PROCEEDINGS OF ECOS 2023 - THE 36TH INTERNATIONAL CONFERENCE ON
EFFICIENCY, COST, OPTIMIZATION, SIMULATION AND ENVIRONMENTAL IMPACT OF ENERGY SYSTEMS
25-30 JUNE 2023, LAS PALMAS DE GRAN CANARIA, SPAIN

BIM2SIM for hydraulic-focussed Energy Simulations – Automatic Generation of pre parametrized Simulation Models

David Jansen[a], Dominik Hering[a] and Dirk Müller[a]

[a] RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Aachen, Germany, david.jansen@eonerc.rwth-aachen.de

Abstract:
The construction of energy-efficient buildings is one of the most important measures to reduce the impact of buildings on our environment. The dynamic simulation of the energy systems of these buildings can improve the design and performance and thus reduce the emissions that impact the environment. However, the process of creating models for dynamic simulations is time-consuming and error-prone. By using already existing digital models from the planning process the process of model generation can partially be automated. This paper presents the tool `bim2sim` and its underlying methodology for the application of automatic model generation for dynamic hydraulic energy system simulation with *Modelica* based in Building Information Modeling models with the open exchange format IFC. The tool is applied to the use case of an example energy system and a manual comparison modeling is performed for the energy system in *Modelica*. The comparison shows that automatic model generation saves a significant amount of time even for comparatively simple systems.

Keywords:
Building Information Modeling, Hydraulic energy simulation, Modelica, Model generation

1. Introduction

Operation of buildings is responsible for 34 % of energy use and 37 % of carbon emissions worldwide [1]. Improving the energy efficiency of building energy systems can help to reduce the environmental impact of buildings. To increase the efficiency of buildings, dynamic simulation models of building energy systems are becoming increasingly important for both design and operation of buildings. However, the generation of these simulation models is a time-consuming process that requires a high level of expertise. Especially because buildings and their energy systems are unique for every building [2]. In order to increase the dissemination of dynamic simulation models in practice, the effort required to create these models must be reduced. The partially automated creation of models on the basis of already existing digital planning data, such as Building Information Modeling (BIM), is a promising approach. In the field of dynamic energy system simulation, however, there are very few approaches that attempt to convert existing BIM models directly into simulation models. By using the open exchange format IFC and the programing language *Python*, this paper presents a tool and the underlying methodologies to achieve a semi automatic model generation of simulation tools based on BIM models.

2. Related Work

The idea of using data models from digital design and, in particular, BIM data is not new. Especially in the field of thermal building simulation, there are already various approaches to avoid redundancy in the double modeling of planning and simulation models [3–7]. In the area of hydraulic-focussed energy simulations or Heating, Ventilation and Air Conditioning (HVAC) simulation in general, the number of existing approaches is smaller. Bazjanac et al. dealt with the issue of transferring HVAC information from IFC in 2002 and 2004 [8] [9]. The presented *IFCtoIDF* tool used EnergyPlus and the IFC2x2 standard, but the tool is not published. Hauer et al. analyze the presentability of HVAC components in the IFC schema [10]. The most important findings are that the IFC schema even in version IFC 4 does not yet provide sufficient options to represent all relevant components of a HVAC system. Accordingly, linked data must be used for a complete description. Furthermore, they give recommendations for possible extensions of the IFC schema with focus on the energy generators. Andriamamonjy et al. present a methodology for deriving thermal building and plant simulations based on the IFC [11]. The IFC is imported to *Python* with *IfcOpenShell* and the output is a *Modelica* simulation model. The focus is on the analysis of different life cycles and planning stages of the building. The approach uses a direct mapping of the BIM data into the *Modelica* library IDEAS. The presented *IFC2Modelica* approach and its code is not published as an open source project. In his PhD thesis, Pauen invented the TUBES System Ontology (TSO) and the tool IFC2TSO to provide better understanding of the HVAC systems [12].

A necessary aspect to create executable simulation models is the creation of a control for the energy system. In order to represent this control in the simulation model, the control logic itself, as well as the linking of this logic with the signals of the associated sensors and actuators, are necessary.

Benndorf et al. present the implementation of a control in the IFC scheme using the example of a heating curve, a time-controlled volume flow controller and a temperature control. For this purpose, the IFC schema had to be supplemented by own new components. Furthermore, the concept of linked data sources was also used by using the *IfcOwl* schema to establish connectivity with the building automation control. However, it is noted that the use of BIM in the context of building control and automation is just being developed and existing BIM models and modeling tools cannot yet represent control, which is why they had to be subsequently added to the Revit export of the BIM model using *IfcOpenShell*. [13]

Sporr et al. use the IFC data format in their work to develop a methodology for mapping the control of an energy system for building heating. The methodology uses a combination of TRNSYS and Simulink and relies on additionally created data in addition to the IFC, since the IFC does not provide the necessary information on the producer side. [14]

Existing research already covers investigation and usage of BIM-data for simulation model generation in the HVAC domain, but non of the existing approaches was released in form of a public available tool. Furthermore, most approaches are based on the assumption of a perfect IFC model that contains no errors or lacks information. In reality, BIM models are currently often not yet perfect, for three main reasons. Authoring software has shortcomings regarding the export to IFC (1), the IFC format itself has shortcomings as not all relevant components and their semantic data is covered by the current IFC4 standard (2) and last but not least, model creators often do not add the relevant data to the BIM model, especially semantic data is often missing. Based on the existing shortcomings, a new approach is developed that provides an easy extendible and open source approach that works with non-perfect IFC data to create *Modelica* simulation models based on IFC4 data.

3. Methodology

Even though this article focuses on creating hydraulic energy simulations of HVAC systems, we want to give a brief overview on what `bim2sim` can do in general. In short, `bim2sim` is a tool that includes methods and concepts to read data from an IFC file, convert it to a simulation-oriented `meta-class` structure by collecting as much information as possible from the IFC, perform various types of processes and simplifications on the `meta-classes`, and export the results to a simulation model. This process can be used for HVAC simulations as presented here, but we also implemented methods and concepts to perform Building Performance Simulation (BPS) .The basic idea was already published [15] as well as the application on BPS [5, 16].

The workflow that is used specific for HVAC simulation is shown in figure 1. The Computer Aided Design (CAD) model that is created with a BIM authoring software is exported to an IFC model. This IFC model is loaded into `bim2sim` using *IfcOpenShell* and the relevant IFC elements are converted into a `meta-class` structure in *Python*. This `meta-class` structure is designed regarding the needs of the simulation domain. The `meta-class` instances are then transferred into a graph network that allows to run various simplification processes on the HVAC system to make it exportable to a simulation model. In the following sections the different steps are explained in more detail.

3.1. Importing data from IFC

Since two different object types are used in the conversion process, they are formatted differently for better distinction. *IfcElements* are displayed in italics, and *Python* `meta-class elements` are displayed in code style.

To get the relevant information from the IFC file to perform a semi-automatic simulation model generation, we use the *Python* implementation of *IfcOpenShell* to import the IFC data into *Python*. As Amor has shown, the number of entities, and thus the ability to represent systems in IFC, has increased significantly with the release of IFC 4 [17]. For this reason, `bim2sim` supports only IFC 4.

IfcOpenShell allows us not only to get the relevant data for every component but also to get metadata about the authoring software which is needed later in the process to enrich missing data.

3.2. Preliminary Model Check

In the next step we perform a model check against the loaded IFC. First basic validations are performed, e.g. to check that all *IfcElements* have a unique GUID. Subsequently, element specific checks are performed regarding the existence and correctness of attributes (e.g. the capacity of a boiler). In the last step HVAC specific checks are performed. E.g. all *IfcDistributionElements* are checked if they have ports assigned via the *IfcRelConnectsPortToElement* relationship

The results are displayed in form of an interactive HTML report. The goal is that the engineer who is in charge

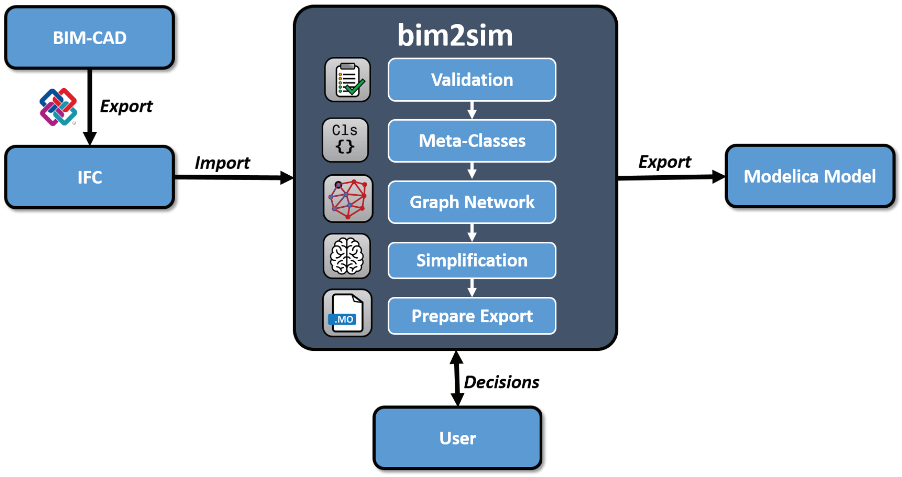


Figure 1: Process of `bim2sim` for HVAC domain.

for the simulations has a document which can be given to the BIM model creator. This document should provide the necessary information about all missing or wrong information inside the IFC model. The BIM model creator can update the IFC export afterwards based on the report.

3.3. Conversion to meta-structure

If the check succeeds the *IfcElements* are converted into the simulation orientated meta structure. This step makes sense, since BIM models for HVAC simulations are still partially insufficiently parameterized. The usage of the meta-structure allows obtaining the relevant information for the simulation from different sources in the IFC. For example, the surface roughness of a pipe can only be taken from the semantic information, but the length and diameter can also be determined via the calculation of the geometry if semantic information is missing. The conversion process consists of two parts. First, the physical instance of the *IfcElement* to be mapped must be identified and second, all relevant information about this class that can be obtained from the BIM data must be collected. The whole process of a `meta-class` element creation for the example of a boiler is shown in figure 2 and is explained in more detail below.

3.3.1. Class mapping

To keep the system modular and easy extendible, we use the class based structure in *Python* to represent each needed element with its own *Python* class. First, we have the base class *IFCBased*, which takes care of general processes such as calculating absolute position and orientation, which is required for all *IfcElements* regardless of their domain. Then we have the *HVACProduct* that inherits from *IfcBased* and adds additional functionality like the connections between ports and elements. This domain specific class is inherited to every element specific class, like the *Boiler* `meta-class`. In this element specific class we define the mapping rules, how to obtain the attributes and additional functions that are required for this process.

The IFC standard uses two information to define an instance of an *IfcElement*: the *IfcElement* itself and the *IfcTypeEnumeration* which allows further specification of the element. In current IFC data, especially in the HVAC domain, we often encounter missing correct declaration of the *IfcElement* and instead dummy classes like *IfcElementProxy* are used. But even if the *IfcElement* is correctly defined, the *IfcTypeEnumeration* is often set to USERDEFINED or completely missing. To overcome these problems and allow `bim2sim` to be used with non-perfect IFC-files, we implemented the possibility to add patterns in the form of regular expressions to look for in the semantic data of the element. This is useful, because even if the *IfcElement* is not correctly defined, the model creator might have entered the relevant information to identify the element as a string in the description of the element.

To make the structure extendible for new elements, these can easily defined as shown in the following example.

```
class Boiler(HVACProduct):
    ifc_types = {'IfcBoiler': ['*', '-EXCLUDING_TYPE']}
    pattern_ifc_type = [
        re.compile('Kessel', flags=re.IGNORECASE),
        re.compile('Boiler', flags=re.IGNORECASE),
    ]
```

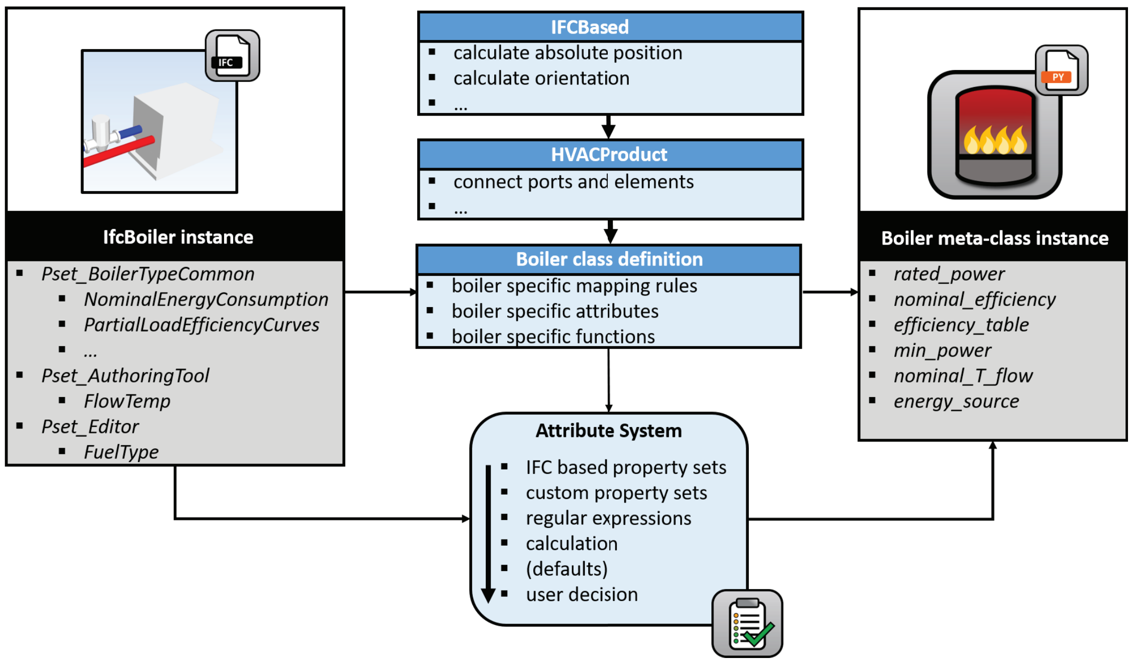


Figure 2: Process of meta class element creation for a boiler.

Based on this definition, the mapping process will first look for elements that are classified as *IfcBoiler* by taking all enumeration types (due to the '*) but the '-EXCLUDING_TYPE' into account. Additionally the process will look for every element that has the strings 'Boiler' or 'Kessel' (german for boiler) in their description. If multiple possible mapping classes are found, a user decision will be triggered.

Based on the defined classes all elements in the IFC will be converted into `meta-class` elements. An overview about all currently implemented classes for HVAC domain and their mapping is listed in Table 1.

Table 1: Mapping between IFC and Meta-structure. *:= all TypeEnumerations are included.

Groups	IFC Element	IfcTypeEnumeration	Meta-Classes
Connections	*IfcDistributionPort*	DUCT, PIPE	`HVACPort`
	IfcDistributionSystem	*	`Medium`
Energy Conversion	*IfcBoiler*	*	`Boiler`
	IfcElectricGenerator	CHP	`CHP`
	-	-	`HeatPump`
	IfcChiller	*	`Chiller`
Hydraulic Distribution	*IfcTank*	STORAGE	`Storage`
	IfcPump	*	`Pump`
	IfcValve	*	`Valve`
	IfcValve	MIXING	`ThreeWayValve`
	IfcPipeSegment	*	`Pipe`
	IfcPipeFitting	*	`PipeFitting`
	IfcPipeFitting	JUNCTION	`Junction`
	IfcDistributionChamberElement	*	`Distributor`
	IfcDistributionSystem	*	`Medium`
Heattransfer	-	-	-
	IfcSpaceHeater	*	`SpaceHeater`
	IfcHeatExchanger	*	`HeatExchanger`
	IfcCoolingTower	*	`CoolingTower`

The *IfcDistributionPort* and *IfcDistributionSystem* have a special role, as they are used to connect the different elements to each other and define the flow direction between the elements. The concept used in IFC to represent connections is shown in figure 3. The `meta-classes` use this concept as well by assigning the respective *HVACPorts* to every `meta-class` element and get additional information about the medium in a circuit from *IfcDistributionSystem*.

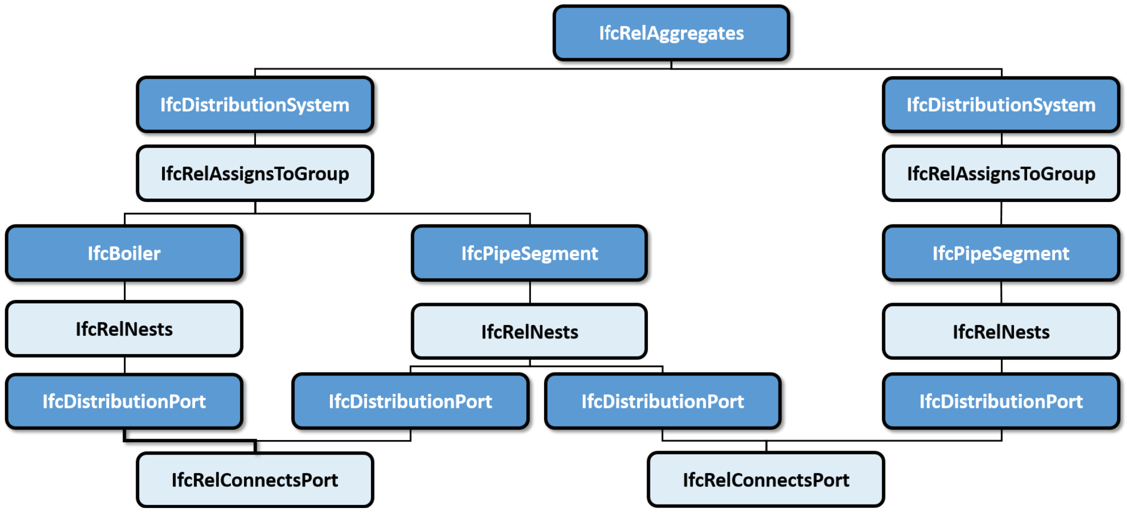


Figure 3: Concept of IFC to represent connections.

It's to mention that due to the fact that the IFC standard does not offer the possibility of directly mapping thermally active components, such as underfloor heating or concrete core activation, are taken care of via the concept of aggregations. Aggregations are explained in more detail in the section 3.4..

3.3.2. Attribute system

The created `meta-class` elements now needs to be filled with the relevant semantic information. On the left side in figure 2, we can see that the *IfcBoiler* provides semantic information included in three *IfcPropertySets* that represent the three different types of property sets that the `bim2sim` tool takes into account.

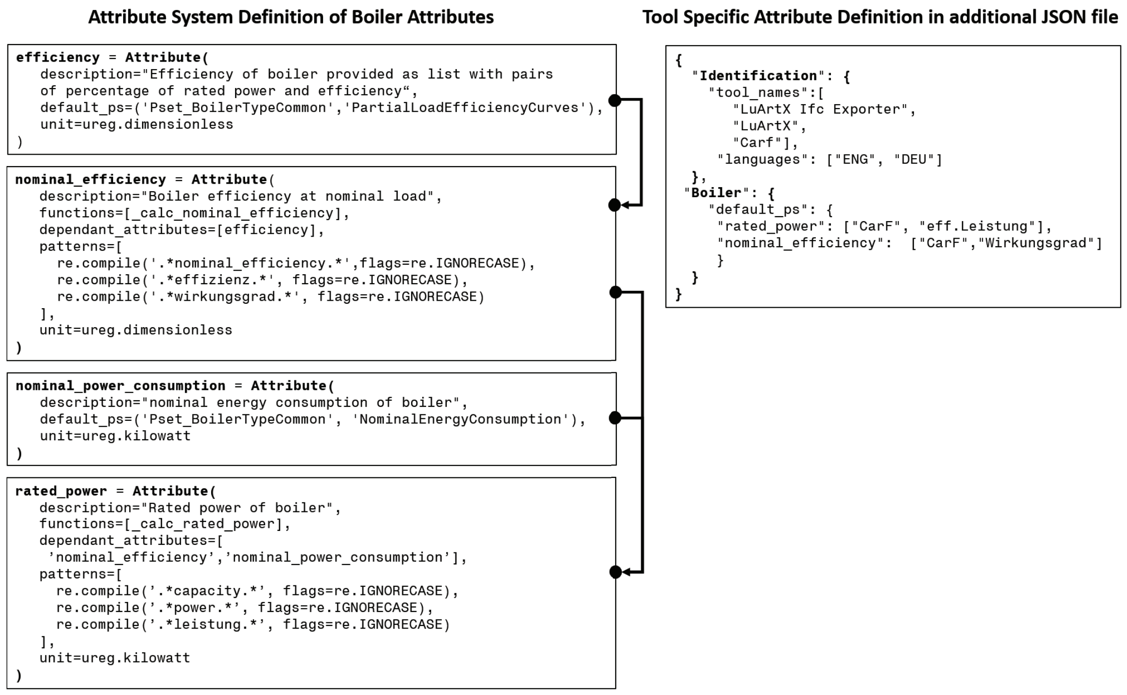
1. The *Pset_BoilerTypeCommon* as well as the belonging properties are IFC-schema compliant.
2. The *Pset_AuthoringTool* represents a property set that is not IFC schema compliant, but which is nevertheless typical for a specific authoring tool.
3. The *Pset_Editor*, which is a fully custom property set added by an editor. This one is not following any rules.

All three *IfcPropertySets* hold information that are relevant for the simulation model. The first one can be taken directly by implementing the IFC schema into the code. For the second one, additional information for every known export tool needs to be stored. For the last one, regular expressions can be used to find relevant information. Another challenge is that information might be stored implicit, but an additional calculation process is needed to obtain the final data that the simulation model requires.

To obtain all relevant information from the IFC-data we implemented the *Attribute*-system. The *Attribte*-system is a hierarchical approach that searches for a defined information by multiple approaches. The structure in figure 4 shows the usage of the *Attribute*-system to obtain the `nominal_efficiency` and the `rated_power` of the boiler as the IFC standard does not offer pre-defined options to input this data.

In this example, all implemented possibilities to retrieve information from the IFC are covered:

- `default_ps`: get information from property set that is defined in IFC standard
- `functions` and `dependant_attributes`: calculate/convert information into the direct form based on other attributes
- `patterns`: search for attributes based on regular expressions
- `JSON`: an additional file which is used to define typical places where specific authoring tools place information


Attribute System Definition of Boiler Attributes
Tool Specific Attribute Definition in additional JSON file

```
efficiency = Attribute(
   description="Efficiency of boiler provided as list with pairs
   of percentage of rated power and efficiency“,
   default_ps=('Pset_BoilerTypeCommon','PartialLoadEfficiencyCurves'),
   unit=ureg.dimensionless
)
```

```
nominal_efficiency = Attribute(
   description="Boiler efficiency at nominal load",
   functions=[_calc_nominal_efficiency],
   dependant_attributes=[efficiency],
   patterns=[
     re.compile('.*nominal_efficiency.*',flags=re.IGNORECASE),
     re.compile('.*effizienz.*', flags=re.IGNORECASE),
     re.compile('.*wirkungsgrad.*', flags=re.IGNORECASE)
   ],
   unit=ureg.dimensionless
)
```

```
nominal_power_consumption = Attribute(
   description="nominal energy consumption of boiler",
   default_ps=('Pset_BoilerTypeCommon', 'NominalEnergyConsumption'),
   unit=ureg.kilowatt
)
```

```
rated_power = Attribute(
   description="Rated power of boiler",
   functions=[_calc_rated_power],
   dependant_attributes=[
    'nominal_efficiency','nominal_power_consumption'],
   patterns=[
     re.compile('.*capacity.*', flags=re.IGNORECASE),
     re.compile('.*power.*', flags=re.IGNORECASE),
     re.compile('.*leistung.*', flags=re.IGNORECASE)
   ],
   unit=ureg.kilowatt
)
```

```
{
  "Identification": {
    "tool_names":[
       "LuArtX Ifc Exporter",
       "LuArtX",
       "Carf"],
     "languages": ["ENG", "DEU"]
  },
 "Boiler": {
    "default_ps": {
     "rated_power": ["CarF", "eff.Leistung"],
     "nominal_efficiency":  ["CarF","Wirkungsgrad"]
     }
  }
}
```

Figure 4: Excerpt of the attribute definition for the boiler `meta-class`.

The IFC-standard only allows defining the efficiency curve as a list of pairs of part load power and part load efficiency, but not `nominal_efficiency` itself. Still, the curve information can be used to calculate the `nominal_efficiency`. The `rated_power` can't be defined in the IFC as well. But based on the calculated `nominal_efficiency` and the `nominal_power_consumption` which can be defined in the IFC, the `rated_power` can be calculated as well.

To meet the goal of being able to generate simulation models even without perfect IFC models, all attributes can also be found in the IFC through the regular expressions as shown. Furthermore, the attribute system searches for dependent properties in the IFC based on the authoring tool defined in `JSON` on the right of the figure. For this purpose, the authoring tool is stored when the IFC file is loaded. As a last option, a user decision is created to retrieve the needed values. However, this is only executed against when the corresponding attribute is needed, either for a calculation of another attribute, or for the export to the simulation model to maximize the level of automation. This is described in more detail in the section 3.5. In addition, the attribute system ensures that all parameters are correctly converted to the specified units, based on the *IfcUnits* specified in the IFC. By using the *Python* package pint [18], it is ensured that later conversions are correct. All these functionalities aren't complex but adding the functionality for all relevant elements in the HVAC domain and in the other domains that `bim2sim` supports without defining a clean, unify and easy to extendible structure would result in non-maintainable code.

3.4. Processing and Simplification

3.4.1. Graph Network Generation

To analyze and simplify the hydraulic circuits the created `meta-classes` and the related HVAC-Ports are converted into a network graph. Network graphs offer the potential to use existing graph algorithms to perform efficient analysis against the hydraulic circuit. `bim2sim` uses two graphs, the `PortGraph` and the `ElementsGraph`. The `PortGraph` uses every `HVACPort` as a node and the edges between the nodes mark the fluid flow between the ports. In the `ElementsGraph` no ports are used, but the elements like a Boiler are the nodes. The `ElementsGraph` graph is an abstraction of the `PortGraph` and is mostly use for visualization. The `PortGraph` is used for most analysis, because it offers information how ports of the same element are connected to each other. This is important as for example a heatpump has four ports, but only two of them are connected with each other in pairs (evaporator and condenser side). `bim2sim` uses the *Python* package *NetworkX* [19] to create network graphs and analyze them.

3.4.2. Aggregations

The concept of `Aggregations` is used in the HVAC part of `bim2sim` to simplify the hydraulic circuit and reduce the numbers of elements to export to the ones relevant for the simulation. This is needed as an export of every `meta-class` instance created based on the IFC directly to a *Modelica* instance would result in an infeasible or at least very slow system of equations in *Modelica*. The simplest example of an `Aggregation` is the `PipeStrand`, which aggregates chains of contiguous connected meta-class elements without junctions to a single `PipeStrand` with an equal total length $l_{total} = \sum_{i=1}^{n} l_i$ and equal diameter $d_m = \sum_{i=1}^{n} l_i \cdot d_i / l_{ges}$. The equivalent parameters are calculated to obtain a simulation model with an equal pressure drop and heat loss. A special case of a `PipeStrand` are coils used for thermal activated building structures, like underfloor heating or concrete core activation. These are specified through the density of pipes (length and number of elements) in a certain area, the distance between the center lines of the pipes.

Using the network graph allows using existing analyze algorithms and classifications. For example, the chain of connected elements can be determined using the degree of the nodes of the graph. The degree of a node is defined as $d_G(v)$ and is calculated based on the number of connections a node has. To find the chain of connected instances without junctions only nodes with a degree of $d_G(v) = [1; 2]$ are taken into account which are of the type `Pipe` or `PipeFitting` or `Valve`.

`bim2sim` also includes more advanced aggregations, two of them will be explained in more detail below, the aggregation of parallel pumps and generator cycles. In practice, parallel circuits consisting of several identical pumps are often used in large hydraulic networks for better scalability. In this way, only the required number of pumps can be switched on, depending on the load. For simulation, these parallel circuits can be converted into a single component to reduce the number of equations of the system of equations to be solved and thus the complexity and simulation time of the corresponding model. In 5, the graph network of an example system is shown, which consists of four pumps in the initial state (a), where one of the pumps has a lower power (purple) and three pumps have an identical power (red). There is also a bypass connected in parallel with the four pumps (green).

The algorithm developed allows both a grouping that groups only parallel pumps of the same power and the option to group all parallel pumps. In addition, the an `AggregatedPipeFitting` is also needed, since other connections at the nodes, such as the bypass shown here, should be kept. The result for the case where only pumps of the same power are aggregated is shown in (b), and in addition to the successful aggregation, it also shows that the bypass (green) is still present and the pump with a different power (purple) was not included in the aggregation. The rest of the graph network remains untouched, since only the aggregation for `ParallelPumps` was performed.

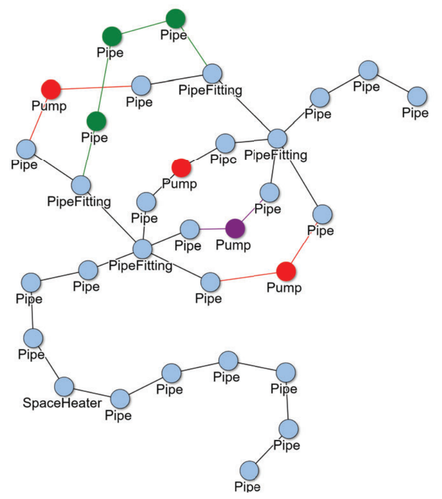
Besides just reducing the elements to be represented later in the simulation model, all relevant information is converted into semantic data. In the case of the aggregation `ParallelPump`, these are in particular the total power of the pumps, the combined nominal volume flow as well as the total length and the average diameter of the adjacent pipelines, which are relevant for the pressure loss.

The aggregation `Generators` is relevant because generator circuits consist of many individual components, most of which have no meaning for the simulation or at least do not need to be represented as individual components.

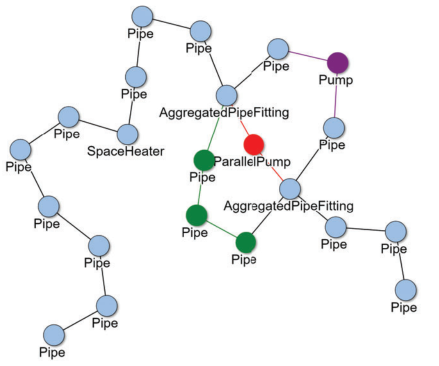
Figure 6 shows the original state of an example in (a). The example consists of a generator circuit with boiler (red), pump (blue), a bypass with valve (green), an expansion tank (purple), and some other pipe elements. The generator circuit is connected to a distributor (gray), to which four other pipe strands are connected, which represent a simplified consumer.

In (b) the result of the aggregation is shown. The simplified consumer circuit with the four pipe elements remains unchanged. However, the generator circuit can be aggregated into a single component, the `GeneratorOneFluid`. This component contains the information about the type and power of the generator, whether there is a separate pump in the circuit, what the power of this pump is, and the information about an existing bypass. Since the expansion tank is not relevant to the simulation, this information is not tracked further. The algorithm can be applied to parallel generators beyond the example shown. In this case, the parallel generators are converted into one generator that provides the total power.

There are limitations in terms of generator types. So far, only algorithms for generators with one external fluid circuit, such as boilers and Combined Heat and Power (CHP), have been included. Generator aggregation for generators with multiple circuits, such as a heat pump, has not yet been created, since mapping a heat pump in the IFC schema is currently only possible with workarounds.



(a) Before Aggregation



(b) After Aggregation

Figure 5: Graph of `meta-class` elements for `ParallelPump` Aggregation

3.5. Exporting to *Modelica*

3.5.1. Export Libraries

The simplified hydraulic network of the `meta-class` elements must now be translated into *Modelica* models in the next step. As introduced before, we implemented a *Plugin* system to allow multiple tools and libraries to take advantage and reuse the concepts that we built with `bim2sim`. Each *Plugin* that uses *Modelica* can build its own export for the used library by creating *Python* classes that inherit from the base instance. In the current version, `bim2sim` holds two *Plugins* for *Modelica* HVAC export: *AixLib* and *HKESim*. *AixLib* is an open source *Modelica* library that holds simulation models for HVAC simulation as well as BPS and is based on the IBPSA core library [20]. *HKESim* is a non-public library used by the ROM Technik company, who significantly contributed to the creation of `bim2sim`. Its focus is on HVAC simulation. Additionally, some basic components are implemented for the *Modelica* Standard Library (MSL).

Every `bim2sim` *Modelica* export class holds information about the path to the *Modelica* model it is exported to, the `bim2sim` instance it represents and the parameters that should be requested before exporting the model. The most basic definition of an export model is as follows:

```
class StaticPipe(StandardLibrary):
    path = "Modelica.Fluid.Pipes.StaticPipe"
    represents = [hvac.Pipe, hvac.PipeFitting, aggregation.PipeStrand]

    def request_params(self):
        self.request_param("length", self.check_length)
        self.request_param("diameter", self.check_diameter)
```

The `self.check_length` and `self.check_diamter` are optional functions to define, to implement plausibility checks against the parameters. Additional functions might but must not be implemented for a basic export. Table 2 shows the current status of which models are used regarding the libraries.

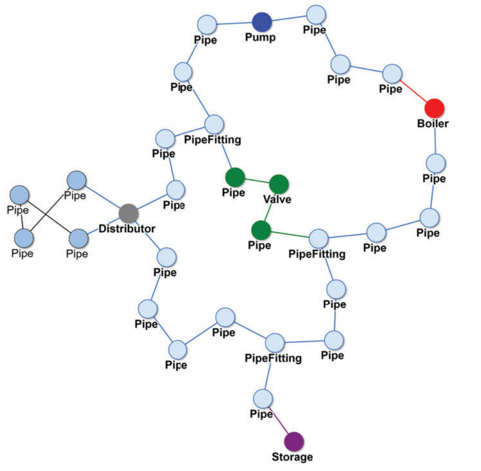
Not all `meta-classes` have a *Modelica* export yet in all libraries, but more export models are currently under development. Furthermore, based on the modular structure, other libraries can be easily added.

3.5.2. Base export instance

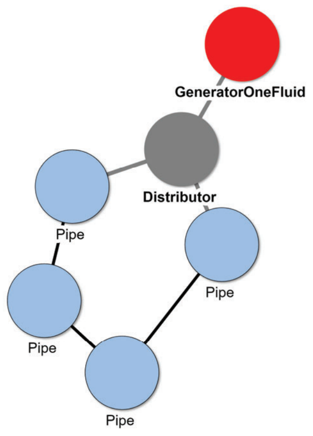
To continue the modularity of the built workflow, the export is based on a class structure that can be easily extended. To ensure modularity, we implemented a base class for a *Modelica* export instance that contains the all relevant functionality and can be extended by any *Modelica* instance, regardless of which library it is used in.

Translate *Python* into *Modelica* parameters

Python and *Modelica* both hold the logic for different types of parameters, like boolean, arrays, lists, integers



(a) Before Aggregation



(b) After Aggregation

Figure 6: Graph of `meta-class` elements for `Generator` Aggregation

and floats with unit. To keep extensibility simple, the conversion from *Python* to *Modelica* is defined in the base class. The conversion thus subsequently runs in the background and no longer needs to be taken into account when extending a library with new models.

Unit conversion As mentioned, we use the package *Pint* inside *Python* to obtain and maintain the correct units from the IFC file. To ensure that no conversion errors occur, the output unit for the *Modelica* model can be defined. By default, the values obtained from IFC are always converted to SI units during export.

Numerical validation checks These checks allow the definition of value ranges within which individual values may lie. This way, it should be prevented that unphysical values get into the model.

Keep track of corresponding IFC element(s) For traceability between IFC model and *Modelica* simulation model, each *Modelica* component is assigned the GUID of the corresponding IFC object as a parameter.

Parameter request system As written before, not all parameters relevant for the simulation can be extracted from the BIM model for various reasons. Nevertheless, it should be ensured at the time of export that all parameters relevant for the simulation model are available or at least have been requested. At the same time, the number of user inputs should be minimized. To achieve this, only those decisions are queried during the process whose result is needed immediately. An example of this would be the decision of what type an IFC element has if the IFC class is not uniquely defined (e.g. when using *IfcBuildingElementProxy*). Such a decision must be made directly, because it has an impact on the further process. Information that is only required for the final simulation model export will only be executed during export.

This avoids in many cases that parameters, which are no longer relevant for the exported model, are queried. For example, the parameter for the volume of a storage, which is identified during the process as a pressure equalizing vessel that is not relevant for the simulation model. If a user decision is skipped, or a parameter fails the final numerical validity check, that parameter is noted as unknown in the exported model. Thus, it can be directly recognized in the model which parameters have to be reworked.

Translate port logic The AixLib and HKESim as well as the MSL use the *FluidPorts* of the MSL to connect instances with each other. The definition inside *Modelica* is that *Port_a* is the incoming port of a Model and *Port_b* is the outgoing port. The `HVACPort` of the `meta-class` system and the simplified graph network hold the needed information to connect the respective *Modelica* instances with each other during export.

3.5.3. Usage of modules

Table 2 already included the `GeneratorOneFluidModule` and `ConsumerHeatingDistributorModule`. Equivalent to the `aggregations` used in the simplification of the graph network, new module-based models for the AixLib and HKESim libraries are currently implemented for export. These modules are pre-configured combinations of already existing components of the library, which reduce the needed number of parameters to minimum. These modules also contain basic control strategies, which allows the export of almost ready to run simulation models. The basic control strategies however can easily be overwritten by user defined control strategies which can be connected to the BUS connectors of the modules. For AixLib we already implemented a boiler module

Table 2: Current state of `bim2sim` *Modelica* export to different libraries.

Meta-class	AixLib	HKESim	MSL
`Boiler`	X	X	
`GeneratorOneFluidModule`	X		X
`HeatPump`	X		
`Chiller`	X		
`CHP`	X		
`Storage`	X		
`Pump`	X	X	
`Valve`			X
`ThreeWayValve`	X	X	
`Pipe`			X
`PipeFitting`			X
`Junction`	X	X	X
`Distributor`	X	X	
`Radiator`	X	X	
`Consumer`	X		
`ConsumerHeatingDistributorModule`	X	X	

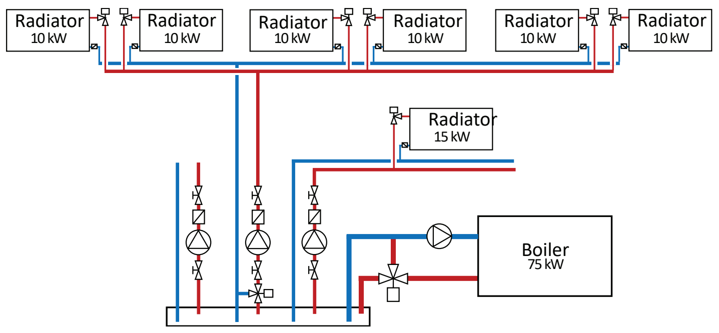
and a consumer module which can be found on GitHub[1]. HKESim won't be discussed because it's an in-house library and not public available.

3.5.4. Export process

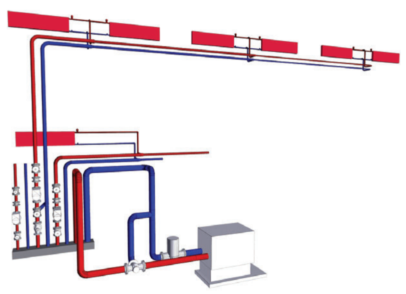
The conversion into *Modelica* models is performed based on a dynamic *Mako* template [21]. This template is filled based on the instances and their parameters and connections that are gathered through the previously discussed concepts. During the export, required, but missing parameter trigger decisions to the user. If these decisions are skipped, the parameters will be left empty in the exported simulation model, but a red annotation will be placed on the top level of the model, that gives feedback which parameters are missing. Additionally, the export tries to arrange the *Modelica* instances in a useful pattern, based on the position information from the IFC.

4. Proof of Concept

To prove the functionality of the developed tool, we have created an example use case in cooperation with the software manufacturer LuArtX. This is shown in figure 7 as schematic drawing (a) and screenshot of the IFC model (b).



(a) Schematic drawing



(b) IFC model

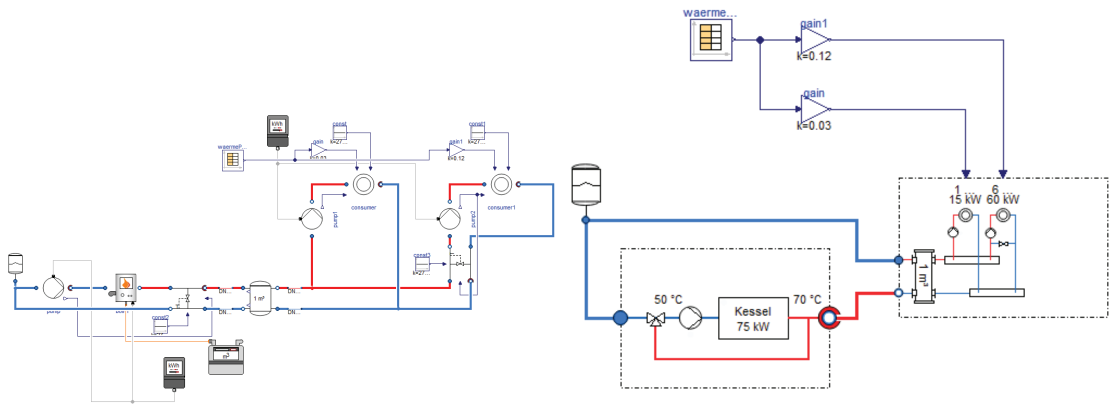
Figure 7: Use case example

The use case consists on the generation side of a boiler for heat generation, a pump, a three-way valve and a bypass for return flow boosting. The consumer side has a distributor, two consumer strands with several

[1]`https://github.com/RWTH-EBC/AixLib/tree/issue1147_ConsumerAndBoiler`

radiators as consumers, of which six radiators are connected in parallel. Each consumer strand has a pump and a valve. One of the consumer strands is an open end and one has a bypass for flow temperature control. The software manufacturer LuArtX improved their CAD software CARF throughout the creation to implement correct export for *IfcPorts*.

Since the creation of *Modelica* models based on plans or 3D models is a common task at the industry partner, both a manual model creation and a partially automated creation by `bim2sim` were performed to demonstrate the potential of the developed tool. The two model exports, both using the in-house library of the industry partner, are shown in figure 8



(a) Manual created model (b) Automatic created model with `bim2sim`

Figure 8: Exported *Modelica* HKESim models

It took a *Modelica* expert from the industry partner 2 hours and 33 minutes (153 minutes) to manually create the model. This includes the understanding of the energy system based on the BIM model, the parameter gathering and entering, modeling in *Modelica* including connections and sensors, creating usage profiles and removing any errors. Creating the simulation model with `bim2sim` took 6 minutes, including adding the missing parameters and demand profiles. This results in a time saving of 95 %.

5. Conclusion

In this paper, the authors presented a brief review of existing approaches in the field of simulation model generation for the HVAC domain based on digital planing data in the form of BIM models. Based on the discovered gap of public available tools which are able to create *Modelica* simulation models based on non-perfect BIM data in form of IFC, the `bim2sim` tool, and it's application for the HVAC domain were introduced. The core of `bim2sim` is the *Python* library with the same. This includes methods easy extendible methods based on orientated programming to convert IFC data into *Modelica* simulation models. The underlying methodology was explained, and the tool was applied to an example use case to prove the concept. The example showed that even for comparatively simple systems, a significant amount of time can be saved.

Currently, the embedded methods are focussed on hydraulic systems. In the future, the authors plan to extend the functionality to duct systems to simulate ventilation as well. Also, the presented use case covered only a small example system. The next step will be bigger systems with multiple generation devices for heating and cooling.

This paper only gives an overview about the functionality and methodology of `bim2sim`. For more information, the authors refer to the GitHub[2] repository, where documentation and the presented example use case can be found.

6. Acknowledgements

The authors gratefully acknowledge the financial support of the German Federal Ministry for Economic Affairs and Energy in the project “BIM2SIM” (project number 03ET1562A) and "BIM2Praxis (project number 3EN1050A). The authors also want to thank the RD department of Rud. Otto Meyer Technik GmbH & Co. KG and especially Christian Warnecke for the extensive contributions to the code.

[2]`https://github.com/BIM2SIM/bim2sim`

References

[1] United Nations Environment Programme. 2022 Global Status Report for Buildings and Construction: Towards a Zero-emission, Efficient and Resilient Buildings and Construction Sector; 2022. Available from: https://wedocs.unep.org/20.500.11822/41133.

[2] Drgoňa J, Arroyo J, Cupeiro Figueroa I, Blum D, Arendt K, Kim D, et al. All you need to know about model predictive control for buildings. Annual Reviews in Control;50:190-232.

[3] Hirth S, Nicolai A. The novel dynamic building energy performance simulation tool SIM-VICUS. vol. 17 of Building Simulation. IBPSA;. p. 0-0. Available from: https://publications.ibpsa.org/conference/paper/?id=bs2021_11116.

[4] Kamel E, Memari AM. Automated Building Energy Modeling and Assessment Tool (ABEMAT). Energy;147:15-24.

[5] Jansen D, Nürenberg M, Müller D. BIM-Basierter Reduced Order Ansatz für Thermische Gebäudesimulationen. In: BauSIM 2020 Proceedings. Verlag der Technischen Universität Graz;. Medium: online Meeting Name: 8. Conference of IBPSA Germany and Austria.

[6] Nytsch-Geusen C, Rädler J, Thorade M, Ribas Tugores C. BIM2Modelica - An open source toolchain for generating and simulating thermal multi-zone building models by using structured data from BIM models;. p. 33-8. 00000. Available from: http://www.ep.liu.se/ecp/article.asp?issue=157%26article=3.

[7] Mediavilla A, Elguezabal P, Lasarte N. Graph-Based methodology for Multi-Scale generation of energy analysis models from IFC. Energy and Buildings;282:112795. Available from: https://www.sciencedirect.com/science/article/pii/S0378778823000257.

[8] Bazjanac, Vladimir, Forester J, Haves, Philip, Sucic, Darko, Xu, Peng. HVAC Component Data Modeling Using Industry Foundation Classes | Building Technology and Urban Systems Division. In: 5th International Conference on System Simulation in Buildings 2002; 2002. .

[9] Bazjanac V, Maile T. IFC HVAC interface to EnergyPlus-A case of expanded interoperability for energy simulation. In: Simbuild;. 00000.

[10] Hauer S, Brès A, Partl R, Monsberger M. An Approach for the Extension of OpenBIM MEP Models with Metadata Focusing on Different Use Cases. In: Proceedings of the 16th IBPSA Conference;. Pages: 189.

[11] Andriamamonjy A, Saelens D, Klein R. An automated IFC-based workflow for building energy performance simulation with Modelica. Automation in Construction. 2018;91:166-81.

[12] Pauen N. Graphbasierte Algorithmen und gesamtheitliche Repräsentation von Systemen der TGA mit BIM und Linked Data [phdthesis];. Available from: https://publications.rwth-aachen.de/record/851025.

[13] Benndorf G, Réhault N, Clairembault M, Rist T, editors. Describing HVAC controls in IFC – Method and application. vol. 122; 2017.

[14] Sporr A, Zucker G, Hofmann R. Automatically Creating HVAC Control Strategies Based on Building Information Modeling (BIM): Heat Provisioning and Distribution. energies. 2020;13:4403.

[15] Jansen D, Fichter E, Richter VE, Barz A, Brunkhorst J, Dahncke M, et al. BIM2SIM - Development of semi-automated methods for the generation of simulation models using Building Information Modeling. In: Proceedings of Building Simulation 2021: 17th Conference of IBPSA;. .

[16] Jansen D, Richter V, Lopez DC, Mehrfeld P, Frisch J, Müller D, et al. Examination of Reduced Order Building Models with Different Zoning Strategies to Simulate Larger Non-Residential Buildings Based on BIM as Single Source of Truth. In: Modelica Conferences;. p. 665-72.

[17] Amor R. Analysis of the Evolving IFC Schema;. .

[18] Pint: makes units easy;. Available from: https://pint.readthedocs.io/en/stable/.

[19] Hagberg A, Schult D, Swart P. Exploring network structure, dynamics, and function using NetworkX. In: Varoquaux G, Vaught T, Millman J, editors. Proceedings of the 7th python in science conference;. p. 11 15.

[20] Maier L, Jansen D, Wüllhorst F, Kremer M, Kümpel A, Blacha T, et al. AixLib: An open-source Modelica library for compound building energy systems from component to district level with automated quality management. submitted to: Journal of Building Performance Simulation.

[21] Bayer M. Mako Templates for Python;. Available from: https://www.makotemplates.org/.