2023 European Conference on Computing in Construction
40th International CIB W78 Conference
Heraklion, Crete, Greece
July 10-12, 2023

REPRESENTING MODELICA MODELS AS KNOWLEDGE GRAPHS USING THE MOONT ONTOLOGY

Elisabeth Eckstädt[1], Karsten Menzel[2], Hervé Pruvost[1] and Dirk Mayer[1]
[1]Fraunhofer IIS/EAS, Dresden, Germany
[2]Institute of Construction Informatics, TU Dresden, Germany

Abstract

Model based design of building energy systems is increasingly used to enable efficient design flows. Different types of models have still to be converted manually, which is time consuming and error-prone. This turns out to be an obstacle for the integration of virtual validation by system simulations. As a contribution for automated conversion and translation of models, an ontology (MoOnt) has been developed that is utilized to represent Modelica simulation models as knowledge graphs (KG). An automated transcriptor that extracts KG from Modelica models has been implemented. A KG carrying the transcribed Modelica library and model representations was successfully queried for test cases. This is the basis towards further steps for automated translation of different types of models in the design process of buildings and a contribution to the model based design of building energy systems.

Introduction/Motivation

The demand for energy efficient buildings leads to increasingly complex system solutions. Renewable energy sources have to be integrated, energy storage has to be dimensioned and consumption for HVAC (heating, ventilation, air conditioning) has to be controlled by predictive algorithms; further, the building owner need economical investments and a high reliability of the systems.

As already established in other branches like microelectronics, aerospace or automotive, model based engineering is more and more used also for building energy systems.

An efficient model based design process integrates tools and methods from the early concept stage to the mechanical or structural design of components. Key to an early validation are building performance simulations including HVAC plant simulations: various concepts can be evaluated, components can be dimensioned and control algorithms developed; i.e. essential development steps can be carried out in simulation tools

Thus building simulation models contain a lot of information on the building under investigation that is beneficial for all stakeholders involved in the building design process, but also useful later in the lifecycle during optimization of operation schedules or refurbishments. This accounts for building performance simulation as well as for HVAC system simulation. To allow for a comprehensive and accurate description of the building behaviour, it becomes necessary to link different sources of information about the building, which is the basic idea of Linked Building Data and also the spirit of Building Information Management (BIM) and Digital Twins (DT). Especially, the link of simulation models with other types of models can accelerate the design flow.

This paper describes how Modelica (Modelica Association, 1997) models can become an integrated part of the DT of a building. Since Digital Twin is not a well defined term, here we understand DT to mean a knowledge graph representing the building, built upon W3C standards RDF, RDFS and OWL and call it a "Semantic Twin".

The representation of Modelica models by knowledge graphs enables a translation of simulation models into other types of models and vice versa. The goal is to have a bidirectional link of Modelica and the OpenBIM format IFC as a contribution to accelerate and automate the design of complex building energy systems. The context of this reseach has been described in (Eckstädt, 2021).

Paper Overview

The main part of the paper is structured according to the 5 steps of the applied knowledge engineering methodology described by (Pinto & Maertin, 2004):

We start by defining the goals of the knowledge engineering process (step 1). In that context an introduction to some basic principles of Modelica is given and the architecture of the involved knowledge graphs is shown. Conceptualisation, formalisation and implementation (steps 2-4) of the developed MoOnt ontology are described in the next sections.

The next steps in knowledge engineering are "Maintainance" and "Evaluation" (step 5). To evaluate the MoOnt ontology we need to complement the MoOnt graph with assertional ("A-Box") statements that represent higher levels of the Modelica library stack. Since this is a lot of information to process a so called "MoTTL transcriptor" has been developed. We will introduce this tool and present the knowledge graphs generated by it. Having done that the final Evaluation step which is "Answering the Competency Questions".

In the end an outlook on the work still pending is given.

Previous Works

Semantic representations of Modelica were considered by (Pop & Fritzson, 2004), this was only shortly after the idea of the semantic web was published by in (Berners Lee, et al., 2001). Pop stated "Modelica users and library developers would benefit from Semantic Web technologies", but no further publications followed for a long time.

(Delgoshaei, et al., 2017) explored the use of semantic representations of different simulation models in Dymola and MATLAB for coupling HVAC and control engineering simulations. They confined to storing merely simulation results in semantic format.

The SPRINT project proposed the "use of OWL ontologies to represent several modeling tool languages, so that full models maintained in different tools could be represented in RDF" (Shani, et al., 2014). In this context the "Wolfram SystemModeler Ontology" (WSM) was published, which was reused and is therefore discussed in depth in this paper.

The whitepaper by (Zeb & Kortelainen, 2017) investigated a semantic representation of Modelica Models as an option for longterm data interoperability. They concluded that RDF representation is not a good solution for long term data interoperability. However, in the authors opinion, this is because they have only tried with a very basic approach without considering appropriate domain libraries. The publication by (Roxin, et al., 2021) has the same shortcoming, whereas they have described the use of domain libraries as a promising approach in their outlook. This has been conducted in this paper and is described in the section "Experiment - Populating the knowledge graph with the MoTTL transcriptor".

(Nachawati, et al., 2022) describe a framework for systems engineering with Modelica called "GitWorks". One part of it is a model catalogue named "GitWork Commons" which is a knowledge graph of existing models. Just as shown in this paper they have some parser that transcribes existing models. As an ontology they use "ModelicaOML" which is their own development. Expressed not in terms of OWL but with the "Ontological Modeling Language" OML, which was formerly developed in the OpenCeasar project but is not a W3C standard. While the ModelicaOML ist published on github (OpenModelica Consortium, 2022), the transcriptor as part of the GitWorks Framework is not.

Modelica Basics

Modelica is an object oriented modeling language. It has been introduced by the (Modelica Association, 1997), that cares for it ever since. Modelica Language Specification 3.5 is the most recent version.

Modelica models consist of components. Components usually have a number of connectors, which can be connected to other suitable connectors. Components are usually instances of library elements, that are created by "Drag-and-Drop" library elements to the "diagram layer" of a model. But they can also be custommade.

Each Model has 4 layers. Most commonly used is the "diagram layer" which shows the model as a "block diagram" with its components and their connections (an example is shown in *Figure 4*). This can be switched to the other layers. The full information is always represented by the text layer, which shows the source code of the model. Most elements, but not all elements have a graphical representation. Furthermore "Documentation" and "icon" layer exist.

Modelica Models are equation based. This means that they are based on equations and not algorithms. A sequencing of the calculation is not given. Therefore connectors and connections are usually undirected.

Since the term "model" has a crucial role in the Modelica terminology it is important to notice, that library elements as well as the instances are both called"models". There are executable models and non executable models. Executable models are by convention marked with a certain icon, but cannot be distinguished by a keyword in their sorce code.

Even though it is not visible in the GUI or by keywords in the source code, there is a library stack in Modelica. It is shown in *Figure 1* (red pyramid). The Basic Library that all others depend on is the Modelica Standard Library (orange box in *Figure 1*). It is a library maintained by the Modelica Association. Its current release is 4.0.0, while the old MSL 3.2.3 ist still widely used. Next level libraries (turquoise levels) depend on the MSL.

In the domain of buildings simulation the so called "IBPSA"(International Building Performance Simulation Association)-libraries are important. These are four libraries sharing common core. Since this core is shared by copying it to the respective libraries it is not consideres a separate layer in the library stack (moreover the core is not longer maintained as a separate artifact). The major Libraries on that level that will be applied in this paper are the "AixLib" (RWTH EBC, 2021) and the "Modelica Buildings Library" (Berkeley Lab Modeling & Simulation, 2014).

On top of that, every user can define their own libraries. Lastly the instance models are created (green level in *Figure* 1) – mostly by "drag-and-drop" the components from the underlying libraries.

Since both libraries (no matter if user defined or from an external source) and instance models are usually saved as "packages" in Modelica, they cannot be distinguished from each other on a technical basis. Just as there is no differentiation between the "model" which can be a library component or a executable instance model – there is also no differentiation between a "package" that is a library and a package that is a collection of executables models. However there is a convention to mark executable models and their packages with a certain icon, but this is not technically ensured by the Modelica Environments. Furthermore users can create Modelica Models without employing a Modelica Environment.

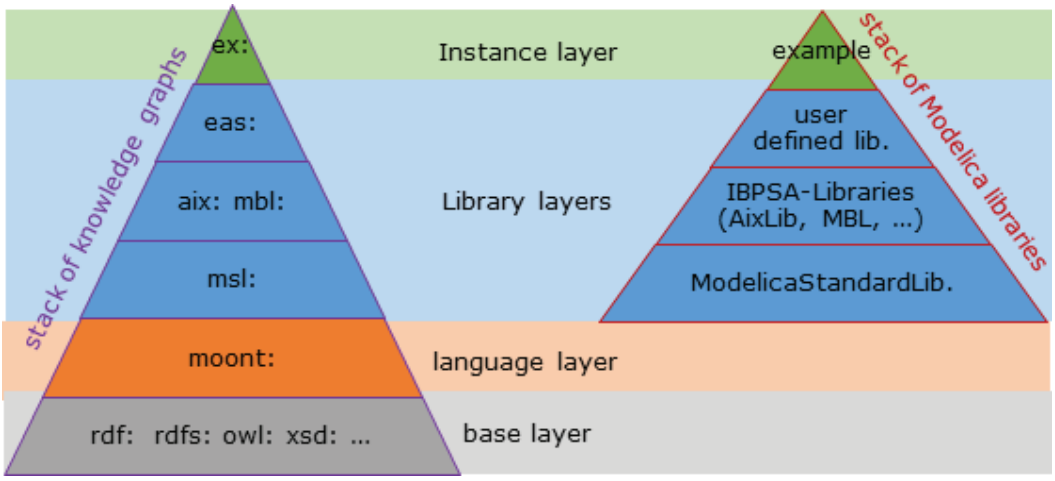

Figure 1: Modelica Library stack and corresponding stack of knowledge graphs ("KG-stack")

Knowledge Engineering

Overview – Methodology

Shortly after the emergence of the basic ideas about ontologies, semantic web and knowledge graphs, the need to develop such knowledge representations with a systematic approach was recognised. (Pinto & Maertin, 2004) compared different methodologies and identified the recurring process steps shown in *Figure 2*. They also described the engineering process as an iterative procedure.

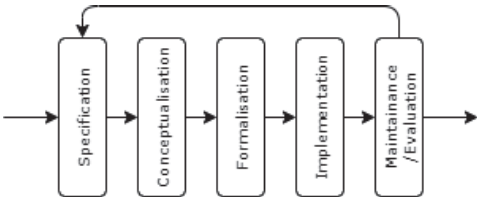

Figure 2: Knowledge Engineering Methodology

At the beginning there are always the specifications. Within this step, the requirements are defined. This is usually done as human-readable free text. This step is followed by "conceptualisation", i.e. working out the essential terms and their relationships to each other. Then these concepts and relationships are formalised and implemented, two steps that take place simultaneously when a software tool is used directly. This is followed by the equally important steps of "Maintenance and Evaluation".

Step 1: Specification

The goal of the knowledge engineering process was to represent any Modelica Model in a form that is ables to answer the following competency questions (CQ). These are representative of expected other questions arising in the context given in the chapter "introduction".

CQ1 Which executable models contain a certain library component?
This is a question typically asked by a modeller still learning or investigating new libraries. Typically one finds a promising component and look for an example on how to use and parametrize it.

CQ2 Which alternative implementations for a certain model component are available in the libraries?
To act as an alternative, "plug compatibility" between the models must be given. This requires that they have the same connectors.

CQ3 What is the nominal power of the HVAC plant components?
For HVAC plants the nominal power of a component is usually the most significant parameter that is also interesting to other participants of the design process.

CQ4 How many data points have to be provided by the building automation system?
This might help with a rough cost estimate for the tendering phase in a design process. Usually costs of the building automation system depend on the number of data points. These map to Modelica RealInput and RealOutputs.

CQ5 Which IfcProducts should be generated from the Modelica models components?
The last question cannot be answered with the Knowledge Graphs presented in this paper, but will be important for further work in the context of BIM and Simulation.

Step 2: Conceptualisation

It was decided to mirror the library stack described before with a corresponding stack of knowledge graphs (see *Figure 1* purple framed pyramid). Below the Modelica Standard Library the are two more layers of knowledge graphs, which are the Modelica Ontology representing the language layer and below that there are the well-known W3C standards. The purple pyramid shows the namespace prefixes of the different knowledge graphs.
For representing the language layer the so called "Modelica Ontology" - abbreviated "MoOnt" – was developed (Eckstädt, 2022). Since the goal was to develop an ontology representing a programming language (Modelica), it was the obvious choice to orient to the language specification. It contains the major terms. The

Modelica Language also contains a class hierarchy that should be reflected in the ontology.
As fundamental concepts we found 27 classes, 19 relations between classes and 21 relations from classes to literals to be necessary. No individuals are foreseen in the ontology, since these belong to the library layers of the KG-stack. Although not all entities identified in the language specification are necessary to answer the CQ given earlier, it was decided to cover all entities in MoOnt. It covers the whole language specification and is therefore more flexible for future use.

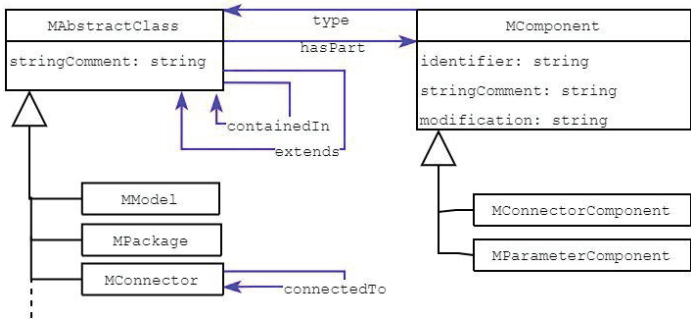

Figure 3: mayor entities of the MoOnt ontology shown as UML class diagram

Steps 3-4: Formalisation, Implementation – MoOnt

The next step in knowledge engineering is to formalize and implement the concepts. When directly working with sofware tools both steps cannot be distinguished.
It is crucial to consider ontology reuse in this step. Wolfram System Modeler (wsm) Ontology (Sprint Consortium, 2014) was mentioned earlier in this paper. It was developed for representing Modelica Models but has – to the authors best knowledge - not been maintained since its release in 2014. Therefore it was reused as a basis for the MoOnt by copying its contents.
The most important parts that have remained unchanged between the predecessor wsm and the MoOnt are shown in the simplified UML diagram in Figure 4.
There is an abstract superclass representing all Modelica classes, where "model" and "package" are the most frequently used ones. Another important entity that was already mentioned is the so called component, with its special cases "ConnectorComponent" and "ParameterComponent". This special cases have been added as they were not present in wsm.
As the affiliation to a certain package or model is also very meaningful for a Modelica classes content, a containedIn-Relation has been added to the ontology with MAbstractClass as domain and range.
Further small adjustments were made, but are not described in this paper.
The implementation of the ontology was done using the free Protégé editor (Protege Team, 2015), it contains 267 axioms. It was exported as a ttl-File that is published on github (Eckstädt, 2022).

Experiment - Populating the knowledge graph with the MoTTL transcriptor

The next step in the knowledge engineering methodology is to evaluate the ontology, to decide whether another iteration is necessary or if only minor adjustments are necessary that can be qualified as "maintainance" works. To be able to evaluate against the competency questions given in step 1 "specification" we need to populate the knowledge graph with information from the upper layers of the library stack

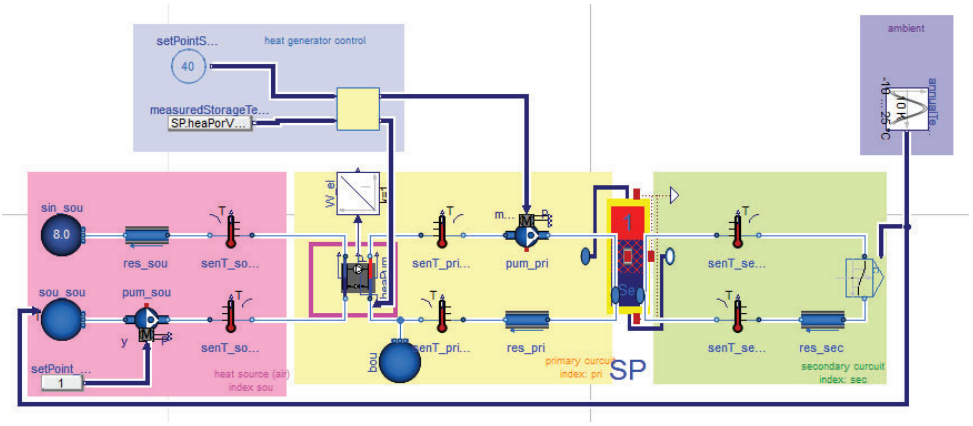

Figure 4: Heat pump plant - example of a Modelica model[1] shown as screenshot from Dymola Environment

[1] https://github.com/ElisEck/MO-x-IFC/blob/main/src/test/resources/C_HeatPumpPlant/LBDCG_example/HeatPumpPlant_V2.mo

```
within Buildings.Fluid.HeatPumps;
model Carnot_y
  "Reversible heat pump with performance curve adjusted based on Carnot efficiency"
  extends Buildings.Fluid.Chillers.BaseClasses.PartialCarnot_y(
    final COP_is_for_cooling = false);
```

```
mbl:Buildings.Fluid.Chillers.Carnot_y rdfs:subClassOf moont:MModel;
    moont:stringComment "Chiller with performance curve adjusted based on Carnot
        efficiency"^^xsd:string;
    moont:containedIn mbl:Buildings.Fluid.Chillers.
mbl:Buildings.Fluid.Chillers.Carnot_y moont:extends
        mbl:Buildings.Fluid.Chillers.BaseClasses.PartialCarnot_y.
```

Listing 1: Comparison of representations of library layer (excerpt): (upper listing) Carnot_y- heat pump model in native Modelica file[2] and (lower listing) turtle syntax[3] of the knowledge graph, equivalents are highlighted with same colour (full files available in repository (Eckstädt, 2022))

```
Buildings.Fluid.HeatPumps.Carnot_y heaPum(
    COP_nominal=4,
    …
    P_nominal=1000*(20/4))
…
connect(heaPum.port_b1, senT_pri_VL.port_a);
```

```
ex:LBDCG_example.HeatPumpPlant moont:hasPart ex:LBDCG_example.HeatPumpPlant.heaPum.
ex:LBDCG_example.HeatPumpPlant.heaPum a moont:MComponent ;
    moont:identifier "heaPum";
    a mbl:Buildings.Fluid.HeatPumps.Carnot_y.
ex:LBDCG_example.HeatPumpPlant.heaPum.COP_nominal moont:modification "4.0"^^xsd:Real;
ex:LBDCG_example.HeatPumpPlant.heaPum moont:hasPart
        ex:LBDCG_example.HeatPumpPlant.heaPum.COP_nominal.
…
ex:LBDCG_example.HeatPumpPlant.heaPum moont:hasPart ex:LBDCG_example.HeatPumpPlant.heaPum.P_nominal.
ex:LBDCG_example.HeatPumpPlant.heaPum.P_nominal moont:modification "5000"^^xsd:Real;
    moont:identifier "P_nominal".
…
ex:LBDCG_example.HeatPumpPlant.heaPum moont:hasPart ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1.
ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1 a moont:MConnectorComponent.
ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1 moont:identifier "port_b1".
ex:LBDCG_example.HeatPumpPlant.heaPum.port_b1 moont:connectedTo
        ex:LBDCG_example.HeatPumpPlant.senT_pri_VL.port_a.
```

Listing 2: Comparison of representations of instance layer (excerpt): heatPumpPlant as shown in Figure 4 as native Modelica file (upper frame) and in turtle syntax[4] (lower frame) of the knowledge graph, equivalents are highlighted with same colour (full files available in repository (Eckstädt, 2022))

Since this is a lot of information to process a tool was developed to generate a knowledge graph from native Modelica files. This tool was named "MoTTL transcriptor" (Eckstädt, 2022) and aims at moving one step forward in comparison to what was described on the section "previous works". The major advantage of the automated approach is, that it can process large amounts of data, also it can be run again if one of the libraries or the models are updated. The MoTTL transcriptor (Eckstädt, 2022) is available as a command line tool, its main input parameter is the path to some Modelica package. As an output the transcriptor generates a knowledge graph of this package in form of a turtle file. The MoTTL transcriptor was implemented in Java. First, a Modelica parser frame-work was created using the tool. „ANTLR (ANother Tool for Language Recognition, (Parr, 2021)) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files". ANTLR requires a grammar file in which the language to be parsed is formally specified. Such a grammar file in BNF form can be found in the appendix of the Modelica Language Specification (Modelica Association, 1997). The grammar file contains e.g. which keywords and separators are used and which order is relevant for the interpretation of the information in the file to be parsed. Based on the grammar, ANTLR generates a parser stub that can be used to traverse the tree represented by the Modelica file. The generated parser stub consists of several Java classes and interfaces, which were implemented to provide the necessary functionality to transform the Modelica file into an internal data model. For the internal data model a method serializeToTTL was implemented, which serializes the internal data model as a graph (in turtle syntax (W3C, 2014)).

Only the necessary entities to answer the competency questions are transcribed to the knowledge graph, most of the DatatypeProperties (that connect instances to literals) are therefore not transcribed.

The sourcecode of the MoTTL transcriptor is available on github along with an executable binary (Eckstädt, 2022). The Modelica example and the python queries describes in the next chapter are also available in this repository.

[2] https://github.com/lbl-srg/modelica-buildings/blob/master/Buildings/Fluid/HeatPumps/Carnot_y.mo
[3] https://github.com/ElisEck/MO-x-IFC/blob/src/main/resources/ontologies/8_ModelicaLibraries/MBL.ttl
[4] https://github.com/ElisEck/MO-x-IFC/blob/src/test/java/output/ex_20221215_1154_fullclean.ttl

Library layer graph

To exemplify this transcription for the Modelica Buildings Library (MBL) is processed. It consists of 12 major subpackages and is delivered in more than 4500 Modelica files, which are ASCII files. Each of this files contains one or more Modelica classes, they require 17.3 MB of hard disk space.The generated knowledge graph contains 644.000 axioms and the turtle-File occupies 42MB of hard disk space.

The two excerpts in *Listing 1* show the Modelica-File of the "Carnot_y"-HeatPump Model and its pendant in the knowledge graph. The "extends" and "containedIn"-Relations are highlighted in green and yellow.

Instance layer graph

Figure 4 shows an example of an executable instance model that contains the aforementioned Carnot_y-heatpump model. It is a model called "HeatPumpPlant" contained in a package names "LBDCG_example". By convention it has the icon designating executable models. It contains 25 components representing the heat source, a primary circuit to heat up a stratified storage, a secondary curcuit to heat the building, a simplified building model and a controller for the heatpump and its primary circuit pump.

The Modelica package occupies 30 kB on the hard drive, while the knowledge graph (represented as a turtle file) needs 88 kB, which is almost triple the size. The graph contains 1063 axioms, while the Modelica-file contents itself with around 600 lines of text.

Listing 2 contrasts brief excerpts from both representations again. In the upper frame is the native Modelica-File, the lower one shows the turtle file of the knowledge graph. The Modelica file contains additional annotations that are not shown for readability, since they are only important for the graphical representation in the diagram layer and can therefore be ignored for the semantic representation. The contrasting file excerpts for the library and instance level graphs show, that the semantic representation is verbose although it is incomplete.

Discussion and Result Analysis – Answering Competency Questions

Formalising competency questions

Main goal of the presented approach is to show the benefit of the semantic representation by querying the KG according to the competency questions stated in the beginning. To be able to answer them, it is necessary to formalize these questions into a computer-readable form. SPARQL queries are an appropriate choice, since it is the main query language for the Semantic Web. The queries are given for CQ1 to 4 in the listings 3 to 7. Answering CQ5 is out of the scope of this paper for now.

Implementing competency questions

The SPARQL queries have been implemented in a python script using the RDFlib (RDFLib developers, 2002). The queries are executed on a joined graph of the respective level and its underlying levels. These python scripts to test the implementation are published along with (Eckstädt, 2022).

The questions 3 and 4 also need knowledge of the executable instance model, therefore its graph needs also to be loaded into the querying engine. The level of user defined libraries is not present in the example and is therefore omitted.

Benefit from querying the MBL-graph with SPARQL CQ1 - Which executable models contain a certain library component?

The query in *Listing 3* is asking for models containing the Chiller.Carnot_y model. It returns 3 models that are either contained in an "Examples" or "Validation" package, which is reasonable. Also the results is inline with the expectations.

```
select ?subj ?obj where {
    ?subj moont:hasPart ?obj .
    ?obj a aix:AixLib.Fluid.Chillers.Carnot_y.}
```

Listing 3: CQ1 as SPARQL query (simple implementation)

Listing 4 shows a more complex form of CQ1: It asks for the implementations of a partial type, which is the equivalent to an abstract class in Modelica. More specifically we look for executable models that contains components which implement the abstract class. This is especially useful since the Fluid components make heavy use of partial Interfaces combined with multi inheritance that can hardly be overseen by users, who have not implemented these components. The query returns 44 models, among them the results of the query 1, which is inline with expectations.

```
select DISTINCT ?subj ?class where {
  ?subj moont:hasPart ?obj .
  ?subj moont:extends
        msl:Modelica.Icons.Example .
  ?obj a ?class .
  ?class moont:extends* aix:AixLib.Fluid.
          Interfaces.PartialFourPortInterface.}
```

Listing 4: CQ1 as SPARQL query (more complex version)

Benefit from querying the MBL-graph with SPARQL CQ2 - Which alternative implementations for a certain model component are available in the libraries?

The query in *Listing 5* asks for models that contain one real Input and two FluidPorts. Developers usually give their models two different ports FluidPort_a and FluidPort_B due to sign conventions. The query is asking for any "controlled flow element", since controller inputs are usually given by a RealInput.

```
select DISTINCT ?subj ?class where {
 ?subj moont:hasPart ?obj .
 ?subj moont:extends msl:Modelica.Icons.Example.
 ?obj a ?class .
 ?class moont:extends* aix:AixLib.Fluid.Interfaces.
                       PartialFourPortInterface.}
```

Listing 5: CQ2

This query delivers the expected results, among them being controlled valves and also the Carnot_y heat Pump model. However there are still problems with this query:

- It is slow: it takes 15 minutes to deliver this 99 models
- It scales badly: when looking for models that have 4 different FluidPorts the query doesn't deliver in hours

- It is not very general: In a first step it has to be determined which connectors a model has, in order to adjust the query in a second step

Although the test with respect to CQ2 can be considered successful, an improved implementation should be considered. The use of alternative triples stores and querying engines also appears promising.

Benefits from querying instance layer KG with CQ3 - What is the nominal power of the HVAC plant components?

The query in *Listing 6* extracts the values of all parameters that are Power values. An alternative implementation would be to identify the important parameters by their name instead of their type. Since different libraries use different naming convetions it was decided to use the quantities "power" and "heatFlowRate" for identifying the relvant parameters.

For answering this question a combined graph of the instance model and the used libraries needs to be queried, since the information of which type a parameter is, is only contained in the library if the parameter belongs to a library component.

The query not only asks for the parameter type, but also for the string comment of the parameter, which gives somes additional context (if available), since the parameter names are sometimes not sufficiently expressive. Also there is a filter for datatype Real to omit indirect parameters that are shown in grey in *Listing 7*.

```
select ?X ?mod ?comment where {{
  ex:LBDCG_example.HeatPumpPlant_V2 moont:hasPart
    ?comp .
  ?comp rdf:type/moont:extends*/moont:hasPart ?partT .
  ?partT moont:identifier ?ident.
  ?partT a moont:MParameterComponent.
  {?partT moont:type msl:Modelica.SIunits.Power. }
   UNION {?partT moont:type
   msl:Modelica.SIunits.HeatFlowRate.}
  ?comp moont:hasPart ?X .
  ?X moont:identifier ?ident.
  ?X moont:modification ?mod.
} UNION {
  ex:LBDCG_example.HeatPumpPlant_V2 moont:hasPart ?X .
  {?X moont:type msl:Modelica.SIunits.Power.} UNION
    {?X moont:type msl:Modelica.SIunits.HeatFlowRate.}
  ?X moont:identifier ?ident.
  ?X moont:modification ?mod.}
?X moont:stringComment ?comment.
FILTER ( datatype(?mod) = xsd:Real) }
```

Listing 6: CQ3

?X	?mod	?comment
ex:LBDCG_example.HeatPumpPlant_V2.heaPum.P_nominal	5000	Nominal compressor power
ex:LBDCG_example.HeatPumpPlant_V2.Qp_2	1000.0	heating load at 5°C
ex:LBDCG_example.HeatPumpPlant_V2.Qp_1	10000.0	heating load at -15°C
ex:LBDCG_example.HeatPumpPlant_V2.Qp_heater_nom	10000.0	Nennleistung Erzeuger
ex:LBDCG_example.HeatPumpPlant_V2.consumer.Qp_1	Qp_1	Stützleistung 1
ex:LBDCG_example.HeatPumpPlant_V2.consumer.Qp_2	Qp_2	Stützleistung 2

Listing 7: Results of CQ3 (grey result lines appear only if FILTER clause is not present

Benefits from querying instance layer graph with CQ4 -How many data points have to be provided by the building automation system?

In Modelica connectors are usually undirected, but there is one well justified exception, which are the `RealInput` and `RealOutputs`. They behave like signal ports, which is why they can be used to extract the information on building automation data points from the model. In the screenshot of the model *Figure 4* signal connections between `RealInputs` and `RealOutputs` are designated with a thick dark blue line (while fluid connections have light blue lines), 15 Signal Ports are visualized and are therefore the expected result.

The information on the interface type, is not available in the knowledge graph that covers the example, it is only available in the federated graph of the example and the libraries it uses. Also the information of the interface type is not attached to the instance of e.g. the heatPump, but it is connected to the class of the heat pump or maybe this class inherits its connectors from some ancestor. The query shown in *Listing 8* queries the full graph comprising of the example and the library graphs for signal ports. Its results are given in *Listing 9*.

In order for the result of the query to be interpreted in the sense of the CQ, however, there are requirements for the design of the Modelica model; it is essential, for example, that all components contained at the top level in the Modelica model (HeatPumpPlant) also have a real physical counterpart, e.g. a controller, a storage unit, ... This means that signal connections (thick dark blue lines in *Figure 4*) in the model must correspond to cables in reality, and not to internal signal connections within a controller. This is given in the example `HeatPumpPlant_V2`, a counter-example is included in the Modelica Package (`HeatPumpPlant`), but cannot be discussed here due to lack of space.

```
select ?comp ?ident ?class where {
  ex:LBDCG_example.HeatPumpPlant moont:hasPart ?comp .
  ?comp rdf:type/moont:extends*/moont:hasPart ?partT .
  ?partT moont:identifier ?ident.
  ?partT a ?class.
  ?comp moont:hasPart ?partA .
  ?partA a moont:MConnectorComponent .
  ?partA moont:identifier ?ident.
  FILTER (?class =
     msl:Modelica.Blocks.Interfaces.RealOutput ||
     ?class =
     msl:Modelica.Blocks.Interfaces.RealInput)
} ORDER BY ASC(?comp)
```

Listing 8: CQ4

?comp	?ident	?class
ex:BDCG_example.HeatPumpPlant_V2.W_el	u	msl:Modelica.Blocks.Interfaces.RealInput
ex:BDCG_example.HeatPumpPlant_V2.annualTemperatureCurve	T_oda	msl:Modelica.Blocks.Interfaces.RealOutput
ex:BDCG_example.HeatPumpPlant_V2.consumer	T_oda	msl:Modelica.Blocks.Interfaces.RealInput
ex:BDCG_example.HeatPumpPlant_V2.controlGeneratorWithPumpMassFlow	T_mea	msl:Modelica.Blocks.Interfaces.RealInput
ex:BDCG_example.HeatPumpPlant_V2.controlGeneratorWithPumpMassFlow	pumpSignal	msl:Modelica.Blocks.Interfaces.RealOutput
ex:BDCG_example.HeatPumpPlant_V2.controlGeneratorWithPumpMassFlow	T_set	msl:Modelica.Blocks.Interfaces.RealInput
ex:BDCG_example.HeatPumpPlant_V2.controlGeneratorWithPumpMassFlow	generatorSignal	msl:Modelica.Blocks.Interfaces.RealOutput
ex:BDCG_example.HeatPumpPlant_V2.heaPum	P	msl:Modelica.Blocks.Interfaces.RealOutput
ex:BDCG_example.HeatPumpPlant_V2.heaPum	y	msl:Modelica.Blocks.Interfaces.RealInput
ex:BDCG_example.HeatPumpPlant_V2.measuredStorageTemperature	y	msl:Modelica.Blocks.Interfaces.RealOutput
ex:BDCG_example.HeatPumpPlant_V2.pum_pri	m_flow_in	msl:Modelica.Blocks.Interfaces.RealInput
ex:BDCG_example.HeatPumpPlant_V2.pum_sou	y	msl:Modelica.Blocks.Interfaces.RealInput
ex:BDCG_example.HeatPumpPlant_V2.setPointStorageTemperature	y	msl:Modelica.Blocks.Interfaces.RealOutput
ex:BDCG_example.HeatPumpPlant_V2.setPoint_ControlValue_pum_sou	y	msl:Modelica.Blocks.Interfaces.RealOutput
ex:BDCG_example.HeatPumpPlant_V2.sou_sou	T_in	msl:Modelica.Blocks.Interfaces.RealInput

Listing 9: Results of CQ4 operating on HeatPumpPlant_V2 (15 "datapoints")

Conclusions and outlook

The knowledge graphs generated by the MoTTL transcriptor based on the MoOnt ontology have been succesfully queried with respect to the competency questions 1 to 4.

Practice-oriented queries such as CQ4 place demands on the modelling of the system on the Modelica side, that have been described. Implementation improvements for CQ2 are preferable, but a proof of concept has been shown.

Next steps are creating an alignment of Modelica to IFC, with the vision to generate IFC models from Modelica models and vice versa as described in (Eckstädt, 2021). In that context CQ5 will become answerable.

Acknowledgments

This work has been accomplished within the FMI4BIM project funded by the German Federal Ministry for Economic Affairs and Energy (BMWI) under reference number 03ET1603A and iECO project funded by the German Federal Ministry for Economic Affairs and Climate Action under grant agreement 68GX21011D. Special Thanks to all authors of the mentioned open-source tools for sharing their work and enabling pursued research.

References

Berkeley Lab Modeling & Simulation, 2014. *Modelica Buildings Library,* https://github.com/lbl-srg/modelica- buildings

Berners Lee, T., Hendler, J. & Lassila, O., 2001. The Semantic Web. *Scientific American*.

Delgoshaei, P., Heidarinejad, M. & Austin, M., 2017. Semantic Inference-Based Control Strategies for Building HVAC Systems Using Modelica-Based Physical Models. *Procedia Engineering*.

Eckstädt, E., 2021. Bidirectional coupling of Building Information Modeling and Building Simulation unsing ontologies. *EGICE*.

Eckstädt, E., 2022. *MoOnt.*: https://github.com/ElisEck/MO-x-IFC

Eckstädt, E., 2022. *MoTTL transcriptor.* https://github.com/ElisEck/MO-x-IFC

Modelica Association, 1997. *Modelica,* https://modelica.org/

Nachawati, M. O. et al., 2022. Towards an Open Platform for Democratized Model-Based. *Proceedings of the American Modelica Conference*.

OpenModelica Consortium, 2022. *ModelicaOML,* https://github.com/OpenModelica/ModelicaOML

Parr, T., 2021. *ANTLR,* https://www.antlr.org/

Pinto, H. S. & Maertin, J. P., 2004. Ontologies: How can They be Built?. *Knowledge Information Systems*.

Pop, A. & Fritzson, P., 2004. The Modelica Standard Library as an Ontology for Modelling and Simulation of Pyhsical Systems. *Whitepaper*.

Protege Team, 2015. *The Protégé Project,* https://protege.stanford.edu/

RDFLib developers, 2002. *RDFLib* https://rdflib.dev/

Roxin, A., Dundee, V. & Vukovic, V., 2021. Investigating Potential Alignments between Modelica Standard Library and SAREF Ontologies. *LDAC*.

RWTH EBC, 2021. *AixLib 1.0.0,* https://github.com/RWTH-EBC/AixLib

Shani, et al., 2014. SPRINT Software Platform for Integration of Eng. and Things D5.11 Final Report.

Sprint Consortium, 2014. *Wolfram System Modeller Ontology.* http://www.sprint-iot.eu/Wolfram-Modelica-ontology.zip

W3C, 2014. *Terse RDF Triple Language.* https://www.w3.org/TR/turtle/

Zeb, A. & Kortelainen, J., 2017. Web Ontology Language data modelling of Modelica simulation models. *Whitepaper*.