A FLEXIBLE MODEL TRANSFORMATION TO LINK BIM WITH DIFFERENT MODELICA LIBRARIES FOR BUILDING ENERGY PERFORMANCE SIMULATION

Jun Cao[1], Reinhard Wimmer[1], Matthis Thorade[2], Tobias Maile[1], James O'Donnell[3], Jörg Rädler[2], Jérôme Frisch[1] and Christoph van Treeck[1]
[1]Institute of Energy Efficient Building E3D, RWTH Aachen, Germany
[2]Berlin University of the Arts, Institute for Architecture and Urban Planning, Germany
[3]School of Mechanical and Materials Engineering and Electricity Research Centre, University College Dublin, Ireland

ABSTRACT

Today, developing models for Building Energy Performance Simulation (BEPS) is a time consuming and costly process. Automated reuse of data from Building Information Models (BIMs) for BEPS model development is a promising approach to improve the process by avoiding manual data input. Recent work focuses on linking BIM to Modelica, a modelling language that is becoming increasingly important in the BEPS community.

This paper presents our technical development of a flexible model transformation system to link BIM with different Modelica libraries to support BEPS. We describe all technical aspects relevant to this model-to-model transformation system: model hierarchy parsing, meta-model creation, model transformation by mapping rules and rule filtering techniques.

INTRODUCTION

Traditional BEPS model development is a mostly manual and, thus, time consuming and costly process. In addition, this manual model creation process can lead to numerous errors and omissions, and inevitably adds dramatically to the cost of the project (Bazjanac et al., 2011; Bazjanac, 2009, 2008). Automated reuse of data from BIM for BEPS model development is a promising concept to address the problem (Yan et al., 2013; Jeong et al., 2014; Cao et al., 2014; Wimmer et al., 2014; Basarkar et al., 2012; Aksamija, 2012). These data exchange techniques between BIM and BEPS improve the model developing process by avoiding large manual data input.

Recent work focuses on linking BIM to Modelica, a modelling language becoming more important in the BEPS community. For example, the methodologies of Yan et al. (2013) and Jeong et al. (2014) use the Application Programming Interface (API) of a proprietary CAD tool to convert a BIM to a specific Modelica library. Their code generation focuses only on geometry conversion for one Modelica library. Cao et al. (2014) and Wimmer et al. (2014) developed a set of data mapping rules from an open BIM to a specific Modelica library. These mapping rules also focus on converting BIM data into one target Modelica library.

Both previously identified shortcomings, namely the existing support for just one library, and the focus on geometry will be addressed in this paper. Hence, this work concentrates on converting Heating, Ventilating and Air Conditioning (HVAC) systems from Industry Foundation Classes (IFC) based BIMs into different Modelica libraries. We use SimModel (O'Donnell et al., 2011) as a placeholder for IFC in our development for distinct reasons explained by Cao et al. (2014) and Wimmer et al. (2014): 1) IFC is the open standard for BIM but does not contain adequate HVAC data definitions for BEPS. SimModel is a BIM format that contains the necessary data for BEPS and the structure of this data model aligns with the structure of IFC; 2) SimModel supports data translations from Input Data Dictionary (IDD), Open Studio IDD, gbXML and IFC; 3) SimModel will form the basis for a new IFC Model View Definition (MVD) that will enable data exchange from HVAC design applications to energy analysis applications; 4) extensions to SimModel could easily support other data formats and simulations. In addition, geometry definitions contained in IFC can be imported into SimModel and HVAC definitions can be added. This acts as starting point for our work. The outcome is a prototype that transforms SimModel into Modelica data based on different Modelica libraries.

SimModel is a data model based on an XML Schema Definition (XSD), using an XML document to save and exchange model data with the other simulation tools. An important work from Reisenbichler et al. (2006) illustrates the possibility to exchange model data between the Modelica simulation platform Dymola and another engineering environment via an XML document. However, they focus on storing the Modelica model in a tree-structured XML document, after loading the XML data to generate the corresponding model for Dymola via an XML file parser. We focus on resolving the model-to-model transformation when the source data model, i.e., the SimModel, is different to the target data model developed in Modelica language and our work uses an XML document as a data container to support the data exchange and conversion between them.

The model transformation from SimModel to Modelica presents a unique challenge that must account for significant differences in their respective model structures. For example, the model component

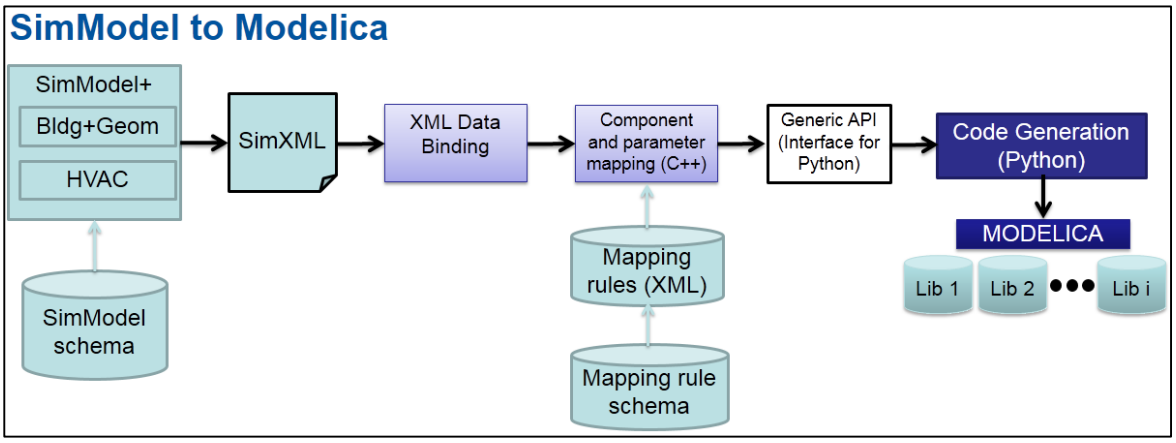

Figure 1 Transformation overview

and parameter definitions, the modeling hierarchy, and level of detail (LOD) of SimModel and Modelica differ substantially.

This paper focuses on detailed technical development of the model transformation system. A system level perspective for the overall conversion process which includes Modelica and use case examples is described in Remmen et al. (2015). The following sections present details of the research methods, system implementation and limitations discussion. The paper concludes with the current progress and future developments.

METHODOLOGY

This section describes the transformation system developed for linking BIM with different Modelica libraries. Figure 1 illustrates the overall system where SimModel acts as the BIM container to save BIM data, for example, HVAC, geometry, property data of HVAC systems and equipment, simulation configurations, etc. The SimModel schema is the XSD of the SimModel data model, defining five different sub-schema namespaces saving and organizing model data and their relationships, such as the building topology. Based on the original work of O'Donnell et al. (2011), we extend the SimModel schema structure for saving additional simulation data required by different Modelica libraries. While SimModel only entails the predefined LOD originating from EnergyPlus, Modelica allows the definition of deeper LODs. The increased LOD of Modelica requires the ability for the user to add additional data to SimModel, and the new, extended SimModel data model is named SimModel+. All the SimModel data are finally saved in an XML-based format file named SimXML.

After saving BIM data into SimXML, the resulting file is then loaded into system memory and parsed by our transformation system based on an API generated by a technique named XML data binding. During the SimXML file parsing, the transformation system builds a dynamic hierarchy structure representing the different data objects of SimModel, for example, different building elements, HVAC systems and their distribution inside the building envelope.

As the SimModel structure is different to commonly used Modelica libraries developed for BEPS (Cao et al., 2014 and Wimmer et al., 2014), we develop a mapping rule schema to represent the data model of mapping rules between SimModel and different Modelica libraries. Simulation engineers can efficiently re-use this mapping rule schema to define different sets of mapping rule instances for different target Modelica libraries. The system loads a set of user-defined mapping rules handling the data transformation from SimModel into a specific Modelica library.

All corresponding parts of our transformation system are implemented in the C++ programming language, in order to satisfy the requirements of model transformation speed and the access to low-level system features, such as virtual memory allocation. C++ offers advanced programming features, e.g., objects, inheritance, and polymorphism, while also providing the facilities for low-level memory manipulation. Thus, C++ is immensely popular, particularly for applications that require speed and/or access to some low-level system features.

Afterwards, the mapped or translated objects and properties for Modelica code generation can be retrieved and controlled by other script-based languages, like Python, via a generic API developed in our transformation system. Script programming languages, like Python, are heavily used for pre- and post-processing of Modelica. They are more flexible and easier to use for controlling the procedure of Modelica code generation based on techniques like pre-defined code templates and interpretation engines. This generic API, interfacing C++ and Python, separates the translation logic from code generation.

The Python-based code generator then performs the last set of data manipulations resulting in a Modelica file that references only one specific Modelica library and is ready for a subsequent simulation.

SimXML Data Binding and Syntax Validation

As described in the former sub-section, the data file of SimModel is an XML-based file named SimXML. It saves all the SimModel data as a structured XML document that is in accordance with the syntax defined

by the SimModel schema. Thus, this sub-section introduces the XML data binding technique for accessing SimXML and validating its syntax.

XML data binding is the process of extracting data from a structure representation of XML documents and presenting it as a hierarchy of objects that correspond to a document vocabulary. This allows us to manipulate SimXML data in a more natural and efficient way. We selected the open source, cross-platform CodeSythesis XSD (CodeSynthesis, 2014) as our system XML binding parser. It is an efficient framework whose parser can be customized for our own applications. The automated XML data binding of CodeSythesis XSD will generate a C++ API for accessing the data stored in SimXML after parsing the SimModel schema. For the SimModel schema, 2611 C++ classes representing different SimModel objects are generated for the data manipulation in a given SimXML file.

The XML syntax validation performs a number of checks on the XML document to prevent the construction of an inconsistent object model, such as an object model with missing required attributes or elements. Our SimXML validation relies on the underlying Xerces-C++ XML parser embed in CodeSythesis XSD. It checks the SimXML data against the given SimModel schema, and outputs the errors found into a log file. The syntax validation is enabled by default and is very useful during the development stage to detect problems with the data model at an early stage. For a user of the framework, however, this validation should not play a significant role anymore.

More detailed explanations of SimModel schema and the XML binding technique can be found in the work of Cao et al. (2014) and O'Donnell et al. (2011).

SimModel Hierarchy Parsing and Visualization

A model hierarchy is an arrangement of the model elements, e.g., objects, names, values, categories, etc., in which the elements are represented as being "above," "below," or "at the same level as" one another. Consequently, the SimModel consists of a hierarchical tree structure saving such relationships between different SimModel data elements. For example, a SimProject class object is normally the root node of this hierarchy tree representing a unique simulation project. This root node can store multiple links that refer to different building design alternatives. Each design alternative also refers to a set of building elements, zones, HVAC systems distributed inside the building, etc. Therefore, the SimModel hierarchy saves a set of different SimModel elements as well as the links between them in a tree-based structure.

In SimXML, each model element is given a unique long type id that distinguishes it from all other elements. Each parent element of the SimModel hierarchy links to a child element by saving its id. As a result, "parse SimModel hierarchy" is a recursive algorithm that detects each SimModel element, locates its position in the hierarchical tree and creates a tree node with a link to its data.

After that, we can also recursively iterate each node of this hierarchy, retrieve the link to the SimXML data element and print out the element data for visualizing the created hierarchy.

SimModel to Modelica Mapping Rule Schema

The mapping rule concept was first introduced in mathematics, representing a particular transformation. This transformation describes the conversion of a source model data into a target model data under the constraints specified by a given equation system. As SimModel is significantly different from the data model of a specific Modelica library (Cao et al., 2014), e.g., AixLib or BuildingSystems, we also need to define a set of mapping rules that can handle the difference between these two different data models.

Wimmer et al. (2015, 2014) proposed a set of different mapping rules that can convert the SimModel data into the Modelica model data defined by a specific BEPS library AixLib (EBC, 2014). Based on this work, we developed a mapping rule schema in XSD, containing the data model of the mapping rules between SimModel and different Modelica libraries. We can thus efficiently re-use this mapping rule schema, originally developed for our transformation system, to define different sets of mapping rule instances for different target Modelica libraries.

Figure 2 illustrates the main structure of the mapping rule schema. Mapping rules are classified according to three different levels in the schema:

1. The first level is library mapping, which is designed to link different mapping rule instances for different Modelica libraries.
2. The second level is component mapping, which is responsible for mapping SimModel components, e.g., a boiler of the HVAC system, into the corresponding component of the Modelica library specified by the first level mapping. At this level, we developed four different mapping rule schemas, i.e., One2One (one component in SimModel maps to another one in Modelica), One2Many, Many2One, and Gap (a component required by Modelica is missing from SimModel, i.e., does not exist yet in SimModel schema. If possible it will be added to SimModel by this rule).
3. The third level deals with the internal properties mapping of the components defined by the upper-level mapping rules. Here we implement the schema for five different property mapping rules: One2One (an One2Many property mapping is just a set of One2One property mappings), Many2One, Gap, Conversion (the property of a SimModel component needs to be converted into another data representation type or result of Modelica by a user-defined equation

system or function, e.g.; the energy efficiency curve from a continuous profile function into a matrix containing discrete signal values) and Combination (a property transformation combing multiple former property mapping rules defined in this level).

For more detailed explanations and examples of different SimModel mapping rules, please refer to the work of Wimmer et al. (2015, 2014).

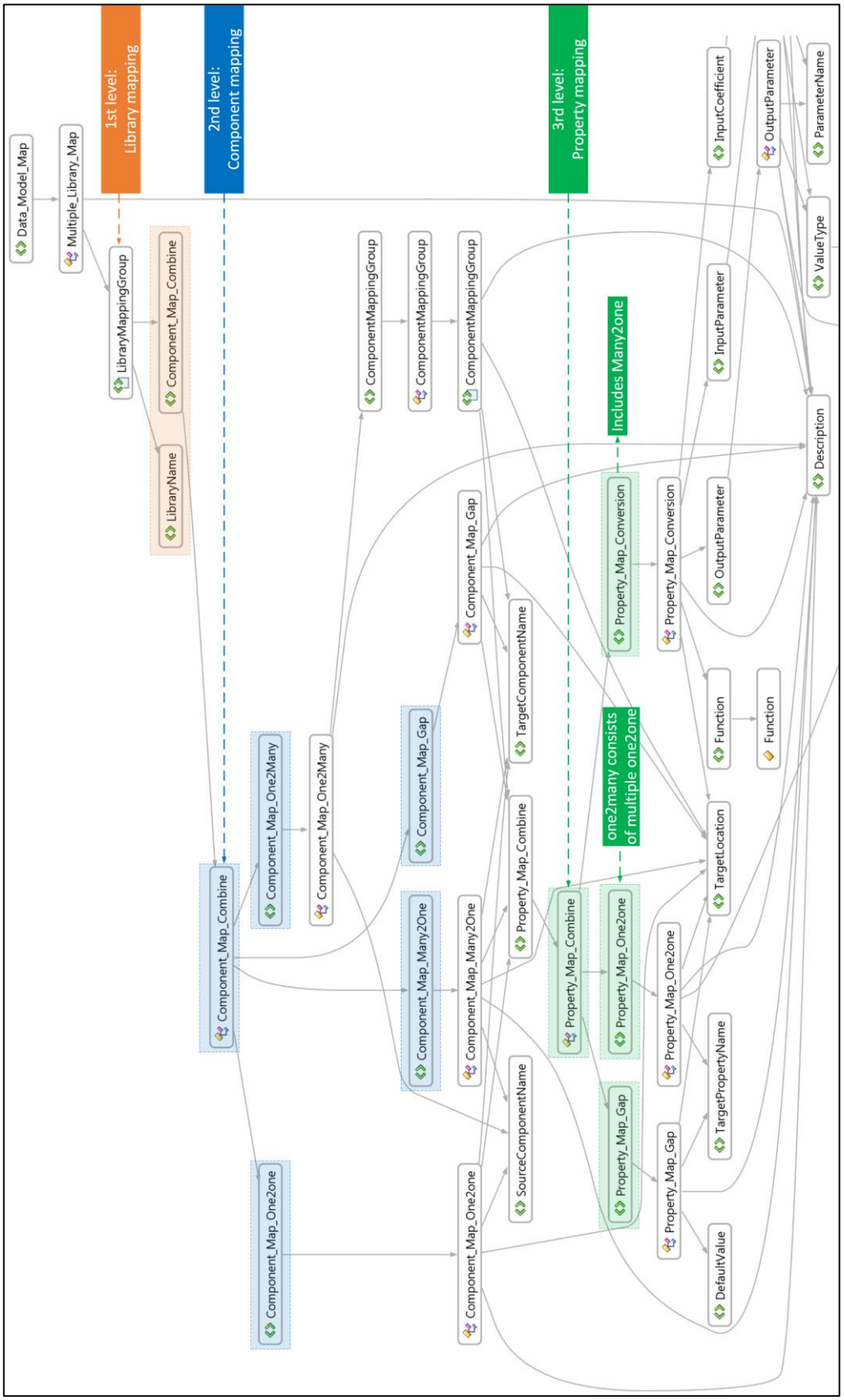

Figure 2 Mapping rule schema. We define different levels of mapping rule for different SimModel elements.

SimModel to Modelica Mapping Rule Filtering

Czarnecki and Helsen (2006, 2003) discussed fundamental work on data model transformation. In context of their work we can view transforming SimModel data model to Modelica code as a special case of model-to-model transformations. We only need to provide the meta-models for the source model and the target programming language as well as the transformation defined with respect to the meta-models. A transformation engine transfers the source model data into the target programming language model (see Figure 3).

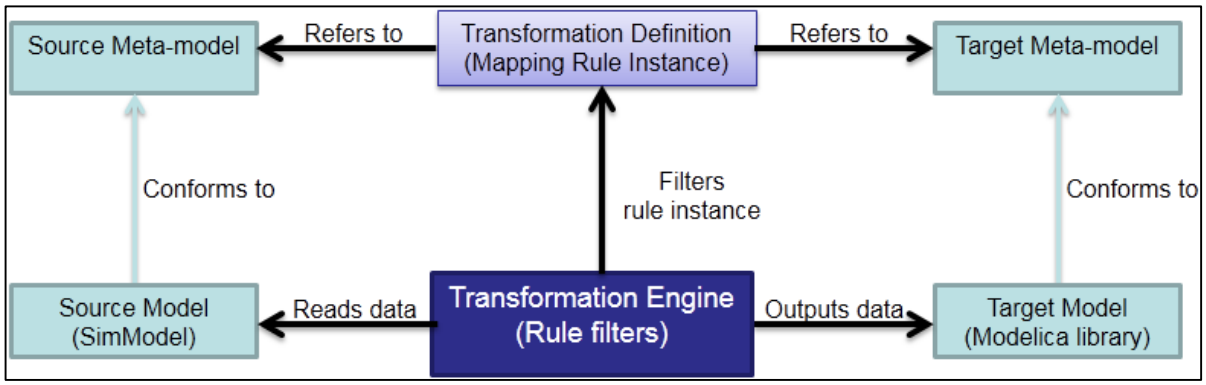

Figure 3 Basic concepts of model transformation

In our case, the source model is SimModel and the target language is Modelica. The transformation is an instance of the mapping rule schema and the transformation engine contains a set of filters translating the model data by filtering the mapping rule instance. The source model data is stored in SimXML file and the target programming language model is the Modelica code, based on a specific BEPS library.

A meta-model describes the type of model and typically defines the abstract syntax of a modelling notation (semantics information of the data model, such as the name list of SimModel APIs and the instance creator for each API at runtime of our system). In addition to this theory, Modelica is a generic programming language designed for different kinds of simulation tasks in engineering. The data models of different Modelica BEPS libraries differ from each other. Therefore, it is not feasible to create the meta-model for each different Modelica library. On the contrary, we analyze differences between their data models and SimModel by defining and implementing the same use cases, both in SimModel and each sub-library developed within the Annex 60 base library (Remmen et al., 2015). Afterwards, we study possible data mapping rules between SimModel and these BEPS libraries by analyzing the use case we implement on both sides. The output of this study describes the mapping rule schema we developed in the former sub-section.

For creating the meta-model of SimModel, we use a technique named reflection or self-reflection. In computer science, reflection is the ability of a computer program to examine and modify the structure and behavior (specifically the values, meta-data, properties and functions) of the program at runtime (Malenfant et al., 1996). This technique allows the handling of a SimModel object indirectly via a SimModel meta-object and represents a particular kind of meta-programming. This program concept was born from object-oriented programing (OOP), e.g., C++ programing. Therefore, as everything in OOP is an object, a SimModel data class can also be treated as an object. With this so-called meta-object, we can manipulate every feature related to that class, like its constructor, methods, attributes, and so on. In our system development, we implement our SimModel meta-model based on the API provided by the QMetaObject class of the Qt library. The Qt Meta-Object System is responsible for maintaining the runtime type information of different SimModel objects, such as HVAC components and properties. A single QMetaObject instance is created for each SimModel data class that is used in our application, which stores all the meta-information for the SimModel data class. The API of a QMetaObject is a very flexible solution for handling large third party libraries, like the SimModel schema.

Based on the transformation theory we introduced in the beginning of this sub-section, the next step consists of calling a transformation engine to transfer the source data model SimModel in SimXML into the target Modelica code. The transformation engine contains a set of different mapping rule filters that correspond to the different mapping rules developed in the mapping rule schema. The corresponding rule filter processes each mapping rule in three stages: 1) The first stage is concerned with parsing the mapping rule content from its XML-based rule data file via the API generated from mapping rule schema based on XML data binding. 2) The rule filter needs to create or call a SimModel meta-object instance to retrieve the SimXML data specified by the mapping rule. 3) The rule filter will execute the data operation, e.g., equations and functions defined by the mapping rule to generate the Modelica data converted from SimModel. The overall filtering process determines

the model transformation strategy. Users are only concerned with providing the mapping rules.

Generic API for Modelica Code Generation

This sub-section introduces a generic API developed for interfacing C++ and the script programming language Python, in order to better control the Modelica code generation.

We developed our first prototype for this generic API based on a technique named language binding. In computer science, a binding from a programming language to a library is an API providing glue code to use that library in a particular programming language (Binding, 2012). In the context of our generic API, bindings are wrapper libraries that bridge the C++ and Python programming languages in order to re-use the SimModel API generated for C++ in Python.

We use ctypes (ctypes, 2014) as one of the binding libraries to wrap the SimModel API for data access out of Python. ctypes provides C compatible data types, and allows calling functions in DLLs or shared libraries from Python. The other binding libraries, like Cython, are also currently under testing in the project development.

In the current prototype development, we exposed the mapped or translated SimModel components and their internal properties into a Python-based Modelica code generator via the ctypes wrapper. In order to provide Python with full controls on SimModel internal data, we also exposed more data objects from the SimModel hierarchy into Python via the generic API.

Figure 4 illustrates our current hierarchy of objects contained in SimModel. As illustrated in the parsed hierarchical structure, SimProject is the root element of the SimModel data model, which represents a BIM-based simulation project. SimProject in turn refers to multiple different child elements including SimSite, which defines a simulation site containing one or more building models, i.e. SimBuilding. A SimBuilding element then refers to the geometry data of the building envelop, such as the space boundary SimSpaceBoundary via the thermal zone element SimZone assigned. The parent HVAC system element is SimSystem, which in turn refers to different HVAC sub-systems.

For example, the experimental use case given by Remmen et al. (2015) defined a hot water loop system as: 1) a water supply side sub-system; 2) a water demand side sub-system and 3) a controls sub-system. These sub-systems contain sets of individual HVAC components such as fans, pumps, heating coils and cooling coils.

LIMITATIONS

The proposed transformation framework is an enhancement of former works, e.g., Yan et al. (2013), Jeong et al. (2014), Cao et al. (2014) and Wimmer et al. (2014). As previous work focused on linking one specific Modelica library to the BIM side, naming

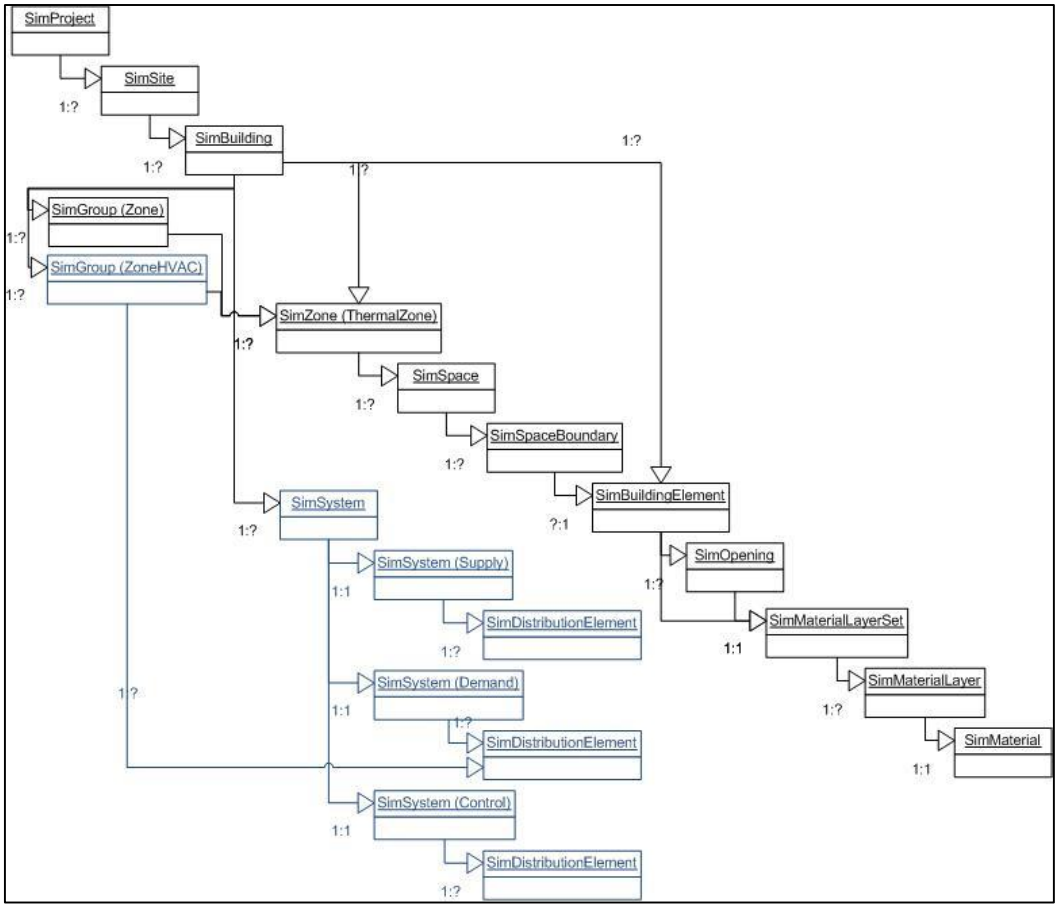

Figure 4 Generic API for Python. We show the hierarchy of SimModel objects exposed via the API in the development.

conventions and data representations differ and are not suitable for linking different libraries.

Yan et al. (2013) and Jeong et al. (2014) use proprietary APIs of Auto Revit to convert only the geometry data of the BIM model into a specific Modelica library. The solution results in a non-open-standard and is non-generic for converting different BIM data formats, e.g., IFC and SimModel. The conversion of the internal HVAC systems from BIM has not yet been resolved by their approaches.

Cao et al. (2014) and Wimmer et al. (2014) developed a set of mapping rules from open BIM to a specific library. Their works do not provide a generic translation framework to reuse the data translation logic (the rule schema) of these rules for different libraries. Thus, they might need a considerable effort to define new mapping rules for different libraries.

Both of these shortcomings were addressed in this work. The authors are aware that the presented framework also has some limitations. The conversion from BIM to a Modelica model is not fully automatic yet, e.g., if a component required by the Modelica side is missing in SimModel, i.e., it does not exist in SimModel schema yet, it has to be added into SimModel manually by a user-defined mapping rule. The mapping rules are an example of adaptable components. To adjust this part, the user needs deep knowledge of both the information model structure and the Modelica model.

As a short summary, this work makes the BIM transformation system able to handle different Modelica libraries developed for different simulation requirements.

CONCLUSIONS AND FUTURE WORK

This paper presents the technical development of a flexible model transformation system to link BIM with different Modelica libraries for supporting integrated architectural design and BEPS. We focus on the schema definition and interpretation of mapping rules within the transformation system:

1) We developed a fundamental data representation schema in XSD to define possible data mapping relationships between BIM and Modelica. This representation schema stores generic mapping rules for translating different levels of BIMs data to Modelica models, such as the library level, HVAC components level, and the internal property level of HVAC components.

2) Based on the representation schema developed in 1), simulation experts can easily define a specific data transformation process between BIMs and different Modelica libraries by using the mapping rules.

3) The third step consisted of interpreting the XML-based mapping rules defined in 1) and 2). This interpretation step converts actual HVAC component instances and their properties from the BIM world into the Modelica world. The result of this process is then used to generate Modelica models within the subsequent steps in the framework.

4) The generic API prototype, interfacing C++ with Python, is developed for the Modelica code generation. This API satisfies low-level application requirements like speed, while also providing the facilities for high-level control of Modelica code generation via script-based language. It separates translation logic from code presentation.

The proposed flexible model transformation is an enhancement to current work by demonstrating the potential to develop a generic solution for linking BIM with different Modelica libraries in order to accelerate the BEPS development in Modelica. While the building simulation domain has its specific challenges, it should be possible to adapt most of the developed concepts and tools to other simulation domains. In particular, the separation of generic code and specific data transformation rules make this process adaptable.

Our future work will concentrate on the following sub-tasks: 1) Enrich the SimModel data exposed by the generic API for the new application requriements. 2) Implement the baseline use cases in the other Modelica libraries we are using, like BuildingSystems (Nytsch-Geusen et al., 2013) and Buildings (Wetter et al., 2014). 3) Further develop more complicated use cases on different Modelica libraries. 4) Test the transformation system comprehensively by a set of use cases developed on different Modelica libraries.

ACKNOWLEDGEMENT

This work emerged from the Annex 60[1] project, an international project conducted under the umbrella of the International Energy Agency (IEA) within the Energy in Buildings and Communities (EBC) Programme. Annex 60 will develop and demonstrate new generation computational tools for building and community energy systems based on Modelica, Functional Mockup Interface and BIM standards.

The work is also conducted within the German research project EnEff-BIM, Energy Efficient Modeling and Simulation Based on Building Information Modeling, was funded by the BMWi[2] under Contract No. 03ET1177A.

Other parts (no duplications) of the preliminary research work in Ireland were supported by a Marie Curie FP7 Integration Grant within the 7th European Union Framework Programme.

REFERENCES

Aksamija, A. (2012). BIM-Based building performance analysis: evaluation and simulation of design decisions. In Fueling Our Future with Efficiency. Presented at the 2012 ACEEE

[1] IEA EBC Annex 60, http://www.iea-annex60.org

[2] The German Federal Ministry for Economic Affairs and Energy, http://www.bmwi.de

Summer Study on Energy Efficiency in Buildings, Pacific Grove, California.

Basarkar, M., O'Donnell, J., Maile T., Settlemyre K., Haves P., 2012. Mapping HVAC systems for simulation in EnergyPlus, in IBPSA Building Simulation - 2012. USA.

Bazjanac, V., Maile, T., Rose, C., et al., "An assessment of the use of building energy performance simulation in early design," in Proceedings of the Building Simulation, Sydney, Australia, 2011.

Bazjanac, V., "Implementation of semi-automated energy performance simulation: building geometry," in Proceedings of the 26th International Conference on Managing IT in Construction (CIB W), vol. 78, pp. 595–602, 2009.

Bazjanac, V., IFC BIM-Based Methodology for Semi-Automated Building Energy Performance Simulation, Lawrence Berkeley National Laboratory, Berkeley, Calif, USA, 2008.

Binding (2012). Language binding. Retrieved from http://en.wikipedia.org/wiki/Language_binding#cite_note-1

Cao, J., Maile, T., O'Donnell, J., Wimmer, R., and van Treeck, C., 2014. Model transformation from SimModel to Modelica for building energy performance simulation. In BauSIM 2014 Conference, Aachen, Germany.

CodeSynthesis, 2014. XSD: XML data binding for C++. Retrieved from http://www.codesynthesis.com/products/xsd/

ctypes, 2014. ctypes — A foreign function library for Python. Retrieved from https://docs.python.org/2/library/ctypes.html

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3), 621-645.

Czarnecki, K., & Helsen, S. (2003, October). Classification of model transformation approaches. In Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture (Vol. 45, No. 3, pp. 1-17).

EBC (Institute for Energy Efficient Buildings and Indoor Climate, Energy Research Center E.ON, RWTH Aachen University), AixLib (2014). [Modelica-based BEPS Library]. Retrieved from https://github.com/RWTH-EBC/AixLib/

Jeong, W., Kim, J., Clayton, M. J., Haberl, J. S., Yan, W. (2014), A Framework to Integrate Object-Oriented Physical Modelling with Building Information Modelling for Building Thermal Simulation, Journal of Building Performance Simulation.

Malenfant, J., Jacques, M., & Demers, F. N. (1996, April). A tutorial on behavioral reflection and its implementation. In Proceedings of the Reflection (Vol. 96, pp. 1-20).

Nytsch-Geusen, C.; Huber, J.; Ljubijankic, M. & Rädler, J. (2013), 'Modelica BuildingSystems − eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme', Bauphysik 35 (1), 21--29 .

O'Donnell, J., See, R., Rose, C., Maile, T., Bazjanac, V., Haves, P., 2011. SimModel: A domain data model for whole building energy simulation, in: IBPSA Building Simulation 2011. Sydney, Australia.

Remmen, P., Cao, J., Ebertshäuser, S., Frisch, J., Lauster, M., Maile, T., O'Donnell, J., Pinheiro, S., Rädler, J., Thorade, M., Wimmer, R., Müller, D., Nytsch-Geusen, C., van Treeck, C. (2015). An Open Framework for Integrated BIM-based Building Performance Simulation Using Modelica, in 14th IBPSA Building Simulation, Hyderabad, India.

Reisenbichler, U., Kapeller, H., Haumer, A., Kral, C., Pirker, F., & Pascoli, G. (2006, September). If we only had used XML. In In 5th Modelica conference, September.

Wetter, M., Zuo, W., Nouidui, T. S., & Pang, X. (2014). Modelica buildings library. Journal of Building Performance Simulation, 7(4), 253-270.

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

Wimmer, R., Cao, J., Remmen, P., Maile, T., O'Donnell, J., Frisch J., Streblow, R., Müller, D., and van Treeck, C. (2015). Implementation of Advanced BIM-Based Mapping Rules For Automated Conversions to Modelica, in 14th IBPSA Building Simulation, Hyderabad, India.

Yan, W., Clayton, M., Haberl, J., Jeong, W., Kim, J. B., Kota, S., Alcocer, J. L. B., Dixit, M., 2013. Interfacing BIM with building thermal and daylighting modeling, in IBPSA Building Simulation - 2013. France.