There are multiple practical implications arising from the three challenges listed above.

(a) *Implementation overhead:* The manual translation to and from a standard as semantically complex as IFC is challenging. In practice, for small pieces of software, developed from scratch to perform a single dedicated task (e.g., a thermal flow simulation tool in the building physics domain), there are simply not sufficient resources (e.g., finances, person hours) for such implementations. This leads to a large number of small tools offering very similar, often state-of-the-art, functionalities that are never used beyond the limits of one project. This results in the loss of domain expert knowledge for the AEC communities.

(b) *No real-time feedback:* Communication via data exchange standards often involves serialization of large amounts of data, which does not allow real-time feedback. The delay between user action and observable results is not measured in milliseconds, but in minutes or hours. For instance, this makes the fine-tuning of a building simulation extremely tedious and time-consuming.

(c) *No single source of truth:* The implicit heterogeneity we described above produces divergence in semantics between implementations and between models. Therefore, instead of a single "source of truth" there are multiple competing ones. This invariably results in translation errors when transferring information from one implementation or model to another [7,20], even within the same software family, e.g., exchanging walls with wall modifiers as tested by us.

To sum up, the practical implications listed above illustrate that reaching semantic as well as pragmatic consensus is a challenge. For better understanding of the issues involved as well as their interdependency and for motivating our approach as well, we present a practical example of the modeling of a wall in the following section, which also accompanies us as a running example throughout the paper. In Section 2.2.1 we will once again return to these issues and formulate three separate technical challenges, for which we will present our solution in the subsequent sections.

1.2. Motivating example

Let us consider the process of exchanging information about a wall between an architect and a structural engineer. Both model the same object, but consider different aspects of it. For the architect (and the building physicist) the wall is a layered construction with thermal, hygric, fire safety and other properties. For the structural engineer the wall is a structural member with a structural behavior within a system. In the IFC standard, there are elements that aid each of them in their modeling task, *IfcWallStandardCase* and *IfcStructuralSurfaceMember*, respectively. However, there is no formal mechanism for establishing that both concepts can describe different aspects of the same object, i.e., no possibility for effective integration. This is the semantic heterogeneity. In practice, each domain expert typically works in her own tool and the information exchange is relegated to a *BIM Collaboration Format (BCF)* file[22] referencing their respective IFC models.This is a topic also addressed by a data-driven method in the work of [28], which concentrates exclusively on the domains of architecture and structural engineering. The pragmatic heterogeneity occurs due to the assumptions of both stakeholders based on their respective point of view. For example, the structural engineer may view the two semantic concepts as complementary categories of the same object, whereas the architect may regard the one concept as an additional representation of the other. Since these points of view are not formally expressed, the difference is hidden and may give the impression of consensus where there isn't one.

In summary, this example involves both semantic and pragmatic heterogeneity. We will examine different solutions to the integration challenge in Section 3.3.

1.3. Contribution

Interoperability in BIM requires integration along both the semantic and the pragmatic dimensions. In this work, we present and evaluate a modeling framework capable of working with multiple semantic type systems inhabiting the same syntax in the context of multiple applications. In addition, the framework provides tools for modeling the pragmatic aspects of a data exchange and converting them from an implicit assumption into an explicit formal specification. The combined model of the semantics and pragmatics of a data model provides an *integration facade* that enables transparency and traceability during interoperability.

The paper is structured as follows. In Section 2, we review the related work and discuss open challenges we face. Section 3 presents our framework step by step by following the workflow for building an integration facade. In this section we describe relevant methods and techniques we employ for the realization of the framework as a prototypical implementation. Section 4 describes the design, evaluation, and results of our conducted case study on the basis of three practical cases. Section 5 concludes this work and outlines future work.

2. Related work

Much of the work on data exchange standards concentrates on bridging legacy, domain or technological gaps. There are several strategies for the implementation of such exchange that have been explored in the past, as shown in the following subsection.

[22] https://technical.buildingsmart.org/standards/bcf/ (last accessed 2021-02-26).

2.1. Data exchange strategies

Here we differentiate between data exchange strategies based on the type of "bridge" offered to cross a certain communication gap. An overview of those strategies is given in Table 2.

2.1.1. Data exchange via common syntax

Data exchange can take place via syntactic containers. Widely used formats employing this strategy include the Co-mma-separated Values (CSV), Extensible Markup Language (XML), the XML Metadata Interchange XMI and the Java Script Object Notation (JSON)–see the entry *common syntax* in Table 2. These formats are often used as storage for arbitrary data, because they offer no semantics of their own. Therefore, there is no interference with the semantics of the stored data. This makes these formats applicable for all domains. Moreover, transformations between them requires establishing only syntactic correspondences, not semantic ones. This transformation task can be automated easily. However, for processing the stored information correctly, each of the interacting applications must have implemented not only the same semantics but, in most cases, also the same serialization routine, e.g., a mandatory ordering of the information.

An example for the strategy of data exchange via common syntax is demonstrated in the Epsilon project [31], in the context of integrating legacy models into new technologies. In the course of this project, Paige et al. [23] implement interoperability between a proprietary modeling tool and an open source model management suite by providing a layer of communication drivers between a Java-based execution engine and a Component Object Model (COM) interface. This layer contains a dedicated driver for each persistence format (e.g., EMF,[23] XML, spreadsheets, etc.). In essence, it demonstrates a technique for loading and manipulating the same semantics by extracting its artifacts from different syntactic containers. Building upon this, the authors use a similar approach in another study, on the integration of Google Spreadsheets coupled with XML-based configuration models. In this study, they bridge the gap between the object-oriented and the table-row-column-oriented (or data-oriented) paradigms via a syntactic translation approach [13]. They do not need to consider the semantics of these formats (since there is none) in order to perform the translation.

2.1.2. Data exchange via common semantics

Implementing not only (*i*) the same semantics, but also (*ii*) the same serialization routine, becomes impractical for the extensive semantics typical for the AEC industries. In order to avoid (*ii*), the data exchange format has to incorporate semantic information in addition to the syntax. An example for this approach is the transfer of geographic data in the domain of urban design. The GeoJSON format builds on the purely syntactical JSON format by adding semantics-carrying keywords to its syntactic structures, e.g., *Feature*, *Point*, *Polygon*, etc. The DXF format, which transfers only geometric information, encodes the geometric semantics in context dependent numerical keys. For example, the key 0 indicates the beginning of an object definition. The key 10 in the context of a line indicates the x-coordinate of its first point in the context of a circle. This means it indicates the x-coordinate of its center. The IFC format also belongs to this category (see entry *common semantics* in Table 2). We already discussed some of its features in Section 1. It has two major advantages over the standards operating on pure syntax: It contains expert knowledge and is maintained by domain experts. This results in a certain level of complexity, which has practical implications. Its textual serialization, specified in ISO 10303-21,[24] contains a keyword for each entity, type, property or quantity set. Thus, for version IFC4 and above there exist over 1500 such keywords. An excerpt of this definition is shown below:

```
simple_entity_instance = entity_instance_name '='
simple_record ';'
simple_record = keyword '(' [ parameter_list ] ')'
entity_instance_name = '#' digit { digit }
```

A major challenge when exchanging data via common semantics is the translation between the common semantics and the internal data model of each application. Even when the data exchange standard is open, this translation remains hidden for the user and this may cause unintended results. For example, if the data exchange standard does not know a specific concept, it cannot transfer it. As an illustration of this drawback let us consider the following scenarios: First, the domain of urban design incorporates concepts for many plants. The IFC standard, however, does not. Therefore, each plant defined in an urban development tool is translated to *IfcProxy* in IFC4, which acts as a placeholder for unknown entities [5]. This loss of information cannot be remedied, even if the data exchange takes place between tools whose internal semantics incorporate the concept of plants. Second, a similar problem occurs when the data exchange standard's level of detail in the description of a concept differs from that of the internal data model of one of the interacting applications. For example, IFC4 differentiates between the material layers in a wall construction. Tools for calculating thermal flux within a wall construction, however, also differentiate between layers within the same material layer. If the translation from the tool to the standard uses the maximum flux value over all layers within a material layer, it will produce different results from a translation that takes the average flux value over all layers within a material layer.

There are even more complex translation problems, when no clear correspondence between data model elements can be established. For example, if the data exchange standard knows only the triangular mesh as the geometric representation of a surface, a Computer Aided Design (CAD) tool that works with Non-Uniform Rational B-Spline (NURBS) surfaces will have to perform triangulation before passing the surface information to the data exchange standard mesh object. Depending on the parameters and constraints of the triangulation, the resulting meshes of the same surface can differ so much between exports that they cannot be reliably identified as representing the same surface [10].

Ontologies: Operating on semantics brings us to the topic of the application of an ontology in the data exchange. IFC itself is based on an ontology, the buildingSmart Data Dictionary.[25] This approach was chosen by Akanbi et al. to address the problem of proprietary data formats resulting in errors and missing data during information exchange for automated cost estimation when employing different BIM software tools (compare to the second limiting factor under entry *common semantics* in Table 2). In [1], the authors present a novel data-driven method for automated quantity takeoff (QTO), since current methods do not address QTO from BIM artifacts created by different tools. For this purpose, they employ a data-driven reverse engineering algorithm to generate QTO algorithms for any building component by covering the variety of BIM representations via a mapping to IFC. The authors state that their approach "is neither an algorithm nor a software but a method for developing interoperable QTO algorithms". They argue that their approach is more robust than (traditional) approaches built on proprietary data formats, and therefore, has a higher level of support for interoperability.

In [25], the authors present an ontology-based approach to support change management as well as traceability of changes throughout all design stages. The authors argue that there are so many approaches presenting standardization methods for different file formats and exchanges, but still the document centric approaches result in parsing, interpretation, and serialization problems. Therefore, their approach builds upon

[23] https://www.eclipse.org/modeling/emf/ (last accessed 2020-11-13).
[24] https://www.iso.org/standard/63141.html (last accessed 2020-11-13).
[25] https://www.buildingsmart.org/users/services/buildingsmart-data-dictionary/ (last accessed 2020-11-13).

Table 2
Comparison of data exchange approaches.

No	Method	Examples	Features	Advantages	Limiting factors
1	Common syntax	CSV[a], XML[b], XMI[c], JSON[d]	Domain-independent, do not contain a specialized semantics. Instead, they offer a structure: a tree, a sequence of containers, etc.	(1) Can store any data and do not interfere with the data semantics. (2) Applicable to all domains. (3) Translations btw. such standards can be easily automated.	(1) The correct interpretation of the data requires that each tool has a separate implementation of a common semantics or the exactly same serialization routine. (2) This can become impractical for large domain-specific standards.
2	Common semantics	GeoJSON[e], DXF, IFC, ontology and description logic	Domain-specific, contain (parts of) the domain's semantics, e.g., a wall, a structural element, etc.	(1) Contain expert knowledge. (2) Can be maintained and extended by domain experts according to the requirements	(1) Depending on the level of detail included in the standard, it can become too large for efficient maintenance. (2) The implementation of the standard involves a translation between the standard and the software's data model, which can be challenging. (3) Missing semantic concepts require workarounds.
3	Copy of reality	digital shadow, digital twin	Multiple domain-specific standards are involved, e.g., architecture, geology, building physics, building automation, etc.	(1) Allows for completely automated information flow btw. real-time and its digital representation. (2) Enables automated as well as continuously decision-making. (3) Provides cross-domain traceability.	(1) Does not allow for abstraction and the digital shadow or twin can become exceedingly large (Tera- or Petabytes in size). (2) The communication between the different domain-specific standards requires a dedicated structure for both hardware and software middleware. (3) Requires a dedicated communications standard.
4	Common knowledge	calculation and simulation methods	Can be based on any of the standards listed above.	(1) Contain operational expert knowledge. (2) Can be used for validation and compliance checking.	(1) Operate on their (often implicit) bespoke data models and are inaccessible for most applications. (2) Implemented in different formal languages and on different platforms.
5	Common data	data base, any domain-independent or domain-specific standard	Can contain any type of monitoring or simulation data, often time-stamped, can be used for automatic semantic classification	(1) Supplies information about the actual state of the corresponding physical object, which can be used by a digital shadow or twin. (2) Arbitrarily fine-grained (values per hour or per millisecond). (3) Data model is rarely complex. (4) Can be reverse-engineered to give insight into an unfamiliar system.	1) Can become very large (Tera- or Petabytes in size). (2) Its structure can be completely arbitrary, and therefore, harder to integrate with any other data exchange standard or model. (3) Can be one-sided and skew the performance of an algorithm based on it.

[a] https://tools.ietf.org/html/rfc4180 (last accessed 2020-11-13).
[b] https://www.w3.org/TR/xmlschema11-1/ (last accessed 2020-11-13).
[c] https://www.omg.org/spec/XMI/ (used as a standard persistence format for the Universal Modeling Language (UML) since 2000 - see https://www.omg.org/spec/UML/About-UML/) (last accessed 2020-11-13).
[d] https://www.ecma-international.org/publications/standards/Ecma-404.htm (last accessed 2020-11-13).
[e] https://tools.ietf.org/html/rfc7946 (last accessed 2020-11-13).

semantic web technologies and ontology engineering. Thereby, they shift the problem of exchanging artifacts to the management of knowledge graphs (ontologies). The authors are convinced that semantic web technologies, such as ontologies, reasoning, and SPARQL[26] queries efficiently manage interrelated project information. Their approach parses building information from IFC files, ontologies from the IFC schema, and rule-sets describing implicit knowledge. In the evaluation they successfully show that calculations as well as inferences computed in different tools by different users can be explicitly described in an interoperable manner. For this purpose, the authors make use of the latest developments of ifcOWL.[27]

When choosing an ontology-based approach for semantic integration, assuming that the domain of interest can be specified in sufficient detail, semantic heterogeneity could be overcome by logical inference. This requires a special class of logics, the so-called description logics. For this purpose a sufficiently detailed ontology has to be present, so that the elements of the export schemata of the involved data sources could be described by logical formulas. Additionally, it has to be possible to represent a global query as a formula covering the concepts that are formally well-defined in the ontology. The data sources relevant for the answer can then be determined by automatic inference. However, this means that such an ontology-based approach for semantic integration is essentially based on local-as-view inclusion relationships, since concepts of data sources are described by a global ontology [30]. In [21] the authors describe a filter implemented as multiple ontologies, one common ontology over all domains and multiple domain-specific ones. This setup allows for a flexible interoperability model.

In [14] five criteria for ontology design can be found: clarity, coherence, extensibility, minimal encoding bias (i.e., clear separation between syntax and semantics), and minimal ontological commitment (i.e., finding the minimal set of terms essential for knowledge communication). This kind of approach is certainly promising and we plan to examine it and compare it to the one we present in our future work.

2.1.3. Data exchange via a copy of reality

The typical BIM model is a proper model according to Kühne's definition introduced in [19]: it is based on an original that already exists or is going to exists (e.g., a building), it is an abstraction of the

[26] https://www.w3.org/TR/rdf-sparql-query/ (last accessed 2020-11-13).
[27] https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/ (last accessed 2020-11-13).

original (e.g., many details are omitted), and it can represent it under certain circumstances (e.g., during the planning process). However, there are developments in the AEC industries aiming at producing *digital twins* of buildings [17]. From a BIM perspective, a digital twin is a BIM model without any abstractions, i.e., a copy of reality and this implies that each software that handles the digital twin has to implement the "semantics of everything".

An example of this development are the numerous attempts to expand the IFC data model by adding more and more detail in various domains [2,9,29]. Many attempts disregard the International Framework for Dictionaries (IFD) mechanism for defining taxonomies for IFC, and consequently, without knowledge of each other, run the risk of producing redundancies and conflicting concepts [7]. As presented in the works of ([2,29], there are not just multiple ways to define a tunnel and its infrastructure, but also multiple possible docking points for new elements in the existing IFC data model. The newest version of IFC, IFC4.3 [6], already contains the basis for infrastructure elements, e.g., *IfcFacility*, *IfcBridge*, in preparation for the definition of, e.g., tunnels. In addition, there is potential for conflict between domains. The data model presented in [2] defines a new subtype of *IfcGeometricRepresentationItem*, the so-called *ProceduralModel*. This model is used to serve as representation of a *TunnelElement*. Another domain that relies heavily on procedural geometry is urban design. This would require a visible and traceable method for establishing a semantic relationship between the concepts of procedural geometry in underground facilities as well as in urban areas.

The developments described above demonstrate the complexity resulting from overlapping semantic and pragmatic heterogeneity.

2.1.4. Data exchange as knowledge communication

The interest in interoperability in the AEC industries includes not just data but also methods for its manipulation. For example, the calculation of the energy efficiency of a building design involves thermal simulation algorithms. New algorithms are routinely developed in the course of research projects and have the potential to bring significant technological advancement, if they could be applied in the industry. However, most of these algorithms operate on their own bespoken data models, and therefore, are inaccessible for other applications (see Section 1.1, practical implication (a)).

Aside from creating a dedicated application, there are multiple approaches to algorithm development in the context of the AEC industries. One of them is the definition of prototypes in applications such as MatLab[28] and Modelica.[29] However, such prototypes are generally only accessible via an Application Programming Interface (API) that only few proprietary software integrate. An additional approach is the use of spreadsheets and small code snippets, e.g., in Microsoft Excel™ and Visual Basic for Applications (VBA), respectively. The drawback of this approach, as well as the previous one, is that both offer no semantic support. A cell in a spreadsheet is a container that holds arbitrary information, which can result in misinterpretation and, ultimately, misapplications of the algorithm. For example, a cell whose upper neighbor holds the text "adjacent zone" could contain either the name of a neighboring thermal zone in a building simulation, or its identifier. Moreover, it gives no indication how the neighbor relationship between zones is to be determined. This example demonstrates that a single cell can potentially lead to multiple interpretation errors due to both semantic and pragmatic heterogeneity. In an additional example, the building automation domain uses the Building Automation Networks (BACnet) protocol[30] to define interoperability between sensors and actuators as well as the control algorithms of components [18]. VDI 3813[31] and VDI 3814[32] define the elements of the standard BACnet workflow. However, BACnet has yet to be adopted as part of BIM, and even if it were fully integrated, it would encounter the same problems we described in Section 1.1.

2.1.5. Data-driven data exchange

This method makes use of the vast amounts of data in the AEC domains as basis for analysis, pattern detection and, ultimately, development of algorithms for automated semantic similarity detection.

In [28] the authors present a data-driven IFC based approach for identifying the same object when transferring information between the architecture and structural engineering domains. The identification process concentrates on features common to both domains, the geometry and the material of the object. In essence, the authors present a method for automated detection and integration of pragmatic heterogeneity.

Petrova et al. couple an ontology persisted as a graph database with numerical data, e.g., geometric, from simulations or monitoring, into a framework that links the graph to the data [24], but does not integrate it into the ontology. In this way, different ontologies can operate on the same data and effectively use it as a communication medium. The drawback of this method is the potential for significant unaddressed pragmatic heterogeneity as the link between an ontological node and the data can be interpreted in a number of different ways.

2.2. Road map: towards full semantic and pragmatic integration

So far, we have listed technological challenges and solutions stemming from different types of heterogeneity. In this subsection we will consolidate our findings and formulate the challenges that need to be addressed to achieve full semantic and pragmatic integration into a *facade* that contains all prerequisites for interoperability not only in BIM but also in any other data exchange context.

2.2.1. Challenges

Full interoperability requires uninterrupted multi-directional information flow through an arbitrary number of applications. The full integration of the underlying data models, or standards, functions as the backbone of this process. In Section 2.1 we examined various data exchange strategies relying on different types of integration. Thereby we established that one vital prerequisite for exchanging not just data, but information, is the semantic and pragmatic consensus.

In an environment as diverse and complex as that of the AEC industries, there is a large volume of explicit domain knowledge distributed over multiple data models. After all, each software tool employed in the AEC industries considers at least one part of the semantics of one or multiple domains.

Considering the issues of both explicit and implicit heterogeneity as well as the interlocking of concerns we discussed in Section 1.1, we can formulate the following challenges on the road to "robust" integration, which we address in this work:

> **Challenge 1**: *Exposing the semantics*. The semantics of a software typically resides within its type system, or data model. In order to interact with a software via another software, we need to discover its semantics, not manually but automatically. Furthermore, we need to make that semantics available in real-time. This requires the ability

[28] https://www.mathworks.com/products/matlab.html (last accessed 2020-11-13).
[29] https://www.modelica.org/ (last accessed 2020-11-13).
[30] http://www.bacnet.org/ (last accessed 2020-11-13).
[31] https://www.vdi.de/richtlinien/details/vdi-3812-blatt-1-automationsfunktionen-fuer-wohngebaeude (last accessed 2020-11-13).
[32] https://www.vdi.de/richtlinien/unsere-richtlinien-highlights/vdi-3814 (last accessed 2020-11-13).

to produce artifacts conforming to it, e.g., valid instances of the semantic types, containing user input on demand.

Challenge 2: *Exposing the functionality*. In addition to discover the semantics of a software, we need access to its functionality, e.g., the methods that trigger all relevant algorithms. This requires the ability to call said functionality with all necessary data with as little additional effort as possible.

Challenge 3: *Staying up-to-date*. Most standards and software in use undergo updates for various reasons, often including their semantics. For example, a change in an algorithm may require a change in the data model. We need to be able to react to such updates as soon as possible. Update cycles of several years, e.g., typical of the IFC standard, place a massive burden on the validity as well as maintenance of already established information flows.

Challenge 4: *Making the pragmatics explicit*. The pragmatics of a standard, concept or software resides in an implicit from in the mind of each user. We need to provide a mechanism for exposing and discussing the pragmatics, in order to reach consensus.

The challenges listed above address a seamless interaction considering the semantics and pragmatics of a single application. Nevertheless, full interoperability requires an unbroken communication network involving multiple applications. This, in turn, requires effective interaction among semantics, which may have partial or full overlap, and between pragmatics that may be contradictory. This brings us to additional challenges concerning (*i*) *translation between semantics*, (*ii*) *translation operations* in cases where there is no one-to-one correspondence between semantic concepts, and (*iii*) *automation of the translation process*. These challenges we will address in our future work.

3. Approach

Our approach hinges on the ability to model and manipulate semantic information in a manner that in no way interferes with it. This means that the syntactic container holding such information does not include any additional semantics, but instead, allows depicting structure in an abstract way, similar to XML or JSON. In addition, unlike XML or JSON, the container is modifiable in real time. This makes model-driven engineering a fitting technique for realising our approach.

3.1. Model-driven engineering: preliminaries

Based on Martin Fowler's classification[33] models are used as: [(1)].

(1) *sketches* for communication purposes, where only partial views of an artifact are specified;
(2) *blueprints* to provide a complete and detailed specification of an artifact;
(3) *programs* instead of code for the development of an artifact.

This means that during development, a team uses models in several different ways as abstraction of reality, which makes the design of an artifact (e.g., software) a model-driven process. Therefore, models are crucial for understanding and sharing knowledge about a domain of interest. *Model-driven Engineering* (MDE) transforms models into so-called "first-class citizens" in the field of software engineering [4]. The purpose of MDE in the software engineering domain ranges from communication between different stakeholders to the executability of the developed software.

According to [4] there are two main concepts in MDE: *models* and *transformations*. The latter is used for employing manipulation operations upon models. The notation for expressing both concepts is known as modeling language. There are different layers of abstraction in MDE. From a bottom-up perspective there are: (*i*) M0, which contains run-time instances of the defined model elements of the next higher layer M1; (*ii*) M1, which describes the domain of interest by a domain model and defines the language describing the semantics of that domain; (*iii*) M2, which defines a modeling language (e.g., UML,[34] SysML,[35] or a domain specific language) for specifying domain models in M1; and the layer (*iv*) M3, which defines a so called meta-language for specifying a metamodel such as MOF.[36] It is essential to note, that the concept of the metamodel in the MDE domain differs considerably from the homonymous concept in the AEC domains, where it is sometimes used to describe correspondences or mappings (see [3]). In the MDE domain, a model is an instance of some more abstract model, or metamodel. This is the reason why we could define an infinite number of model levels. However, a model has to *conform to* its metamodel, which means that all its elements can be expressed as instances of the corresponding elements of the metamodel, as shown in Fig. 1 and implemented in our framework presented in Section 3.2. In a nutshell, a modeling language is a tool that supports engineers in specifying models, be it in graphical or textual representation.

Transformations are used for mapping between models specified at any level. For example, in model-driven software engineering, transformations are used for the automatic transformation of model elements (M1) to corresponding code statements (also M1), which could be executed at a platform and produce run-time instances corresponding to the model run-time instances (M0). Generally, transformations are defined at a model level higher than the level they are applied on [4]. Transformation rules could be defined manually from scratch, or by defining specific mapping rules by means of a model transformation language such as the Atlas Transformation Language[37] (ATL), which is the most widely used rule-based transformation language, both in academia and in industry. ATL contains a mixture of declarative as well as imperative constructs, is uni-directional (i.e., transformation from language A to language B), and transforms *read-only* input or source models into *write-only* output or target models [4].

In the context of this work, we employ the described MDE techniques for extracting a representation of the semantics of any of the involved applications, and for producing valid instances of the types necessary for executing a function call within it. Thereby, MDE enables us to make the sources of semantic conflicts between applications explicit, and to provide a reproducible path towards their resolution. In addition, it provides tools for expressing and negotiating pragmatics. In our case study presented in Section 4, we examine the feasibility of a direct information exchange between applications via our MDE-based framework. We forego of

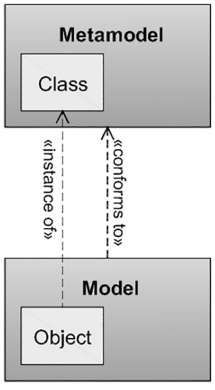

Fig. 1. Relationship between Metamodel (M2) and Model (M1) [4].

[33] https://martinfowler.com/bliki/UmlMode.html (last accessed 2020-11-13).

[34] https://www.uml.org/ (last accessed 2020-11-13).
[35] https://sysml.org/ (last accessed 2020-11-13).
[36] https://www.omg.org/mof/ (last accessed 2020-11-13).
[37] https://www.eclipse.org/atl/ (last accessed 2020-11-13).

employing serialization or using shared memory. Instead, we compile a minimal set of preconditions, coupled with an estimation of the required effort, that need to be met by the involved software to ensure that all input- and output-relevant data structures are exposed as public types, and that there is a suitable *entry point*. Finally, we evaluate the implications of the presented framework for the data exchange between multiple domains, such as building physics and structural engineering.

3.2. The modeling framework for integration facades

Our approach can be demonstrated by a workflow incorporating our central modeling framework that allows us to overcome the challenges we formulated in Section 2. In this section, we give a broad overview of the workflow as well as the framework and take a deeper look at the framework's modeling language. We will elaborate on other aspects of our approach step by step in the course of evaluating our case study results in Section 4.

3.2.1. The workflow

Fig. 2(b) shows the structure of our modeling framework on the left, and the type structure of an application written in an object-oriented programming language on the right, as an example. The two structures are similar. Both require a language for describing the structure. In the target application, the instances of the language build a type system. In most cases this is the semantics of the application. The language provides the syntactic containers to hold the semantics. For example, the C# programming language allows us to create classes. The *class* as a syntactic container resides in the C# language (see the box *Language* in the top right corner of Fig. 2(b)). Class *Person*, on the other hand, is a syntactic instance of *class* and carries semantics representing the concept *person* in the real world. Class *Person* would occupy box *Types* in Fig. 2(b). The same relationship between syntactic containers and semantics holds true in the modeling framework. However, in it, the instances of the language build a model. The *Component* as a syntactic container resides in the language of the modeling framework (see the box *Language* in the top left corner of Fig. 2(b) and the top element in Fig. 2(a)). Metamodel *Person*, in analogy to class *Person*, is a syntactic instance of *Component* and also carries semantics representing the concept *person* in the real world. Metamodel *Person* would occupy the box *Metamodel* in Fig. 2(b).

As we elaborated in **Challenge 1**: *Exposing the semantics*, in order to be able to work with the semantics of any application, it needs to be discovered first. Steps 1, 2 and 3 of the workflow depict the discovery process. Fig. 2(b) also shows the required interfaces. In order to access the target application's type system without the help of the source code, we need a reflection mechanism or a dedicated API. This is what the first step accomplishes - retrieving information about types, e.g., **T1** (see steps 1 and 3 in Fig. 2(b)). In the next step, using a syntax mapping interface integrated into the modeling framework, or its API, we choose the appropriate language elements and build a model of the discovered type system, e.g., the metamodel **MM1**. Finally, in step 3, we establish a link between each type and its corresponding metamodel. However, this is not yet sufficient to overcome **Challenge 1**.

The run-time data resides not in the type system of the target application, but in its instances. For this reason, the modeling framework enables the construction of a model of each instance, which can receive user input. This is made possible by the syntactic relationship *instance-of* implemented by the language of the modeling framework (see relationship *OfType* between *Typed Component* and *Component* in Fig. 2(a)). The models of the instances of the target application are constructed as syntactic instances of the metamodel corresponding to its type system (see the box *Model* in Fig. 2(b)). This enables step 4, in which the user edits the model of each instance. Step 5 uses the established association between each type of the target application and a metamodel to produce models of the run-time instances of the target application's type system. Those models can inject their information into the target application (step 6), interact with instances produced only in the target application (step 7) and supply data for function calls on the target application (step 8), which result in valid output (step 9). This allows us to overcome **Challenge 1**.

At this point, let us take a deeper look into the language of the modeling framework. The relevant excerpt is depicted in Fig. 2(a). Its core consists of two classes, *Component* and *Parameter*. A component can contain an arbitrary number of parameters and other components. It can also reference other components, be an instance of or the representation of another one. In particular, the syntactic relationships our modeling framework can accommodate are the following:

(1) **Association** (unidirectional or bidirectional). This relationship allows coupling, or referencing, between model elements without restrictions. See *ref. Components* and *ReferencedBy* in Fig. 2(a).
(2) **Containment**. This relationship ensures that one element is completely contained in exactly one other element or not contained in any element at all. See *Subcomponents* and *ContainedParameters* in Fig. 2(a).
(3) **Instance-of**. This relationship allows one element to be declared as the type of another. See relationship *OfType* between the classes *Component* and *Typed Component* in Fig. 2(a).
(4) **Representation-of**. This relationship allows one element to be declared as the representation of another. See relationship *Representing* between the classes *Component* and *Typed Component* in Fig. 2(a).

This language allows us to build a model corresponding to any semantic type. *Component* can model reference-type elements (e.g., a class, which is addressed by reference or pointer), *Parameter* can model value-type elements (e.g., an attribute of type integer, which is addressed by value). Both *Component* and *Parameter* as well as each of the above listed syntactic relationships can act as a carrier for arbitrary semantics. In fact, the differentiation between those four kinds of relationships is motivated mainly by convenience. What we truly need is simply the concept of two syntactic elements having a relationship. The specification of its type can be left entirely to the semantics of its content. For example, in this case, composition relationships between classes are expressed as one component being a sub-component of the other, i.e., a true part of it. Association and aggregation relationships, on the other hand, are expressed as one component referencing another. Bidirectional associations require each component to reference the other. The relationship modeling the one between a class and its instances is contained in the *OfType* relationship container. Finally, the relationship modeling the one between a real-world concept and its representation in a model is contained in the *Representing* relationship container. An example of the application of the language of our framework to model class *Person* can be found on our project website.[38]

We will examine the modeling of classes and their instances in more detail in Section 4. Now we turn to the methods of a class and the functionality they implement. Overcoming **Challenge 2**: *Exposing the functionality* depends entirely on the target application and the methods of its types. Unless we misappropriate the reflection mechanism to call private methods, we are dependent on its public ones. Moreover, we need to be able to call them with objects, or instances, as input parameters, to enable real-time interaction. This is step 8 of **Workflow 1**. We will demonstrate two approaches to overcoming this challenge in Section 4, when we discuss specific use cases and programming styles.

Challenge 3: *Staying up-to-date*, on the other hand, is met by steps 1 to 3 already. As the discovery process is fully automated, any change in the semantics of the target application on the right translates into an

[38] https://cdl-mint.se.jku.at/artefacts-for-semantic-integration-for-big-open-bim/##class-Person.

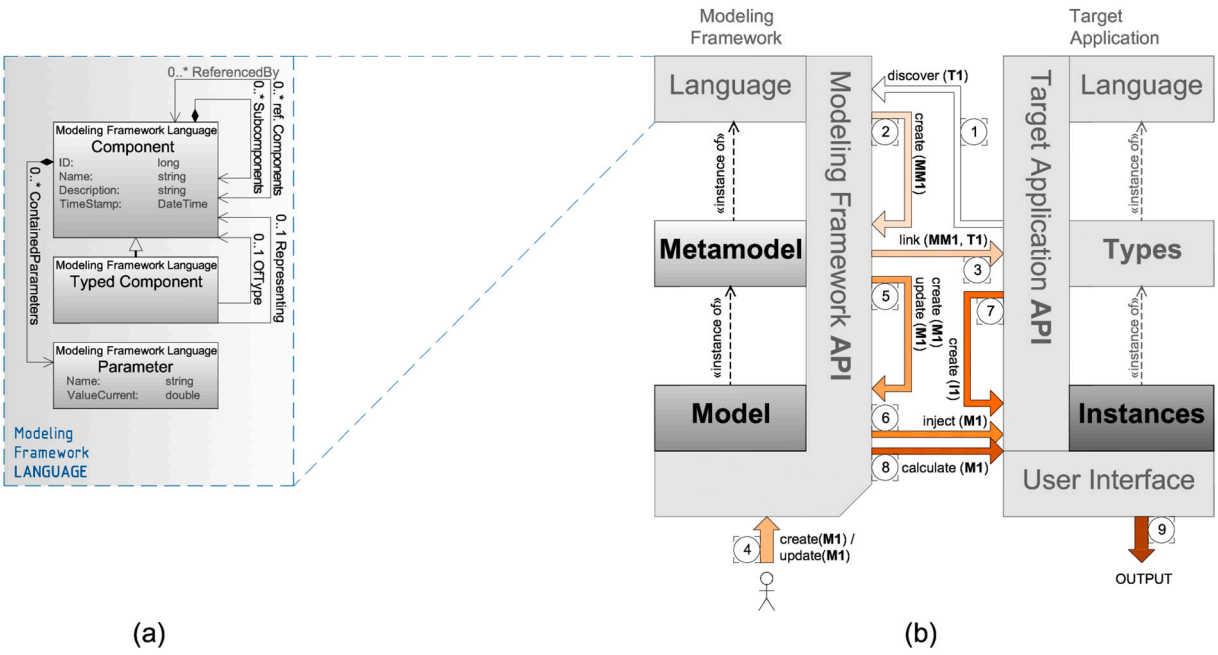

Fig. 2. (a) The language of the modeling framework and (b) Workflow 1, which answers Challenges 1 and 3.

adapted metamodel on the left. This, in turn, enables the subsequent steps to proceed with updated instances. Only step 4 may require more attention from the users, as they would have to comprehend the update and adapt to it.

The modeling framework offers all tools to meet **Challenge 4**: *Make the pragmatics explicit* since we can enrich the model of the type system of the target application by additional model elements and relationships to add the pragmatics to the semantic model and produce a true *integration facade* for the target application.

Addressing challenges 1 to 4 will allow us to perform a loss- and distortion-free translation considering both semantics and pragmatics, even in cases where there is no one-to-one correspondence between concepts. We will examine the workflows involved in translation between multiple source and target applications and present their evaluation in our future work.

3.3. Integration facades

Our approach, as described in the previous sections, provides *integration facades* for data models that include semantics and pragmatics. It is, therefore, necessary to differentiate between semantics, syntax and representation. These three dimensions are generally handled within the same data model, as exemplified by IFC (see Section 1). A separation can significantly simplify the integration process. The pragmatics, on the other hand, consists of implicit assumptions without a formal expression. We will take a closer look after separating the three explicit dimensions. In Fig. 3 (a) we do just that: semantics are depicted along the *ontological axis*, syntax - along the *linguistic* axis, as it is the formal language of the data model, and the representation of an object of the real world is depicted along the *conceptual axis*. The latter one is where we cross over from abstract modeling into the real world. In Fig. 3 (b) we show the typical case when we model a real-world concept, e.g., that of a wall construction. This concepts is represented by a model element, e.g., *Ontological Type Wall Construction*. This type can be instantiated as an object in the model, in this case *Object wc01* to represent one specific object in the real world, e.g., one specific wall construction that we can interact with in the real world.

However, real world objects are often part of the extension of more than one concept [19]. In our case, the same physical object can be regarded simultaneously as a wall construction and as a vertical slab, as shown in Fig. 3(c). In other words, it has multiple aspects, or it can be viewed from different points of view as discussed in our motivating example (see Section 1.2). On the other hand, the typical object-oriented modeling language does not allow the instantiating of multiple concept representations into the same modeling object. Fig. 3 (d) demonstrates that the *instance-of* relationship is defined only between one class and one object. Therefore, the typical modeling scenario involves modeling each concept by a separate ontological type, instantiating of each type into a separate ontological instance and modeling a connection between them by applying various software patterns. In this case, objects **wc01** and **vs01** can be used as dynamic types in an object-type pattern, either to assume the role of a dynamic type for the other or to provide multiple dynamic typing for a third object.

In either case, such workaround misuses syntactic tools (the syntactic association relationship) to model semantics (the semantic instance-of relationship). Therefore, a change in the syntax in the process of data exchange can inadvertently cause a hidden change in semantics that can be very difficult to trace and needs handling on a case-to-case basis, depending on the syntax used by the data models of interacting tools. For example, if, during data exchange, we transition from a syntax that allows dynamic associations to a syntax that allows only association queries on demand, the semantic information stored in that association will not always be present to restrict the instance behavior as a true static type would. This would completely remove type safety from the translated data model.

The modeling language we use in our approach offers several additional modeling constructs that avoid this type of intermixing of syntax and semantics. Fig. 4(a) shows an attempt to add a semantic connection to object **wc01** and **vs01** by adding an additional semantic abstraction level along the ontological axis. We declare that *Ontological Type Wall Construction* is an instance of *Ontological Type Layered Design*, *Ontological Type Vertical Slab* - an instance of *Ontological Type Structural Design*, and that both are specialisations (or sub-types) of the *Ontological Type Design* (see Fig. 4(b)). This would allow to establish that **wc01** and **vs01** have a common type and, in that type's definition, that they represent aspects of the same real-world physical object. Since the traditional object-oriented paradigm allows only one level of ontological instantiating, between a class and an object, we need a different construct to allow

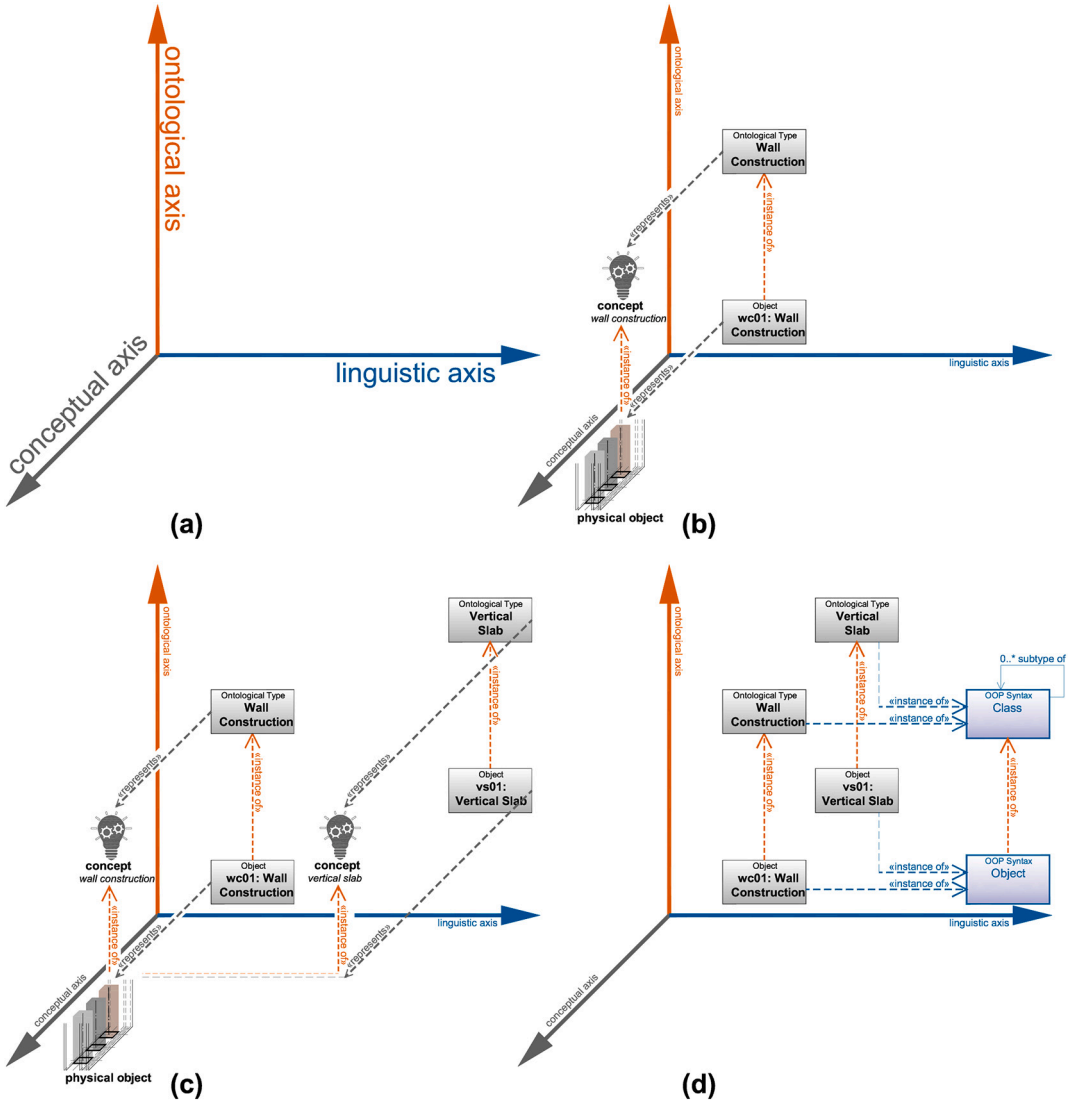

Fig. 3. The three axes of integration. (a) ontology, linguistics and conceptual thinking (b) a typical relationship between a concept and its model (c) the same object classified as different concepts (d) a typical relationship between a model and its formal language.

multiple levels of instantiating. Fig. 4(b) shows that we cannot actually instantiate all ontological types we need by employing the object-oriented approach alone. Fig. 4(c) demonstrates the model-driven engineering approach. The *Typed Component* element allows an arbitrary number of instantiating steps via the relationship *Of Type*. Thus, *Layered Design*, *Structural Design* and *Design* can all be syntactically constructed as syntactic instances of *Typed Component* and semantically connected along the ontological axis.

The declaration of additional types we described above is an example of handling the pragmatic dimension of a data model by making one possible implicit assumption explicit, e.g., that *Wall Construction* and *Vertical Slab* are aspects of the same concept (see the motivating example in Section 1.2), and integrating it in the model of the semantics, even if it is not expressed in the type system of the target application.

The pragmatics can be expressed along the conceptual axis as well, since our approach allows representational relationships along the conceptual axis. Fig. 4(d) shows a different type of relationship between the ontological types *Wall Construction* and *Vertical Slab*, via the *Representing* relationship declared between different syntactic instances of element *Typed Component*. Here, the pragmatics of another user can be made explicit by stating that the vertical slab is regarded as a representation of the wall construction (in a particular context). The situations depicted in Fig. 4(c) and (d) can be regarded as conflicting pragmatics made explicit. This, in turn, makes it possible to resolve the conflict as part of the model and to produce a coherent integration facade of the target application.

Finally, our approach allows us to confine the data model syntax purely to its role as a linguistic tool and to model ontology and representation separately from it. This removes the syntax of the data model from the list of potential semantic error sources during translation between data models.

In the next section, we present an embedded case study consisting of three cases, or units of analysis, on the basis of the guidelines of [26]. The case study enables us to evaluate the approach we presented in this section in a real-world environment, and to examine the separated workflow steps in detail.

4. Case study

The prototype of the modeling framework, we introduced in Section 3 is implemented as a Windows Presentation Foundation (WPF) application in C#. Therefore, we focus our attention on software written in C#, or on applications that offer a C# API for direct access (e.g., Microsoft Excel™).

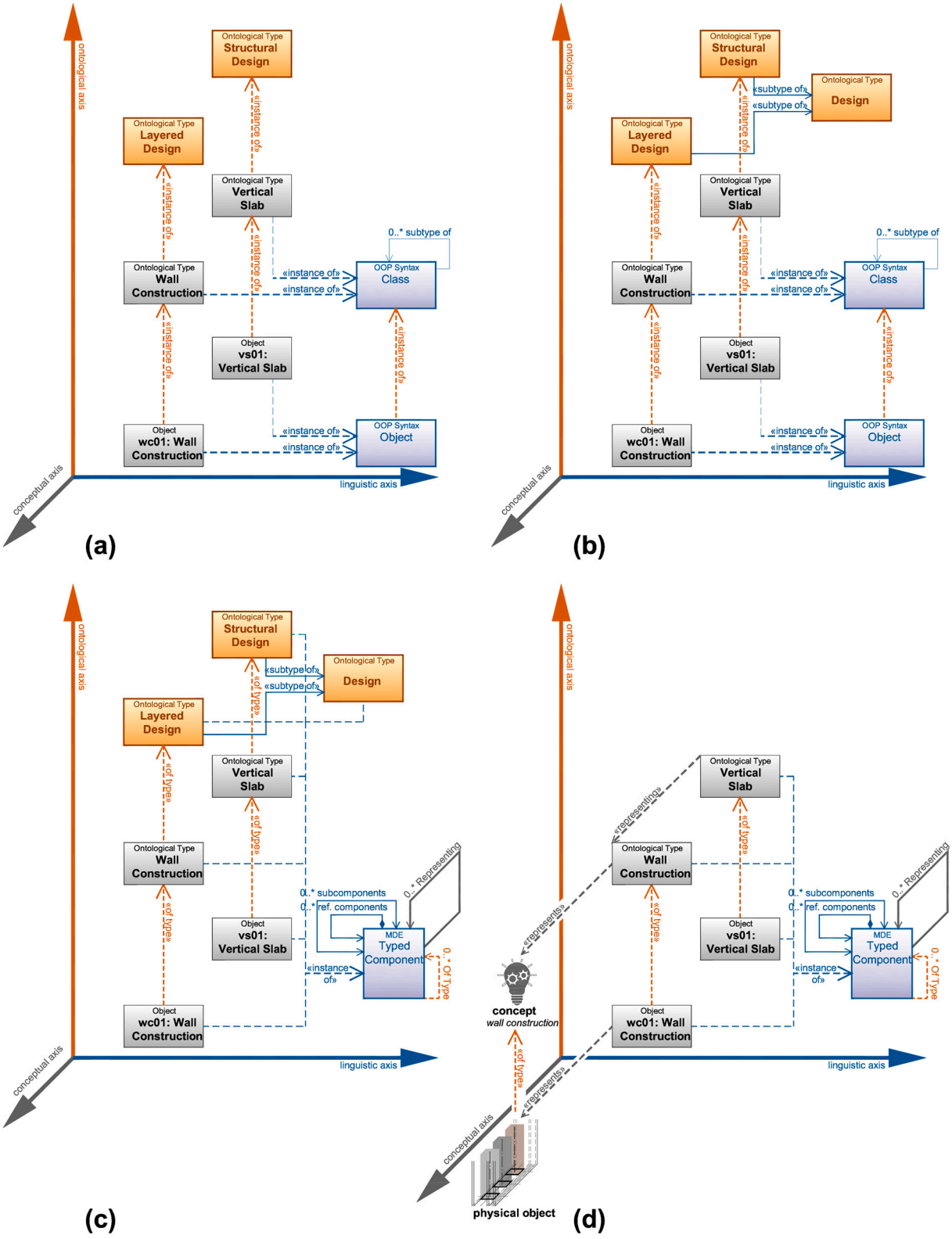

Fig. 4. The three axes of integration continued. (a) adding ontological types (b) connecting the ontological types via a common super-type is not possible with the typical class-object syntax (c) realising the additional types via the syntax of our modeling language (d) realising a representational relationship between ontological types along the conceptual axis.

4.1. Design

We chose small non-web based simulation software, which is typical for simulation tools in the AEC domains. Usually, such tools are prototypically implemented as part of a research project, but seldom advance from this stage into a product one. Furthermore, as we outlined in Section 1.1, maintaining them in the context of very complex and constantly evolving data exchange standards is not cost-effective. According to our own online search, such tools are effectively not available beyond the duration of the project they accompany.

For this reason, we have chosen two simulation applications available on the website Code Project[39] that closely mimic the typical features of simulation software in the field of building physics. The first one is a gravity simulator, and the second a sound wave propagation simulator (from the acoustics sub-domain of buildng physics). Both handle input and output via a Graphical User Interface (GUI). The third application is a Microsoft Excel™ tool for calculating the temperature, humidity and CO2 concentration in a single space developed by domain experts at TU Wien [22].

4.1.1. Case 1: a windows forms application simulating particle motion under gravity

This software simulates the motion of a swarm of particles, each with an initial mass and velocity, under the influence of gravity. The result is represented by an animation on a two-dimensional canvas and as a table containing mass, positions and velocities. Both in its functionality and in the presentation of the simulation results, it resembles existing tools in the AEC industries, e.g., a tool for the calculation of light distribution. The main difference lies in the input handling. This tool allows the user to set the initial position of each particle per mouse-click, which makes the result dependent on the user's hand movements. However, simulation tools in the AEC industries aim to deliver reproducible results. For this reason, they require predictable input, most commonly in the form of a human-readable text file. Therefore, a first step in adapting a tool with an interactive user interface would be to replace the imprecise interaction by precise textual instructions in an input file. This requires the tool to implement custom serialization, which is a non-trivial task depending on multiple factors, e.g., the tool's data model complexity. In any case, a direct communication with the application's data model is not possible.

4.1.2. Case 2: a windows presentation foundation application simulating the propagation of sound waves

This software simulates the propagation of sound waves generated by user-defined sound emitting point or line sources. The resulting interference pattern is calculated numerically over a discrete grid and is displayed as an animation on a canvas in one, two, or three dimensions. Its functionality is very similar to existing tools in the AEC industries. There is an entire sub-domain in building physics dedicated to sound protection, which employs similar methods for the calculation of the sound-proofing properties of materials and constructions. As in the previous case, the main difference, from the viewpoint of the user, is the input handling. Sound emitters and sound blockers are defined by mouse-click. An exact numerical definition in a text file is not possible, as the tool is not equipped with a custom serializer.

Cases 1 and 2 will demonstrate each step of **Workflow 1** in detail. In particular, we will focus on the part of the implementation that addresses **Challenge 1**: *Exposing the semantics* and **Challenge 2**: *Exposing the functionality*.

4.1.3. Case 3: a Microsoft Excel™ application simulating the thermal behavior of a single space

This case takes a deeper look at a Microsoft Excel™ appli-cation developed as part of a research project in the domain of building physics [22]. It determines the temperature, humidity and CO2 concentration in a single enclosed space over the course of a week during a heat wave. In addition, it takes climate, wall construction, orientation, user behavior, and building services into account.

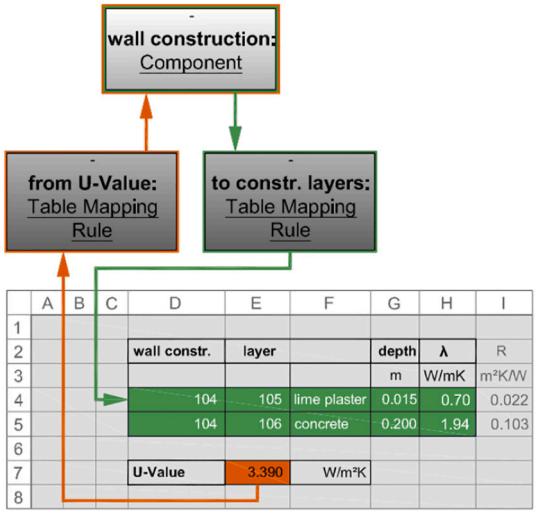

Fig. 5. Case 3. A sample calculation of the U-Value of a wall construction consisting of two layers in an Excel sheet.

Since this configuration is far too complex to depict here, we have used a small example of one possible configuration of the input and output as a representation of the results we obtained in this study. The bottom part of Fig. 5 shows an Excel sheet prepared for the calculation of the U-Value[40] of a wall construction consisting of two layers. The input cells are green, with each row corresponding to a material layer in the wall construction. The output cell is orange. The green and orange arrows indicate the input and output information flow, respectively. Both the source and the target of the information flow is a wall construction object. In the full case configuration, the input is distributed over multiple sheets and multiple cell ranges within each sheet. The simulation routine is written in a global module in VBA and is called by clicking a button. The output is a time series saved in a dedicated output sheet. The application supplying the input BIM model is object-oriented and defines buildings, spaces, walls etc. as objects.

This case demonstrates the steps involved in addressing **Challenge 4**: *Making the pragmatics explicit*. In particular, it focuses on the expression of the pragmatics as a representational relationship. This means it shows the mechanisms involved in declaring one semantic concept as representative of another.

4.2. Evaluation focus

In all three cases, we will make adaptations in the source code and evaluate both the adaptation itself and the implementation of the workflow described in Section 3. During the evaluation, we will underline the aspects that directly address the challenges we listed in Section 2.2.1. Since **Challenge 2**: *Exposing the functionality* is entirely dependent on the target application, we will answer the following additional questions:

CQ 1. What amount of effort, measured in change in program length as defined in [15], is necessary to expose an alternative entry point in an existing software that accepts an object of arbitrary complexity as input?

[39] https://www.codeproject.com/ (last accessed 2020-11-13).

[40] EN ISO 6946:2017 https://www.iso.org/standard/65708.html (last accessed 2020-11-13).

CQ 2. What amount of effort, measured in change in program length (see above), is necessary to adapt an existing software, not conforming to object-oriented programming conventions, e.g., using arrays of primitive types instead of objects, to an object-oriented entry point requirement?

4.3. Case study procedures

4.3.1. Data collection procedures

The subjects of this study are pieces of software. Our selection criteria included (*i*) full access to the source code, (*ii*) the software not being familiar to us or, if it is familiar to us, also having a real-world application, (*iii*) the software either being of the AEC domains or very similar in function and (*iv*) the software simulating a physical process based on a one-time input and returning a one-time output. We searched for prototypes of simulation tools from the building physics domain online. For the first two cases we chose applications published on the Code Project Website. For the third, TU Wien provided us with access to a simulation tool for the purpose of this study (the Microsoft Excel™ tool, [22]). In all cases, we were able to examine the code fully and test various adaptations. The purpose of the examination was not to review the quality of the code, but to work with unfamiliar and/or real-world examples and be confronted with real-world challenges.

4.3.2. Analysis procedures

For the analysis of the cases we gathered quantitative data: the change in program length, the change in lines of code, and the number of new types created in the target software (see questions **CQ 1** and **CQ 2**). We further evaluated qualitative data. We tested various approaches to creating an alternative entry point in the target application. We also documented the programming changes required for exposing enough of the underlying semantics through the entry point to enable passing all necessary input data and receiving output.

4.3.3. Validity procedures

The validity of the obtained results can be compromised by several factors. Firstly, the programming expertise of the researchers has an influence on the data collection. For this reason we do not measure the time spent in unsuccessful adaptation attempts, but analyze only the most efficient adaptations achieved. Secondly, the applied technologies can impact both the qualitative and the quantitative evaluation. Quantitative evaluation measured in lines of code can vary greatly based on the programming language and the developer's programming style. Therefore we have added a more robust measure, the program length, which is defined by [15] as the sum of the total number of operators and the total number of operands in the program.

The creation of the entry point depends strongly on technology as well–therefore we only answer the question, if it is possible, not how expensive it is. The difficulty in exposing the underlying semantics via an entry point, on the other hand, depends on the programming style. We have selected cases with very different programming styles to demonstrate the wide range of challenges an adaptation could face.

4.4. Evaluation

In this section we apply the workflow steps we presented in Section 3 and evaluate the results. We start with **Workflow 1** (see Fig. 2) and Case 1 (see Section 4.1.1).

4.4.1. Evaluation of case 1: a windows forms application simulating particle motion under gravity

Workflow 1 starts with the discovery of the semantics of the target application. Step 1 is shown in Fig. 6. The involved parts of the modeling framework and of the target application are highlighted in blue and green, respectively, in the overview image in the top right corner of the figure. Each of the highlighted modules is expanded to show the relevant portion of its content. For example, the semantics of the target application (the green block in the overview image labeled *Types*) can be seen

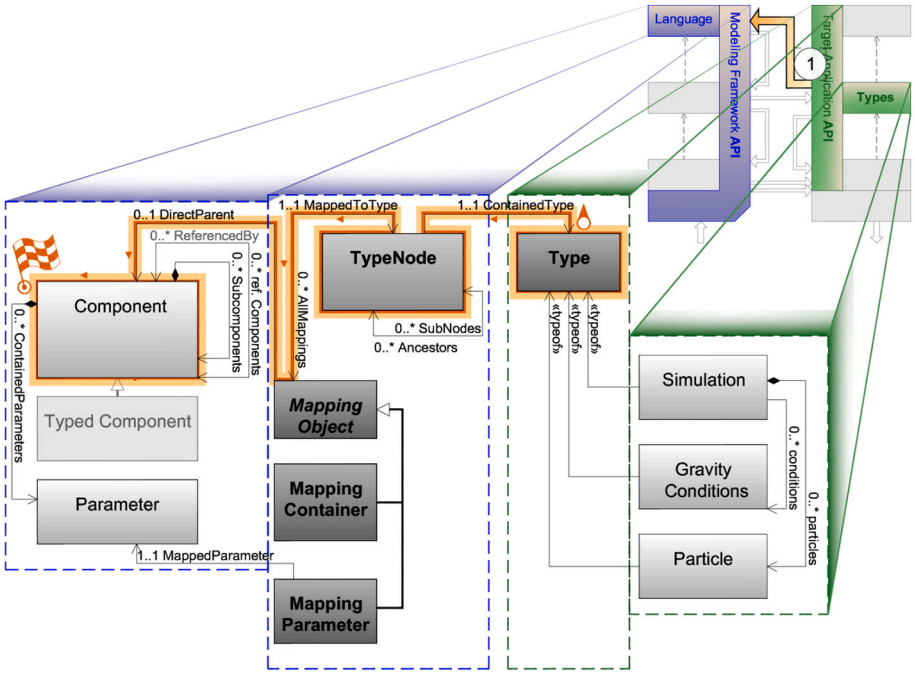

Fig. 6. Case 1. Realization of step 1 of Workflow 1.

in detail on the right in the main portion of the figure, in the box with a dashed greed border. This semantic data model consists of three types. There is a *Simulation*, which refers to *Gravity Conditions* and contains some *Particles*. The language of the modeling framework we presented in Section 3 is the in the top left corner of the overview image. The relevant details are depicted on the left in the main portion of the figure.

In addition to the data models, Fig. 6 shows the interfaces involved in step 1 of **Workflow 1**. On the side of the target application, we use the.net Reflection API to gather information about the target application's type system via type *Type*. The modeling framework realizes the transfer to its own language by employing the types *TypeNode*, *MappingObject* and its subtypes *MappingContainer* and *MappingParameter*. The application of step 1 of **Workflow 1** is depicted as the path highlighted in orange that originates at *Type* and ends in *Component*, thereby completing the information transfer from the target application to the modeling framework.

The result from this transfer is shown in Fig. 7. In step 2 of **Workflow 1**, the instances of *TypeNode* extract the information necessary both for the construction of the corresponding metamodel of each target type and for its instantiation. Once the type structure has been discovered, the modeling framework uses a subtype of *Component*, *Typed Component*, to set a direct relationship between a type and its metamodel element (see the highlighted association *OfType* between *Typed Component* and *Type* in Fig. 7). This is a generic procedure, completely independent of the specific types of the target application. The only requirement is that the application has types exposable by some mechanism, e.g., the meta-information extracted by the Reflection API.

Fig. 8 depicts step 3 of **Workflow 1**. We put particular emphasis on the completed metamodel corresponding to the type system of the target application and represented by the blue block labeled *Metamodel* in the overview image. It consists of elements **M1**, **M2** and **M3**. **M1** models type *Simulation*, with attribute *Name* set to 'M1' and attribute *OfType-Name* set to'Simulation'. Effectively, **M1** has two types, *Typed Component* and *Simulation*. Those types, however, play different roles. The linguistic type is indicated by the dashed orange line connecting **M1** and the language element *Typed Component*. It provides structure, or syntax. The ontological type is indicated by the association *OfType* between **M1** and *Simulation* highlighted in orange. It determines the meaning, or semantics. Therefore, **M1** corresponds to, or is a model of, type *Simulation*.

Type *Simulation* includes the attributes *Name* and *Size*. Those are modeled by the Parameter instances **pM1** and **pM2**, respectively, contained in **M1** via the relationship *ContainedParameters*. Also included in the structure of **M1** is the reference to type *Gravity Conditions*. It is modeled by **M2** of linguistic type *Typed Component* and ontological type *Gravity Conditions*. It is associated with **M1** via the *ref. Components* relationship. Fig. 8 shows the model **M3** of type *Particle* as well.

In effect, the metamodel is the model of the target application's type system. It is shown both in Fig. 8 and in Fig. 9. It enables the creation of run-time instances via reflection. The process begins with the instantiation of this metamodel, or type model, into instance models (see the expanded module *Model* in blue in Fig. 9).

Let us take a closer look at the instance models in Fig. 9 and compare them to the automatically generated target application run-time instances on the right, in module *Instances*. Model instance **s1 Model** is of syntactic type **M1**, which is itself of semantic type *Simulation*. The syntactic *instance-of* relationship between **s1 Model** and **M1** relies on the association *OfType* between *Typed Component* and *Component* in the language utility of the modeling framework (see Fig. 2). The *instance-of* relationship between an object and its type in the the target application, on the other hand, is enforced by the type system of its programming language. All such relationships are depicted as dashed orange lines annotated *« instance-of»* in Fig. 9.

The structure of the run-time instance of *Simulation*, **s1**, includes the slots *Name* and *Size*. Those are modeled by the *Parameter* instances **pM Name** and **pM Size**, respectively, contained in **s1 Model** via the relationship *ContainedParameters*. Also included in the structure of **s1** is the reference to instance **gc** of type *Gravity Conditions*. It is modeled by **gc Model** of syntactic type **M2**, associated with **M1** via the *ref. Components* relationship. In addition, Fig. 9 shows the models **pa Model** and **pb Model** of the instances **pa** and **pb** of type *Particle*, respectively. Each slot *Mass* in a run-time instance is modeled by a *Parameter* instance with the Name 'Mass' and the corresponding value.

Just as the target application permits editing of its run-time data

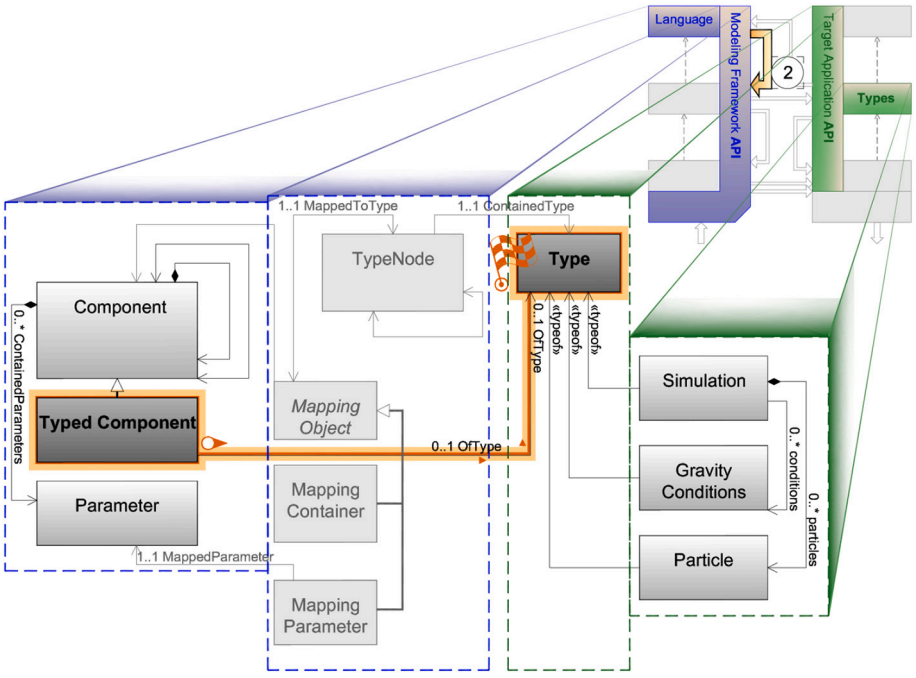

Fig. 7. Case 1. Realization of step 2 of Workflow 1.

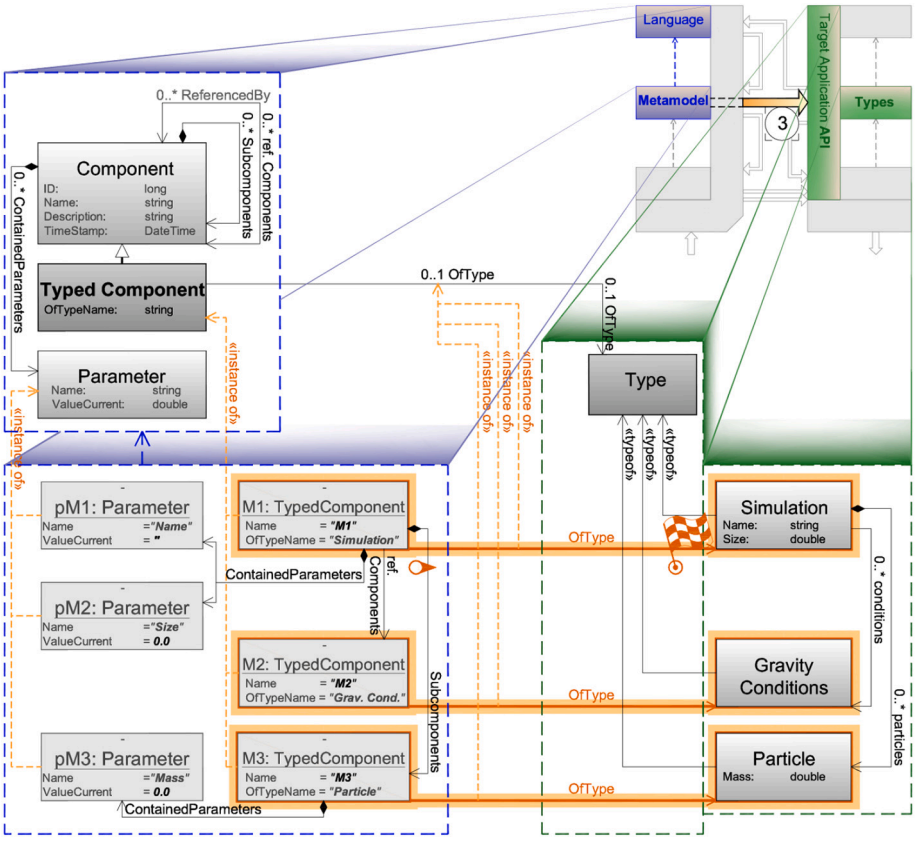

Fig. 8. Case 1. Realization of step 3 of Workflow 1.

structure, so does the modeling framework. The size of the simulation, the number and mass of the particles as well as the gravitational conditions can all be adjusted by the user. In steps 4 and 5 of **Workflow 1**, the model instances receive the user's input (see the orange highlight annotated *4* in the bottom left corner of Fig. 9) and convert it to model instances. In step 6, the corresponding run-time instances of the target application are updated via reflection. In effect, the user input is injected into the target application.

An alternative, simplified solution, which dispenses with the separate modeling of the types and run-time instances of the target application, is shown in Fig. 12 in Appendix A.

The correspondence in structure between the target application data structure and its model in Fig. 9 is syntactic, or linguistic. The semantic correspondence is guaranteed by the association to the same types - *Simulation*, *Gravity Conditions* and *Particle*. Both the target application's data structure and the modeling framework model carry the same meaning - a particle simulation named 'Earth' with size 10.5 containing two particles with mass 0.25 and 0.5 and referencing some gravitational conditions.

This concludes the demonstration of steps 1 to 6 of **Workflow 1**. Those steps fulfill the requirements of **Challenge 1**: *Exposing the semantics*. The modeling framework makes the semantics of the target application available in real-time. This is achieved by producing artifacts conforming to it, e.g., valid run-time instances of its semantic types, containing user input, on demand.

Challenge 2: *Exposing the functionality* requires actual adaptations of the target application and involves steps 7 to 9 of **Workflow 1**. So far, we have gained full access to the run-time instances without implementing any semantics by hand. However, calling an application's functionality depends on the public methods it exposes. In this case, the application uses Windows Forms for the GUI design. This means that there is a standard entry point without input parameters, which calls the standard constructor of the form:

```
1 [ STAThread ]
2 static void Main () {
3 Application .Run ( new Form1 ());
4 }
```

An alternative entry point, which takes an object, possibly representing the entire simulation, as an input parameter, and calls an adapted form constructor could look like this:

```
1 public static void Go( Type_1 input ) {
2 Application . Run( new Form1 ( input ));
3 }
```

The programming style of this tool is object-oriented and there is, in fact, a type representing the entire particle simulation. Therefore, the introduction of the alternative entry point is straightforward. The type representing a particle simulation, *Simulation*, simply replaces *Type_1* in the code above. An additional method for triggering the simulation programmatically completes the list of necessary changes in the source code. After those, in steps 6 to 9 of **Workflow 1**, the tool can be passed input and executed from the modeling framework. The quantitative evaluation of the code adaptation is shown in Table 3. The additional type serves to organize input and output and implements the above mentioned method.

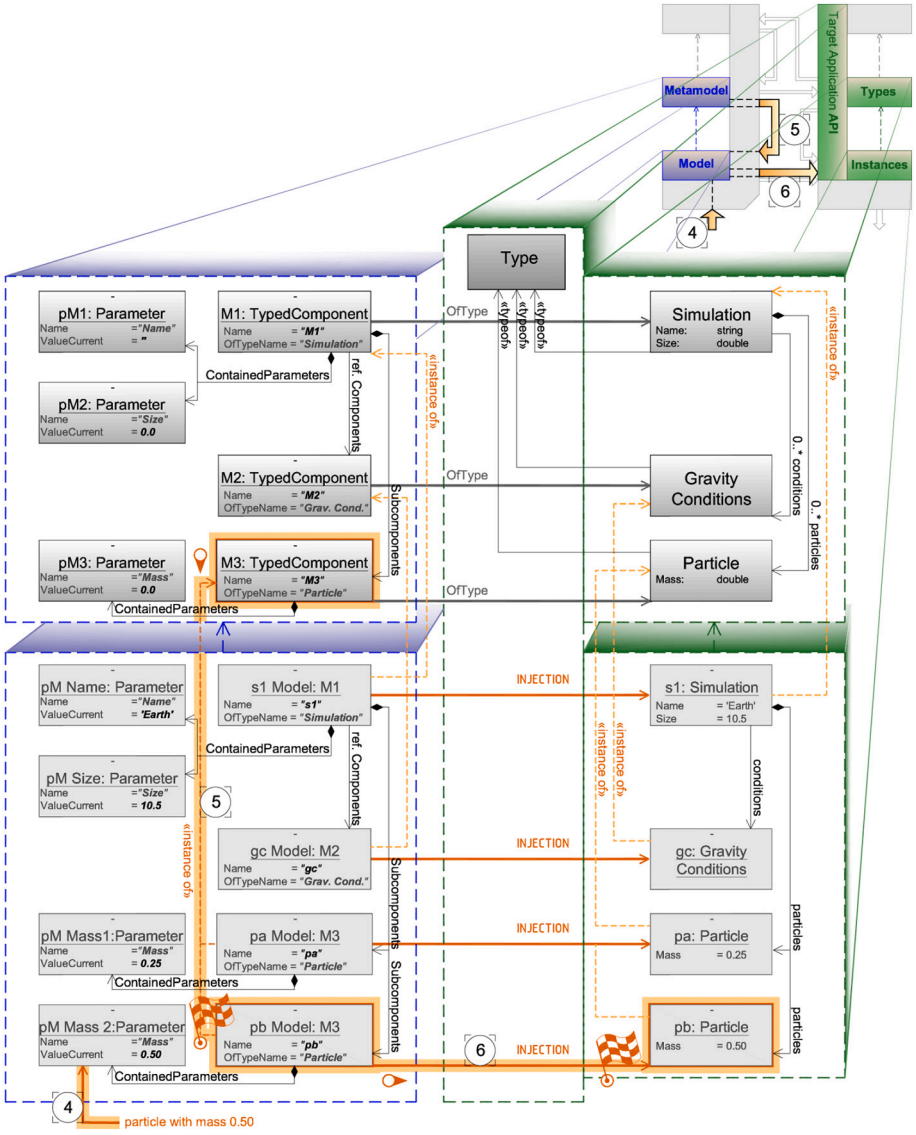

Fig. 9. Case 1. Realization of steps 4, 5 and 6 of Workflow 1.

Table 3
Case 1. Quantitative evaluation of the code adaptation.

Metric	Original	Adapted	Diff.	Increase in %
Program length	7791	8125	334	4.29
Lines of code	714	755	41	5.74
Number of types	8	9	1	–

In conclusion, we present the adapted workflow.

(1) Initially, the modeling framework establishes the connection to the particle simulation tool.
(2) It creates a default model of a particle simulation containing only one particle with a default mass zero. This can be viewed as a model of the types of the particle simulation.
(3) The user edits the instance model to change the properties of the single particle as well as to add more particles and modify the gravitational conditions. Since the modeling framework enables copying, it means that the user can produce an arbitrary number of simulation instances from that one initial model.
(4) The user triggers the simulation with any one of those instances via the alternative entry point **Go**, which accepts a particle simulation instance as input.
(5) The particle simulation tool creates both an animation and a table containing the results.

4.4.2. *Evaluation of case 2: a windows presentation foundation application simulating the propagation of sound waves*

The programming style of this application is data-oriented. It utilizes numerical methods for the calculation of the interference pattern of sound waves and, therefore, uses a grid, implemented as a two-dimensional array, as its main data structure and input source. There are different approaches to handling two-dimensional arrays, or matrices. If the matrix is sparse, the non-zero entries can be represented as objects. In this case, those would be points in two or more dimensions. In this manner, the entire matrix can be represented by a collection of objects. This scenario is very similar to the one in Case 1. However, instead of using already existing types in the target application, such as *Simulation*, *Gravity Conditions* and *Particle*, we can adapt the source code and add a new type *Point2D* to represent a two-dimensional emitting point source, and a new type *Simulation* that wraps a list of *Point2D* instances.

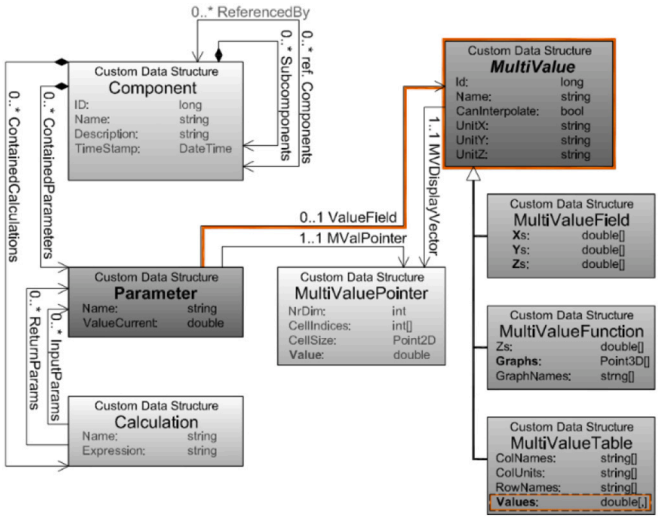

Fig. 10. Handling large numeric data in the modeling framework.

However, we want to address **Challenge 1**: *Exposing the semantics* without modifying the target application. In addition, the scenario described above is less suitable for large dense matrices. Such matrices are produced, for example, by simulation or monitoring software in the building physics domain. A matrix of monitoring measurements taken every 15 s over the course of a year has more than 2 million rows and may have many hundreds or even thousands of columns.

The language utility of the modeling framework we presented in Section 3 has classes for handling data-oriented approaches. Those are the sub-types of *MultiValue*, an abstract container for multiple numeric values, shown in Fig. 10. The values can be organized in scalar fields (*MultiValueField*), graph fields (*MultiValueFunction*) or tables (*MultiValueTable*). An instance of *MultiValue* can be referenced by an instance of *Parameter* by means of the *ValueField* relationship. In this way, it allows the parameter access to all the data via a pointer (see *MultiValuePointer*). Since a parameter can exist only when contained in a component, the data is automatically associated with one or more components, which can take on multiple roles (see Section 2.1.5). On the one hand, the component can act as a simple container, or wrapper, of the data. On the other hand, if the component has a type, i.e., if it is an instance of *Typed Component*, it can act as a typing mechanism.

Using a component as a wrapper of the two-dimensional array produces a model comparable to Case 1. Since the data-oriented programming style still uses the C# typing utility, the Reflection API allows the discovery of all relevant types. As we will see below, we need to add at least one new type to the application to act as input parameter of the alternative entry point. We can utilize this additional type, *Simulation*, as a wrapper for the two-dimensional binary array that carries the input information.

Just as in the previous case, meeting **Challenge 2**: *Exposing the functionality* requires code adaptations. The application uses WPF for the implementation of the GUI. This means that the standard entry point is located in an automatically generated file:

```
1 [ System.STAThreadAttribute ()]
2 [ System.Diagnostics.DebuggerNonUserCodeAttribute
()]
3 [ System.CodeDom.Compiler.
GeneratedCodeAttribute ("PresentationBuildTasks ",
" 4.0.0.0 ")]
4 public static void Main () {
5 WpfTransmissionLineMatrixCS.App app =
6 new WpfTransmissionLineMatrixCS.App ();
7 app.InitializeComponent();
8 app.Run ();
9 }
```

An alternative entry point definition may involve generating a new application domain with its own domain manager for the WPF application to run in. The type of the object we use for passing the input to the application has to be declared as serializable; however, no custom serialization needs to be implemented.

```
1 public static void Go( Simulation input )
2 {
3 var dm_type = typeof ( DomainManager );
4 string codeBase = Assembly.GetExecutingAssembly().
CodeBase ;
5 UriBuilder uri = new UriBuilder( codeBase );
6 string path = Uri.UnescapeDataString( uri. Path );
7 string ab_path = Path.GetDirectoryName( path );
8
9 var setup = new AppDomainSetup
10 {
11 ApplicationBase = ab_path,
```

Table 4
Case 2. Quantitative evaluation of the code adaptation.

Metric	Original	Adapted	Diff.	Increase in %
Program length	8422	8783	361	4.29
Lines of code	1085	1133	48	4.44
Number of types	6	10	4	–

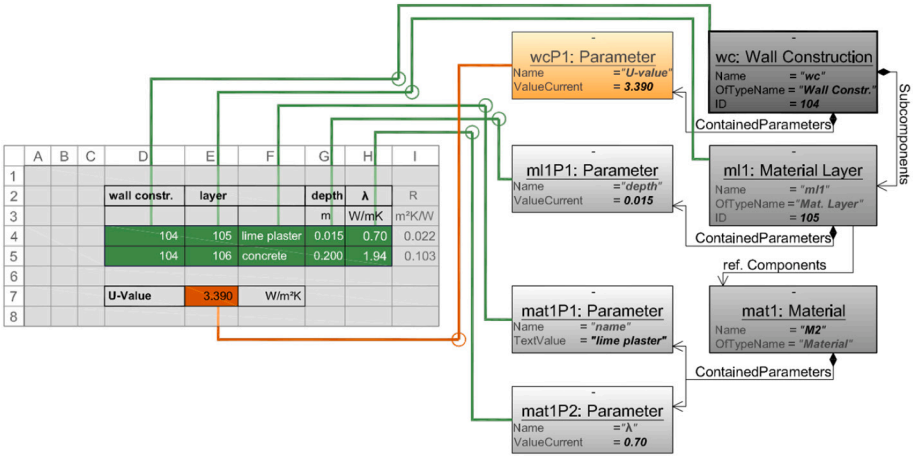

Fig. 11. Case 3. Translating between type *Wall Construction* and an Excel sheet.

```
12 AppDomainManagerAssembly = dm_type.Assembly.
FullName,
13 AppDomainManagerType = dm_type.FullName
14 };
15 AppDomain domain =
AppDomain.CreateDomain( "TempDomain", null , setup );
16 CallBackContext ctx = new CallBackContext();
17 ctx.SourceContainer = input;
18 CrossAppDomainDelegate action = ctx.AppEntry;
19 domain.DoCallBack( action );
20 }
```

As mentioned above, the class *Simulation* (see the input parameter type in the alternative entry point **Go** above) plays the role of a wrapper for the information we need to pass. These adaptations result in the addition of 4 classes to the tool (see Table 4): two for the entry point, one as a wrapper for the input and one for translation from the data to the object-oriented paradigm. Both in terms of lines of code and program length increase, this case is very similar to the previous one.

This completes the adaptation of **Workflow 1** to this case. As in the previous case, **Challenges 1** and **2** were overcome. **Challenge 3**: *Staying up-to-date* is met as a direct consequence of the dynamic connection between the modeling framework and the target application. Any changes in the type structure of the target application results in an automatic update of the dynamically created models in the modeling framework.

In conclusion, we present the adapted workflow.

(1) Initially, the modeling framework establishes the connection to the sound wave propagation simulation tool.
(2) The modeling framework creates an empty table referenced by a parameter in a component instance of type *Simulation*. This configuration can be viewed as a model of the default instance of the sound simulation.
(3) The user populates the table by marking the sound emitting sources as ones and leaving all other entries zero.
(4) As in case 1, the user triggers the simulation by injecting a model of a *Simulation* instance into the alternative entry point, **Go**.
(5) The tool creates the resulting sound interference pattern.

4.4.3. Evaluation of case 3: a Microsoft Excel™ application simulating the thermal behavior of a single space

In this case, we evaluate the ability of the modeling framework to overcome **Challenge 4**: *Making the pragmatics explicit*. As we laid out in Section 4.1.3, the setup in Case 3 is complex. The source application has an object-oriented type system for representing an entire building. The target application, which simulates the thermal behavior of the building, consists of multiple Excel sheets and VBA macros. As an illustration of the task, we chose one small excerpt, which translates a wall construction to an Excel sheet, as shown in Fig. 5 and, in more detail, in Fig. 11.

On the right in Fig. 11 is the model of the wall construction, consisting of two material layers, of which only one (**ml1**) is shown. On the left is the Excel sheet that calculates the U-value of wall constructions. The translation pairs are connected by colored lines. Green stands for translation to the spreadsheet, orange for translation from the spreadsheet. For example, the wall construction **wc** itself maps to the cell under the label *wall constr.*, by supplying only its ID. Parameter **mlP1**, contained in the material layer **ml1**, maps to the cell under the label *depth*, by supplying only its current numeric value, 0.015. Parameter **wcP1**, contained in the wall construction **wc**, on the other hand, is mapped to from the cell where the U-value is located in the spreadsheet.

This case example presents us with the task of dealing with a collection of cells in a table, whose meaning is known only to the designer of the simulation the table realizes. In essence, we need to extract the pragmatics and make them explicit in the corresponding model of the modeling framework. As mentioned in Section 2.1.1, Francis et al. were confronted with a similar challenge in the case study presented in [13]. Their approach involved the creation of a metamodel–a *Spreadsheet* containing multiple *Worksheets*, which contain *Columns*, which can reference each other by means of *Reference*. This metamodel allows the automatic detection of dependencies between cells and tables. However, the semantics of those connections still needs to be supplied by the user.

Addressing **Challenge 4**: *Making the pragmatics explicit* involves making the an explicit connection between cells or ranges of cells within a spreadsheet and a semantic type (see Fig. 4(c)) or concept (see Fig. 4 (d)). For example, the user can create a model element in the modeling framework that corresponds to the concept of an U-Value. Subsequently, the user can define a semantic instance-of relationship between this element and the models of cells in a particular range that store U-Values. This would make the implicit assumptions of the simulation designer explicit and greatly aid any user by removing a significant potential error source, the misinterpretation of the simulation's data model.

This concludes the evaluation of our approach. In the following subsection we will discuss some threats to validity with respect to the four challenges (see their definitions in Section 2.2.1) we aim to overcome.

4.5. Evaluation of validity

It is of note that in all three cases we work with applications with clearly defined functionality and a data model consisting of less than 15 types.

4.5.1. Construct validity

As we stated in Section 2.2.1, the goal of our approach is to provide full integration or *integration facade* for any data model, i.e., a semantic and pragmatic consensus, as this is the backbone of interoperability in any data exchange environment, including *Big Open BIM*. In order to account for the arbitrary component in our evaluation, in Case 1 and Case 2 we chose applications that were not developed by us or known to us prior to the study. In Case 3 we chose an Excel simulation (see [22]) well known to us. The reason for this is threefold. First, the algorithm and its input and output are far more complex than those in Case 1 or Case 2–this is the complexity of a real-world applications. Second, the Excel environment is typically used as a prototyping tool in multiple AEC domains. And third, the programming paradigm of this tool is data-oriented instead of object-oriented and gives us the opportunity to account for information flow between semantic-carrying and semantic-agnostic applications.

4.5.2. Hidden factors

In spite of being confronted with a data-oriented programming paradigm in Case 2, we still had the ability, due to the chosen programming language, C#, to utilize object-oriented methods. Our results do not apply to software written in languages that do not allow object-oriented access of any kind.

We have not considered any security aspects in this study.

4.5.3. Generalization

A data exchange standard cannot rely on our approach unless it is generalizable to include much more complex applications.

Challenge 1: *Exposing the semantics*. Our approach employs automated methods for extracting the data model of any application. Even a data model as extensive as IFC4.3 can be automatically loaded.

Challenge 2: *Exposing the functionality*. The three cases we examined allow us to formulate the following requirements that need to be fulfilled by a software with no API, no implementations of memory sharing or of data exchange standards, in order to be fully accessible from other applications:

R2.1. The type structure should contain one type with a pre-defined name, e.g., *MainObject*, providing controlled access to all objects managing input and output.

R2.2. The application should have an entry point that accepts an instance of type *MainObject* as input.

R2.3. The application should trigger its main functionality automatically on receiving input over the entry point described above.

Challenge 3: *Staying up-to-date*. Since both Challenge 1 and Challenge 2 are generalizable, staying up-to-date, in terms of data model and functionality, is generalizable as well.

Challenge 4: *Making the pragmatics explicit*. The modeling framework allows the definition of additional semantic types as well as relationships along the ontological and conceptual axes. These tools allow the extraction of the pragmatics and making it explicit for all users, thus contributing to the consensus of the *integration facade*.

4.5.4. Reliability

The researchers involved in this case study have expertise in the following fields: architecture, building physics, model driven engineering, multi-level modeling and ontology engineering.

5. Conclusion and future work

In Section 2.2.1 we formulated four challenges that need to be overcome in any data exchange process in order to achieve uninterrupted multi-directional information flow. This also applies to Big Open BIM in the AEC industries. The modeling framework we presented in Section 3 addresses those challenges to varying degrees, as summarized in Section 4.5. It enables the integration of semantics at run-time and of pragmatics at design-time. The result is that implicit assumptions are made explicit. Consequently, they can be discussed and a consensus both in semantic and pragmatic terms can be reached, since, as stated by Fetzer [12], *"meaning is at the heart of both semantics and pragmatics"*. In our approach we regard syntax, semantics and representation as the three independent axes of the space in which data exchange operates. This independence, or separation of concerns, allows the semantics to be modeled independently of the notation or representation of information in any available tool, which makes our approach universally applicable to data exchange processes. In addition, we can model pragmatics both along the semantic and the representational dimensions, and produce a full integration of both semantics and pragmatics into an *integration facade*, which is a prerequisite for producing a single source of truth.

Our future work will focus on utilizing the integration facades in the translation between the data models of various tools and standards, even in cases where no one-to-one correspondence between concepts exists. In addition, we will develop algorithms for the detection of semantic patterns as basis for automated translation procedures and of translation rules between semantic models involving calculations. Furthermore, we will examine alternative approaches to defining a semantic model. In this work we demonstrated the power of Model-Driven Engineering (MDE). We intend to perform a comparison between MDE and ontology design in our future work.

CRediT authorship contribution statement

Galina Paskaleva: Conceptualization, Methodology, Software. **Alexandra Mazak-Huemer**: Investigation, Writing - review & editing, Supervision. **Manuel Wimmer**: Supervision. **Thomas Bednar**: Supervision, Software.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Workflow 1: non-level-respecting modeling style

An alternative, simplified solution to the one in Fig. 9 in Section 4.4.1, which dispenses with the separate modeling of the types and runtime instances of the target application, is shown in Fig. 12. The model presented in the bottom left corner contains both the type and instance information and is ready for user input injection. This is a more efficient, however, not level-respecting, model according to [19].

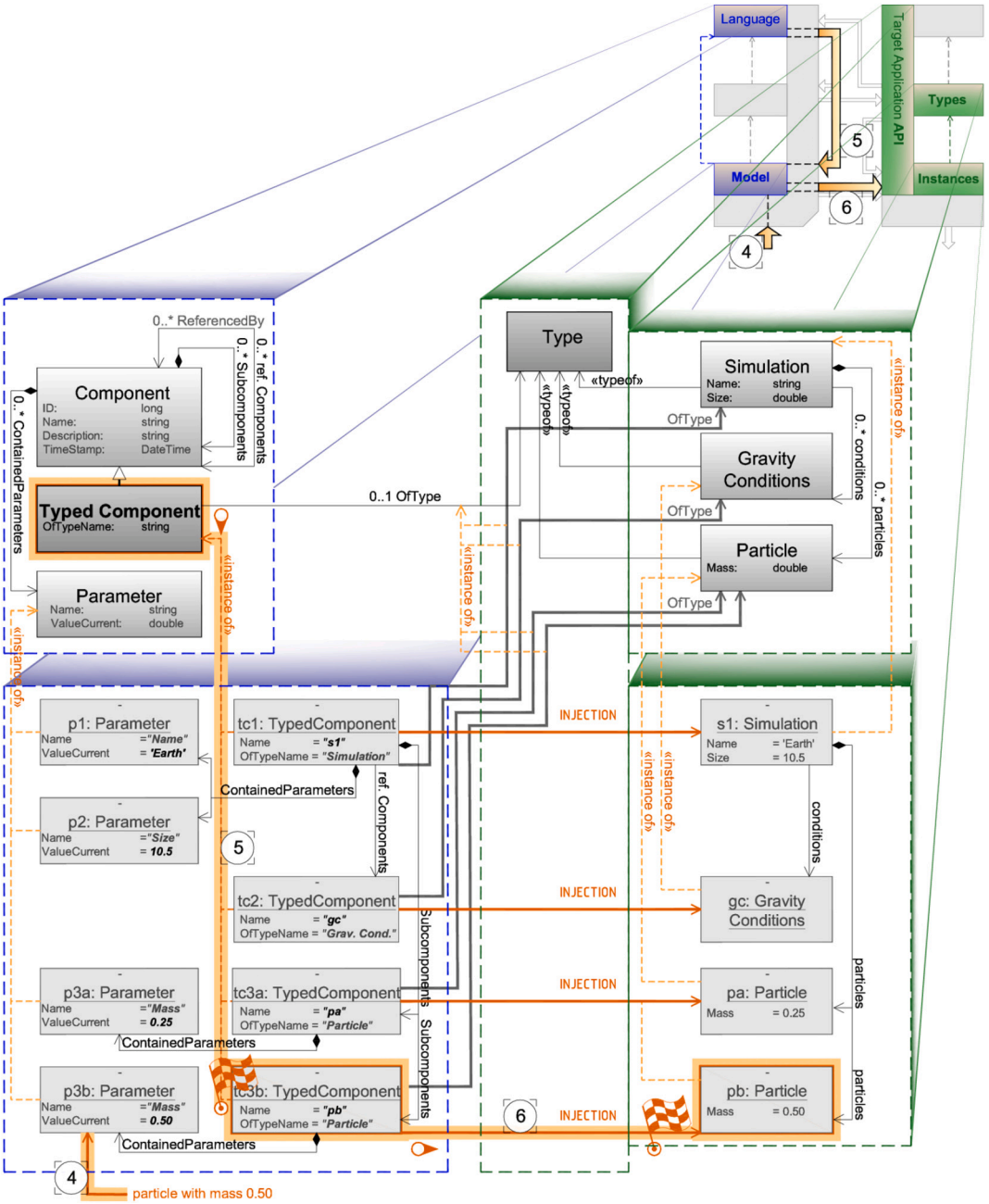

Fig. 12. Case 1. An alternative solution to the one shown in Fig. 8 and Fig. 9 in Section 4.4.1.

References

[1] T. Akanbi, J. Zhang, Y.C. Lee, Data-driven reverse engineering algorithm development method for developing interoperable quantity takeoff algorithms using IFC-based BIM, J. Comput. Civ. Eng. 34 (2020), https://doi.org/10.1061/(ASCE)CP.1943-5487.0000909.

[2] A. Borrmann, T. Kolbe, A. Donaubauer, H. Steuer, J. Jubierre, M. Flurl, Multi-scale geometric-semantic modeling of shield tunnels for GIS and BIM applications, Comput.-Aided Civil Infrastruct. Eng. 30 (2015) 263–281, https://doi.org/10.1111/mice.12090.

[3] M. Bracht, A. Melo, R. Lamberts, A metamodel for building information modeling-building energy modeling integration in early design stage, Autom. Constr. 121 (2021), https://doi.org/10.1016/j.autcon.2020.103422.

[4] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, 2 ed., Morgan & Claypool, USA, 2017. ISBN: 978-1627057080.

[5] buildingSMART, IFC4 Release, URL, https://standards.buildingsmart.org/IFC/RELEASE/IFC4/FINAL/HTML/, 2013. last accessed on November 13, 2020.

[6] buildingSMART, IFC4.3 Release Candidate 2, URL: https://standards.buildingsmart.org/IFC/DEV/IFC4_3/RC2/HTML/, 2020. last accessed on February 26, 2021.

[7] T. Cerovsek, A review and outlook for a 'building information model' (BIM): a multi-standpoint framework for technological development, Adv. Eng. Inform. 25 (2011) 224–244, https://doi.org/10.1016/j.aei.2010.06.003.

[8] W. Chen, M. Das, V.J.L. Gan, J.C.P. Cheng, Integrated data model and mapping for interoperable information exchange between BIM and energy simulation tools, in: E. Toledo Santos, S. Scheer (Eds.), Proceedings of the 18th International Conference on Computing in Civil and Building Engineering, Springer International Publishing, 2021, pp. 496–506, https://doi.org/10.1007/978-3-030-51295-8_35.

[9] A. Costin, N. Nawari, A. Adibfar, C. Eastman, Preliminary evaluation of the industry foundation classes (IFC) to enable smart city applications, in: CIB World Building Congress, 2019, pp. 1–10.

[10] F. Dong, R. Zhang, R. Xiao, B. Lei, Mesh simplification with global contour feature preservation, in: WRI World Congress on Computer Science and Information Engineering, 2009, pp. 679–685, https://doi.org/10.1109/CSIE.2009.636.

[11] W. East, Construction Operations Building Information Exchange (COBIE), Defence Technical Information Center URL, https://apps.dtic.mil/sti/citations/ADA491899, 2007. last accessed on February 26, 2021.

[12] A. Fetzer, Recontextualizing Context, John Benjamins Publishing Company, Amsterdam, The Netherlands, 2004. ISBN 9789027295712.

[13] M. Francis, D.S. Kolovos, N. Matragkas, R.F. Paige, Adding spreadsheets to the MDE toolkit, in: A. Moreira, B. Schätz, J. Gray, A. Vallecillo, P. Clarke (Eds.),

Model-Driven Engineering Languages and Systems, Springer, Berlin Heidelberg, 2013, pp. 35–51. ISBN 978-3-642-41533-3.
[14] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing? Int. J. Hum.-Comput. Stud. 43 (1995) 907–928, https://doi.org/10.1006/ijhc.1995.1081.
[15] M.H. Halstead, Elements of software science, in: Elsevier Computer Science Library: Operational Programming Systems Series, North-Holland, New York, NY, 1977. ISBN 978-0444002150.
[16] J. Haymaker, P. Keel, E. Ackermann, W. Porter, Filter mediated design: generating coherence in collaborative design, Des. Stud. 21 (2000) 205–220, https://doi.org/10.1016/S0142-694X(99)00042-3.
[17] S. Kaewunruen, P. Rungskunroch, J. Welsh, A digital-twin evaluation of net zero energy building for existing buildings, Sustainability 11 (2018), https://doi.org/10.3390/su11010159.
[18] H.R. Kranz, BACnet Gebäudeautomation 1.12. cci Dialog, 2013. ISBN 978-3922420255.
[19] T. Kühne, Matters of (meta-) modeling, Softw. Syst. Model. 5 (2006) 369–385, https://doi.org/10.1007/s10270-006-0017-9.
[20] H. Lai, X. Deng, Interoperability analysis of IFC-based data exchange between heterogeneous BIM software, J. Civ. Eng. Manag. 24 (2018) 537–555, https://doi.org/10.3846/jcem.2018.6132.
[21] J. Lee, Y. Jeong, User-centric knowledge representations based on ontology for AEC design collaboration, Comput. Aided Des. 44 (2012) 735–748, https://doi.org/10.1016/j.cad.2012.03.011.
[22] J.N. Nackler, Sommerlicher Wärmeschutz: Vergleich von Berechnungsansätzen eines Planungsinstrumentes für Entwurfsfindung, Ph.D. thesis, TU Wien, Vienna, Austria, 2017.
[23] R.F. Paige, A. Zolotas, D.S. Kolovos, The changing face of model-driven engineering, in: Present and Ulterior Software Engineering, Springer, 2017, pp. 103–118, https://doi.org/10.1007/978-3-319-67425-4_7.
[24] E. Petrova, P. Pauwels, K. Svidt, R. Jensen, Towards data-driven sustainable design: decision support based on knowledge discovery in disparate building data, Architect. Eng. Design Manag. 15 (2019) 334–356, https://doi.org/10.1080/17452007.2018.1530092.
[25] M. Rasmussen, M. Lefrançois, P. Pauwels, C. Hviid, J. Karlshøj, Managing interrelated project information in AEC knowledge graphs, Autom. Constr. 108 (2019), https://doi.org/10.1016/j.autcon.2019.102956.
[26] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2008) 131–164, https://doi.org/10.1007/s10664-008-9102-8.
[27] J. Steel, R. Drogemuller, B. Toth, Model interoperability in building information modelling, Softw. Syst. Model. 11 (2012) 99–109, https://doi.org/10.1007/s10270-010-0178-4.
[28] J. Wu, H. Sadraddin, R. Ren, J. Zhang, X. Shao, Invariant signatures of architecture, engineering, and construction objects to support BIM interoperability between architectural design and structural analysis, J. Constr. Eng. Manag. 147 (2021), https://doi.org/10.1061/(ASCE)CO.1943-7862.0001943.
[29] N. Yabuki, T. Aruga, H. Furuya, Development and application of a product model for shield tunnels, in: 30th ISARC, 2013, pp. 435–447, https://doi.org/10.22260/ISARC2013/0047.
[30] Q. Yang, Y. Zhang, Semantic interoperability in building design: methods and tools, Comput. Aided Des. 38 (2006) 1099–1112, https://doi.org/10.1016/j.cad.2006.06.003.
[31] A. Zolotas, H.H. Rodriguez, D.S. Kolovos, R.F. Paige, S. Hutchesson, Bridging proprietary modelling and open-source model management tools: the case of PTC integrity modeller and epsilon, in: 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), IEEE, 2017, pp. 237–247, https://doi.org/10.1109/MODELS.2017.18.

Galina Paskaleva, M.Sc. from TU Wien, is a researcher at Montanuniversität Leoben at the chair of Subsurface Engineering. She is an architect and a student of informatics. Her main research focus is the application of model-driven software development methods to the data exchange in the domains of Architecture, Engineering and Construction. She is a member of the development team for a multi-stakeholder IT-environment SIMULTAN at TU Wien.

Alexandra Mazak-Huemer, is a professor at Montanuniversität Leoben at the chair of Subsurface Engineering. Before she headed Module 3 in the Christian Doppler Laboratory for Model-Integrated Smart Production (CDL-MINT) at JKU Linz. Her research field focuses on digital transformation in tunnel information modeling as well as in modeling of production systems. Her research interests are on information integration, model-driven engineering and model-based data analytics. She headed numerous interdisciplinary national funded projects in the field of digital transformation.

Manuel Wimmer is a full professor leading the Institute of Business Informatics - Software Engineering at the Johannes Kepler University Linz and he is the head of the Christian Doppler Laboratory CDL-MINT. His research interests comprise foundations of model engineering techniques as well as their application in domains such as tool interoperability, legacy modeling tool modernization, model versioning and evolution, and industrial engineering.

Thomas Bednar, M.Sc. and PhD from TU Wien, is a full professor for Building Physics at TU Wien. His main research focus are prediction models for hygrothermal and acoustic performance of buildings. He has written many papers related to energy use, moisture and building acoustics both on calculations and measurements and is now finishing projects on plus-plus energy high rise office buildings or safeguarding wooden beam ends after renovations of brick-buildings from 1900. He is head of the development of multi-stakeholder IT-environment SIMULTAN at TU Wien for developing and understanding the built environment.