www.itcon.org - Journal of Information Technology in Construction - ISSN 1874-4753

# A MODEL-DRIVEN APPROACH TO THE INTEGRATION OF PRODUCT MODELS INTO CROSS-DOMAIN ANALYSES

REVISED: February 2015
PUBLISHED: March 2015 at http://www.itcon.org/2015/17
EDITOR: Rezgui Y.

***Ulrich Hartmann, Associate Research Scientist (until Oct. 2014) and PhD student,***
***Karlsruhe Institute of Technology KIT, Department Building Lifecycle Management;***
***Ulrich_Hartmann@gmx.de***

***Petra von Both, Professor,***
***Karlsruhe Institute of Technology KIT, Department Building Lifecycle Management;***
***Petra.vonBoth@kit.edu***

***SUMMARY:*** *During the many phases in the lifecycle of a building, an ever growing number of dynamic and static aspects are encountered that seem worthy of modelling in a computer-readable representation. Product models that attempt to cover the concepts from the many disciplines or phases in a product's lifecycle face a dilemma, namely maintaining a balance between growing model complexity on the one hand, and user requests for additional concept coverage on the other. In examining the main principles of model theory (Stachowiak 1973) more closely, this paper tries to identify tendencies that may hamper the balance between complexity and completeness. However, a scientific evaluation of model complexity needs objective measures. Metrics for the assessment of software complexity are readily available and can also be applied to models, since models (schema or instance) can automatically be transformed into a programming language (Hartmann, von Both 2010) or an instance structure. These metrics can therefore be helpful in the discussion about model complexity. Different approaches have been taken to handling model complexity and the complex process of modelling itself. Leal S. et al. (2014) develop a template-based model generating tool for energy simulation models to evade the lossy and complex recourse to IFC or gbXML models. Cao J. et al. (2014) use a transformation tool to generate building energy performance simulation models in order to reduce the complexity otherwise encountered in traditional building simulation programs. Thomas D. et al (2014) combine a CitySim model holding a simplified representation of the surrounding buildings with models for the EnergyPlus building performance engine. This allows for rapid assessment of the performance of the early design-stage building information models (BIM) on both the building and urban systems scale. Koene F. et al. (2014) take the reductionist idea of model theory a step further by reducing a building to a simple model consisting of two thermal masses.*

*The Building Lifecycle Management Department at Karlsruhe Institute of Technology KIT conducts analyses beyond the scope of a single building. Analyses often have to take the urban environment into account, meaning that not only building models but also models of the surrounding city area are potentially involved. From the modelling standpoint this raises the question as to whether the city model and building models should be unified into one "super model", or whether models should be kept separate and the relations between them expressed by means of meta data.*

*The work presented is a result of the research project "A model-driven approach to the integration of product models into cross-domain analysis processes" (original title: "Ein modellgetriebener Ansatz zur Integration von Produktmodellen in domänenübergreifende Analyseprozesse") sponsored by the federal German research funding organization 'Deutsche Forschungsgemeinschaft (DFG)'.*

***KEYWORDS:*** *model integration ,complexity, dominant decomposition, cross-domain model analysis, building lifecycle management, city models, building models, product models IFC, CityGML.*

***REFERENCE:*** *Ulrich Hartmann, Petra von Both (2014). A model-driven approach to the integration of product models into cross-domain analyses, Journal of Information Technology in Construction (ITcon), Vol. 20, pg. 253-274, http://www.itcon.org/2015/17*

# 1. INTRODUCTION

Consulting the principles of model theory and the metrics for model complexity, the approach of keeping models separate has been taken. The conceptual framework presented here includes a concept for the specification of interfaces between models. This specification is not restricted to the description of schematic (type-based) relationships, but also allows the expression of constraints on model objects (instances). The separate specification of the interface between the analysis model on one side and the source model(s) on the other, not only facilitates the re-use and replacement of source models but also enables the distribution of work between experts of different disciplines: Domain experts can concentrate on the design of the analysis model, while data modelling experts can concentrate on the mapping expressions between models.

In the AEC sector, the use of models as a means for communication, analysis or collaboration has been common for centuries, including following the arrival of computers. While models were previously expressed using physical or symbolic replicas, computers have given us virtually limitless power of expression. In the field of AEC, many disciplines work together, with each discipline having its own logical export model. With the emergence of computers it became common to transfer working steps to the computer. This initial step didn't change the conceptual working method, it simply moved the usual steps to the computer to take advantage of improved data handling, e.g. by using it as a digital drawing board or a central repository for the management of drawings, etc. Handling drawings turned out to be a cumbersome process due to the many different formats used by applications to store them. This led to the idea of having a STandard for the Exchange of Product model data, initiated under the abbreviation STEP in 1984. Early research work focused on the distribution of model data in networks using a common typeset for model entities and entity relations (Hartmann, U.; Huhnt, W; Laabs, A; Pahl, P. J., 1990). While the exchange of data between applications was the primary focus of STEP, the aim of its follower, the Industry Foundation Classes (IFC), was the digital representation of buildings a.k.a modelling. The Industry Alliance for Interoperability (now BuildingSMART) was the organizational body that set the goals for the IFCs: to create a formal basis for digital building data in all phases of its lifecycle. Starting in 1995, national and international workgroups of all relevant disciplines shaped out a common product model, intended to be a sufficient basis for collaboration and improve the quality of computer-supported work.

The well-intended approach of bringing all related disciplines under the hood of a single common model faces some obstacles, however. The disciplines often do not share the same perception of an artifact. A wall may be reduced to an axis in structural engineering, a quantity of necessary materials in calculation, a bidding position, etc. In some cases, an entity may be modelled as the unification of all attributes of all disciplines. In other cases, artifacts of the original models of disciplines involved may not be entirely compatible. This may cause losses or redundancies in the unified model.

In this paper, we follow a three-step guideline for the assessment of models. First, we analyze the effects of the so-called "dominant decomposition" (D'Hondt, 2002) to various disciplines, then we look at the main principles of model theory as founded by Stachowiak in 1973 (Stachowiak 1973), and third we examine the reasons for the complexity of models schemas and model instances (discussed in-depth in Hartmann, von Both 2010). As mentioned, multi-model scenarios are of primary interest in this paper. The specifics of these scenarios lead to special requirements for cross-domain modelling.

Taking all the steps into account, we shape out a solution approach for a common modelling problem in urban architectural analysis: the analysis of problems related to buildings within an urban context. Depending on the problem, this often turns out to be a multi-model problem. Although applicable to analyses embracing more than one model, the approach may also ease the mapping between the object model of an application and a (complex) standard model. It may encourage implementers to work on top of a standard model without facing a complexity that is implausible in the light of the tasks the application aims to deal with.

As a generalization of the solution approach, a conceptual framework has been developed. Its intention is to formalize the analysis within a multi-model scenario. Finally, two sample scenarios are given. Also, a software prototype has been developed as a proof of concept.

## 2. MODEL ASSESSMENT BACKGROUND

### 2.1 Separation of concerns and the dominant decomposition

The first notion of the idea of separation of concerns (SoC) goes back to Dijkstra in 1976 and Parnas in 1972. It denotes the identification of different concerns in a software system. In the face of the so-called software crisis of the 1970s (Naur and Randell 1969), the phrase was formulated to express the intended encapsulation and modularisation in software systems. Aspect-oriented software development (AOSD), originally presented by Kiczales in 1997, builds on this idea in order to find new ways of identifying cross-cutting concerns. Concerns are regarded as cross-cutting if behaviour of the same kind is spread or repeated over several modules, instead of being encapsulated in a single module.

The unification of concepts from different disciplines in a single model may be seen as a compromise for the sake of interoperability. Those disciplines that see a significant syntactical and/or conceptual gap between their original native model and the integrated model may experience the dominance of the new decomposition as troublesome. This effect has been impressively described in "The Tyranny of the Dominant Model Decomposition" (D'Hondt, 2002). The dominance of the dominant decomposition may cause side effects. The larger the gap between the unified model and the discipline's native model, the larger the amount of required knowledge about the relationship between them in the application logic that makes use of the unified model. Figure 1 (Filman 2004) illustrates this. The reality, in this case an unsorted set of coloured shapes of different sizes, may be sorted or decomposed using different criteria: size, colour or shape. As soon as one of three criteria becomes the dominant criterion, the other two are scattered across, and are thus cross-cutting concerns. Expressed in terms of Figure 1, we can see that taking the colour-sorted model as the common unified model "disciplines" focusing on size, as the main modelling criteria have to query the unified model in order to re-establish their native decomposition. They will most likely view the colour-sorted model as being dominant.

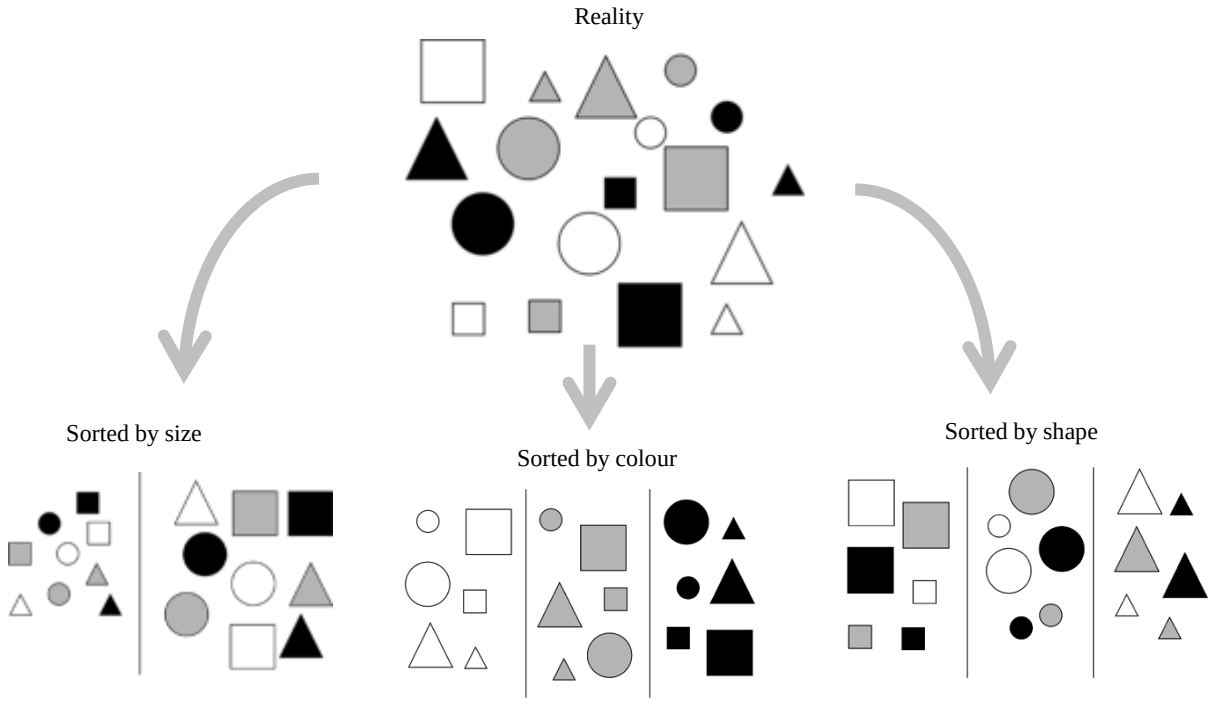



*Figure 1: Abstract concern space and some decomposition examples (Filman 2004)*

**Dominant decompositions and product models**

Domain models in the AEC disciplines clearly have a higher complexity than the entities shown in Figure 1. Therefore, the unification of domain models into one unified model is an even bigger challenge. The likelihood

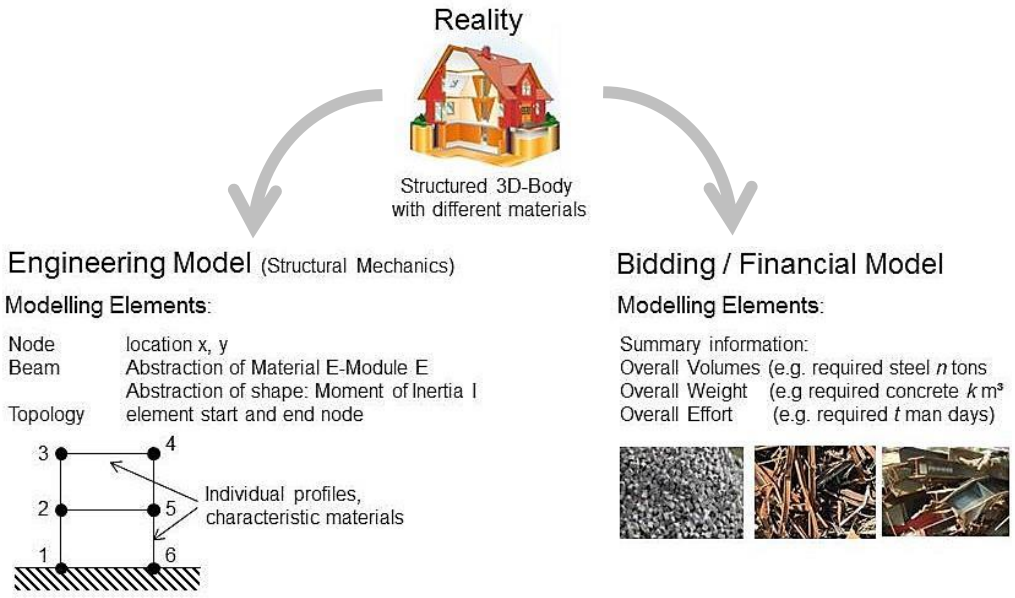



Figure 2: Schema integration (a) by aggregation of structural elements and (b) by decomposition of aggregated values

of the resulting decomposition being regarded as dominant is higher the larger the semantical gap between the resulting unified model and the original domain model. Figure 2 illustrates this. Bidding positions hold summary information on the entire construction work by defining an object structure advantageous for calculation (Figure 2, right). This decomposition of objects (e.g. aggregates or whole/part-relations) is useful (pragmatic) for solving the underlying problem of concluding contracts on the basis of bidding positions where the controlling of costs is the main focus. The same decomposition is totally useless, for example, for structural mechanics calculations. Summing up all E-Modules and Moments of Inertia to conduct a structural mechanics calculation (Figure 2, left) would simply be wrong. If vice versa the decomposition of the engineering model is also taken for modelling costs, then costs will cross-cut the model, since the intention of the bidding model - to expose aggregated cost values - cannot be modelled. Therefore, from the viewpoint of the bidding domain, the engineering model is a dominant decomposition. Product models happen to be pure data models; they do not implement any behaviour. Therefore the relation between the aggregated position costs and the origin of the cost information (amount of material for each single element) is implicit. In other words, the algorithmic relationship between summary cost and the many source elements with their single cost values is not explicitly expressed by the model. Both sides may be altered independently and unnoticed. This redundancy inside the model may cause contradictions and inconsistent model states. In our case, there is no functional relation between the cost of a single part and the summary costs information. In well-established product models like the IFCs, this phenomenon is commonly observable. It is in the responsibility of applications to handle inherent and implicit model design decisions correctly by means of the application logic.

## 2.2 Modelling principles

Models are made for a purpose. This reflects the modelling principle of *pragmatism*. The purpose may be to conduct analysis for a set of given problems, lossless exchange of product model data between participants in a workflow, description of physical product properties, description of the construction processes, etc. The modelling principles of representation, reduction, abstraction and pragmatism expressed in general by model theory acquire their own shape in specific disciplines. Engineering sciences define their own concepts and models (representations) by reducing the more general concepts of physics to the level of detail that is necessary

to describe the respective problem (pragmatism). As a result, in structural mechanics a beam will not be calculated on a molecular level. The engineering model consists of just a few parameters yielding results that can be calculated with reasonable effort. Interfaces between model elements specify the action and reaction between related elements. Computer sciences have developed their own specific adaptations of modelling principles. The Interface Segregation Principle (ISP) states that modules should depend on each other and on the leanest possible interface. Module users should not be forced to call interfaces unrelated to their context. The motivation for this postulation lies in the limitation of dependencies between modules. Conceptually unnecessary dependencies can be the reason for annoying conflicts or unnecessary complexity. For example, a building may either be modelled as a collection of rooms or as a collection of walls and plates. Rooms can be calculated using the distances between walls and slabs, or alternatively, walls and slabs can be calculated using the gaps between rooms. In a building model, the concept of rooms cannot be expressed independently from the concept of walls and slabs. Otherwise it would be possible to specify a room whose geometry collides with walls and slabs. When using model types in which this type of redundancy is possible (e.g. IFC), and it is up to the application logic to solve these conflicts. An example may serve to illustrate this: In the object-oriented paradigm, properties of an object are modelled in a "has-a" relationship. For example, an object *has a* geometry or a storey *has a* collection of rooms. This modelling approach ensures that subordinate structures are discarded when the parent structure is deleted. The list of rooms does not make sense once the storey has been removed. IFC follows another approach called INVERSE reference. Rooms are assigned to a storey via relation objects (IfcRelContainedInSpatialStructure). Consequently, if the storey is deleted, its rooms and the relations object remain. It needs internal knowledge of applications to implement the *has-a* behaviour, and merely removing the storey object is insufficient. Applications are forced to implement interfaces in order to maintain model consistency otherwise unnecessary for the native purposes of the application not in line with the ISP. While this is an example of a structural-modelling pitfall, redundancies can also be found. For example, IFC models may hold calculated floor areas alongside a room's length and width values, the calculated thermal transmittance values of walls alongside layered walls, etc. Changing one side does not trigger an update of the other side of the dependency. Where cause and reason are not obvious, model users either do not trust the calculated results (and prefer to recalculate the values on the basis of their original input parameters) or otherwise accept possible inconsistencies. There is another issue with this approach: Applications that decide to recalculate room geometry from wall positions can be terribly wrong. If rooms were the leading elements, which one of the original geometrical attributes should be recalculated? This modelling decision cannot be expressed explicitly in most model types (e.g. IFC). As a layer between an application and a standard model like IFC, the approach presented in this paper can encapsulate solutions to the above-mentioned problems. It can free applications from having to fill design gaps between the native object model and an external object model.

## 2.3 Model complexity

BuildingSMART, the makers of the IFCs, want to support the participation of different disciplines in building lifetime processes by defining subschemas of the entire IFC model schema. The so-called Model View Definitions (MVDs) provide a subset of model elements appropriate for collaboration between specific disciplines. This encapsulation of concepts also helps reduce complexity. So far, MVDs have been published for the overall coordination of model integration, for structural analysis, and for the exchange of basic facility management data. Being subschemas, they do not introduce new IFC schema elements, but are assembled using the available IFC concepts. New decompositions, e.g. aggregated values, are therefore only possible within the complete framework of the IFCs. When working with the IFCs, applications are still faced with the high formal complexity of IFC constructs. The formal complexity of the IFCs could be one of the main reasons why there are not more than just a few small downstream applications on the market using IFC. The BuildingSMART website lists around 160 applications supporting IFC import, with roughly half of them also supporting IFC export. Only the big players have the resources to implement full IFC support. The participation of around 30 companies in the IFC certification process shows this. The deep chains of indirections in the IFC schema are relics of the relational database background in the long history of this data model. Unfortunately, the relational constructs have made their way into the IFC XML schema. This becomes obvious in constructs such as IFCRelation (and derived classes). They resemble foreign key constructs of the relational modelling paradigm and do not align with object-oriented design patterns. As an example, in the object-oriented world, all the attributes of an object are deleted at the end of the lifetime of the containing object (part-of relationship). Not so with IFC objects. Since the attributes of an object are being stored in a second separate object, and the object itself is unaware of

this second object, the lifetime of the main object is not in sync with the object holding the attributes. In a relational database this would not pose a problem, since the referential integrity would be maintained by the database management system. This phenomenon - known as the "object-relational impedance mismatch"- makes it even more difficult for object-oriented applications to work with the IFCs. Loose coupling in the object-oriented world decouples the lifetimes of the two related objects by intention. Since IFC lacks the capability to model part-of relations correctly, it is up to applications to decide whether to apply strong or loose coupling of sub-elements. Another issue is the existence of two parallel ID-systems. IDs have to be unique inside their area of validity (scope). In the IFCs, the two ID-systems have two different scopes: the GlobalID-system has global (worldwide) scope, whereas the simple ID-system has file-wide scope. The reference mechanism used in IFCRelation classes uses the ID-system with file-wide scope. Therefore, aggregates can only be defined if all referenced elements are contained in the file. This physical linkage to the storage representation is especially painful, because it inhibits the modelling of components that could be re-used and assembled in different contexts - an established and well-proven modelling scenario in mechanical engineering disciplines. When talking about complexity, a discussion can become potentially vague and based on personal assumptions. Metrics have been developed to express the complexity of software in general and models in particular, and also to prevent vagueness. Metrics for the analysis of product model complexity are discussed in "Metrics for the Analysis of Product Model Complexity" (Hartmann, Ulrich; von Both, Petra 2010), where a comparison is made between the complexity of the IFCs and CityGML. The metrics presented there can also be used to estimate the influence on complexity of the approach presented in this paper.

## 2.4 Multi-model scenarios

Real-world data is of primary interest for conducting realistic analyses. It is stored in databases or files using a data model that has been previously defined (e.g. XML schema, database schema, etc.). Before conducting a model-based analysis on real-world data, it must be ensured that the data model of the data source contains sufficient data for the analysis. This is not a matter of quantity but of the element types and properties encountered in the source data model. Sometimes it becomes apparent that there is no single source data model that holds all the information (types, attributes, etc.) necessary to conduct the envisioned analysis. Many problems in the context of architecture go beyond the scope of one single building. This is the case if the urban environment also needs to be taken into account. In these cases it is tempting to demand a unified city+building data model for such domain-spanning problem sets. In fact, efforts have been made to import IFC building information into city models (e.g. the LOD concept in CityGML). Enriching existing models by adding concepts or attributes of other model schemas may be reasonable in some cases, but in view of the considerations regarding model theory and model design referred to previously, format conversions and data imports cannot serve as a general approach for multi-model management. Taken to the extremes, the huge bandwidth of imaginable problems would in the end lead to complete unification of countless different model types. The resulting supermodel would either be ambiguous and inherently inconsistent because of the many intertwined and unrelated decompositions, or favour one decomposition. For those huge models like the IFCs, much work has been done to avoid ambiguities between participating subdomains. This harmonization process cannot always be 100% complete, but it always ends up with a decomposition being partly viewed as dominant by specific participants.

Multi-model analysis scenarios can either include data models that are completely disjunct or have a conceptual overlap. In the first case, models do not share similar concepts. For example, comparing the element types of a model for future energy cost trends is totally unrelated to a building model. This is crucial, since energy saving investments can only be realistic if the costs for energy conservation (e.g. better thermal isolation) and the lifetime energy costs are both included in the analysis. The dependency between energy costs and the costs for energy conservation can then be analysed by accessing both independent models plus a linking model in between. In this case, a model of the thermal energy flow is the link between the physical characteristics (structure, material) of the building and the absolute amount of thermal energy that can be transformed into costs, given a cost-per-energy unit factor.

If a conceptual overlap between the models exists, it may be the enabling factor for the joined participation of both models in an analysis. Imagine a citywide analysis of the potential solar energy that might be captured within the city area. In the city model, the concept of a building exists, but it is not precise enough to extract relevant information about building roofs such as size and orientation (where solar equipment could be placed)

or even the suitability of the underlying construction. A building model on the other hand does not have enough urban context information (location, orientation, cadastral assignment).

Another issue of practical importance for data handling is the following: Assuming that logically-identical buildings will have identical IDs in both data models is unrealistic. Other characteristics have to be consulted to identify associated objects in both models. In this example, street name and number may occur in both models, thus enabling a non-ambiguous paring of both occurrences.

So far, we have seen some of the problems that can arise if data models form the basis for cross-domain analysis. The main problems are:

- ID ambiguities between the different data models
- Different ("dominant") decompositions
- Different conceptual design decisions (loose coupling versus fixed compositions, indirections, aggregated values vs. detailed attribution)
- Paradigm breaks (impedance mismatch, e.g. object-oriented vs. relational)
- Different syntax

## 2.5 Requirements for cross-domain models

Analysis models constructed from source models of different domains should not suffer from the formal and conceptual complexity mentioned above. Pushing the characteristics and complexity of source models forward into the analysis model would obviously hamper the implementation of analysis algorithms. A level of abstraction between analysis models and source models is necessary. In this sense, the analysis model is a model of a model, that is, an additional meta level on top of the existing abstraction level. Therefore, the well-known principles of modelling (abstraction, reduction, representation, and pragmatism) also hold true for the creation of analysis models from source models. Analysis models in particular have to be more abstract than the base models they rely upon, because analysis models address specific problem domains, in contrast to common standard models such as IFC or CityGML, which address a whole spectrum of participating disciplines. This goal can only be reached by the reduction of the types exposed by the source models to the absolute minimum required for the analysis. Currently, some common standard models are envisioned as an all-purpose model for a whole league of disciplines. Although conceptually striking, the concept of a universal language has not yet reached a significant level of relevance. Humans prefer to represent their thoughts in many different languages specific to their cultures. All languages have their own advantages and expressiveness that make them a unique part of their culture. Putting it this way, an analysis model uses concepts unique to the respective problem domain. The language concepts are special to this particular domain, expressing the expertise of the discipline in a characteristic and well-chosen terminology. Compared to the overall common language, the set of concepts in a terminology is reduced by number, but extended by semantics. Words from natural languages may be reused, obtaining special meaning in an expert terminology, but the expert language itself is reduced to the absolute minimum necessary to express the concepts of the expert domain. Reduction to this relatively small set of language elements means an expert language can be more precise, less error-prone, and less eloquent. This has been done intentionally for pragmatic reasons: to supply a compact lingo for the expression of problems typical to the domain, and communication between domain experts.

A cross-domain multi-model analysis makes the necessity of an additional abstraction layer on top of the base models especially obvious. However, the implications of model theory and the different perspectives between expert domains and global models still hold true even in the case of only one base model being involved.

Consequently, the abstraction layer between base models and the expert domain model (in this case the analysis model) should support:

- Analysis-specific ontologies
- Formal notations of the problem domain (e.g. naming conventions specific to the expert domain)
- Integration on different scales

- A reduced type set
- Explicit mapping between source models and analysis model

Analysis algorithms could then concentrate on the complexity of the analysis itself by using the leanest possible interface to the model.

## 3. META-LEVEL MODELLING APPROACH

How can the requirements for cross-domain models be met? As mentioned earlier, the software crisis gave a special impetus to computer science to overcome complexity problems. The new discipline of software engineering has been created as a bounding box. Not only does it include the adoption of new paradigms such as object-orientedness, it also sets the focus on the software production process, for example by defining new roles and stakeholders. A similar process can be observed in the field of product lifecycle data management. In the building industry, some product models for different purposes have already emerged. Given their complexity, similar problems, already known from the software crisis, arise. As a solution approach, the principles of general model theory have been adopted and applied, and sample solutions and best practices specified. One of these best practices is the famous set of design patterns for software, originally published by Gamma et al. in 1994, which still remains a standard work today. Although not intended to be a universal remedy, design patterns serve as a library of strategies for commonly-encountered structural and behavioural problems in software design. Product models can take advantage of the expertise and solutions found by computer science, because models are just another piece of software. In our approach, we use the façade pattern to hide complexity from applications and to facilitate a generative implementation approach.

### 3.1 Applying the model façade pattern

The façade pattern provides a simple interface for packages of higher complexity. As such, it applies the black-box-principle. The reduced complexity exposed through the interface is designed to serve a pragmatic reason: to provide the leanest-possible dependency between the consumer and the supplier of functionality. The interface is a contract that consumers can rely on and implementers have to comply with. Any implementer who fulfils the interface contract is a valid supplier of that functionality. This abstraction creates the desired independence and paves the way for reuse and interchangeability. These general interface considerations have been further specialized in the façade pattern. It provides an abstraction of the underlying packages. In fact, consumers do not need to know about the details of the underlying packages at all.

Transferred to the model scenario, the façade pattern reads as follows (Figure 3): The analysis model provides an interface with an appropriate degree of complexity to hide underlying models and their complexity, and which may be higher than desirable in the given problem context. The mapping façade implements this interface (Figure 3, top). As long as models can comply with the interface contract, they can take part in the implementation of the interface. In Figure 3, model A and model B together yield the necessary information for the implementation of the façade. As an alternative, model B could be replaced by model B', which also exposes the right interface to fulfil contract a. This abstract design enables the exchange and reuse of models as long as

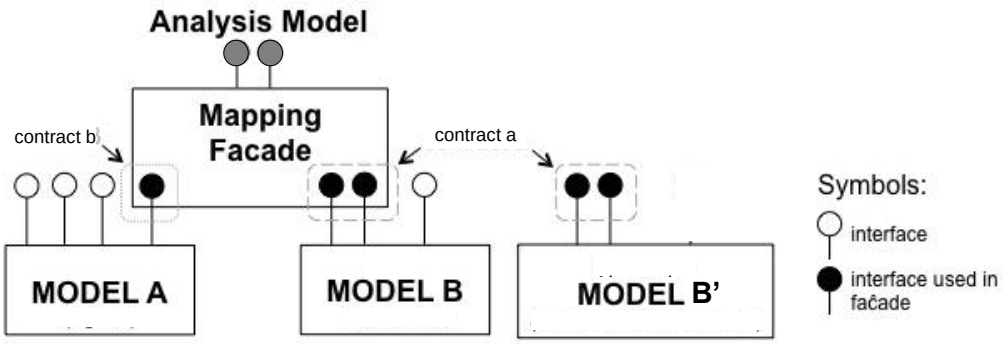



Figure 3: Façade pattern and interchangeability of models

they fulfil the required contract. This also implies that more than one model may be capable of supplying the complete interface requirements. Interface functionality can be provided by a whole set of models if necessary.

Now that abstractions and dependencies have been defined, the implementation process for the interface has to be shaped out. After the target model (here: the analysis model) and the source models have been specified by domain experts, the question is, who implements the mapping between them? Each element in the target model can be assigned to at least one element in the source model. This assignment can be expressed by relating element types, but a pure type-based mapping definition alone may be insufficient. Depending on the problem, it also has to take characteristics of the model objects (instances) into account. As an example, the cardinality of both endpoints may be as general as an m:n-relationship, where m and n can be anything from 1 to infinite. This depends on the problem (e.g. model analysis) and the actual model element instances. Therefore, the implementation of the mapping may include an aggregation of elements from source models to the target model (1: many) or a decomposition of source model elements over target model elements (many: 1) (see Figure 4). Only the implementer of the façade needs to know about this problem-dependent mapping logic. The aggregation of cost elements into a summary cost value is an example of such a mapping. Not only do cardinalities have an impact on the mapping logic, the mapping may be influenced by characteristics that are not direct parts of the mapping partners. This may give the mapping the appearance of a constraint. As an example, in summing up single values, there might be a specific threshold value below which values are ignored. We will see in a later example (section 5) that rooftop segments too small for a solar panel are not counted when calculating the potential rooftop area available for power generation. In Figure 4 the mapping ignores all the properties of the elements in model D except the cost values, which are summed up in a mapping algorithm. The cost information is scattered over the model, therefore the mapping algorithm has to collect this information. To do this, the algorithm needs to know how to calculate volumes and how to apply unit costs.

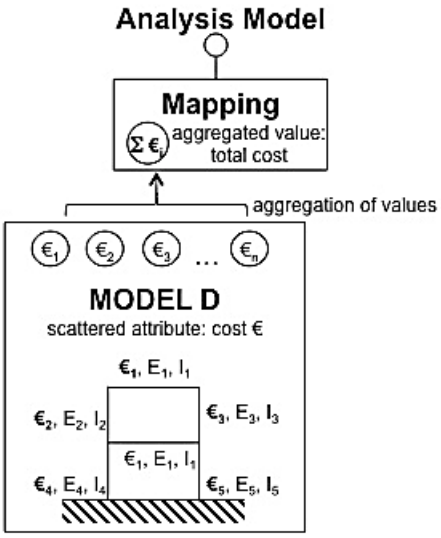



Figure 4: Mapping logic and cardinalities

**Target model ontology**

Experts use the vocabulary and structures commonly accepted in their domains of expertise. The framework concept presented here allows the definition of these elements so that the ontology of the target model reflects the perception of domain experts. The decomposition of a model used in an analysis can then no longer be perceived as dominant (or even tyrannical), because experts can design exactly the decomposition they find useful for a given type of analysis. Formal complexity resulting from the necessity to transform a given structure of a standard model into the usual format of a domain can be kept out of the formulation of the analysis itself, especially out of the logic of the analysis algorithm. The formal complexity of specifying the mapping between elements of two model types has been completely encapsulated inside the mapping component that is managed separately. Through this, the mapping complexity is kept entirely out of the scope of the domain expert, who can concentrate on the analysis problem itself.

**Schemaless specification**

When talking about data models, we do this mostly on the assumption that the schemas of all participating data models are clearly defined. However, the generation of an analysis model cannot be performed merely as a schema transformation between source model schemas and the schema of the analysis model. As mentioned before, the characteristics not only of types but also of instances can be important. The façade implementation must be able to put this requirement into practice. The question is whether the mapping logic needs to know any detail about each participating schema. More precisely, does it need to know about each complete schema or only about the part that is going to be mapped? Obviously, the schema of the target model must be known to the mapping logic, otherwise it would be impossible to produce the required objects in a valid format. Or to put it another way, the only elements of the target model that are instantiated are those for which a mapping specification exists. In contrast to this, only those elements of the source models occurring in a mapping specification are of interest to the mapping logic. Since the reduction of complexity is the main reason for creating target models from source models, it is very unlikely that the whole range of source model element types will be processed by the mapping logic. The instantiation of the relevant analysis model objects is the main functionality of a façade implementation; the resulting model must of course be a valid representation of the analysis model schema. In an implementation, this can be checked in a post processing step. It frees the mapping process from the huge type convention overhead. Type-based mapping relations can be specified declaratively without complete coverage of all model element types encountered. Moreover, the type-based mapping specification can be further extended with object-based mapping constraints.

**Elements of a mapping specification**

Element sets are commonly specified declaratively with query languages such as SQL or OCL. They consist at the very least of the specification of the source elements on which the query operates and the specification of target elements onto which source elements will be projected (mapped). The projection does not need to be one-on-one; it can also include functional or structural relationships. Functional relationships can be simple conversions such as data type or unit conversions, but also algorithmic constructs of any kind, e.g. a sum of cost values. Structural relationships can be aggregations of sub-elements into elements on a higher hierarchical level, e.g. an element for summary cost information. Beyond the simple pass-through of elements, the result set can be further constrained by specifying conditions on source elements. While a correct specification of source elements and projections requires knowledge of the source and the target types, constraints can include characteristics of element types involved (schema knowledge) as well as of element instances (e.g. attribute values) as shown in Figure 5. The mapping specification may reference one, two, or more source models. In SQL or OCL, queries on distinct sets of elements can occasionally be joined based on their concordant IDs. As mentioned above, it cannot be assumed that participating object-oriented data models implement object identity by means of IDs. Therefore, an alternate mechanism is required to express object identity and to support target object construction through the merger of two source model objects.

A complete mapping specification consists of a source model element specification, a target model projection, and optional mapping constraints and – if necessary – a generic ID definition capable of expressing cross-model links (joins) between associated model concepts.

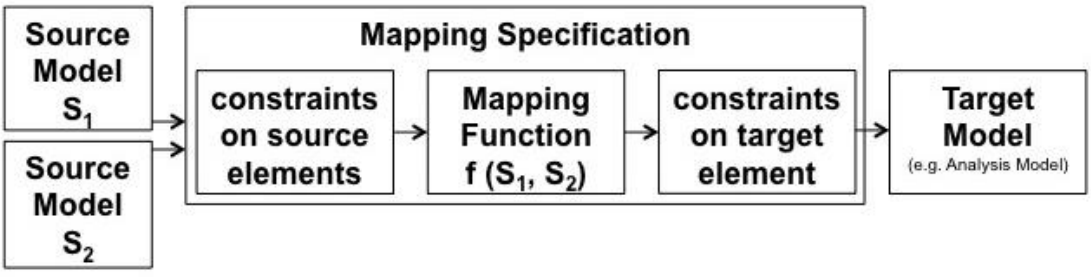



*Figure 5*: Target model element projection

**Referencing source model elements**

Data model types are valid in their distinct namespaces. To avoid ambiguities between participating source models, it is necessary to specify the namespace of a source-model element in a source-model element specification by specifying its fully-qualified name. Additionally, a method for matching the identities of objects across source models must be specified. This specification allows for the subsequent creation of objects in which the characteristics of both source objects are combined. This is equivalent to inner join declarations in SQL and OCL. The source model element specification has knowledge about the source data model but is completely unaware of the target data model.

**Target element projections**

Once source model elements have been specified, the target model ontology can be instantiated according to a projection specification that specifies the name and data type of the target model object. As a simple example, an attribute value of a source-model object could be copied onto an attribute value of the target model. Slightly more complex, the sum of many attribute values of a source model could be stored in the respective aggregated attribute value of the target model. There could also be a complex calculation taking different source-model attributes as input parameters and writing the result of the calculation into the target model, e.g. the calculation of a thermal transmission value from layers of a wall element (Figure 5, mapping function). The projection is the only place in the whole mapping specification where elements of both the source and target model are known.

**Mapping constraints**

Restrictions on the resulting element set can be specified by expressions. Figure 5 shows constraints on source elements and/or on the resulting target elements. Only matching elements on the source-model side will then be respected and instantiated on the target-model side. Theoretically, conditions can be specified on source element attributes or on target model attributes. In the latter case, a target model object will be constructed and potentially discarded at a later stage if constraints are violated. In the former case, source model objects that do not meet conditions will not be part of the set of elements projected into target model elements. The difference can be substantial, since accumulated values may show different results depending on the order in which restrictions are applied. An appropriate order of constraint application cannot be given without knowledge of the problem. In one case the intention may be to filter out values below a certain threshold value, while in other cases the accumulated value itself may be subject to restrictions.

**Cross-model identity**

Some data models support the concept of global unique identifiers (GUIDs) for all objects. This makes ID-based object mapping an easy task, but the concept is not generally supported. Moreover, in the object-oriented world there is no need for IDs conceptually, and the uniqueness of objects can be achieved another way. It can be achieved through special values for an object's attributes, thus linking uniqueness closely with the object semantics and not an artificial id value. Since specific attribute values make an object unique, these attributes must be specified in an identity definition. Besides the fully qualified name of the object type, the definition specifies a global name for the identity definition. As an example, the global id of a building in IFC (ifc:IfcBuilding\@GlobalId) could be linked to the id of a CityGML Building (city:Building\@gml:id). Based on these global names, identities can be matched across two or more models. As long as identity attributes in the identity definition of one participating data model can be transformed into the attributes of another model's identity definition, the matching process can work automatically. This would be the case in simple scenarios where both parts, for example, contain attributes for street name and number in different attributes and different data types. Transformation is then possible in at least one direction.

A different case arises if automatic transformation is impossible. This would be the case, for example, if one data model supports the GUID concept and the other relies on object semantics to support uniqueness. In this case, it might be necessary to specify the mapping relation explicitly in a name value pair construct (e.g. a mapping table). Although the manual creation of a mapping table is a tedious and error-prone task, it might be inevitable in some cases. The functional relation between the IDs of both data model objects then includes a mapping table lookup, as might be the case when mapping IFC building objects (GUIDs) onto CityGML objects (street name / number or cadastral ID). The mapping table would be required for each target / source model pair - clearly an additional effort compared to a merely type-based declarative mapping.

## 3.2 Formal mapping notation

The schema of a formal mapping notation has to reflect the requirements mentioned in section 2.5. It has to be able to specify the mapping relation between ontologies of the source models and the ontology of the analysis model. It has to be able to express simple data types as well as complex types. Instances of types (a.k.a. objects) can be organized in different topologies such as deeply nested part-of relations, trees, lists and more. The schema has to support these different kinds of aggregational relations. Figure 6 shows the mapping schema in the form of a class diagram.

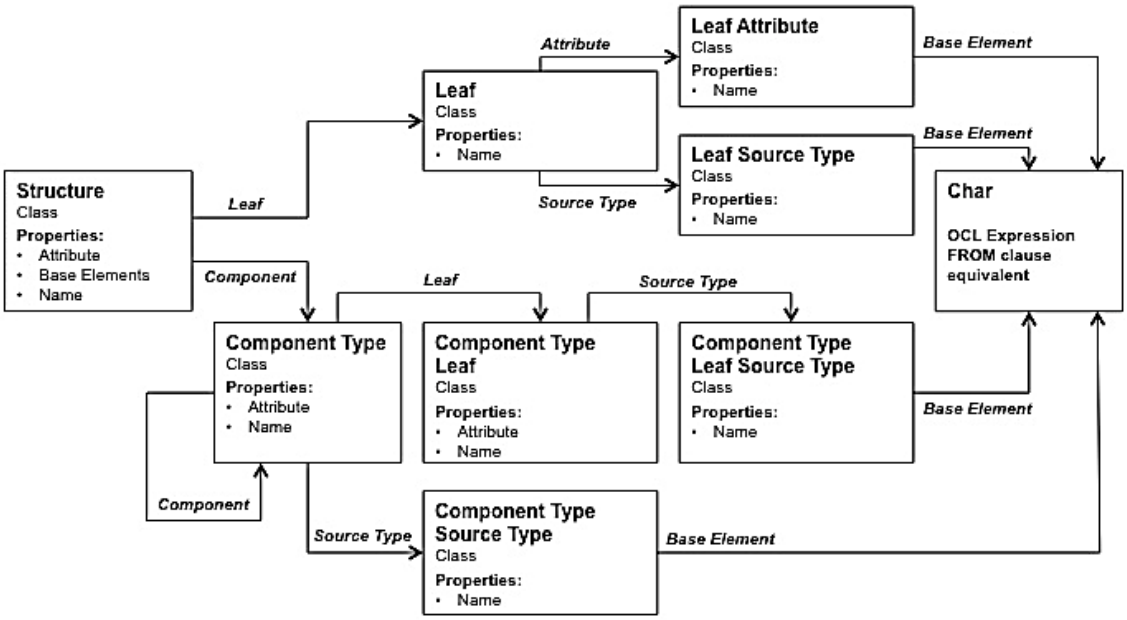



Figure 6: Elements of the model mapping schema (class diagram)

As shown in Figure 6, complex types (class ComponentType) can be specified that consist of simple types (class ComponentTypeLeaf) or hold a subordinate level of one or more complex types. With these simple type definitions, objects of arbitrary complexity can be generated. The Name attribute of each schema element specifies the type name the generated object will have in the expert domain. The Structure class serves as a container for a whole set of objects. This, so far, meets the requirement of being able to create arbitrary ontologies on the target side. What about the source of data that makes up generated analysis model objects? The BaseElement class holds the specification of the pool of objects that will be queried. It resembles the FROM clause in the query language OCL used in the framework prototype (section 4.1). The SourceType class resembles the SELECT clause in a query language. It specifies the projection of source data onto target data. As an example, if BaseElement specifies all the objects in a model (e.g. by specifying its enclosing container), then setting the SourceType attribute to "IfcWallStandardCase" and the Name attribute to "Wall" would result in a set of objects all of which were of the Wall type. One Wall object is instantiated for each object of type IfcWallStandardCase encountered. The Wall object does not as yet have any attributes or aggregated sub-objects. This does not satisfy requirements in most cases, and the result set usually needs to be populated and refined further. This can be achieved by constraining the source objects, thus limiting the instantiated target model objects to the amount of source model objects with special characteristics, such as attributes with predetermined values (wall thickness, material and the like). Assigning a child element of the type ComponentTypeLeaf will cause an attribute to be generated in the target model object. The generation of the attribute will follow the same logic as in the case of the generation of the ComponentType object itself: The BaseElement attribute of the mapping specification will specify the pool of objects queried, SourceType will specify the projection. Constraints, the equivalent to a WHERE clause, can be specified in the Attribute field. It specifies a query (SourceType, BaseElement) and an expression that the query result has to meet for the object to be created.

**Model-driven façade generation**

The concept of a façade between source models and an analysis model can be specified using the formal mapping notation described in section 3.1. In order to put this concept to work, a software prototype has been implemented. In this prototype, the façade is a component that exposes the specified analysis model instance through its programmable interface. In order to generate this analysis model instance, one or more source model instances are specified. Source model types and generated analysis models may vary with every given specification; the mapping logic inside the façade has to reflect this. Therefore the façade logic must be capable of adapting to a new specification either through interpretation of the mapping notation, or by creating the implementation of the mapping logic anew. A broad spectrum of participating model types is possible. Not only can an analysis model expose a wide range of different ontologies, the source models can also be of any type (and not just IFC and CityGML). An interpreter (Figure 7 a) capable of handling all these possible different types and mapping constructs would be overly complex. Therefore, the model-driven software development approach has been taken. It generates a new façade implementation from every specification (Figure 7 b). The code generator is much simpler than a one-fits-all interpreter. In addition, this approach is more abstract, because the façade implements an additional level of abstraction between the Platform-Independent Model (PIM) and the analysis model. In contrast to an interpreter, which would implement the whole analysis model generation process, the model-driven approach separates the mapping logic from the model generation process, especially because the generator has no knowledge of specific model schemas, as shown in Figure 7.

**Platform-Independent Model (PIM)**

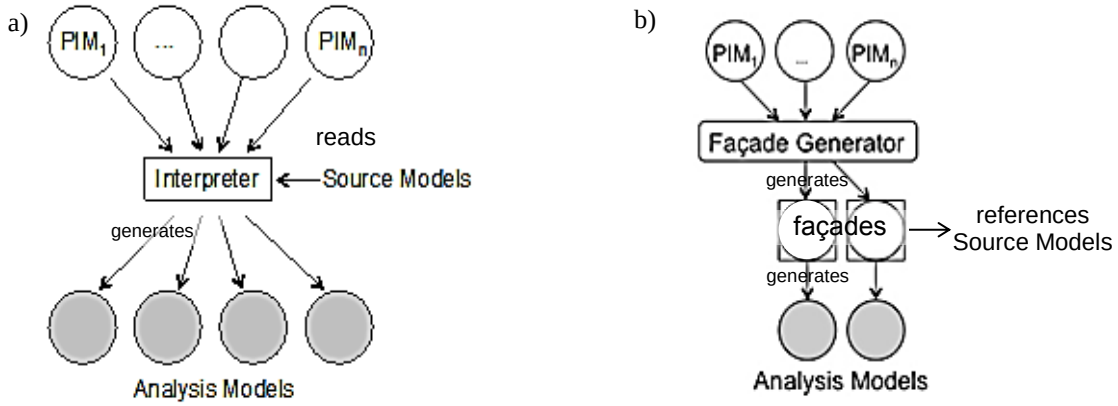



Figure 7: Mapping logic implementation alternatives a) interpretation and b) generated mapping code

In the model-driven approach, a platform-specific model is generated from a platform-independent model (PIM). In our case, the mapping specification is the PIM that describes the relationship between the analysis model and the source models on an abstract conceptual level in a software-platform-independent way. There are no implications for implementation or software environments. Furthermore, it does not imply any design decision about the software components to be generated. The decision about the structure of the generated façade is made solely in the generator that transfers the PIM into a platform-specific software format that can later be executed in a specific software environment.

**PIM-to-PSM Transformation**

A generator takes a PIM as input and translates it into a platform specific format, the platform-specific model (PSM), e.g. a programming language. This so-called PIM-to-PSM transformation transforms a model from a higher level of abstraction into a concrete environment, e.g. a software runtime environment. In our case, the model mapping specification is transformed into C# code for the .Net runtime environment. The generated software follows the façade design pattern. Generators could be exchanged for addressing different target platforms or different software designs. This approach is aligned with the concept of model-driven software development (MDSD).

**Platform-Specific Model (PSM)**

The end product of the model-driven generation of software is the platform-specific model, in this case a software component that implements the façade design pattern. The component is ready to be integrated into the modelling framework. It can access instances of specific source model types as input, and create instances of one specific type of analysis model as output. For each mapping specification a façade component will be generated and integrated into the framework.

## 4. CONCEPTUAL FRAMEWORK AND SOFTWARE PROTOTYPE

In the approach presented here cross-domain model analysis workflow consists of the following steps (Figure 8):

- PIM Specification
  1. Analysis model schema definition
  2. Selection of appropriate source model types
  3. Specification of model mapping
- Generation of a façade component in the PIM-to-PSM Generator using the PIM Specification
- Selection of source model instances
- Creation of an analysis model instance in the model generator using the generated façade
- Specification of an analysis algorithm module
- Application of analysis logic to the analysis model in the analysis manager
- Presentation of the analysis results (e.g. visualizations)
- Storage of analysis results

The PIM Specification steps are covered conceptually and can be specified using the model mapping notation. The following logical steps are performed by further processing the PIM Specification. The software prototype, as a physical realization of the framework concept, consecutively executes all these steps, from façade generation to the application of analysis algorithm. Figure 8 shows the sequence of steps performed by the framework.

Separation into these steps enables the division of tasks between the domain expert and modelling expert. Both experts must collaborate closely to get the mapping of concepts between the analysis model and base models aligned. The modelling expert needs a basic understanding of the analysis model semantics in order to specify an appropriate mapping, but because of the separation, both experts are not confronted with the full complexity of

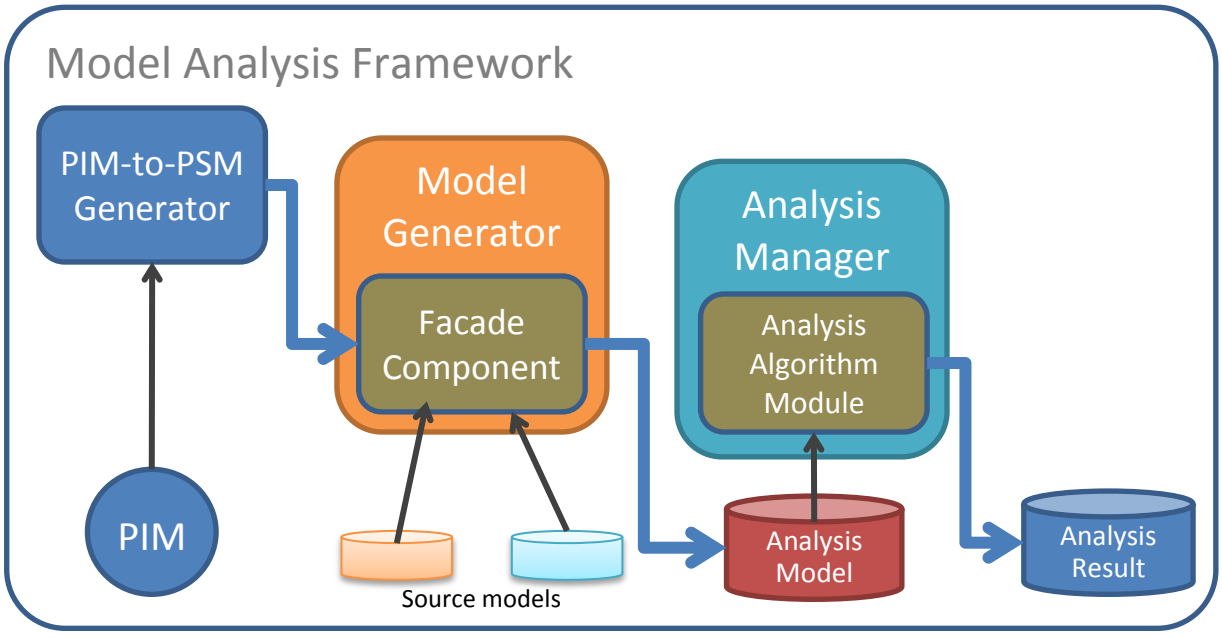



Figure 8: Workflow in the Model Analysis framework

all the participating models. All steps are clearly separated and communicate through well-defined interfaces;

components that implement one step can be exchanged and reused independently. Therefore, the same analysis model definition can be reused, while the source models can be exchanged or the analysis model enhanced while maintaining the underlying source models. Further along in the process, the analysis logic can be re-applied or improved, while the analysis model remains the same.

## 4.1 Physical framework prototype

The modelling framework runs as a service, without any user interface. It exposes interfaces for analysis model specification, mapping specification, analysis algorithm specification, analysis result storage and visualization. A graphical editor has been developed to ease the specification of the analysis model and model mappings. For the administration of analysis projects, especially the management of participating models, code generation and integration of analysis algorithms, a framework management console has been developed on top of the framework (Figure 8).

The schema of the analysis model is physically expressed in the XML-based model mapping notation. It can be specified in any text editor or XML editing tool and then passed over to the modelling framework. The framework will then validate the model mapping document against the model mapping schema, and formally invalid specifications will be rejected. The manual development of complex schemas would be a tedious and error-prone task. For this reason, the graphical analysis model editor supports domain experts in designing an analysis model schema. Figure 9 shows a screenshot of the graphical editor. The editor generates the XML equivalent of the graphically specified analysis model in the model mapping notation. This XML document holds the notation in a validated format and is ready to be processed by the framework. Analysis projects can be defined in the Framework Management Console (Figure 10). A project consists of a PIM (Tab "Analysis"), as many references to model instances (Tab "Source Models") as were referenced in the PIM, and an analysis module that implements the analysis algorithm. The framework can then process the project by running through the steps shown in Figure 8.

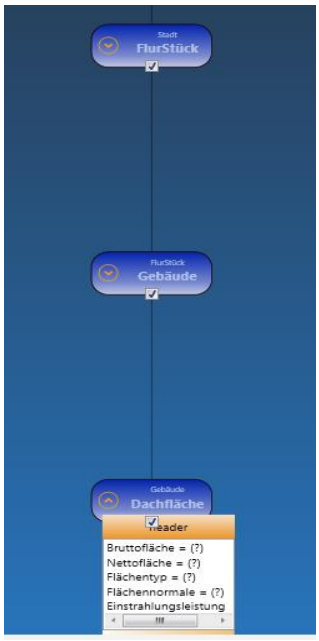
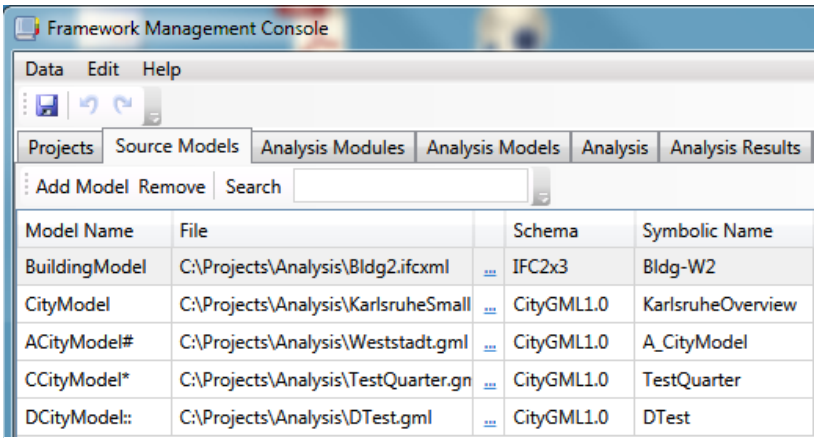



Figure 9: Graphical editor for analysis model definitions




| Model Name | File | | Schema | Symbolic Name |
|---|---|---|---|---|
| BuildingModel | C:\Projects\Analysis\Bldg2.ifcxml | ... | IFC2x3 | Bldg-W2 |
| CityModel | C:\Projects\Analysis\KarlsruheSmall | ... | CityGML1.0 | KarlsruheOverview |
| ACityModel# | C:\Projects\Analysis\Weststadt.gml | ... | CityGML1.0 | A_CityModel |
| CCityModel* | C:\Projects\Analysis\TestQuarter.gn | ... | CityGML1.0 | TestQuarter |
| DCityModel:: | C:\Projects\Analysis\DTest.gml | ... | CityGML1.0 | DTest |

Figure 10: Framework Management Console

A domain expert can specify an analysis model agnostic of any concrete model instance. References to model instances in the analysis model definition can be purely symbolic. These references are resolved when the PIM is processed by the framework. The creation of analysis model elements depends on the availability of appropriate

source model data. Data may be available in different sources of different types. The mapping specification between the analysis model and source models has to take the schema of the source models into account. Therefore, a decision has to be made on the model types (IFC, CityGML, etc.) serving as data sources before a modelling expert can specify the mapping expressions. The modelling expert uses the XML-based modelling mapping notation (Figure 6). This again can be a challenging task, therefore the graphical analysis model editor supports the specification of model mappings. The editor assures that only valid source model element types can be inserted into the mapping expression. The resulting XML document is then ready to be processed by the framework.

The steps are summarized in Figure 11. In the definition phase, the domain expert and modelling expert work together to specify meta information for the relation between the analysis model and related source models. The mapping specification contains all the information necessary to create the software component that implements the mapping logic. In the processing phase, a model façade using the meta information is created by the framework. The code generation component of the framework takes the model mapping meta information as input and creates platform-specific code (C#.Net), which will be dynamically compiled within the framework into an analysis model component. In the management console, source models are specified and the generated façade can generate an analysis model. The framework manages all the different types of model mapping specifications and the resulting analysis models.

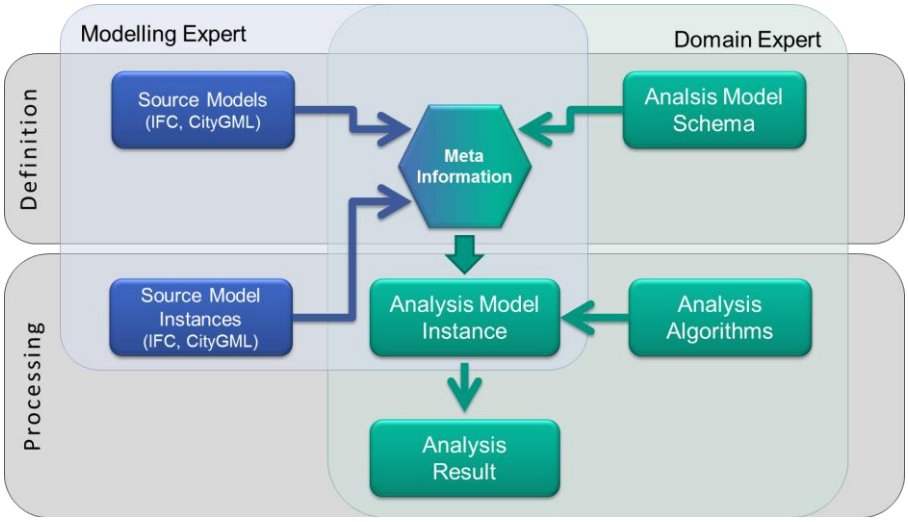



Figure 11: Workflow of analysis definition and processing.
Cooperative work of the modelling and domain expert

# 5. EXAMPLE ANALYSES

## 5.1 Example 1: Public transportation and city quarter information

In this example we want to show cross-model analysis in the urban context. A city administration responsible for public transportation is planning a new bus route to enhance the system. The route should link business areas and residential areas and provide an alternative option to travelling by car, with the aim of reducing office-hour traffic significantly.

**Algorithms for analysis**

Different route alternatives have to be compared (Figure 12). The catchment area of the optimal route would collect most commuters in the morning, and allow them to disembark as close as possible to their workplaces and vice versa in the evening. Although linking everyone's home with their workplace is impossible, optimizing the traffic route by identifying the embarking and disembarking areas that provide the best coverage still seems statistically sound. The floor area of all the buildings in the covered region is summed up, separated into the different occupancy types, such as 'office', 'private', etc. In this simple example we exclude other considerations

such as route length, travelling time, route switching and so on. The focus is on domain-spanning analysis and the advantages of inserting an abstraction layer between the software-centred and domain-knowledge-centred view. We could use virtually the same algorithms to place block heat power plants at optimal locations in the urban area. More importantly, the separation of concerns into two separate physical layers would promote the re-use not only of concepts but also of components. Changing the investigation from bus routes to block heat power plants requires references to be changed in the PIM and minor changes in the algorithm but the underlying models can be re-used.

**Data sources**

The communal cadastral system, often simply a Geographic Information System (GIS), contains the city map with real-estate and city road information. Each real estate entry has a reference to an electronic building

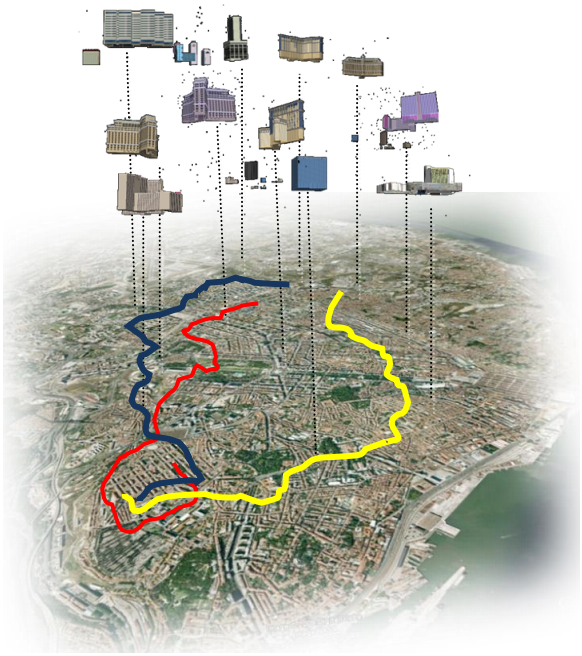


Figure 12: Potential routes for a new bus line

document containing an IFC-based model of the building. The technical specification is of no relevance for the concept.

Basic assumptions

- The coverage area is calculated assuming a maximum walking distance to the route. Beyond that distance it would not be attractive to utilize the bus line.
- An estimation of the number of people involved is given by the person ratio per square metre of office floor and per square metre of private home floor respectively.

**Involving different algorithms**

As a first rough estimation, the coverage area could be calculated by applying a direct surface-to-surface connection between building and bus route. Buildings within the maximum distance would belong to the coverage area.

In a more meaningful (but also more time-consuming) calculation, the exact walking distance between the building location and bus route could be calculated using a navigation service (e.g. Google Maps).

Pursuing the concept of dynamic business logic involvement through the use of components loaded at runtime, different algorithms for calculation could be consulted declaratively.

## 5.2 Example 2: Potential for inner-city green energy production

In this analysis, the potential for solar energy production in a city environment is investigated. To this end, rooftop surfaces suitable for the exploitation of solar energy are calculated. CityGML models with a low level of detail (LOD) can be derived from the base areas of buildings using cadastral information. This approach is simple and inexpensive, but lacks precise rooftop information, as shown in Figure 13. A higher level of detail can be achieved by airborne laser scanning, which is quite expensive, or by importing the required information from IFC models. The latter approach is not taken here since we want to keep redundancies out of our modelling strategy. The mere shape of a rooftop might not supply enough information for the analysis. Also, the construction type of the roof and the utilization of underlying spaces might have to be taken into account, a level of detail not envisaged in CityGML.

From this perspective it becomes apparent that one single model type cannot provide the necessary information for conducting the analysis. Cadastral databases or city models (e.g. in the CityGML format) may - depending on their level of detail - include only rough shapes of buildings, with imprecise or missing information about

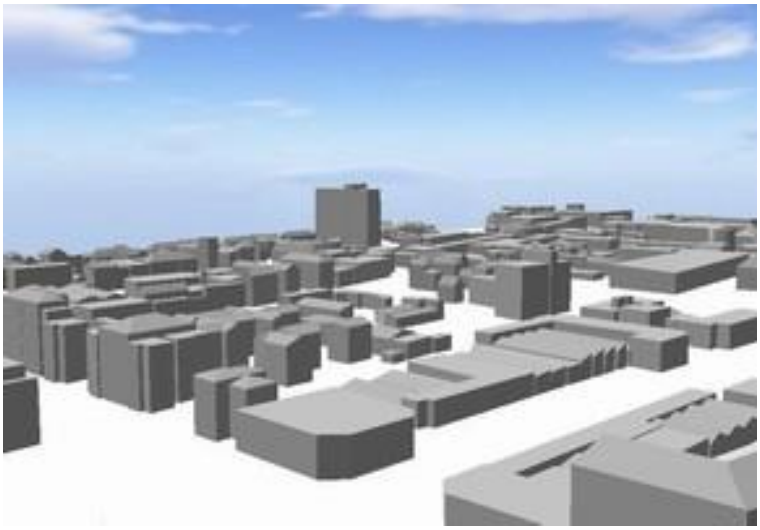


*Figure 13*: Building representation in CityGML with low level of detail.
*Source: www.citygml.org/index.php?id=1539*

rooftop areas. On the other hand, building models (e.g. in IFC format) lack the information about the urban context, e.g. shadowing effects by other buildings, orientation, and city quarter or cadastral field assignment. If no appropriate city model with the necessary level of details is at hand, only the selective combination of information contained in both models yields the required result. Other investigations (Alam, N. et al. 2012) either used a CityGML model with the necessary level of detail or considered using airborne laser scanning point clouds (LIDAR) to generate sufficient 3D model data. In our example we assume that IFC models for all city buildings are available. In combination with a city model this is a relatively simple approach for generating the necessary 3D data. The effects of shadows caused by nearby objects could also be examined with this 3D data. This would be a good example of model refinement and re-use, which is clearly possible with our approach. However, the focus here is not on presenting a precise solar insulation model but on showing a modelling approach aligned to the principles of model theory and a complexity scaled to the problem statement.

In this scenario the types of data sources are clear. The city model is given in a file containing CityGML data, and the buildings are supplied in IFC-format files. After the domain expert has passed the model to the modelling expert, mapping assignments between IFC building elements and CityGML building elements can be specified. Each building in a CityGML model needs to be associated with an IFC building. The modelling expert has to find attributes in both building concepts that are suitable for expressing this association. In this example

we follow a generic approach, and assume that the street name and number attributes are suitable as pseudo-IDs in both models[1]. For each building element in the city model, a building element for the analysis model can now be instantiated. Through the pseudo-ID, the size and orientation attributes of this element can be initialized with the respective values of the IFC building element.

The mapping notation for the attribute assignment of a rooftop segment appears as follows:

```
<Structure xmlns:xsd="… omitted for brevity .."
           xmlns="http://analysis.com/Namespace">
  <!-- DomainObjects $IfcBuildings and $CityGmlBuildings -->
  <Name>SolarInsulationModel</Name>
  <Component>
    <Name>Roofs</Name>
    <SourceType Name="Roof" BaseElement="$CityGmlBuildings"/>
    <!-- create a Roof object for every CityGML building encountered -->
      <Attribute>
        <!-- The Roof object has an attribute named CityGmlID -->
        <!-- copy the CityGML building ID into the attribute -->
        <SourceType Name="ID" BaseElement="$CityGmlBuildings.Building"/>
        <Name>CityGmlID</Name>
      </Attribute>
    <Component>
      <!-- The Roof object aggegates an object named Segment for each Slab encountered
      for every IfcRoof.Segments element -->
      <Name>Segment</Name>
      <SourceType Name="Slab" BaseElement="$IfcBuildings.IfcRoof.Segments"/>
      <Attribute>
        <!-- Attibute SegmentSize copied from
              DomainObject IfcBuildings subordinate attribute -->
        <SourceType Name="Size" BaseElement="$IfcBuildings.IfcRoof.Segments.Slab"/>
        <Name>SegmentSize</Name>
        <Value>Size>2.0</Value>
      </Attribute>
      <Attribute>
        <!-- Attibute SegmentOrientation copied from
              DomainObject IfcBuildings subordinate attribute -->
       <SourceType Name="Orientation"
                   BaseElement="$IfcBuildings.IfcRoof.RoofSegment.Slab"/>
        <Name>SegmentOrientation</Name>
      </Attribute>
      <Attribute>
        <SourceType Name="ID" BaseElement="$IfcBuildings"/>
        <Name>IfcID</Name>
        <!--joining element sets based on associated IDs -->
        <Value>$IfcBuildings.ID==$CityGmlBuildings.Building.ID</Value>
      </Attribute>
    </Component>
  </Component>
</Structure>
```

[1] In this case mapping would also be possible based on building IDs. This would require a manually-created ID mapping table between the IFC building id and the CityGML building id.

Given the fact that this mapping specification operates on IFC elements, the above XML notation looks surprisingly simple. Normally one would have to dereference a tedious chain of dereferencing expressions in order to get from IfcRoof elements to its related segments (roof slabs), and from there to the surface area of a segment and the normal vector of the segment surface. This would clearly overstrain the patience of the reader, and also of the user of the mapping notation. Therefore, the framework supplies a facility called 'DomainObjects'. It supports transient object set-creation by processing an OCL query string at runtime. The result of the query is a set of objects matching the projection part of the query. This query expression would otherwise have to be put into the `BaseElement` attribute, which would harm readability of the notation significantly. In our case, `CityGmlBuildings` objects and `IfcBuildings` are created beforehand in a pre-processing step. The `CityGmlBuildings` object contains just as many `Building` objects as there are buildings in the CityGML model. The only reason the `Building` object holds the CityGML building ID is for subsequent joining with the equivalent IFC building object. The `IfcBuildings` object is also a condensed view of the IfcBuilding objects of the IFC source model. For the planned analysis we need to know more not about the building, but about its roof and the segments (slabs) being used for the roof. Even the segments are stripped-down versions of the original source, holding only a size and an orientation attribute. The size attribute has a constraint, prohibiting the creation of Segment objects if the size value is less than 2.0. The last attribute in the Segment component ensures that Segment objects are only created if the IDs of the building in the IFC and in the CityGML model match. Remember that we do this on the assumption that a generic way of transferring the IDs of one model type into the IDs of the other model type exists. If not, the DomainObjects facility could also handle explicit mapping via a manually created mapping table.

After the analysis model specification and the mapping specification have been completed, actual model data can be added to the analysis project of the framework. This is done using the framework management console by adding the respective model files to the analysis project. The project then contains all the data needed to create an instance of the analysis model. This can also be done through the framework management console which calls the respective interfaces of the framework. While the mapping specification itself is purely based on types, constraints work primarily on instance data. In this case, only roof segments with a size larger than the specified minimum value should be recognized and used in the analysis model.

The constraint expression can either contain constant values as stated above, or variables that need to be replaced with the respective values[2]. The creation of an analysis model as a pragmatic and condensed view of the whole problem field is an important part of the whole analysis. With this model the algorithm of the analysis can be less complex. In this case it degenerates to just two nested loops, summing up all the segment sizes of one building and the summary information for each building. This simple algorithm can be passed over to the framework directly and be processed on the current instance of the analysis model. This algorithm can naturally be further enhanced, taking into account as well the orientation of rooftop segments. The resulting document would then reflect the dependency between the daily movement of the sun and the energy input of the rooftop elements. This can be done by simply replacing the analysis algorithm in the framework management console, while keeping the analysis model untouched (since it already contains the normal vector of each rooftop segment in the Orientation attribute). The scope of this example encompasses multi-model analysis, but the analysis algorithm could be further enhanced to take the daily movements of the sun and shadow effects into account. For further information on the latter, refer to Alam, N. et al. (2012).

In order to conduct the same analysis on different cities, the city model and building model instances have to be exchanged and the analysis model created anew. The analysis algorithms can then be reused on the newly-created analysis model instance.

Variation of constrained values is another flexibility offered by the framework concept. When a constrained value is varied, the analysis model has to be regenerated. The analysis algorithm then operates once more on the updated model. The different results of constraint-based model analyses are stored for subsequent comparison.

[2] For example, by using the DomainObjects facility.

## 6. DISCUSSION AND CONCLUSION

In this paper we discussed a conceptual framework for analysis involving more than one product model type. The interface layer between the application data model and an external data model (e.g. IFC) provided by the approach presented here allows for any relationship between the source and destination object to be implemented inside this layer. New compositions and decompositions as shown in Figure 2 may be established based on the requirements of the analyses. Structural and functional relations between source objects and resulting objects can be implemented. A dominance of decompositions, as seen from a special analysis point of view, may be relaxed or dissolved.

Our examples show cross-model analysis using a CityGML model and IFC models. In developing the framework we found three aspects of premier importance for the structure and the manageability of analysis projects conducted with the framework: 1. The ability to generate ontologies appropriate to the analysis. Although this sounds commonplace, the underlying source models can implement decompositions, which might be considered disadvantageous from the viewpoint of the analysis. The interface can generate decompositions aligned to the requirements of analysis. 2. Standard models such as CityGML and the IFCs exhibit a significant amount of complexity. Objective measures for model complexity have been presented in a separate publication (Hartmann, Ulrich; von Both, Petra, 2010). Applying complexity measures shows that a significant reduction in the complexity exposed to the analysis algorithm is possible with the presented framework. The reduction of types (classes) available in the underlying model types to the extent necessary for a specific analysis is an obvious indicator for reduced complexity. In addition it has been shown that an inappropriate class design may cause added algorithmic complexity. Mapping existing class design of a source model to a class design advantageous for the analysis algorithm moves this complexity to the generated interface, thus freeing the algorithm itself from this task. 3. It has been shown that model principles, namely reduction, abstraction and pragmatism are shaped out differently in different model types because of their different intentions. Putting an analysis model on top of some source models can be seen as another modelling step in itself. During this modelling step, modelling principles can be applied as appropriate for the analysis. This helps modelling principles to take effect with regard to the analysis problem domain.

The approach presented here is not limited to cross-model analyses. Encapsulation of complexity and object model mapping could encourage the development of applications that previously shied away from the effort of using complex standard models.

## 7. FUTURE WORK

The framework concept that has been developed divides an analysis process into several interchangeable parts. Source models, analysis models, mapping specifications and algorithms are the components of this scenario. This componentized approach offers flexibility in several directions. Components can be separately enhanced or completely replaced, making reuse of components in several analyses possible. Single components expose significantly lower complexity than the combined whole. Separation into areas of expertise makes it possible to spread the distinct steps in the entire analysis process over several specific experts.

The concept of the creation of problem-specific models with streamlined complexity is not limited to analysis processes. It can also be useful in any environment where processes require condensed information that is otherwise scattered over several complex models or general data sources. The approach offers a systematic way of exposing the required pragmatic interface between the model and process in a dynamic and declarative way.

## REFERENCES


Alam, N. et al. (2012). Shadow Effect on Photovoltaic Potentiality Analysis Using 3D City Models. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. 39.

Blind, M.; Gregersen, J. B (2005). Towards an Open Modelling Interface (OpenMI) the HarmonIT project. In: Advances in Geosciences. 4, S. 69-74.

Cao J., Maile T., O'Donnell James, Wimmer R., van Treeck C. (2014). Model Transformation from SimModel to Modelica for Building Energy Performance Simulation. IBPSA-Germany, RWTH Aachen University, Aachen.

Crout, N. M. J.; Tarsitano, D.; Wood, A. T. (2009). Is my model too complex? Evaluating model formulation using model reduction. In: Environmental Modelling & Software. 24 (1), S. 1–7.

Dirksen, P. W; Blind, M. W; Bomhof, T.; Nagandla, S. (2005). Proof of Concept of OpenMI for Visual DSS Development. In: Proceedings of the MODSIM 2005 conference, Melbourne, Australia.

Filman, Robert E.; Elrad, Tzilla; Clarke, Siobhn; Aksit, Mehmet (2004). Aspect-Oriented Software Development. Addison Wesley Professional. ISBN 0-321-21976-7.

Gamma, E (1994). Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley Longman, Amsterdam; 1st ed., Reprint. (31. October 1994)

Hartmann, Ulrich; von Both, Petra (2010). Metrics for the Analysis of Product Model Complexity, Managing IT in Construction, in: Proc. CIB W78 Conference, Cairo.

Hartmann, U.; Huhnt, W; Laabs, A; Pahl, P. J. (1990). "COMPAD: Kommunikation produktanalysierender Daten im ISDN-B: Meilenstein 2: Neutrale Dateien", Institut für Allgemeine Bauingenieurmethoden, TU Berlin, Januar.

Hartmann, U.; Huhnt, W; Laabs, A; Pahl, P. J. (1990). "COMPAD: Kommunikation produktanalysierender Daten im ISDN-B: Meilenstein 3: Inter-Prozess-Kommunikation", Institut für Allgemeine Bauingenieur-methoden, TU Berlin.

D'Hondt, Maja; D'Hondt, Theo (2002). The Tyranny of the Dominant Model Decomposition. Vrije Universiteit Brussel, Belgium.

Kiczales, G. (1997). Aspect-oriented programming. In ECOOP '97: European Conference on Object-Oriented Programming, Invited presentation.

Koene F., Bakker L., Lanceta D., Narmsara S. (2014). Simplified Building Model of Districts IBPSA-Germany, RWTH Aachen University, Aachen.

Leal S., Hauer S., Judex F., Eder K., Gahr S. (2014). A prototypical automated building modelling tool. IBPSA-Germany, RWTH Aachen University, Aachen.

Mutis, Ivan; Raja, R. A. Issa (2012). Framework for semantic reconciliation of construction project information. Journal of Information Technology in Construction (ITcon), Vol. 17, pg. 1-24, http://www.itcon.org/2012/1

Stachowiak, H. (1973). Allgemeine Modelltheorie. Springer Verlag, Wien.

Stinner S., Streblow R., Müller D. (2014). Dynamic uncertainty analysis of the buidling energy performance in city districts. IBPSA-Germany, RWTH Aachen University, Aachen.

Thomas D., Miller C., Kämpf J., Schlüter A. (2014). Multiscale Co-simulation of EnergyPlus and CitySim models derived from a Building Information Model. IBPSA-Germany, RWTH Aachen University, Aachen.

Wimmer R., Maile T., O'Donnell J., Cao J., van Treeck C. (2014). Data-requirements specification to support BIM-based HVAC-definitions in Modelica. IBPSA-Germany, RWTH Aachen University, Aachen.