Web Ontology Language data modelling of Modelica simulation models

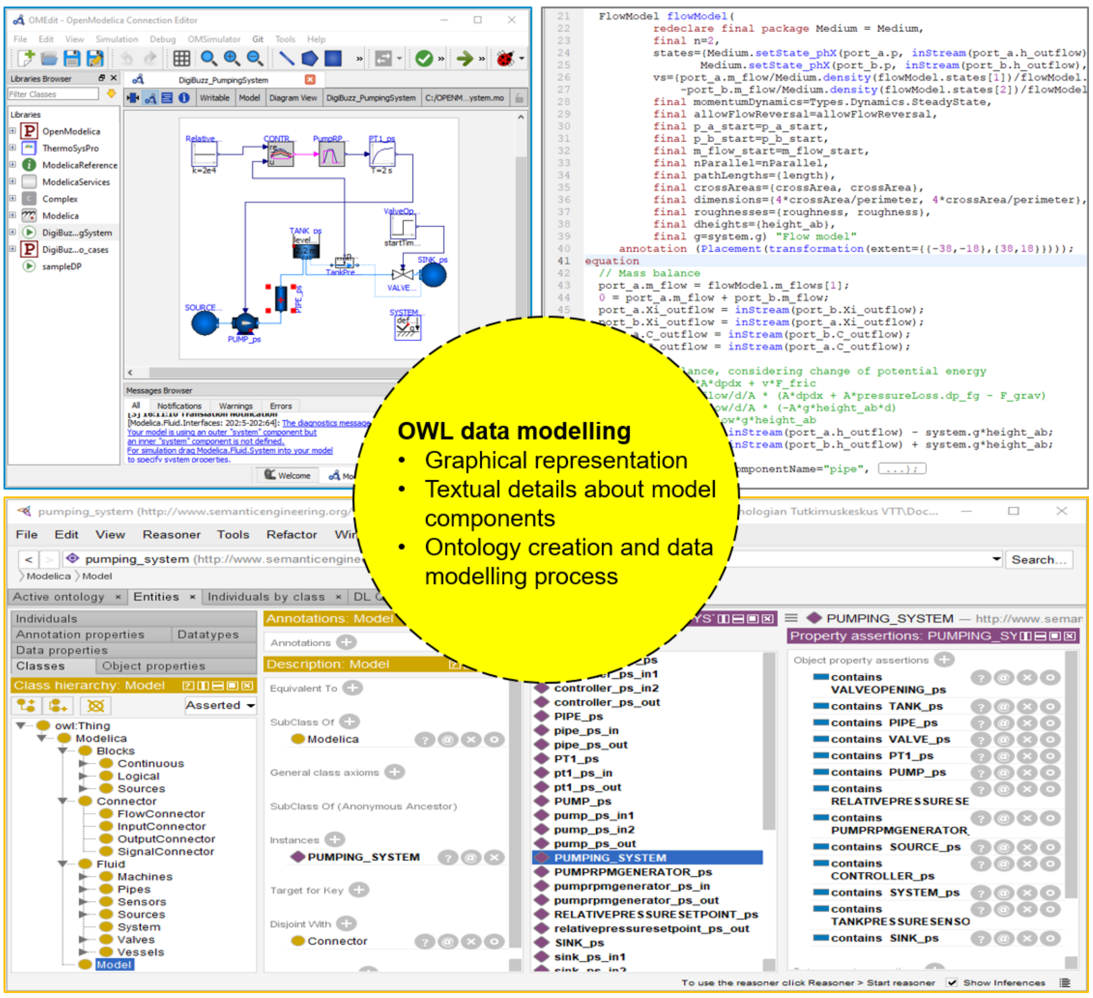
Zeb, Akhtar; Kortelainen, Juha

Published: 26/03/2021

Document Version
Publisher's final version

RESEARCH REPORT

VTT-R-01517-20


OWL data modelling
Graphical representation
Textual details about model components
Ontology creation and data modelling process

```
  FlowModel flowModel(
          redeclare final package Medium = Medium,
          final n=2,
          states={Medium.setState_phX(port_a.p, inStream(port_a.h_outflow)
                Medium.setState_phX(port_b.p, inStream(port_b.h_outflow),
          vs={port_a.m_flow/Medium.density(flowModel.states[1])/flowModel.
              -port_b.m_flow/Medium.density(flowModel.states[2])/flowModel
          final momentumDynamics=Types.Dynamics.SteadyState,
          final allowFlowReversal=allowFlowReversal,
          final p_a_start=p_a_start,
          final p_b_start=p_b_start,
          final m_flow_start=m_flow_start,
          final nParallel=nParallel,
          final pathLengths={length},
          final crossAreas={crossArea, crossArea},
          final dimensions={4*crossArea/perimeter, 4*crossArea/perimeter},
          final roughnesses={roughness, roughness},
          final dheights={height_ab},
          final g=system.g) "Flow model"
     annotation (Placement(transformation(extent={{-38,-18},{38,18}})));
equation
  // Mass balance
  port_a.m_flow = flowModel.m_flows[1];
  0 = port_a.m_flow + port_b.m_flow;
  port_a.Xi_outflow = inStream(port_b.Xi_outflow);
  port_b.Xi_outflow = inStream(port_a.Xi_outflow);
  a.C_outflow = inStream(port_b.C_outflow);
  outflow = inStream(port_a.C_outflow);

  lance, considering change of potential energy
  A*dpdx + v*F_fric
  low/d/A * (A*dpdx + A*pressureLoss.dp_fg - F_grav)
  ow/d/A * (-A*g*height_ab*d)
  ow*g*height_ab
  inStream(port_a.h_outflow) - system.g*height_ab;
  inStream(port_b.h_outflow) + system.g*height_ab;

  mponentName="pipe", ...);
```

Web Ontology Language data modelling of Modelica simulation models

Authors:	Akhtar Zeb, Juha Kortelainen
Confidentiality:	Public

Report's title Web Ontology Language data modelling of Modelica simulation models	
Customer, contact person, address Innovaatiorahoituskeskus Business Finland, Kari Koskela P.O. Box 69, FI-00101 Helsinki, Finland	**Order reference** 4437/31/2019
Project name Kohti digitaalisen kaksosen konkreettista kaupallista hyödyntämistä	**Project number/Short name** 125078/BF_DigiBuzz
Author(s) Akhtar Zeb, Juha Kortelainen	**Pages** 31/
Keywords Simulation, digital twin, data preservation, OWL, Modelica	**Report identification code** VTT-R-01517-20

Summary

Digital twins are promising several benefits, including better monitoring and control, improved performance, and lower maintenance costs of their physical counterparts. However, there are still some challenges associated with the development and management of digital twins that need to be addressed before the actual realisation of digital twins' applications. In this study, we look at the data management issue of digital twins that are based on system simulation.

The problem with most of the commercial modelling and simulation software is the use of proprietary data models and specific formats to store these models, resulting in poor interoperability. Since digital twins of certain assets need to be operated for a couple of decades, there is a possibility that the underlying application software may not be available through the assets' life cycles. In order to avoid unknown risks or business disruption, it is advisable to preserve information about the simulation models of digital twins in long-lasting formats. This is to ensure that in situations where the original software is inappropriate/inaccessible for opening/running a digital twin' simulation model, one should be able to revive the model, using the preserved information, and adopt alternative tools.

This work illustrates the long-term data preservation of digital twins by exploring two Modelica system simulation models. Modelica is an open-source language for modelling complex systems and provides detailed information about the models of components that can be preserved in user-specific formats. The models were built in OMEdit environment, a free tool for Modelica implementation. Data models of the corresponding simulation models were developed in Web Ontology Language (OWL), which is used for standardised representation of information on the Semantic Web. Protégé ontology editor was used to design four ontologies that provide the necessary concepts and relationships for describing the simulation models. Using these concepts and relationships, the OWL data models were generated. It was found that OWL lacks certain features and the visualisation of data models gets complex as the model size grows.

Confidentiality	Public	
Espoo 26.3.2021 **Written by** Akhtar Zeb, Research Scientist	**Reviewed by** Emmanuel Ory, Research Team Leader	**Accepted by** Tuomo Tuikka, Research Manager
VTT's contact address VTT Technical Research Centre of Finland Ltd, P.O. Box 1000, FI-02044 VTT, Finland		
Distribution (customer and VTT) Business Finland, Registry, 1 copy VTT, Archive, 1 copy		

Preface

Digital twins are under active research and development, both in the industry as well as in the research community. The concept of digital twin promises benefits and added value due to the improved use of data and better control of products, systems and assets. While the new opportunities are introduced and the benefits are emphasised throughout the life cycle of the physical twin, the life cycle of the digital twin seems to gain less attention. Similar to the implementation of all the new digital technologies, such as the Internet of Things, edge computing and advanced data analytics, digital twins would also increase the overall complexity of the solution and bring new dependencies to the solutions' life cycle management. This issue becomes even more challenging when the wide variety of technologies, the pace in the technology development, and the expected life cycles of technologies are considered. With industrial machines and systems, the life cycle expectation of several decades is common. On the other hand, in information technology and especially in computing, a decade is a long time and, e.g., the technical support of computer operating systems is typically shorter.

This work is part of the DigiBuzz research project in which the business aspects of digital twin technologies are taken into account, together with technical opportunities and new business opportunities, but also raising risks are considered. Being critical and pointing out challenges does not mean that we think the progress in the concept of digital twin and the related technologies is a waste of resources, but, to the contrary, we try to highlight the potential challenges that could arise later during the application of digital twins. This would help us to tackle the risks before they can even emerge. Digital twin is a powerful concept and like many other breakthrough technologies, it requires new kind of thinking to fully realise its potential but also to avoid the associated pitfalls.

We thank the funding parties, Business Finland and the participating companies and organisations, for enabling an interesting research project that clearly has relevance. The combination of technology and business research in the same package is strong and enables the partners to understand the elements of complex solutions from all the important directions.

Espoo 26.3.2021

Authors

Contents

1. Introduction

Today, the digital twin is a buzzword in academia as well as in the industrial sector. Developing digital twins of the physical assets (e.g. components, products, systems or processes) operating in the real-world are promising several benefits, including better monitoring and control, improved performance, and lower maintenance costs. However, there are still some challenges associated with the development and management of digital twins that need to be addressed before the actual realisation of digital twins' applications. In this work, we look at the data management issue of simulation-based digital twins.

As digital twins can be categorised in several ways, nevertheless, they are mainly data-driven (i.e. based on static and dynamic data of the real-world system and, e.g., AI and ML algorithms), simulation-based (i.e. based on first-principles or other kinds of simulation models), or hybrid (i.e. a combination of the first two). In this work, we focus on the simulation-based digital twins that are built using simulation software. The problem with most of the commercial modelling and simulation software is their proprietary data models and storing models in software application-specific formats. The data model of one software can hardly be related to the data model of another software and, in the worst-case, their simulation model formats are black boxes that do not provide information about the mathematics and physics involved in the simulation process. Since digital twins of certain assets (e.g. aircraft, ships, defence systems, and process plants) need to be operated for several decades, there is a possibility that the underlying application software will not be available through the assets' life cycles. Thus, for the development of simulation-based digital twins, it is advisable to consider only those application software that provide detailed information about the simulation models and preserve this information in long-lasting formats. This is to ensure that in situations where the original software is inappropriate/inaccessible for opening/running a digital twin model, one should be able to revive the models, using the preserved information, and adopt alternative tools without substantial costs or business disruption.

The operation of a computer simulation of a system or phenomenon requires a computer system, a simulation software and a description of the actual simulation model. The computer system typically contains the hardware, i.e. the physical computer system, and the operating system that provides the common functions and services for the computation, such as networking and computing process scheduling. The simulation software is a dedicated software application enabling the execution of a simulation. The simulation software can be a general simulation software application for the selected type of simulations, or it can be a simulation model dedicated software application that contains the integrated simulation model description and numerical solver algorithm. In the first case, the general simulation software can be called a solver, as it is used for solving a mathematical representation of the simulation model. Examples of this kind of solver software are majority of the computational fluid dynamics and finite element method software applications, such as Ansys Fluent [1] and Dassault Systèmes Abaqus [2]. In the second case, i.e. when the simulation model and the solver algorithm are integrated, the overall software application is usually implemented by coding it with the appropriate programming language and programming tools. There are simulation solutions that combine the two, i.e. they are general simulation applications, but also produce a dedicated software application that contains the description of the dedicated simulation model and the numerical solver to run the simulation. Many Modelica language implementations represent this category. A simulation model is the representation of a particular simulation case in a format that can be used with the selected simulation software application. The simulation model contains information about the components and features included in the simulation. For example, in the case of a thermodynamic system, the simulation model has information about what kind of system components are included, what are the parameters and initial simulation values of the components, and how the components are connected and interacting with each other. Together with the simulation software application, the simulation model defines the mathematical representation of the physics, phenomenon and features that are included in the simulation of the physical product or system.

In this study, we consider the example of Modelica [3], which is an open, object-oriented and equation-based language for modelling and simulation of complex systems. The Modelica libraries [4], especially the open-source libraries, provide detailed information about the models of components. For example, the source code of the Modelica Standard Library [5] is publicly available for exploration and implementation. In addition, the different versions of the Modelica Specification [6] are freely available for developing Modelica translator or compiler. Furthermore, different commercial and open-source tools [7] can be used for the implementation of Modelica Specification. For example, OpenModelica [8], [9], which is used in this study, is a free Modelica environment available for modelling and simulation of Modelica-based models. The two Modelica system simulation models presented in this study are developed in the OpenModelica Connection Editor (OMEdit) environment [10], which is a free graphical user interface for model creation, connection editing, simulation of models, and visualisation of simulation results. These models do not correspond to any real-world system but are used for illustration purposes only.

For the management and long-term preservation of simulation-based digital twin's data (i.e. simulation model data), we consider the example of Web Ontology Language (OWL) [11], which is used for standardised representation of ontologies and knowledge models on the Semantic Web [12]. The concept of semantic data management of multibody system modelling was introduced by Kortelainen in 2011 [13]. We designed four ontologies that provide the necessary concepts and relationships for information representation of the simulation models. These concepts and relationships were utilised to generate OWL data models of the two Modelica system simulation models using the Protégé tool [14], which is an open-source ontology editor and a knowledge management system.

This work demonstrates the chain of information in different forms that enables reviving the functionality and accuracy of the original simulation models in other tools if the original model cannot be used. In this work, the chain contains the information about the model components, their parameters and relations in the system simulation model in the form of OWL data model, information about the mathematical implementation of the simulation model components in the form of openly available Modelica libraries, and finally the semantics and syntax of the Modelica language itself in the form of the openly available Modelica language specification documentation. The chain of information discussed in this work does not contain the description of the physical phenomena involved and the mathematical representation of the physics. The numerical representation of the mathematical presentation and the implementation of the numerical representation in the form of a software application are assumed to be implemented based on the information that is available in open Modelica domain libraries and the specification of the Modelica language itself. The whole chain, i.e. the fundamental elements of a computer simulation, is illustrated in Figure 1. The three topics, enclosed in a dash-lined box in the figure, are emphasised within this study. As Modelica is an open and well-specified language, it would be possible to only rely on it as the means to represent the model information. The OWL was selected to demonstrate the overall approach for data modelling. In addition, general data modelling technologies can provide better tools for data transformation from one representation to another.

The rest of the report is organised as follows. Section 2 briefly discusses the Modelica language and the OMEdit tool. Section 3 introduces the OWL language and Protégé tool. Section 4 presents two Modelica system simulation models built in OMEdit environment. Section 5 shows the development process of four ontologies using the Protégé tool. Section 6 explains the OWL data modelling of the simulation models. Finally, Section 7 summarises and concludes the findings of this study.

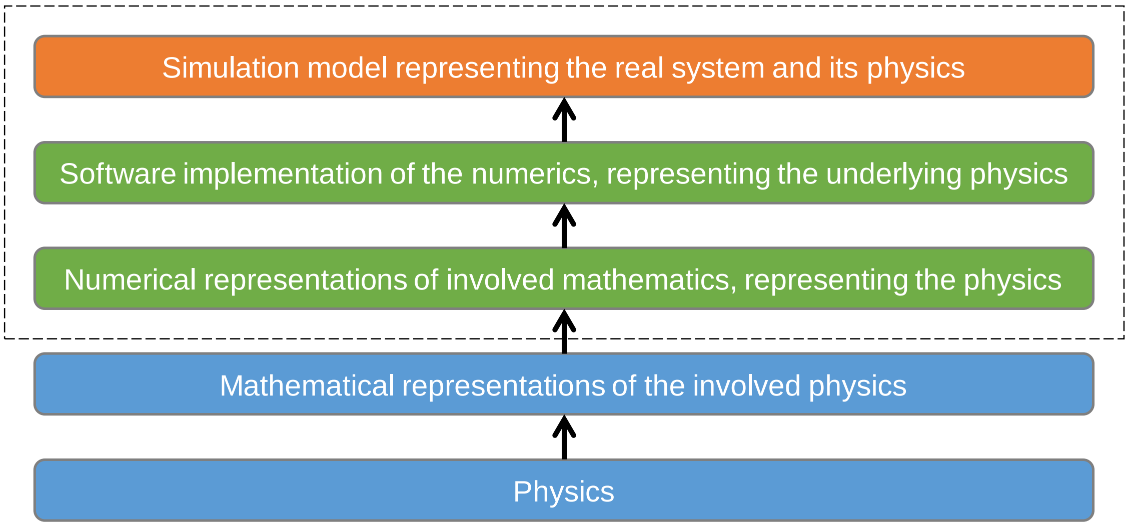

Simulation model representing the real system and its physics
Software implementation of the numerics, representing the underlying physics
Numerical representations of involved mathematics, representing the physics
Mathematical representations of the involved physics
Physics

Figure 1: The fundamental elements of computer simulation.

2. Modelica language and OMEdit tool

In this section, we briefly introduce the Modelica language and OMEdit tool. We chose Modelica because it is open-source and provides detailed information (mathematics and physics) about the components of a simulation model. By preserving this information in long-lasting formats, one would be able to reproduce a simulation model by using up-to-date tools, in case the original application software is not available. For the implementation of Modelica system simulation models, we chose the OMEdit tool, which is a free graphical user interface for model creation, connection editing, simulation of models, and visualisation of simulation results.

2.1 Modelica language

Modelica language [15] is a free language developed and maintained by the non-profit Modelica Association [16] since 1996. It is an object-oriented and equation-based language for modelling and simulation of complex systems consisting of components from, e.g., mechanical, electrical, thermal, hydraulic, control, and other domains. The different versions of the Modelica Specification [6] are publicly available for the implementation of Modelica compiler that transforms a Modelica model into a form (usually C-code), which can be simulated by standard tools. Both commercial and free simulation tools [7] that implement the Modelica Specification are available, for example, Simplorer (by ANSYS), Dymola (by Dassault Systèmes) and solidThinking Activate (by Altair) are among the commercial tools, whereas, OpenModelica (by the Open Source Modelica Consortium [8], [9]) is an open environment for modelling and simulation of Modelica models. Similarly, there are both free and commercial Modelica libraries [4] available covering different industries including automotive, aerospace, building and energy. The Modelica Standard Library [5] is a free library from the Modelica Association containing basic components for several domains, such as control, fluid and media, mechanical and electrical systems.

The expressive power of Modelica is large, supporting both high-level modelling by composition and detailed library component modelling by equations. Models of standard components are typically available in model libraries and users can also define their own model components. Using a graphical model editor (such as OMEdit), a model can be defined by drawing a composition diagram (also called schematics) by positioning icons that represent the models of the components, drawing connections between components and giving parameter values in dialogue boxes.

Figure 2 shows an example of a composition diagram of a double pendulum system, which is a "DoublePendulum" example available under the Mechanics package of the Modelica Standard Library. The system contains a set of components that are connected by drawing connections between their connectors. The textual representation of this model is shown in Figure 3, and the detailed model representation (definitions and equations) of the damper component of the system is shown in Figure 4.

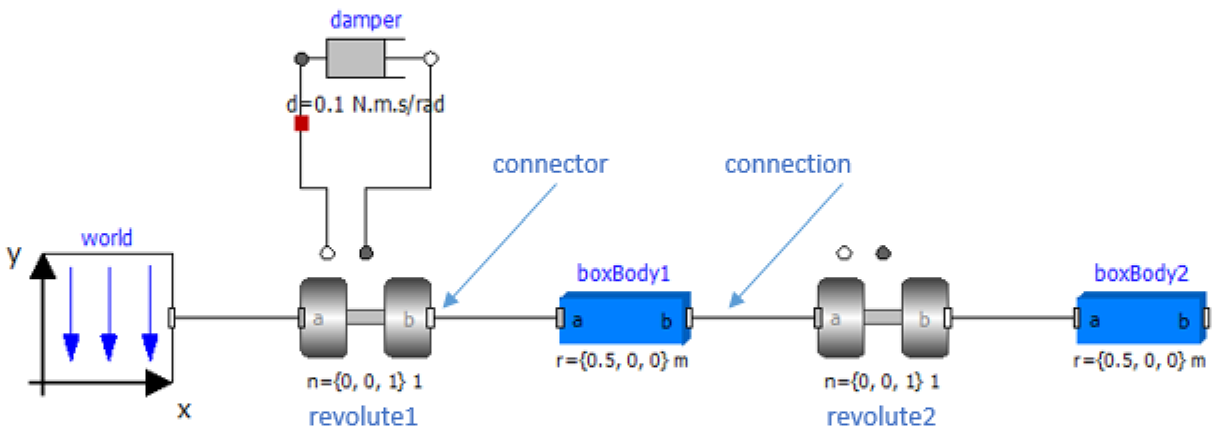
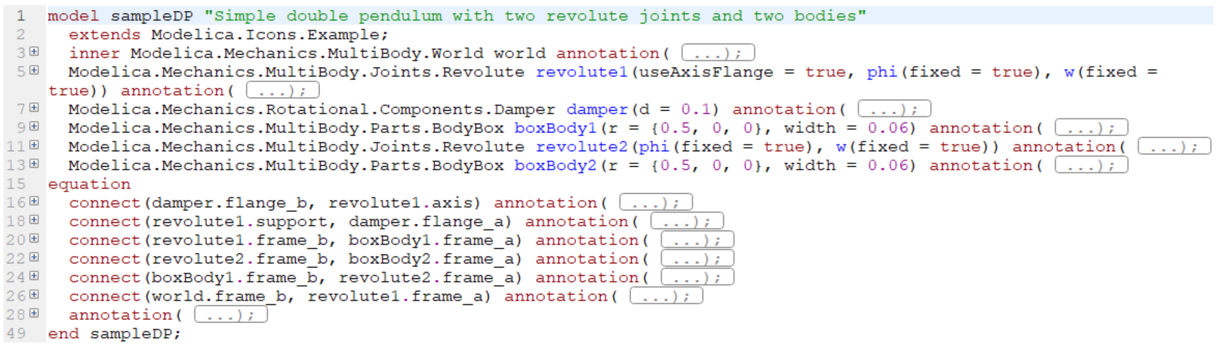
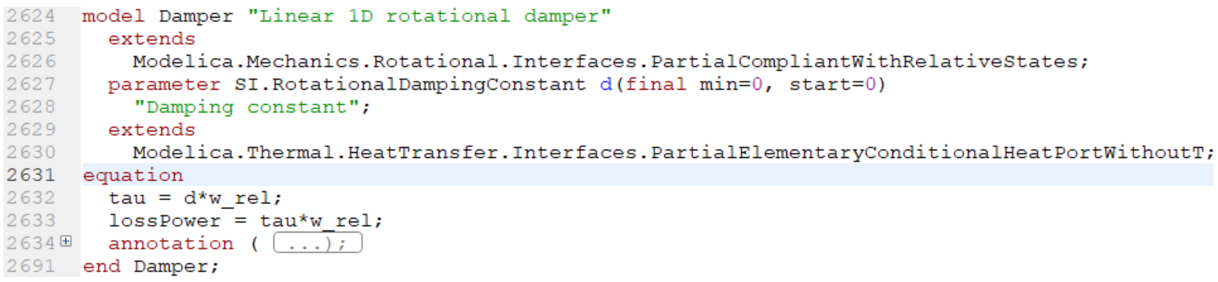


Figure 2: An example of a composition diagram of a double pendulum system.



```
model sampleDP "Simple double pendulum with two revolute joints and two bodies"
  extends Modelica.Icons.Example;
  inner Modelica.Mechanics.MultiBody.World world annotation( ...);
  Modelica.Mechanics.MultiBody.Joints.Revolute revolute1(useAxisFlange = true, phi(fixed = true), w(fixed =
true)) annotation( ...);
  Modelica.Mechanics.Rotational.Components.Damper damper(d = 0.1) annotation( ...);
  Modelica.Mechanics.MultiBody.Parts.BodyBox boxBody1(r = {0.5, 0, 0}, width = 0.06) annotation( ...);
  Modelica.Mechanics.MultiBody.Joints.Revolute revolute2(phi(fixed = true), w(fixed = true)) annotation( ...);
  Modelica.Mechanics.MultiBody.Parts.BodyBox boxBody2(r = {0.5, 0, 0}, width = 0.06) annotation( ...);
equation
  connect(damper.flange_b, revolute1.axis) annotation( ...);
  connect(revolute1.support, damper.flange_a) annotation( ...);
  connect(revolute1.frame_b, boxBody1.frame_a) annotation( ...);
  connect(revolute2.frame_b, boxBody2.frame_a) annotation( ...);
  connect(boxBody1.frame_b, revolute2.frame_a) annotation( ...);
  connect(world.frame_b, revolute1.frame_a) annotation( ...);
  annotation( ...);
end sampleDP;
```

Figure 3: A textual representation of the example double pendulum system.



```
model Damper "Linear 1D rotational damper"
  extends
    Modelica.Mechanics.Rotational.Interfaces.PartialCompliantWithRelativeStates;
  parameter SI.RotationalDampingConstant d(final min=0, start=0)
    "Damping constant";
  extends
    Modelica.Thermal.HeatTransfer.Interfaces.PartialElementaryConditionalHeatPortWithoutT;
equation
  tau = d*w_rel;
  lossPower = tau*w_rel;
  annotation ( ...);
end Damper;
```

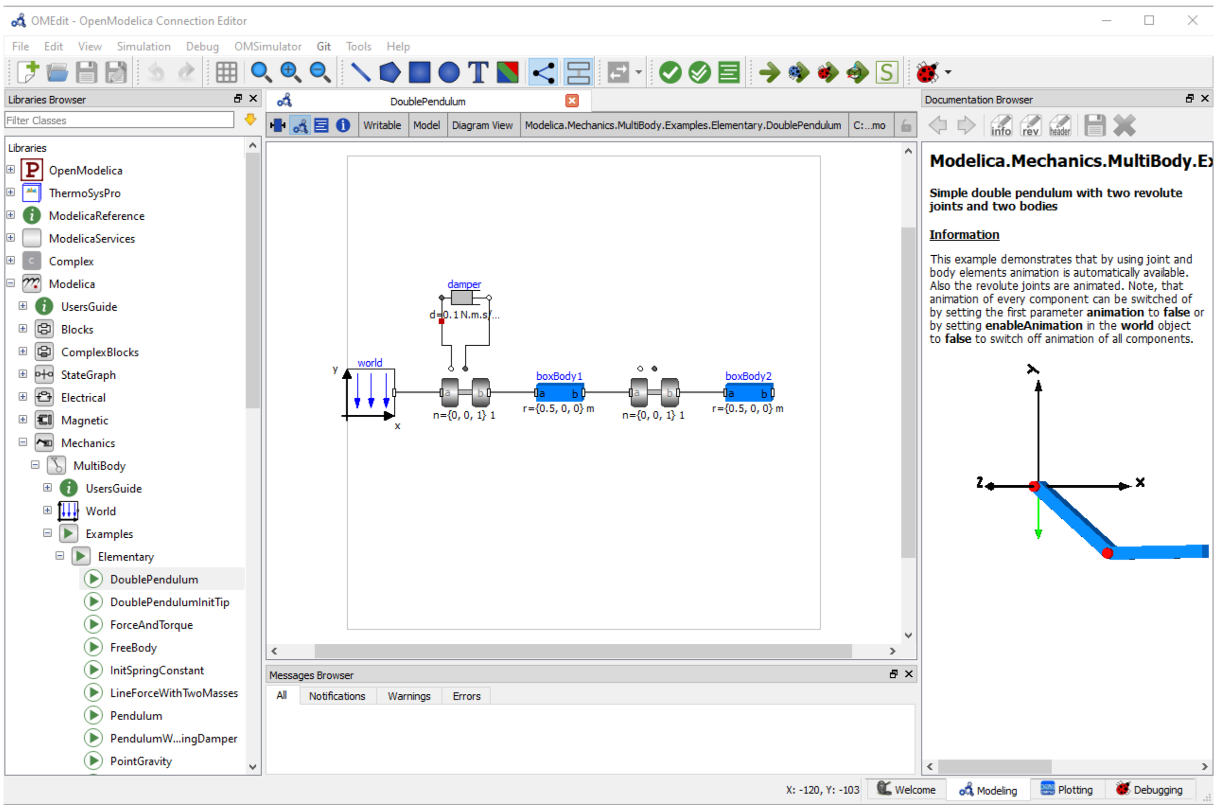
Figure 4: Detailed model representation of damper component of the double pendulum system example.

By preserving this information (mathematics and physics) about the components of a simulation model in a certain general format (e.g. in OWL), one can reproduce a similar component or system model using other languages or tools than Modelica, if needed. In this way, the simulation-based digital twins of long-lasting products and systems can be made less dependent on the computing infrastructure (software, operating system and computer system), which usually have shorter lifespan.

From the model description point of view, however, Modelica language has its own semantics and syntax, which may introduce some challenges in data modelling. For example, the connector type is a feature of the Modelica language that enables specifying restrictions on how the components of the system model can be connected and how the information is exchanged between the components. In Modelica library development, the type system of the Modelica language can be used for restricting connections to be defined only between limited types of connectors. Another noteworthy detail with Modelica is that the models and more specifically the connections between components are acausal, i.e. a connection does not have a direction and connecting component A to component B is equivalent to connecting component B to component A. In the case of data modelling, this is different from OWL, where the connections or relations are always directional. This is discussed in more detail in Section 5. The semantic and syntactic differences may cause challenges in data modelling due to the need for using complex data model structures to capture the semantics and details of the description of the target information.

2.2 OMEdit tool

OMEdit [10] is an open-source graphical user interface for model creation, connection editing, simulation of models, and plotting of results. The interface supports the creation of user-defined models or extensions. The models can be viewed as graphical or textual (see Figure 2 and Figure 3). Also, there is a possibility to add information about the model as documentation. The tool contains the Modelica Standard Library as well as several other open-source libraries (e.g. ThermoSysPro, Nuclear, SolarTherm), which can be found in System Libraries under the File tab. Figure 5 shows the user interface of OMEdit environment.



Modelica.Mechanics.MultiBody.Ex

Simple double pendulum with two revolute joints and two bodies

Information

This example demonstrates that by using joint and body elements animation is automatically available. Also the revolute joints are animated. Note, that animation of every component can be switched of by setting the first parameter **animation** to **false** or by setting **enableAnimation** in the **world** object to **false** to switch off animation of all components.

Figure 5: A user interface of the OMEdit tool.

3. Web Ontology Language and Protégé tool

3.1 Web Ontology Language

OWL [11] is used for standardised representation of ontologies and knowledge models on the Semantic Web [12]. The language supports defining libraries of domain concepts and using the concepts for modelling the data, information or knowledge in the selected cases. The libraries of concepts are called ontologies and they contain descriptions of the concepts (classes) and their attributes (data properties), named and typed relations between the concepts (object properties), and restrictions how the concepts are used and connected. In addition to OWL, the umbrella of the Semantic Web contains a large set of technologies that provide the bases for OWL, such as Resource Description Framework or RDF [17], the SPARQL query language for RDF [18], and the Semantic Web Rule Language or SWRL [19]. RDF defines the basis for the information model by defining the concept of data triple (*subject* – *predicate* – *object*, or *object* – *relation* – *object*), SPARQL defines a query language that unifies data retrieval from semantic databases, and SWRL adds a mechanism of simple rules and restrictions to OWL and extends it.

The purpose of the resulting ontology is to provide a library of concepts or a vocabulary that can be used for generating a data model of something concrete within a domain of interest. A data model, based on the ontology, then uses the concepts and relationships defined in the ontology. For example, the concepts (classes), *Person* and *Pet*, and relationships (object properties and data properties), *ownsPet* and *hasFurColor*, are defined in an ontology, which can be used to generate a data model of some facts. For example, *Sally* (an individual of *Person*) *ownsPet Fido* (an individual of *Pet*); *Fid hasFurColor 'grey'* (data value), and so forth.

In this work, we developed the same kind of statements describing the design of simulation models. Such simulation models, as shown in Figure 2, are composed of components and connections between these components. Thus, the components and connectors will serve as the individuals of classes and the parameters of the components will serve as the data properties. For example, we can state by using triples (subject – predicate – object, see Figure 6):

- *DoublePendulum* (individual of the class *model*) *contains* (object property) *damper* (individual of the class *Damper*)
- *damper* (individual of the class *Damper*) *hasDataProperty Diameter = 7.0* (individual of the data property *Diameter* with value '7.0')

In this way, we can preserve detailed information about a simulation model and in situations where the underlying system is not available for running the models, we can utilise the information and create new solutions. In our work, we use the Protégé tool for data modelling.

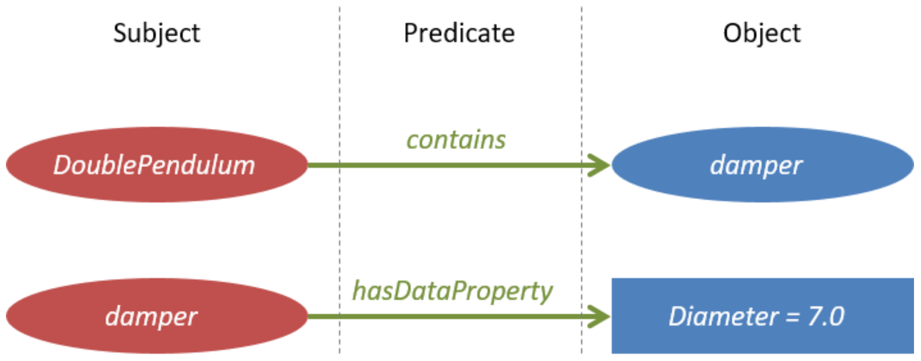


Figure 6: Examples of data triples for representing the information of a simulation model.

3.2 Protégé tool

Protégé [14] is an open-source ontology editor and a knowledge management system. There are both desktop and web versions of Protégé tool available. The desktop version has more functionalities than the web version and can be further extended by installing various plugins. Figure 7 shows the desktop version of Protégé tool used in this work.

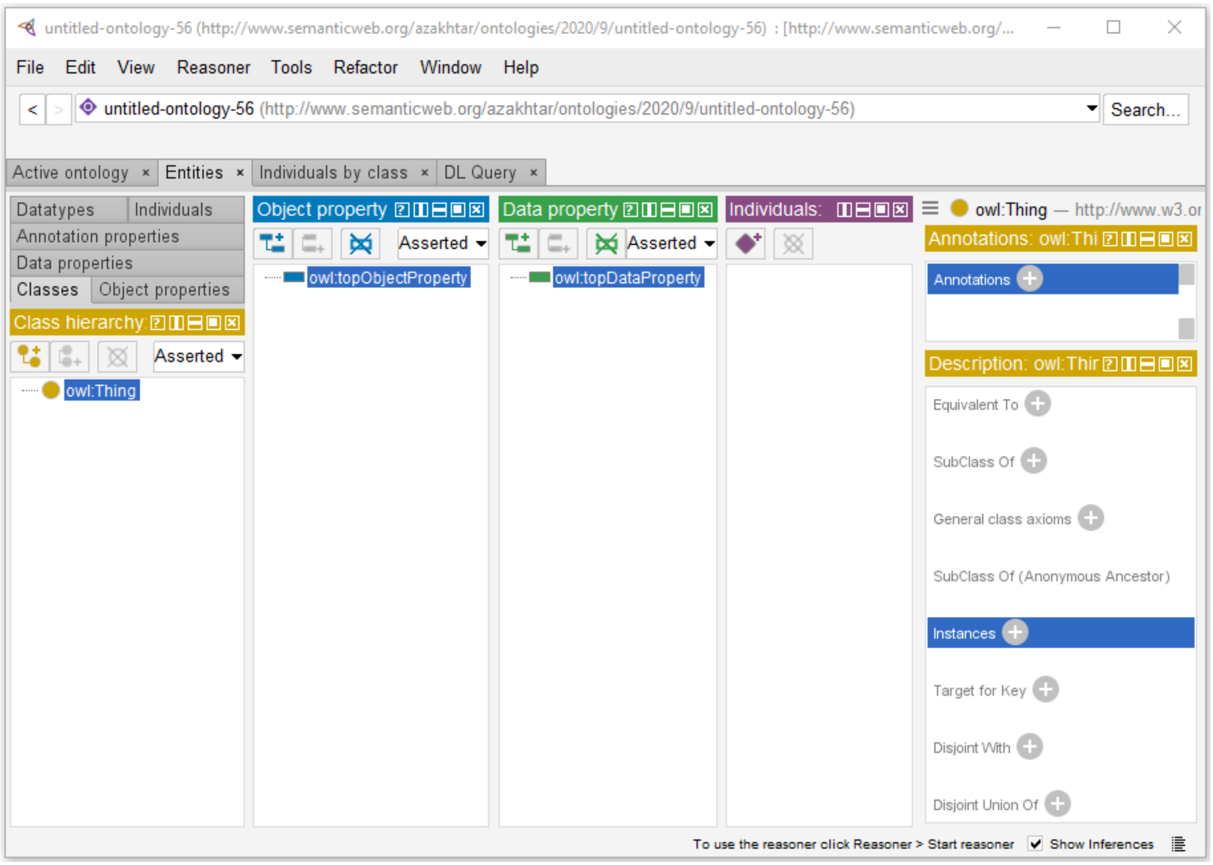


Figure 7: A user interface of the Protégé tool (Desktop version).

4. Modelica system simulation models

In this section, we present two Modelica system simulation models that are built in the OMEdit environment (see Section 6 for the OWL data modelling of the corresponding models using the Protégé tool). These system simulation models do not correspond to any real-world application and are used for illustration purposes only. The first model is named as Irrigation_System (shortly “irs”) and the second model as Pumping_System (shortly “ps”). The Irrigation_System is comprised of model components available in the ThermoSysPro library, and the Pumping_System consists of components available in the Modelica Standard Library.

4.1 ThermoSysPro library and the Irrigation_System model

ThermoSysPro [20] is an open-source Modelica library that has been mainly designed for the static and dynamic modelling of power plants, but can also be used for other energy systems such as industrial processes, buildings, etc. [21], [22]. It is developed by Électricité de France (EDF) and contains over 100 model components such as heat exchangers, steam and gas turbines, compressors, pumps, furnaces, and combustion chambers. In particular, one- and two-phase water/steam flow, as well as flue gases flow are handled [21]. Figure 8 shows the packages (e.g. Solar, Thermal, WaterSteam) and some of the model components (e.g. Compressor, Generator, StaticCentrifugalPump) of the ThermoSysPro library in the OMEdit.

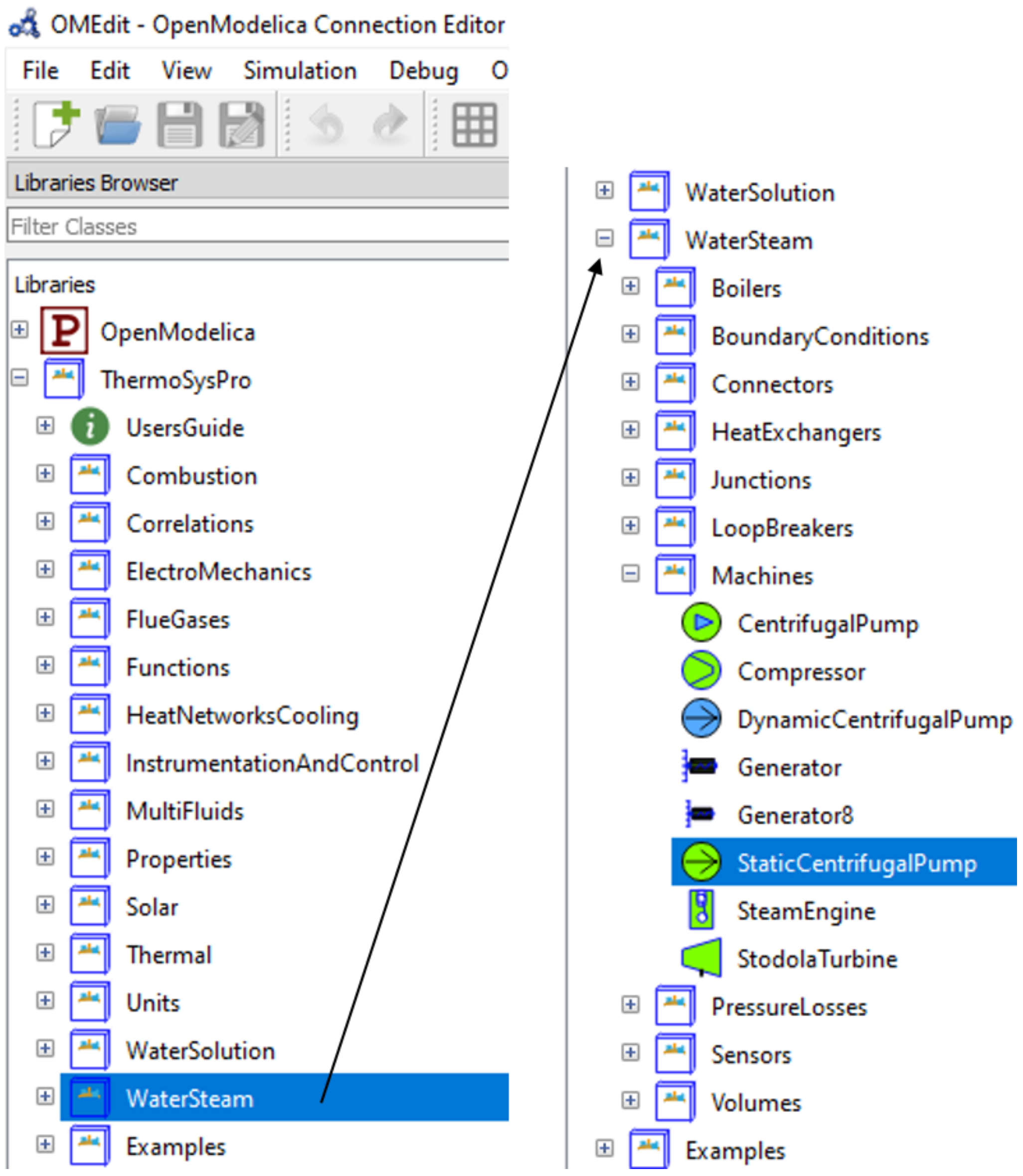


Figure 8: ThermoSysPro library in OMEdit tool, its sub-libraries and model components.

The model components of the ThermoSysPro library are used to build a Modelica system simulation model in the OMEdit environment, as shown in Figure 9. We named this system simulation model as “Irrigation_System” and abbreviated as “irs”. The Irrigation_System consists of five components: 1) source, 2) pump, 3) pipe, 4) valve, and 5) sink. These components are connected with each other, or more precisely, with the help of connectors. Each component of the Irrigation_System (and every component of the ThermoSysPro library) has one or more inlet and outlet connectors. For example, the pipe component has one inlet (dark blue square) and one outlet (dark red square) connectors as shown in Figure 9.

In this case, the pressure, temperature and elevation of water at the source (i.e. inlet boundary) is 1 bar, 30 ºC, and 0 m, respectively. At the sink (i.e. outlet boundary), the water pressure is 8 bar and elevation 5 m, whereas, the temperature of the water does not change during the process. The purpose of the system is to pump water from the lower pressure level (source) to the elevated higher pressure level (sink) with a mass flow rate of 10 kg/s through the pipe.

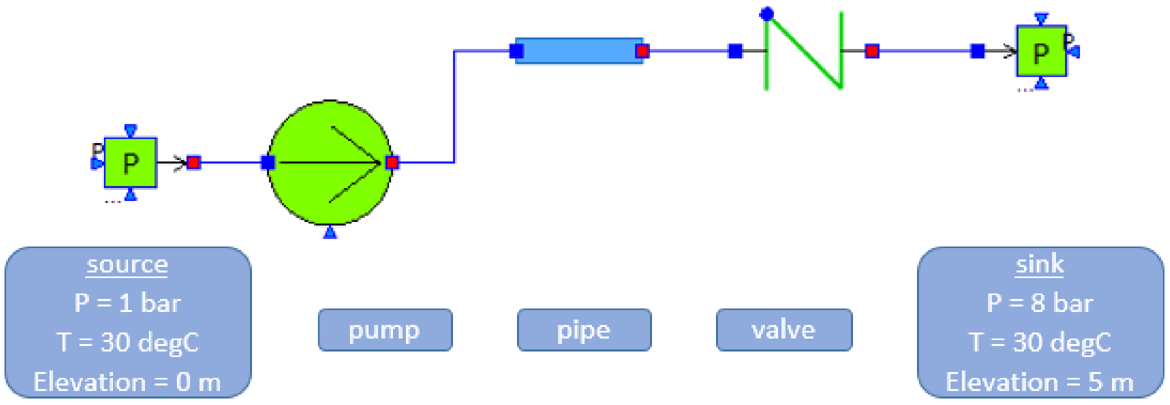
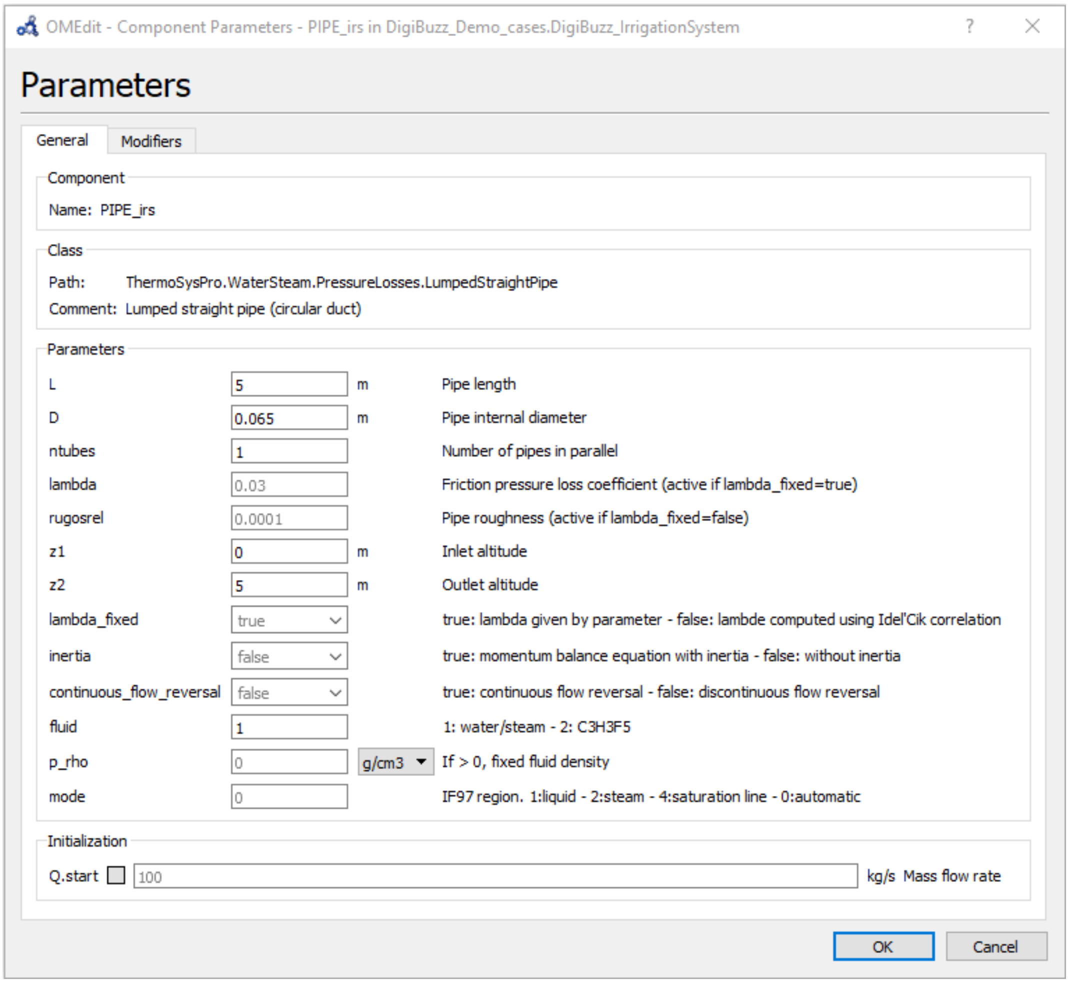


Figure 9: Simulation model of the Irrigation_System consisting of model components of the ThermoSysPro library and developed in the OMEdit environment.

In the Irrigation_System above, the water elevation is defined using the inlet and outlet altitude parameters of the pipe component (e.g. z1 and z2 in Figure 10). Each component of the Irrigation_System (as well as every component of the ThermoSysPro library) has several parameters and attributes with default values. Figure 10 shows a pipe component (named as “PIPE_irs” which is also an attribute) has parameters “L” (pipe length), “D” (pipe internal diameter), “ntubes” (number of pipes in parallel), and so forth. The values of these parameters and attributes are modified by the modelling and simulation experts according to their needs.


OMEdit - Component Parameters - PIPE_irs in DigiBuzz_Demo_cases.DigiBuzz_IrrigationSystem
?
Parameters
General
Modifiers
OK
Cancel

Component

Name: PIPE_irs

Class

Path: ThermoSysPro.WaterSteam.PressureLosses.LumpedStraightPipe

Comment: Lumped straight pipe (circular duct)

Parameters

L	5	m	Pipe length
D	0.065	m	Pipe internal diameter
ntubes	1		Number of pipes in parallel
lambda	0.03		Friction pressure loss coefficient (active if lambda_fixed=true)
rugosrel	0.0001		Pipe roughness (active if lambda_fixed=false)
z1	0	m	Inlet altitude
z2	5	m	Outlet altitude
lambda_fixed	true		true: lambda given by parameter - false: lambde computed using Idel'Cik correlation
inertia	false		true: momentum balance equation with inertia - false: without inertia
continuous_flow_reversal	false		true: continuous flow reversal - false: discontinuous flow reversal
fluid	1		1: water/steam - 2: C3H3F5
p_rho	0	g/cm3	If > 0, fixed fluid density
mode	0		IF97 region. 1:liquid - 2:steam - 4:saturation line - 0:automatic

Initialization

Q.start ☐ 100 kg/s Mass flow rate

Figure 10: An example of parameters and attributes of a pipe component of the ThermoSysPro library opened with OMEdit tool.

In this study, the focus is not on the development of an optimal simulation model but on the creation of a data model that can provide enough information for the reverse engineering of simulation models. The need may come up when the original modelling and simulation software is not available or cannot be used for running and editing an already existing simulation model. Thus, just for the sake of keeping the target data model simple, we modified only a few parameters and attributes of the Irrigation_System's components; most of the components' parameters and attributes were left unchanged with their default values and are not documented here. All the attributes and parameters of the components with their default values are available from the ThermoSysPro library and can be saved in a convenient format (see Figure 3 and Figure 4).

Table 1 shows only the modified parameters and attributes of the components constituting the Irrigation_System (irs), which are then used in the data modelling process, as described in Section 6. Similarly, Table 2 shows the modified names and types (inlet, outlet) of the connectors associated with each component of the Irrigation_System simulation model.

Table 1: Original names (left), and modified parameters and attributes (right) of the components constituting simulation model of the Irrigation_System (irs).

Original name of component in the ThermoSysPro library	**Modified parameters and attributes of component constituting simulation model of the Irrigation_System (irs)**		
	Modified name	**Modified parameter**	**Left as default**

SourceP	SOURCE_irs	P0 = 1 bar
		T0 = 30 ºC
			All other parameters

StaticCentrifugalPump	PUMP_irs	VRot = 1880 rev/min
		fluid = 1 (water/steam)
		MPower = 100000 W
			All other parameters

LumpedStraightPipe	PIPE_irs	L = 5 m
		D = 0.065 m
		ntubes = 1
		z1 = 0 m
		z2 = 5 m
		fluid = 1 (water/steam)
			All other parameters

CheckValve	VALVE_irs	fluid = 1 (water/steam)	All other parameters

SinkP	SINK_irs	P0 = 8 bar
		T0 = 30 ºC
			All other parameters

Table 2: Proposed names for the connectors of different components constituting the Irrigation_System simulation model.

Modified component name of the "irs" simulation model	**Proposed connector name for component**	**Connector name description**
SOURCE_irs	source_irs_out	outlet connector of SOURCE_irs
PUMP_irs	pump_irs_in	inlet connector of PUMP_irs
	pump_irs_in1	inlet signal connector of PUMP_irs
	pump_irs_out	outlet connector of PUMP_irs
PIPE_irs	pipe_irs_in	inlet connector of PIPE_irs
	pipe_irs_out	outlet connector of PIPE_irs
VALVE_irs	valve_irs_in	inlet connector of VALVE_irs
	valve_irs_out	outlet connector of VALVE_irs
SINK_irs	sink_irs_in	inlet connector of Sink_irs

4.2 Modelica Standard Library and the Pumping_System model

Modelica Standard Library [5] is developed together with the Modelica language from the Modelica Association [16]. It provides constants, types, connectors, partial models and model components in various disciplines. It consists of many sub-libraries or packages, including Fluid, Blocks, Machines, and MultiBody. Figure 11 shows the Modelica Standard Library, its sub-libraries or packages (e.g. Blocks, Mechanic, Fluid), and model components (e.g. Constant, OpenTank). In this study, we only use components of the Fluid and Blocks packages of the Modelica Standard Library for building the simulation model of Pumping_System.

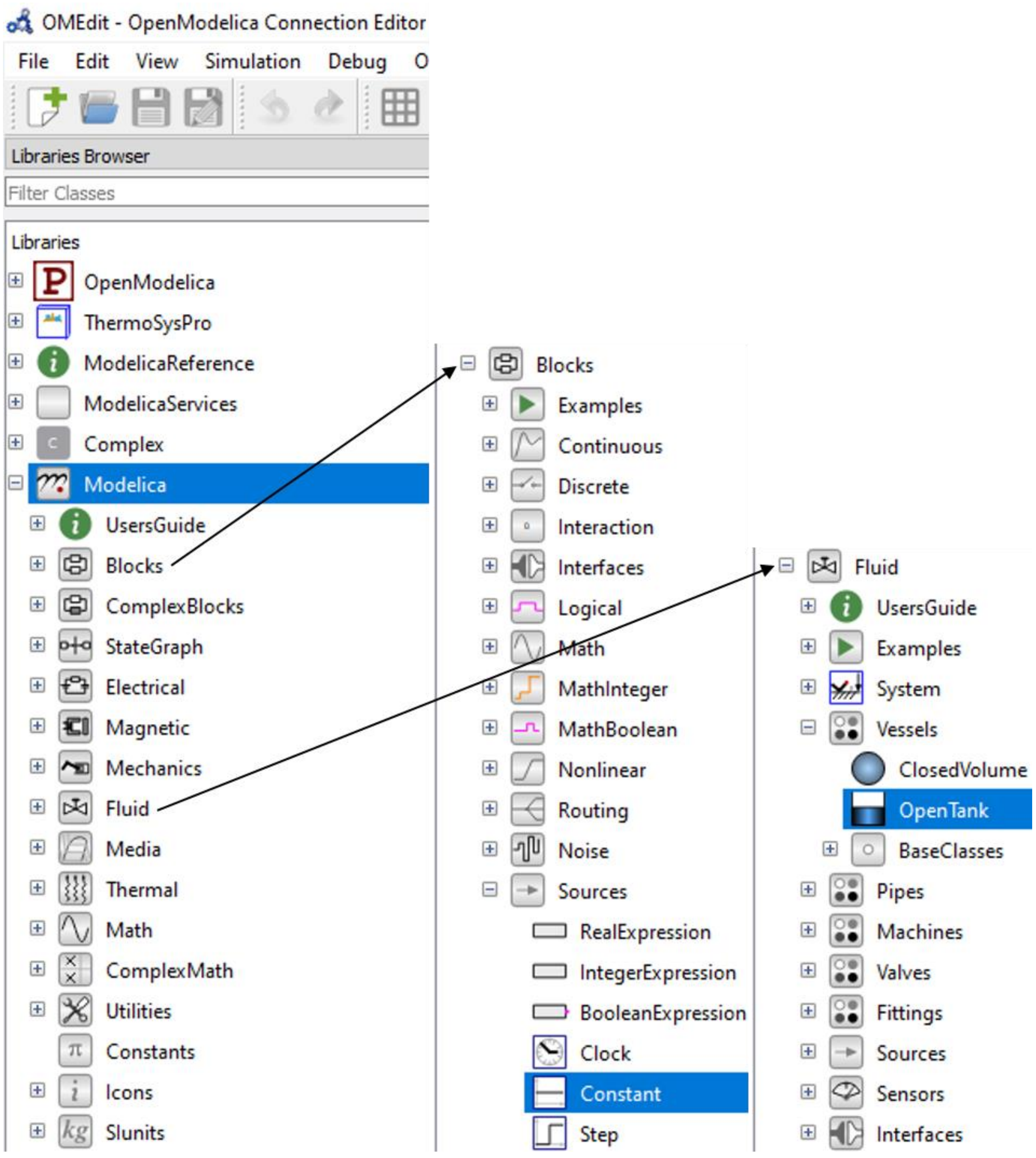


Figure 11: Modelica standard library in OMEdit tool, its sub-libraries and model components.

The system simulation model shown in Figure 12 is actually an example under the Fluid package of the open-source Modelica Standard Library. We named this simulation model as Pumping_System and abbreviated as "ps". The Pumping_System is composed of 13 components both from the Blocks and Fluid packages of the Modelica Standard Library. The component "SYSTEM_ps" (modified name in this study) is needed in each fluid model to provide system-wide settings, such as ambient conditions and overall modelling assumptions. The purpose of the system is to pump water from a source by a pump, through a pipe whose outlet is 50 m higher than the source, into a tank. The water level in the tank is controlled by a valve. The water controller is a simple on-off controller, regulating on the gauge pressure measured at the base of the tank. The output of the controller is the rotational speed of the pump, which is represented by the output of a first-order system. In order to avoid singularities in the flow characteristic, a small but non-zero rotational speed is used to represent the standby state of the pump. The system is simulated for 2000 s. When the valve is opened at time t = 200 s, the pump starts turning on and off to keep the tank level around 2.2 m, which roughly corresponds to a gauge pressure of 200 mbar, as shown in Figure 13.

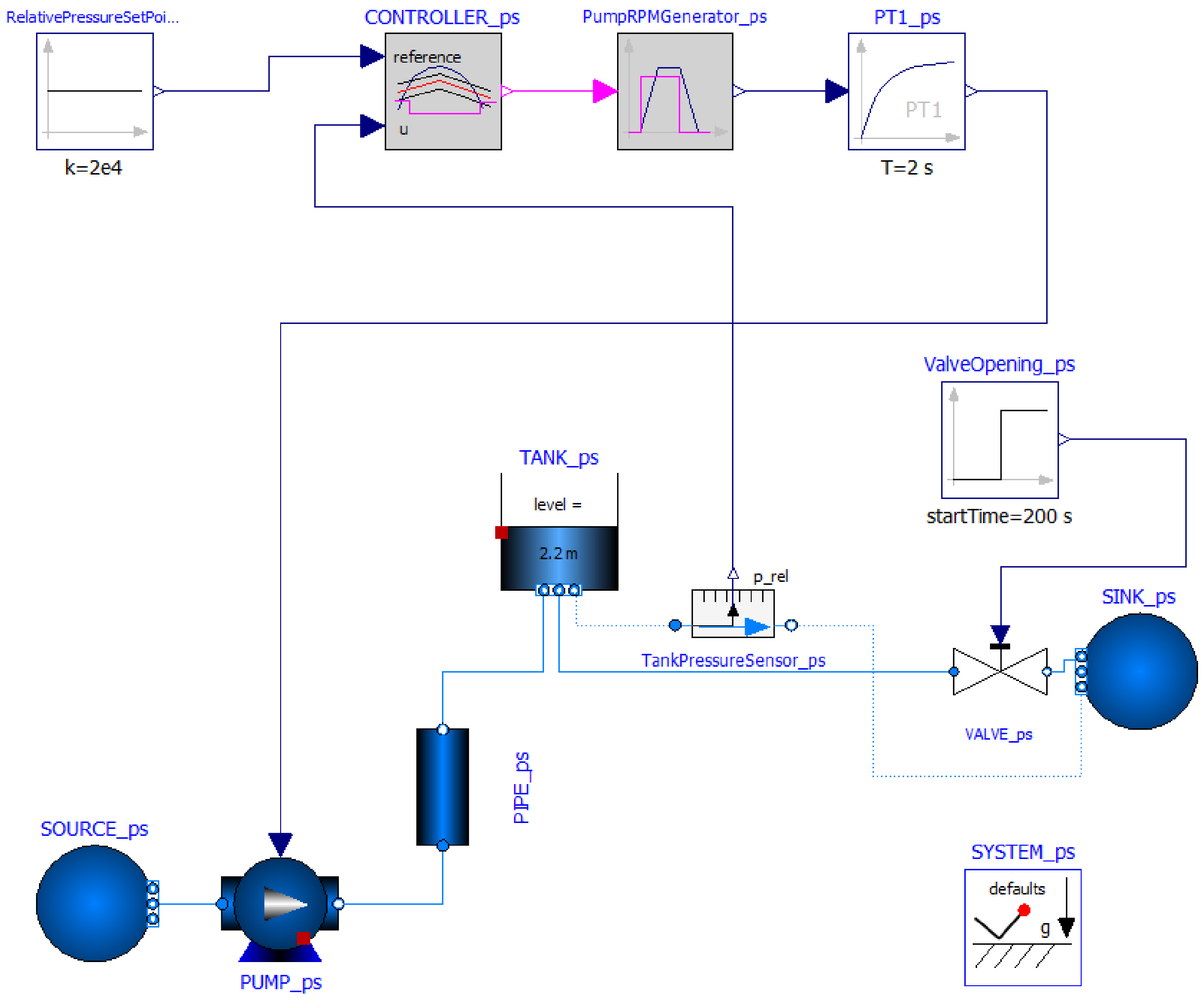
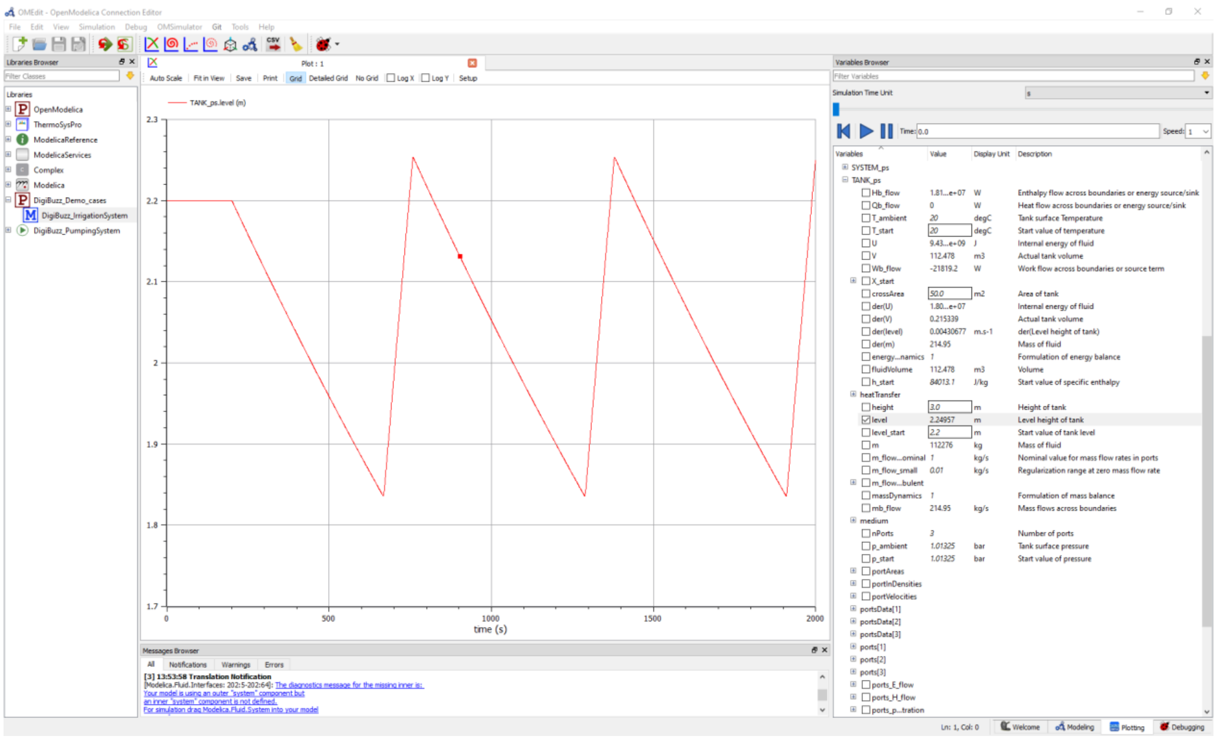


Figure 12: Simulation model of the Pumping_System consisting of model components of the Modelica Fluid and Blocks sub-libraries, and developed in OMEdit environment.



Variables	Value	Display Unit	Description
SYSTEM_ps			
TANK_ps			
Hb_flow	1.81...e+07	W	Enthalpy flow across boundaries or energy source/sink
Qb_flow	0	W	Heat flow across boundaries or energy source/sink
T_ambient	20	degC	Tank surface Temperature
T_start	20	degC	Start value of temperature
U	9.43...e+09	J	Internal energy of fluid
V	112.478	m3	Actual tank volume
Wb_flow	-21819.2	W	Work flow across boundaries or source term
X_start			
crossArea	50.0	m2	Area of tank
der(U)	1.80...e+07		Internal energy of fluid
der(V)	0.215339		Actual tank volume
der(level)	0.00430677	m.s-1	der(Level height of tank)
der(m)	214.95		Mass of fluid
energy...namics	1		Formulation of energy balance
fluidVolume	112.478	m3	Volume
h_start	84013.1	J/kg	Start value of specific enthalpy
heatTransfer			
height	3.0	m	Height of tank
☑ level	2.24957	m	Level height of tank
level_start	2.2	m	Start value of tank level
m	112276	kg	Mass of fluid
m_flow...ominal	1	kg/s	Nominal value for mass flow rates in ports
m_flow_small	0.01	kg/s	Regularization range at zero mass flow rate
m_flow...bulent			
massDynamics	1		Formulation of mass balance
mb_flow	214.95	kg/s	Mass flows across boundaries
medium			
nPorts	3		Number of ports
p_ambient	1.01325	bar	Tank surface pressure
p_start	1.01325	bar	Start value of pressure
portAreas			
portInDensities			
portVelocities			
portsData[1]			
portsData[2]			
portsData[3]			
ports[1]			
ports[2]			
ports[3]			
ports_E_flow			
ports_H_flow			
ports_p...tration			

Messages Browser

```
[3] 13:53:58 Translation Notification
[Modelica.Fluid.Interfaces: 202:5-202:64]: The diagnostics message for the missing inner is:
Your model is using an outer "system" component but
an inner "system" component is not defined.
For simulation drag Modelica.Fluid.System into your model
```

Ln: 1, Col: 0

Figure 13: Plotting of the simulation results showing the tank level of Pumping_System.

As illustrated in the case of the Irrigation_System model, we followed the same procedure for defining various elements of the Pumping_System that are used in the data modelling process. In order to keep the data modelling process simple, we modified only few parameters and attributes of the components constituting the simulation model of the Pumping_System (ps), as shown in Table 3. A complete list of parameters and attributes with default values is available from the Modelica Standard Library under the Fluid sub-library in the "Examples" package and "PumpingSystem" example.

Table 3: Original names (left), and modified parameters and attributes of the components constituting simulation model of the Pumping_System (ps).

Original name of component in the Modelica Standard Library	**Modified parameters and attributes of the components constituting simulation model of the Pumping_System (ps)**		
	Modified name	**Modified parameter**	**Left as default**

FixedBoundary	SOURCE_ps	p_ambient = 1.01325
		T_ambient = 20.0
			All other parameters

PrescribedPump	PUMP_ps	use_N_in = true
		N_nominal = 1200.0
		nParallel = 1.0
			All other parameters

StaticPipe	PIPE_ps	diameter = 0.3
		nParallel = 1.0
		length = 100.0
		height_ab = 50.0
			All other parameters

OpenTank	TANK_ps	height = 3.0
		corssArea = 50.0
			All other parameters

RelativePressure	TANK PRESSURE SENSOR_ps	–	All other parameters
ValveLinear	VALVE_ps	dp_nominal = 2.0	All other parameters

		m_flow_nominal = 400.0	

FixedBoundary	SINK_ps	p = 1.01325
		T = 20.0
			All other parameters

Step	VALVE OPEN-ING_ps	offset = 1.0E-6
		height = 1.0
		startTime = 200.0
			All other parameters

FirstOrder	PT1_ps	k = 1.0
		T = 2.0
			All other parameters

TriggeredTrapezoid	PUMP RPM GEN-ERATOR_ps	rising = 3.0
		amplitude = 1200.0
		falling = 3.0
		offset = 0.001
			All other parameters

OnOffController	CONTROLLER_ps	bandwidth = 4000.0
		pre_y_start = false
			All other parameters

Constant	RELATIVE PRES-SURE SET-POINT_ps	k = 20000.0	All other parameters
System	SYSTEM_ps	–	All other parameters

Similarly, the proposed names for each connector of the components constituting the simulation model of Pumping_System are shown in Table 4. As stated earlier, some components may have two or more inlet connectors (e.g. PUMP_ps), and two or more outlet connectors (e.g. TANK_ps). Also, in the data modelling process, the signal type of connectors are differentiated from the flow type connectors using the Modelica general ontology class “Connector” and its sub-classes “FlowConnector” and SingalConnector” (see Section 5.1).

Table 4: Proposed names for the connectors of different components constituting Pumping_System (ps) simulation model.

Modified component name of the “ps” simulation model	**Proposed connector name for component**	**Connector name description**
SOURCE_ps	source_ps_in	inlet connector of SOURCE_ps
	source_ps_out	outlet connector of SOURCE_ps
PUMP_ps	pump_ps_in1	inlet connector of PUMP_ps
	pump_ps_in2	inlet signal connector of PUMP_ps
	pump_ps_out	outlet connector of PUMP_ps
PIPE_ps	pipe_ps_in	inlet connector of PIPE_ps
	pipe_ps_out	outlet connector of PIPE_ps
TANK_ps	tank_ps_in	inlet connector of TANK_ps
	tank_ps_out1	outlet connector 1 of TANK_ps
	tank_ps_out2	outlet connector 2 of TANK_ps
TANK PRESSURE SENSOR_ps	tank pressure sensor_ps_in	inlet connector of TANK PRESSURE SENSOR_ps
	tank pressure sen-sor_ps_out1	outlet connector of TANK PRESSURE SENSOR_ps
	tank pressure sen-sor_ps_out2	outlet signal connector of TANK PRES-SURE SENSOR_ps

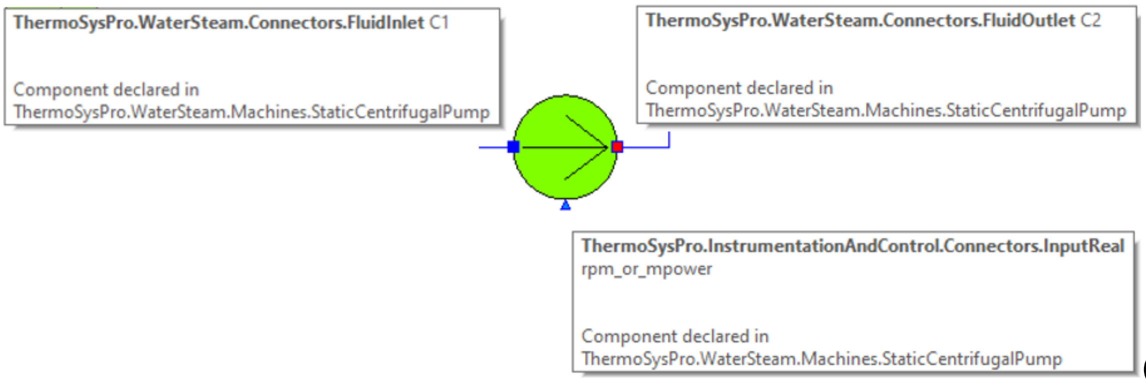
VALVE_ps	valve_ps_in1	inlet connector of VALVE_ps
	valve_ps_in2	inlet signal connector of VALVE_ps
	valve_ps_out	outlet connector of VALVE_ps
SINK_ps	sink_ps_in1	inlet connector 1 of SINK_ps
	sink_ps_in2	inlet connector 2 of SINK_ps
	sink_ps_out	outlet connector 1 of SINK_ps
VALVE OPENING_ps	valve opening_ps_in	inlet connector of VALVE OPENING_ps
	valve opening_ps_out	outlet connector of VALVE OPEN-ING_ps
PT1_ps	pt1_ps_in	inlet connector of PT1_ps
	pt1_ps_out	outlet connector of PT1_ps
PUMP RPM GENER-ATOR_ps	pump rpm generator_ps_in	inlet connector of PUMP RPM GENER-ATOR_ps
	pump rpm generator_ps_out	outlet connector of PUMP RPM GEN-ERATOR_ps
CONTROLLER_ps	controller_ps_in1	inlet connector 1 of CONTROLLER_ps
	controller_ps_in2	inlet connector 2 of CONTROLLER_ps
	controller_ps_out	outlet connector of CONTROLLER_ps
RELATIVE PRES-SURE SETPOINT_ps	relative pressure set-point_ps_in	inlet connector of RELATIVE PRES-SURE SETPOINT_ps
	relative pressure set-point_ps_out	outlet connector of RELATIVE PRES-SURE SETPOINT_ps

5. Ontology development

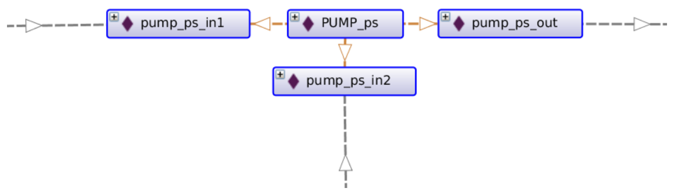
This section introduces four ontologies designed for the demonstration of OWL data modelling of the two Modelica system simulation models (i.e. Irrigation_System and Pumping_System) described in the previous Section 4. The ontologies are developed using the Protégé tool, providing the following concepts:

- The *Modelica general ontology* contains the concepts of “Model” (i.e. Irrigation_System or Pumping_System) and “Connector” (i.e. inlet/outlet and flow/signal)
- The *Modelica blocks ontology* contains concepts for the model components available in the Blocks package of the Modelica Standard Library
- The *Modelica fluid ontology* contains concepts for the model components available in the Fluid package of the Modelica Standard Library
- The *ThermoSysPro ontology* contains concepts for the model components available in the ThermoSysPro library

As mentioned earlier, every component of the ThermoSysPro library contains one or more inlet and outlet connectors that are used to connect the model components of a simulation model/system. For example, Figure 14 shows a pump component having two inlet (dark blue square and triangle) and one outlet (dark red square) connectors. Thus, from the data modelling perspective, there are 4 separate entities to be defined, i.e. pump, inlet 1, inlet 2, and outlet (see Section 5.1). In addition to the inlet and outlet connectors, it should be noted that in this study we differentiated the signal type connectors from the flow type connectors by using separate ontology classes. Table 2 and Table 4 show, respectively, the name and type of connectors for each component of the Irrigation_System and Pumping_system models used in the data modelling process.


ThermoSysPro.WaterSteam.Connectors.FluidInlet C1
Component declared in ThermoSysPro.WaterSteam.Machines.StaticCentrifugalPump
ThermoSysPro.WaterSteam.Connectors.FluidOutlet C2
Component declared in ThermoSysPro.WaterSteam.Machines.StaticCentrifugalPump
ThermoSysPro.InstrumentationAndControl.Connectors.InputReal rpm_or_mpower
Component declared in ThermoSysPro.WaterSteam.Machines.StaticCentrifugalPump

(a)



(b)

Figure 14: An example of inlet and outlet connectors of a pump component. Above (a) the Modelica ThermoSysPro library component presented in Modelica graphical notation, and below (b) the equivalent data model in OWL, visualised in the Protègè editor. In the OWL model, the individual pump_ps_in1 represent the blue rectangular inlet, pump_ps_in2 the blue triangular inlet and pump_ps_out the out red rectangular outlet connector point in the Modelica component, respectively.

5.1 Modelica general ontology

A Modelica general ontology is developed for demonstrating the data models of the Irrigation_System and Pumping_System. Figure 15 shows the classes or concepts and object properties defined in the ontology. It consists of “Modelica” as a superclass with two sub-classes “Connector” and “Model”. The “Model” class or concept of this ontology can be used to represent a Modelica model (e.g. Irrigation_System or Pumping_System). The class “Connector” contains four sub-classes and can be used to describe the connectors of each component constituting a Modelica simulation model. For example, a component having an inlet connector can be instantiated using the concept “InputConnector”, a flow type connector can be instantiated using the concept “FlowConnector”, and so forth.

Figure 15 also shows the five object properties defined in the Modelica general ontology. The property:

- “contains” can be used to describe a Modelica model, e.g. “Irrigation_System contains PIPE_irs”
- “hasConnector” can be used to describe a connector of the model component, e.g. “PIPE_irs hasConnector pipe_irs_out”
- “isConnectedTo” can be used to describe the connection between the connectors of model components, e.g. “pipe_irs_out isConnectedTo valve_irs_in”
- “isConnectorOf” is the inverse of the property “isConnectedTo”
- “isLogicallyConnectedTo” can be used to describe the connection between two model components, e.g. “PIPE_irs isLogicallyConnectedTo VALVE_irs”

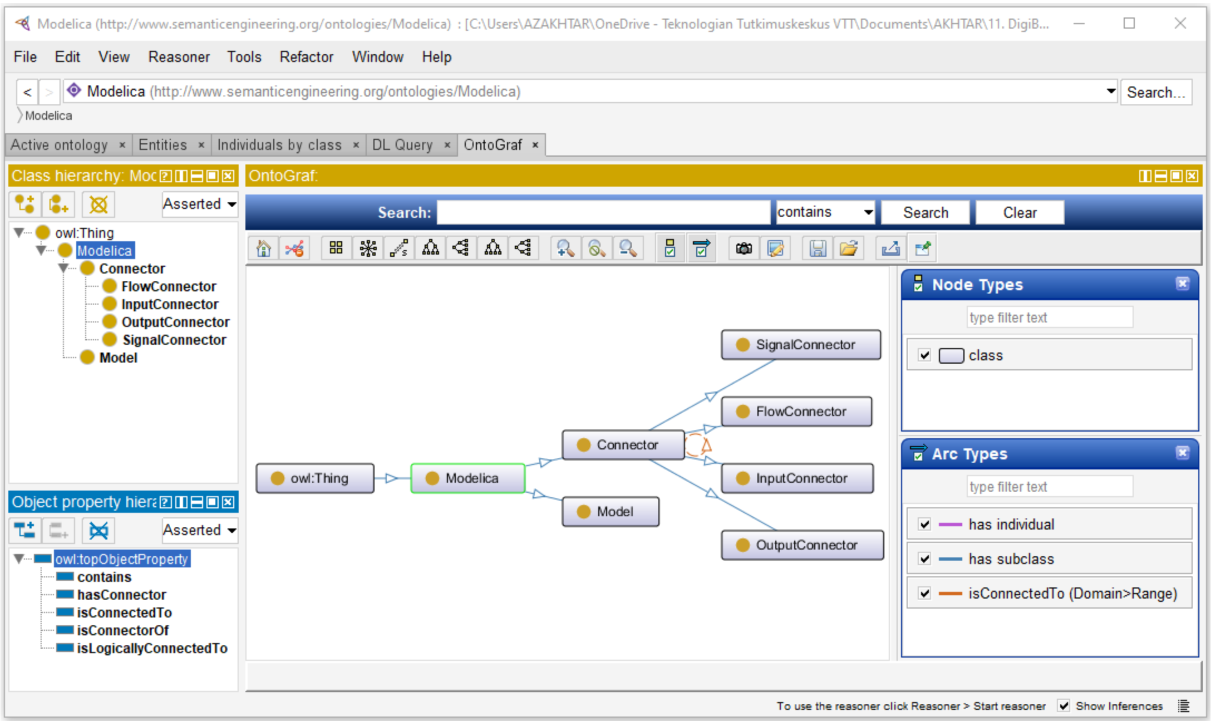
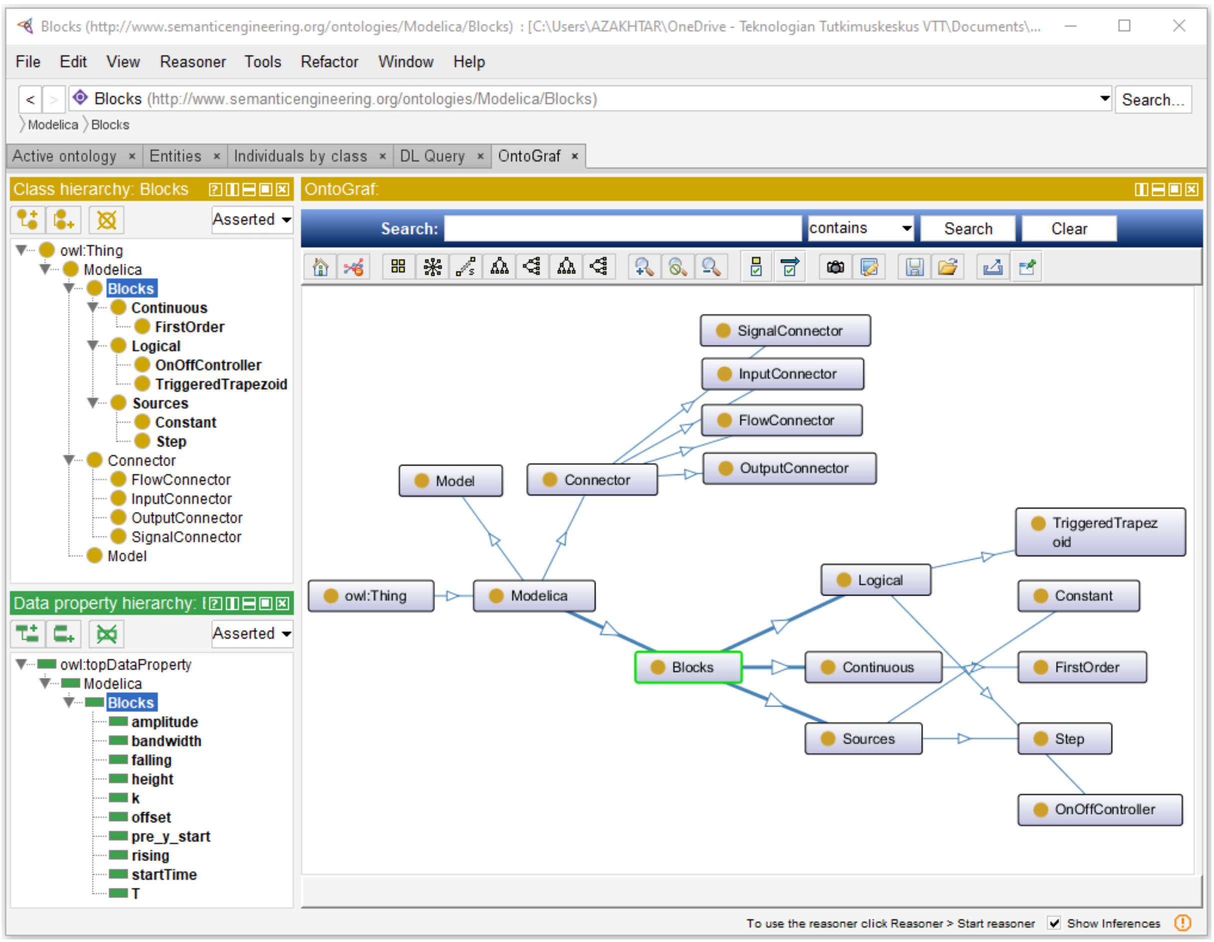


Figure 15: A Modelica general ontology, its concepts and object properties.

5.2 Modelica blocks ontology

The Modelica blocks ontology is a subset of the Modelica Standard Library and its Blocks package. The hierarchical structure of the classes/concepts of this ontology is defined in exactly the same manner as presented in the Blocks package (see Figure 11). In addition to its own concepts and parameters of the model components defined as data properties (e.g. amplitude, bandwidth, falling, height, etc.), this ontology also imports the Modelica general ontology, as shown in Figure 16.

It should be noted that the Modelica blocks ontology does not include all the concepts and all the parameters of model components available in the Blocks package. It contains only the necessary concepts and components' parameters (as data properties) that are required for the OWL data modelling of the Modelica model (i.e. Pumping_System) presented in this study.



Blocks (http://www.semanticengineering.org/ontologies/Modelica/Blocks) : [C:\Users\AZAKHTAR\OneDrive - Teknologian Tutkimuskeskus VTT\Documents\...

Class hierarchy: Blocks (Asserted)

- owl:Thing
  - Modelica
    - Blocks
      - Continuous
        - FirstOrder
      - Logical
        - OnOffController
        - TriggeredTrapezoid
      - Sources
        - Constant
        - Step
    - Connector
      - FlowConnector
      - InputConnector
      - OutputConnector
      - SignalConnector
    - Model

Data property hierarchy: (Asserted)

- owl:topDataProperty
  - Modelica
    - Blocks
      - amplitude
      - bandwidth
      - falling
      - height
      - k
      - offset
      - pre_y_start
      - rising
      - startTime
      - T

To use the reasoner click Reasoner > Start reasoner ✓ Show Inferences

Figure 16: A Modelica blocks ontology, its classes and data properties. Also, the Modelica general ontology is imported.

5.3 Modelica fluid ontology

The Modelica fluid ontology is a subset of the Modelica Standard Library and its Fluid package. This ontology is designed the same way as the Modelica blocks ontology, i.e. its classes or concepts hierarchy is defined according to the structure presented in the Fluid package (see Figure 11).

Figure 17 shows the concepts and parameters of the model components defined as data properties (e.g. crossArea, diameter, height, etc.) in the Modelica fluid ontology. Also, it imports the Modelica general ontology. Furthermore, it should be noted that this ontology does not include all the concepts and components' parameters as available in the Fluid package, but contains only the necessary concepts that are required for the OWL data modelling of the Modelica model (i.e. Pumping_System) presented in this study.

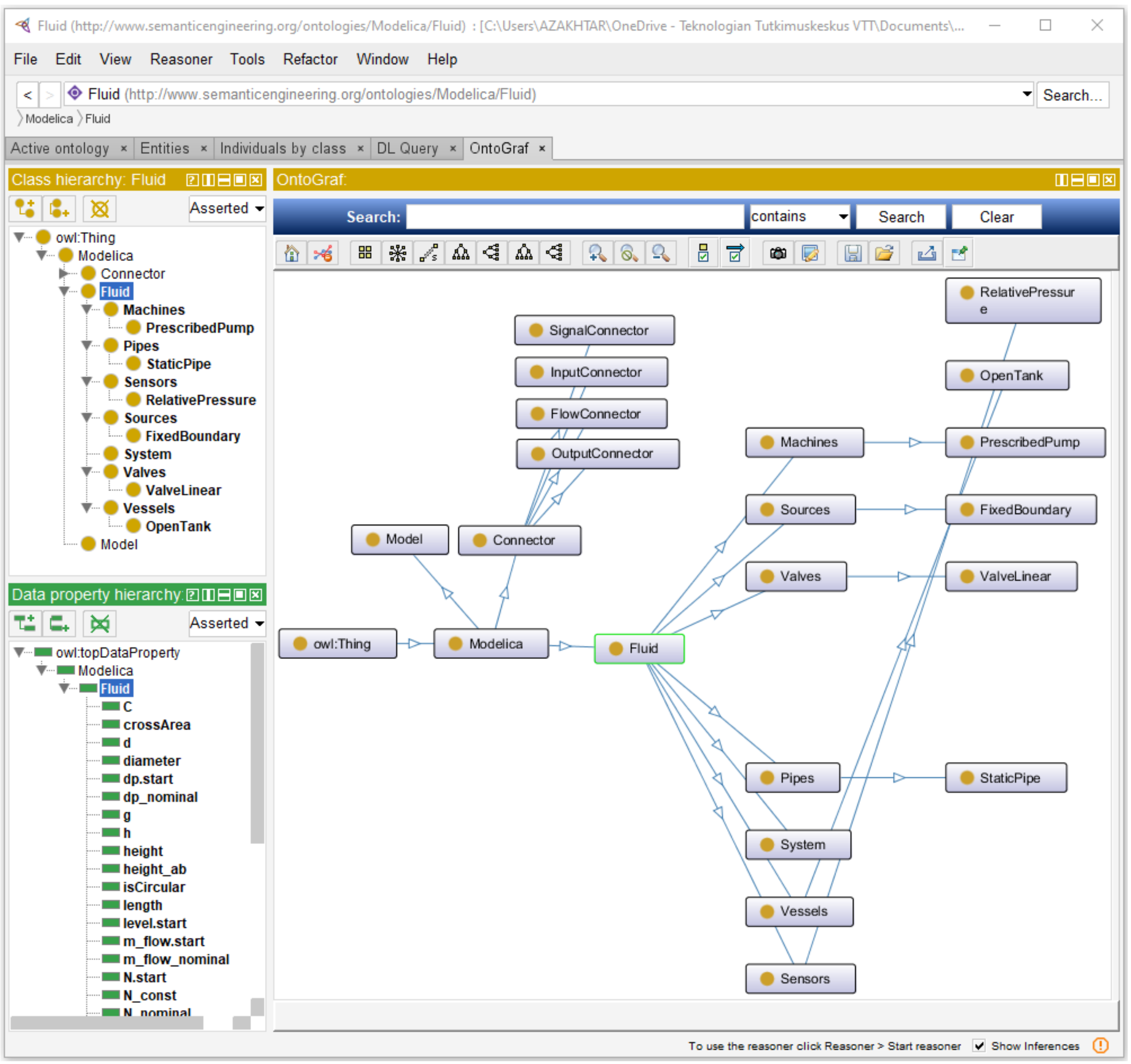


Figure 17: A Modelica fluid ontology, its classes and data properties. Also, the Modelica general ontology is imported.

5.4 ThermoSysPro ontology

The ThermoSysPro ontology is a subset of the ThermoSysPro library. It is designed the same way as the Modelica blocks and Modelica fluid ontologies, i.e. its classes or concepts hierarchy is defined according to the structure presented in the ThermoSysPro library (see Figure 8).

Figure 18 shows the concepts and parameters of the model components defined as data properties (e.g. adiabatic_compression, continuous_flow_reversal, fluid, etc.) in the ThermoSysPro ontology. Also, it imports the Modelica general ontology. Furthermore, it should be noted that this ontology does not include all the concepts and components' parameters as available in the ThermoSysPro library, but contains only the necessary concepts that are required for the OWL data modelling of the Modelica model (i.e. Irrigation_System) presented in this study.

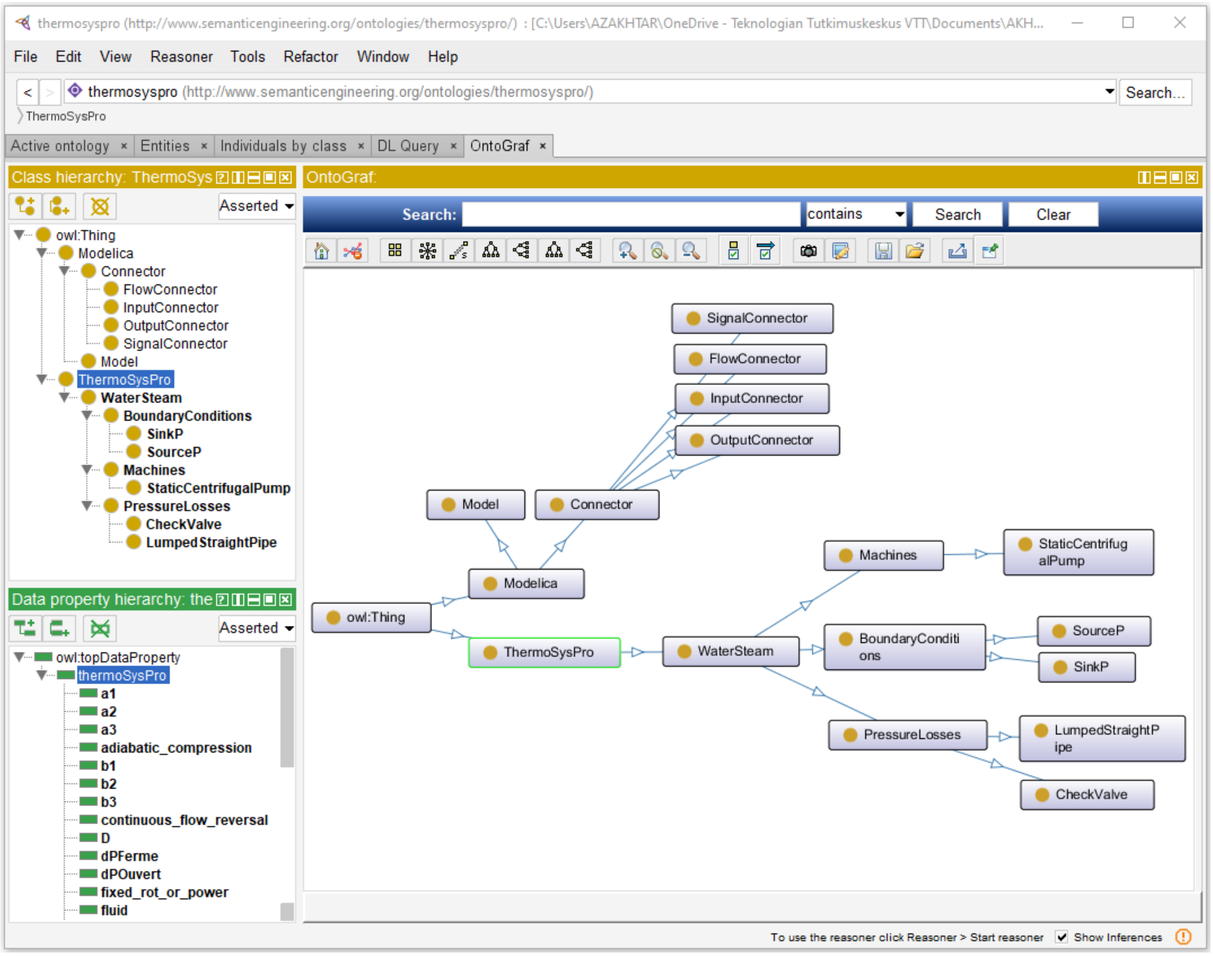


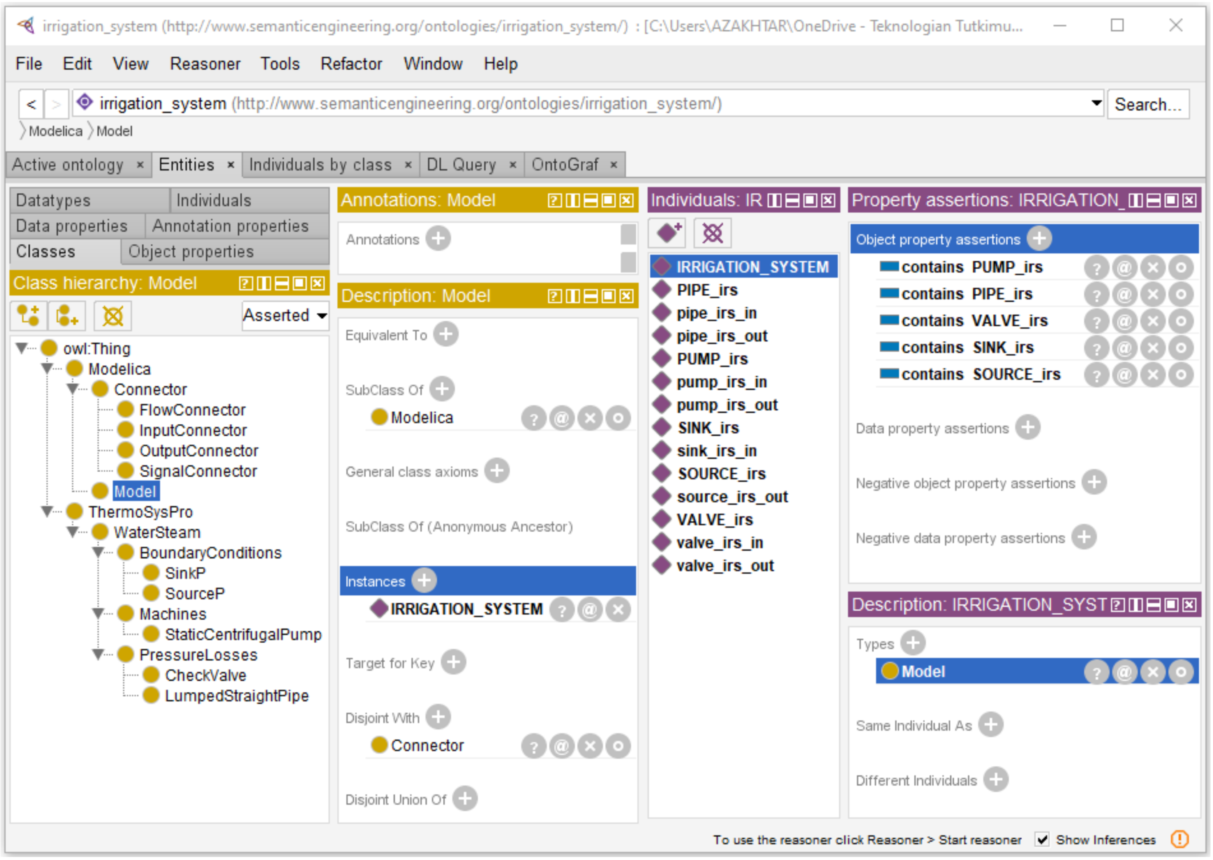
Figure 18: A ThermoSysPro ontology, its classes and data properties. Also, the Modelica general ontology is imported.

6. OWL data modelling of Modelica system simulation models

This section discusses the OWL data modelling of Irrigation_System and Pumping_System introduced in Section 4.1 and Section 4.2, respectively. As the Irrigation_System consists of components from the ThermoSysPro library only, thus its data modelling requires the ThermoSysPro ontology and the Modelica general ontology. Whereas, the Pumping_System contains components from both the Blocks package and the Fluid package of the Modelica Standard Library, and therefore its data modelling utilises the Modelica blocks ontology, Modelica fluid ontology and Modelica general ontology.

6.1 OWL data modelling of the Irrigation_System model

The OWL data model of Irrigation_System was developed using the Protégé tool. Both the Modelica general ontology and the ThermoSysPro ontology were imported, which contain the necessary classes/concepts, data properties, and object properties. As shown in Figure 19, the *IRRIGATION_SYSTEM* is instantiated using the *Model* sub-class of the *Modelica* class. The *IRRIGATION_SYSTEM* consists of various components that were defined using the data property *contains*, e.g. “IRRIGATION_SYSTEM contains PUMP_irs”, “IRRIGATION_SYSTEM contains PIPE_irs” and so forth.



irrigation_system (http://www.semanticengineering.org/ontologies/irrigation_system/) : [C:\Users\AZAKHTAR\OneDrive - Teknologian Tutkimu...

irrigation_system (http://www.semanticengineering.org/ontologies/irrigation_system/)

Modelica 〉 Model

Class hierarchy: Model

- owl:Thing
  - Modelica
    - Connector
      - FlowConnector
      - InputConnector
      - OutputConnector
      - SignalConnector
    - Model
  - ThermoSysPro
    - WaterSteam
      - BoundaryConditions
        - SinkP
        - SourceP
      - Machines
        - StaticCentrifugalPump
      - PressureLosses
        - CheckValve
        - LumpedStraightPipe

Annotations: Model

Description: Model

SubClass Of: Modelica

Instances: IRRIGATION_SYSTEM

Disjoint With: Connector

Individuals: IR

IRRIGATION_SYSTEM
PIPE_irs
pipe_irs_in
pipe_irs_out
PUMP_irs
pump_irs_in
pump_irs_out
SINK_irs
sink_irs_in
SOURCE_irs
source_irs_out
VALVE_irs
valve_irs_in
valve_irs_out

Property assertions: IRRIGATION_

Object property assertions:
contains PUMP_irs
contains PIPE_irs
contains VALVE_irs
contains SINK_irs
contains SOURCE_irs

Description: IRRIGATION_SYST

Types: Model

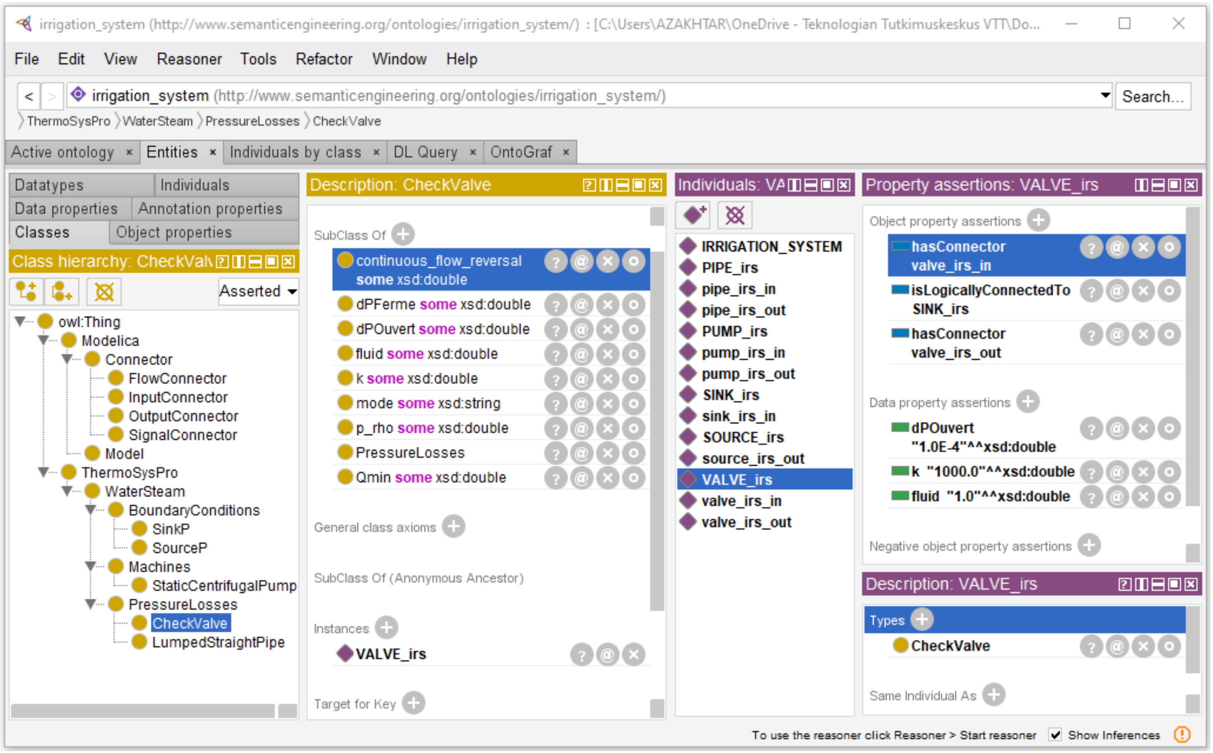
To use the reasoner click Reasoner > Start reasoner

Figure 19: OWL data model of Irrigation_System. Composition of the model.

Also, the model components were instantiated using the concepts/classes of the ThermoSysPro ontology, as well as the restriction properties for the model components were defined, as shown in Figure 20. For example, the component "VALVE_irs" was instantiated using the "CheckValve" concept, and its restriction properties were defined, such as "continuous_flow_reversal some xsd:double". In addition, the object properties were defined, such as "VALVE_irs hasConnector valve_irs_in". Furthermore, the data properties were also asserted, such as "VALVE_irs fluid 1.0".

Similarly, the connectors for each model component were defined, as shown in Figure 21. For example, the pipe outlet (pipe_irs_out) was defined as an outlet connector (OutputConnector) of flow type (FlowConnector) and is connected to (isConnectedTo) the valve inlet (valve_irs_in).

Figure 22 shows the complete OWL data model of Irrigation_System, i.e. its model components, parameters of the components, and the connectors of each component and their types. We can see that the visualisation of the complete model is inconvenient. Hence, simulation models comprised of larger number of components have complex visualisation, as we will see in the case of OWL data modelling of Pumping_System consisting of 13 components.


irrigation_system (http://www.semanticengineering.org/ontologies/irrigation_system/) : [C:\Users\AZAKHTAR\OneDrive - Teknologian Tutkimuskeskus VTT\Do...

File Edit View Reasoner Tools Refactor Window Help

irrigation_system (http://www.semanticengineering.org/ontologies/irrigation_system/)

ThermoSysPro 〉WaterSteam 〉PressureLosses 〉CheckValve

Active ontology × Entities × Individuals by class × DL Query × OntoGraf ×

Class hierarchy: CheckValve — Asserted

- owl:Thing
  - Modelica
    - Connector
      - FlowConnector
      - InputConnector
      - OutputConnector
      - SignalConnector
    - Model
  - ThermoSysPro
    - WaterSteam
      - BoundaryConditions
        - SinkP
        - SourceP
      - Machines
        - StaticCentrifugalPump
      - PressureLosses
        - CheckValve
        - LumpedStraightPipe

Description: CheckValve

SubClass Of
- continuous_flow_reversal some xsd:double
- dPFerme some xsd:double
- dPOuvert some xsd:double
- fluid some xsd:double
- k some xsd:double
- mode some xsd:string
- p_rho some xsd:double
- PressureLosses
- Qmin some xsd:double

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
- VALVE_irs

Target for Key

Individuals: VALVE_irs
- IRRIGATION_SYSTEM
- PIPE_irs
- pipe_irs_in
- pipe_irs_out
- PUMP_irs
- pump_irs_in
- pump_irs_out
- SINK_irs
- sink_irs_in
- SOURCE_irs
- source_irs_out
- VALVE_irs
- valve_irs_in
- valve_irs_out

Property assertions: VALVE_irs

Object property assertions
- hasConnector valve_irs_in
- isLogicallyConnectedTo SINK_irs
- hasConnector valve_irs_out

Data property assertions
- dPOuvert "1.0E-4"^^xsd:double
- k "1000.0"^^xsd:double
- fluid "1.0"^^xsd:double

Negative object property assertions

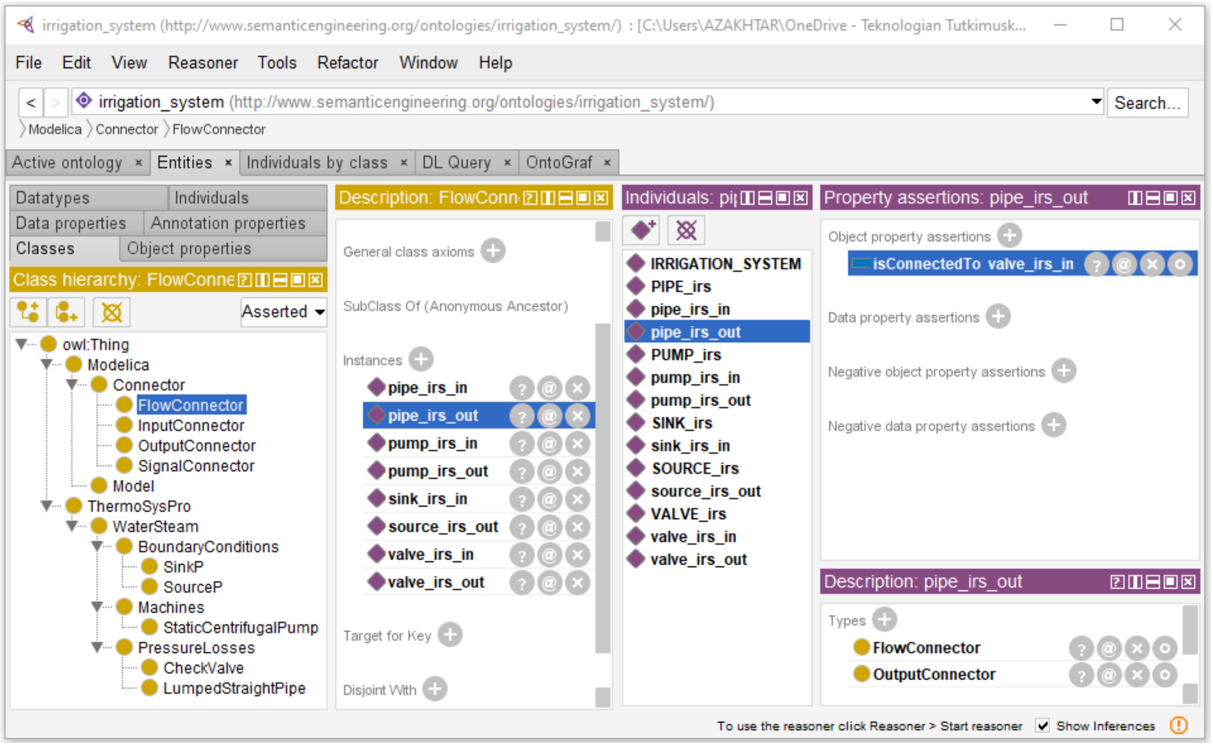
Description: VALVE_irs

Types
- CheckValve

Same Individual As

To use the reasoner click Reasoner > Start reasoner ✓ Show Inferences

Figure 20: OWL data model of Irrigation_System. Description of model component.


irrigation_system (http://www.semanticengineering.org/ontologies/irrigation_system/) : [C:\Users\AZAKHTAR\OneDrive - Teknologian Tutkimusk...

File Edit View Reasoner Tools Refactor Window Help

irrigation_system (http://www.semanticengineering.org/ontologies/irrigation_system/)

〉Modelica 〉Connector 〉FlowConnector

Active ontology × Entities × Individuals by class × DL Query × OntoGraf ×

Class hierarchy: FlowConnector — Asserted

- owl:Thing
  - Modelica
    - Connector
      - FlowConnector
      - InputConnector
      - OutputConnector
      - SignalConnector
    - Model
  - ThermoSysPro
    - WaterSteam
      - BoundaryConditions
        - SinkP
        - SourceP
      - Machines
        - StaticCentrifugalPump
      - PressureLosses
        - CheckValve
        - LumpedStraightPipe

Description: FlowConnector

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
- pipe_irs_in
- pipe_irs_out
- pump_irs_in
- pump_irs_out
- sink_irs_in
- source_irs_out
- valve_irs_in
- valve_irs_out

Target for Key

Disjoint With

Individuals: pipe_irs_out
- IRRIGATION_SYSTEM
- PIPE_irs
- pipe_irs_in
- pipe_irs_out
- PUMP_irs
- pump_irs_in
- pump_irs_out
- SINK_irs
- sink_irs_in
- SOURCE_irs
- source_irs_out
- VALVE_irs
- valve_irs_in
- valve_irs_out

Property assertions: pipe_irs_out

Object property assertions
- isConnectedTo valve_irs_in

Data property assertions

Negative object property assertions

Negative data property assertions

Description: pipe_irs_out

Types
- FlowConnector
- OutputConnector

To use the reasoner click Reasoner > Start reasoner ✓ Show Inferences

Figure 21: OWL data model of Irrigation_System. Connector type and its connection.

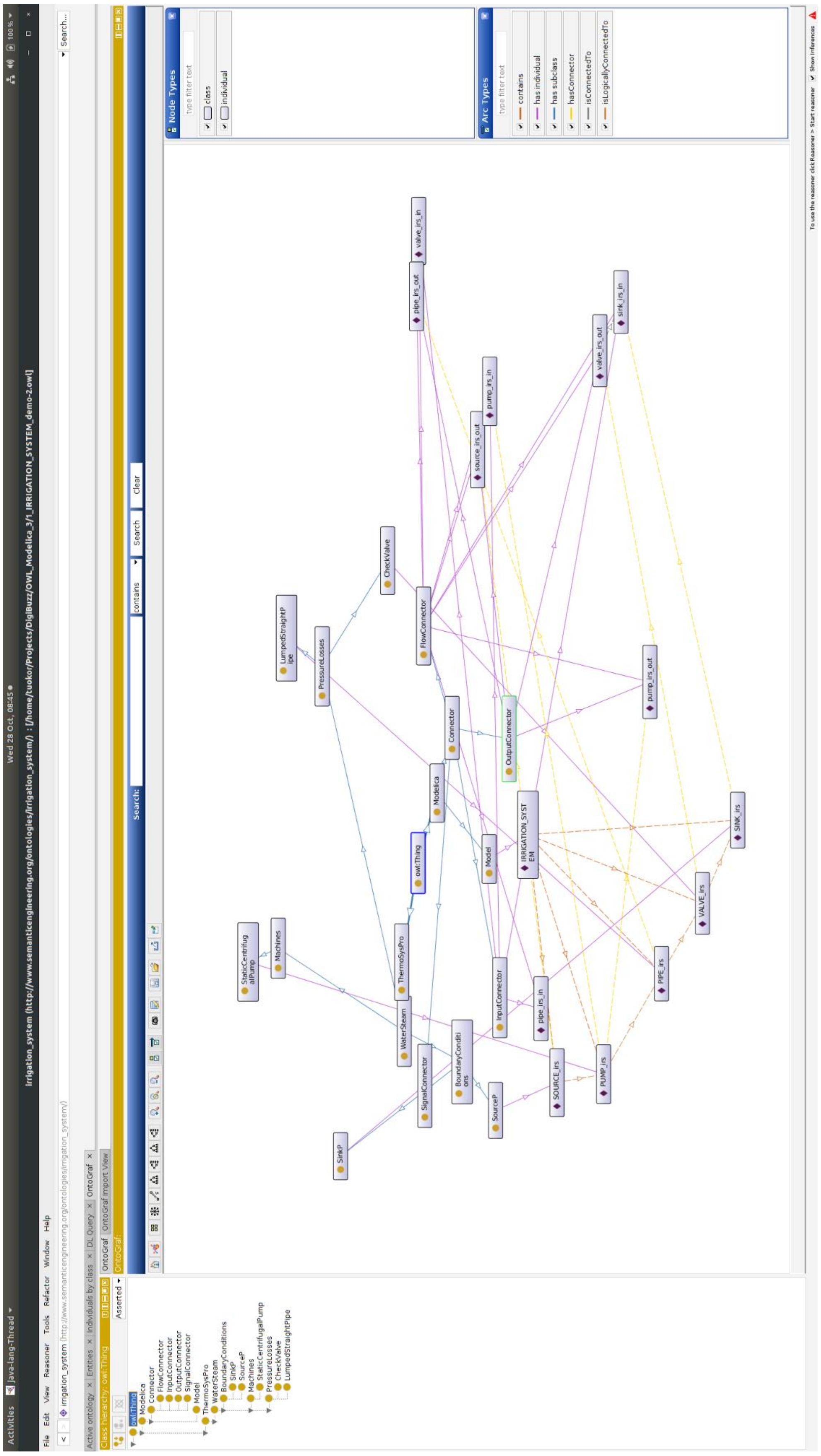
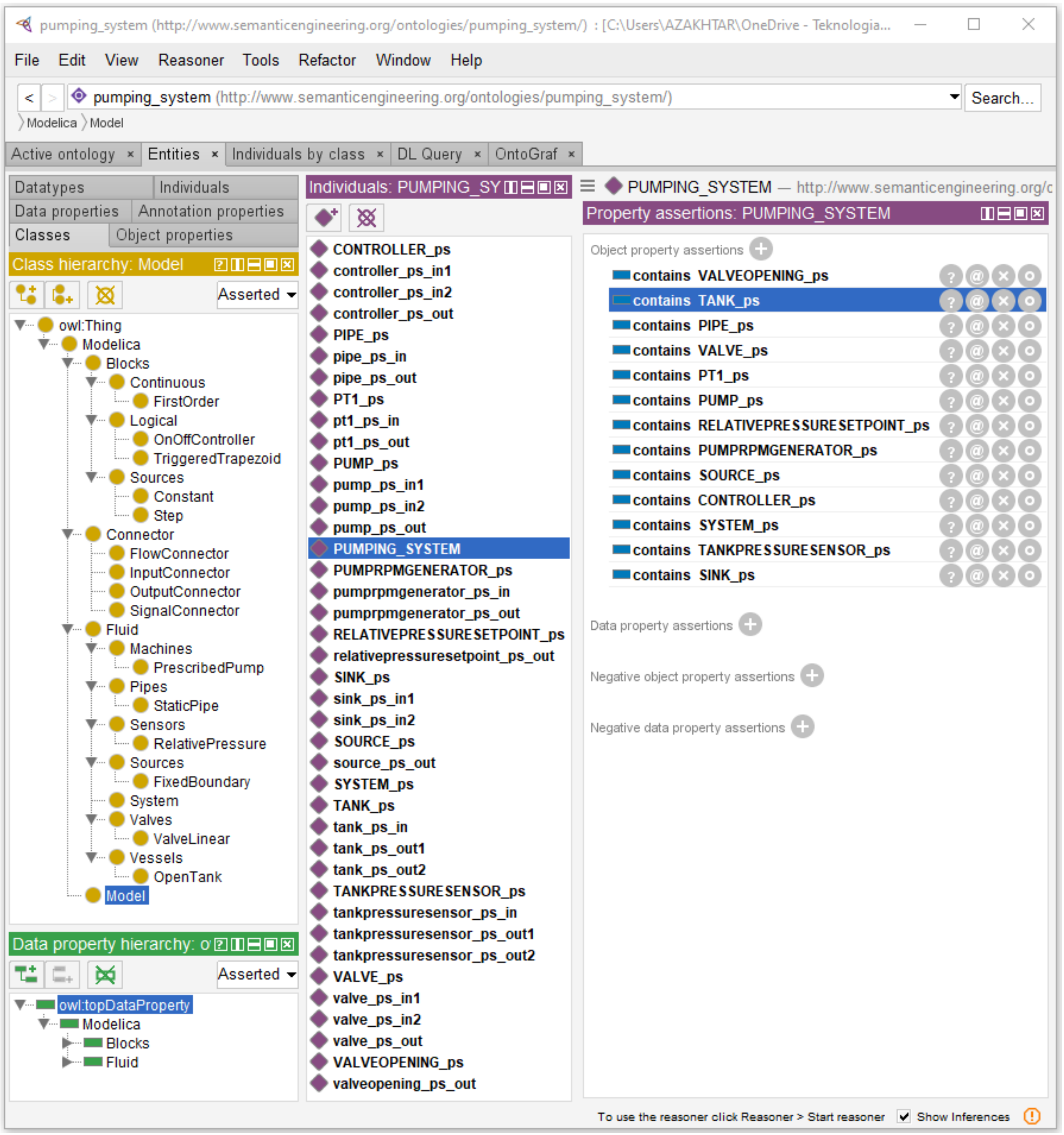


Figure 22: OWL data model of Irrigation_System. Model components, parameters of components, connectors of each components and their types.

6.2 OWL data modelling of the Pumping_System model

The OWL data model for the Pumping_System was created by following the same procedure as in the case of Irrigation_System. Basically, there is no difference between the two systems from the data modelling perspective. The only differences are the number of components in each model (i.e. 5 components in Irrigation_System, and 13 components in Pumping_System) and the ontologies needed (i.e. Modelica general and ThermoSysPro ontologies for Irrigation_System, whereas, Modelica general, Modelica fluid and Modelica Blocks ontologies for Pumping_System).

As shown in Figure 23, the Modelica general, Modelica Block, and Modelica Fluid ontologies were imported into the Protégé tool. The *PUMPING_SYSTEM* was instantiated using the class *Model*, and its components were defined using the object property *contains*.


pumping_system (http://www.semanticengineering.org/ontologies/pumping_system/) : [C:\Users\AZAKHTAR\OneDrive - Teknologia...

File Edit View Reasoner Tools Refactor Window Help

pumping_system (http://www.semanticengineering.org/ontologies/pumping_system/)

Search...

Modelica 〉 Model

Active ontology × | Entities × | Individuals by class × | DL Query × | OntoGraf ×

Datatypes | Individuals
Data properties | Annotation properties
Classes | Object properties

Class hierarchy: Model

Asserted

- owl:Thing
  - Modelica
    - Blocks
      - Continuous
        - FirstOrder
      - Logical
        - OnOffController
        - TriggeredTrapezoid
      - Sources
        - Constant
        - Step
    - Connector
      - FlowConnector
      - InputConnector
      - OutputConnector
      - SignalConnector
    - Fluid
      - Machines
        - PrescribedPump
      - Pipes
        - StaticPipe
      - Sensors
        - RelativePressure
      - Sources
        - FixedBoundary
      - System
      - Valves
        - ValveLinear
      - Vessels
        - OpenTank
    - Model

Data property hierarchy: o

Asserted

- owl:topDataProperty
  - Modelica
    - Blocks
    - Fluid

Individuals: PUMPING_SY

CONTROLLER_ps
controller_ps_in1
controller_ps_in2
controller_ps_out
PIPE_ps
pipe_ps_in
pipe_ps_out
PT1_ps
pt1_ps_in
pt1_ps_out
PUMP_ps
pump_ps_in1
pump_ps_in2
pump_ps_out
PUMPING_SYSTEM
PUMPRPMGENERATOR_ps
pumprpmgenerator_ps_in
pumprpmgenerator_ps_out
RELATIVEPRESSURESETPOINT_ps
relativepressuresetpoint_ps_out
SINK_ps
sink_ps_in1
sink_ps_in2
SOURCE_ps
source_ps_out
SYSTEM_ps
TANK_ps
tank_ps_in
tank_ps_out1
tank_ps_out2
TANKPRESSURESENSOR_ps
tankpressuresensor_ps_in
tankpressuresensor_ps_out1
tankpressuresensor_ps_out2
VALVE_ps
valve_ps_in1
valve_ps_in2
valve_ps_out
VALVEOPENING_ps
valveopening_ps_out

PUMPING_SYSTEM — http://www.semanticengineering.org/c

Property assertions: PUMPING_SYSTEM

Object property assertions

contains VALVEOPENING_ps
contains TANK_ps
contains PIPE_ps
contains VALVE_ps
contains PT1_ps
contains PUMP_ps
contains RELATIVEPRESSURESETPOINT_ps
contains PUMPRPMGENERATOR_ps
contains SOURCE_ps
contains CONTROLLER_ps
contains SYSTEM_ps
contains TANKPRESSURESENSOR_ps
contains SINK_ps

Data property assertions

Negative object property assertions

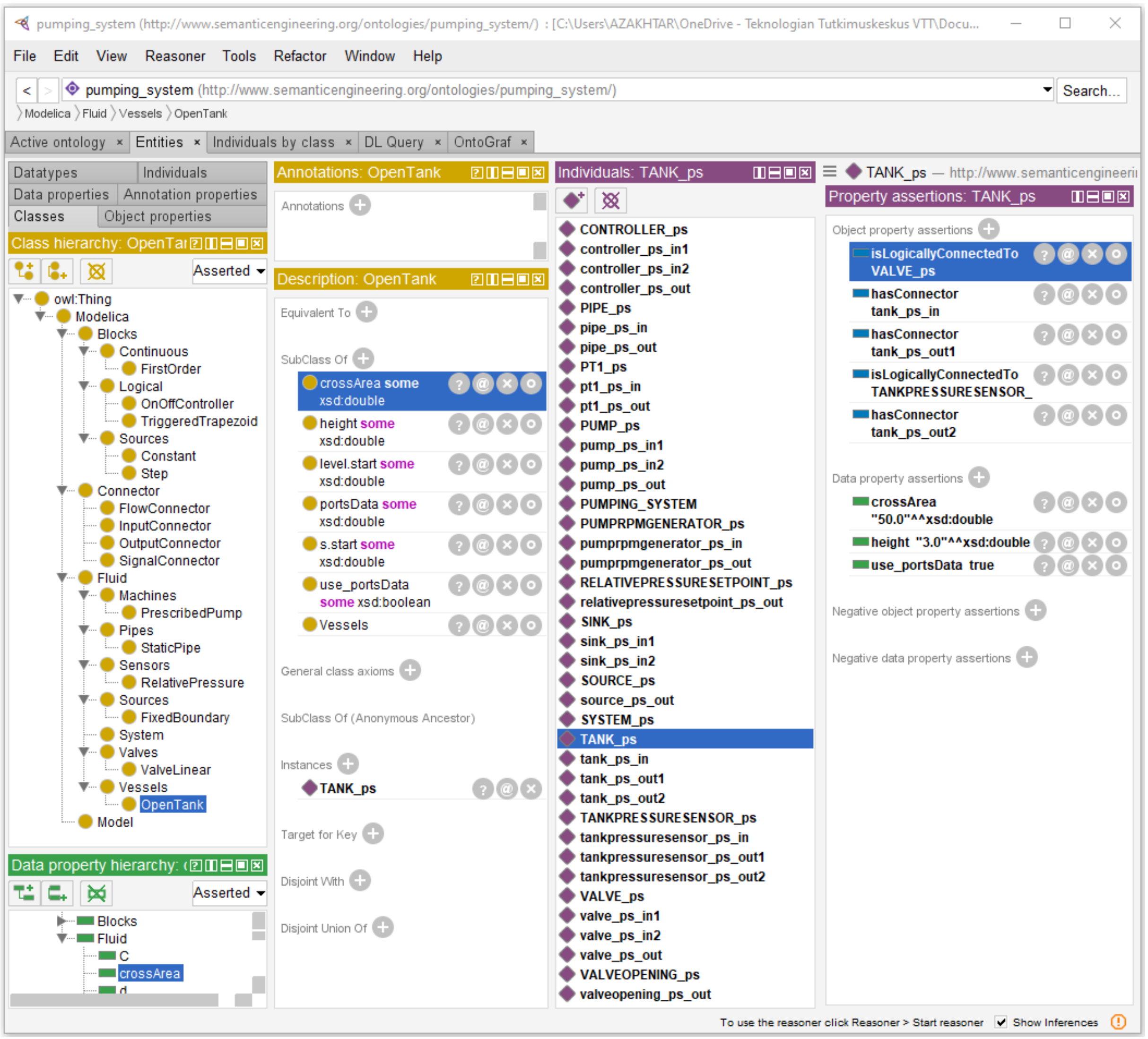
Negative data property assertions

To use the reasoner click Reasoner > Start reasoner ✓ Show Inferences

Figure 23: OWL data model of Pumping_System. Composition of the model.

Similarly, the model components (e.g. PUMP_ps, VALVE_ps, TANK_ps) were instantiated using the classes/concepts of the Modelica Blocks and Modelica Fluid ontologies. Likewise, the inlet, outlet, flow type and signal type connectors of each model components were instantiated and defined, as shown in Figure 24.

Figure 25 shows the complete OWL data model of Pumping_System. As we can see, it is quite difficult to visualise the components, instances of components, parameters of components, connectors of each components and their types.


pumping_system (http://www.semanticengineering.org/ontologies/pumping_system/) : [C:\Users\AZAKHTAR\OneDrive - Teknologian Tutkimuskeskus VTT\Docu...
File
Edit
View
Reasoner
Tools
Refactor
Window
Help
pumping_system (http://www.semanticengineering.org/ontologies/pumping_system/)
Search...
Modelica 〉Fluid 〉Vessels 〉OpenTank
Active ontology
Entities
Individuals by class
DL Query
OntoGraf
Datatypes
Individuals
Data properties
Annotation properties
Classes
Object properties
Class hierarchy: OpenTa
Asserted
Annotations: OpenTank
Description: OpenTank
Individuals: TANK_ps
TANK_ps — http://www.semanticengineeri
Property assertions: TANK_ps
Data property hierarchy:
Asserted
To use the reasoner click Reasoner > Start reasoner
Show Inferences

Class hierarchy:
owl:Thing
- Modelica
  - Blocks
    - Continuous
      - FirstOrder
    - Logical
      - OnOffController
      - TriggeredTrapezoid
    - Sources
      - Constant
      - Step
  - Connector
    - FlowConnector
    - InputConnector
    - OutputConnector
    - SignalConnector
  - Fluid
    - Machines
      - PrescribedPump
    - Pipes
      - StaticPipe
    - Sensors
      - RelativePressure
    - Sources
      - FixedBoundary
    - System
    - Valves
      - ValveLinear
    - Vessels
      - OpenTank
  - Model

Data property hierarchy:
- Blocks
- Fluid
  - C
  - crossArea
  - d

Annotations

Equivalent To

SubClass Of
- crossArea some xsd:double
- height some xsd:double
- level.start some xsd:double
- portsData some xsd:double
- s.start some xsd:double
- use_portsData some xsd:boolean
- Vessels

General class axioms

SubClass Of (Anonymous Ancestor)

Instances
- TANK_ps

Target for Key

Disjoint With

Disjoint Union Of

Individuals:
CONTROLLER_ps
controller_ps_in1
controller_ps_in2
controller_ps_out
PIPE_ps
pipe_ps_in
pipe_ps_out
PT1_ps
pt1_ps_in
pt1_ps_out
PUMP_ps
pump_ps_in1
pump_ps_in2
pump_ps_out
PUMPING_SYSTEM
PUMPRPMGENERATOR_ps
pumprpmgenerator_ps_in
pumprpmgenerator_ps_out
RELATIVEPRESSURESETPOINT_ps
relativepressuresetpoint_ps_out
SINK_ps
sink_ps_in1
sink_ps_in2
SOURCE_ps
source_ps_out
SYSTEM_ps
TANK_ps
tank_ps_in
tank_ps_out1
tank_ps_out2
TANKPRESSURESENSOR_ps
tankpressuresensor_ps_in
tankpressuresensor_ps_out1
tankpressuresensor_ps_out2
VALVE_ps
valve_ps_in1
valve_ps_in2
valve_ps_out
VALVEOPENING_ps
valveopening_ps_out

Object property assertions
- isLogicallyConnectedTo VALVE_ps
- hasConnector tank_ps_in
- hasConnector tank_ps_out1
- isLogicallyConnectedTo TANKPRESSURESENSOR_
- hasConnector tank_ps_out2

Data property assertions
- crossArea "50.0"^^xsd:double
- height "3.0"^^xsd:double
- use_portsData true

Negative object property assertions

Negative data property assertions

Figure 24: OWL data model of Pumping_System. Description of model component.

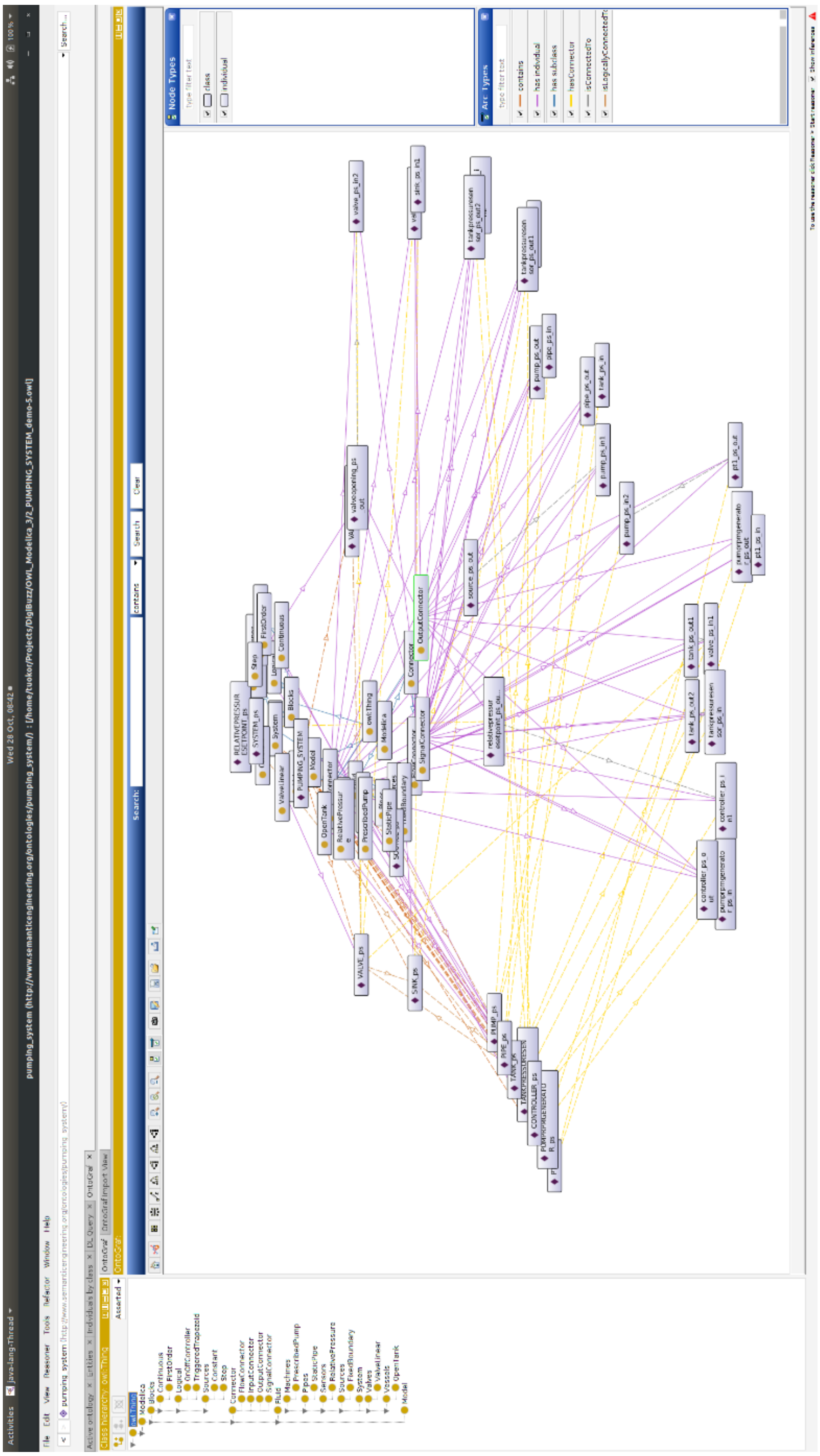


Figure 25: OWL data model of the Pumping_System simulation model, including model components, parameters of components, connectors of each components and their type.

7. Discussion and conclusions

This study focuses on the data management aspect of simulation-based digital twins that are expected to run for a couple of decades. During such a long time period, there is a possibility that the underlying computer infrastructure (software, computer hardware and operating system) may change or advance to the level where it would be impossible to open and run the previously developed computer models. The problem is especially associated with the application software used for developing the simulation models of digital twins. Most of the commercial modelling and simulation software use proprietary data models and store models in vendor-specific formats. The data model of one software can hardly be related to the data model of another software and, in the worst-case, their simulation model formats are black boxes that do not provide information about the mathematics and physics involved in the simulation process. For this reason, there is a need to preserve enough information about the simulation models of digital twins in formats that can be easily processed with tools other than the ones used for their creation.

To illustrate the preservation of simulation-based digital twins' data for longer periods, two Modelica system simulation models were presented. Modelica is an open-source language for modelling of complex systems and provides detailed information about the models of the components. The models were built in the OMEdit environment (a free tool for Modelica). Four OWL ontologies were created that provide the necessary concepts and relationships for describing the simulation models. Using the Protégé ontology editor, the OWL data models of the corresponding models were developed. The resulting OWL data models, containing information about the simulation models, can be used for preserving information of the simulation models in detail and, together with the Modelica libraries and Modelica Specification, they provide the needed basis for reproducing the simulation models and the simulation solver application, if required.

During the study, it was noticed that the visualisation of the complete data model gets complicated as the size of the simulation model grows (i.e. a model with more components is difficult to visualise in the Protégé tool). This is illustrated by a visualisation of a data model representing a relatively simple simulation model in Figure 25. The value of visualising the data graphs comes with the ability to browse the model, study the connections and neighbouring components and relations between the components. Another finding was that the OWL language has its own limitations and is not the best tool for general data representation in diverse applications. Therefore, the future work should be focused on the feasibility of other data modelling methods, languages, and tools for preserving simulation models' data. Especially domain-specific modelling languages, such as UML profiles, and available technologies and tools for using them would be worth investigating. In the applications of simulation model data representation, data transformation from one representation to another is required and is often challenging due to complex semantics and information structures.

References

[1] Ansys, “Ansys Fluent: Fluid Simulation Software,” 2020. https://www.ansys.com/products/fluids/ansys-fluent (accessed Nov. 04, 2020).

[2] Abaqus, “Abaqus Unified FEA — SIMULIA^{TM} by Dassault Systèmes®,” 2020. https://www.3ds.com/products-services/simulia/products/abaqus/ (accessed Nov. 04, 2020).

[3] Modelica, “The Modelica Association,” 2020. https://www.modelica.org/ (accessed Jul. 24, 2020).

[4] Modelica Association, “Modelica Libraries,” 2020. https://www.modelica.org/libraries (accessed Jul. 24, 2020).

[5] Modelica Standard Library, “Modelica.UsersGuide.Overview.” https://doc.modelica.org/om/Modelica.UsersGuide.Overview.html (accessed Oct. 30, 2020).

[6] Modelica Association, “Modelica Language Documents,” 2020. https://www.modelica.org/documents (accessed Jul. 24, 2020).

[7] Modelica Association, “Modelica Tools,” 2020. https://www.modelica.org/tools (accessed Jul. 24, 2020).

[8] OpenModelica, “Introduction,” 2020. https://www.openmodelica.org/ (accessed Jul. 24, 2020).

[9] P. Fritzson *et al.*, “The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development,” *Model. Identif. Control A Nor. Res. Bull.*, vol. 41, no. 4, pp. 241–295, 2020, doi: 10.4173/mic.2020.4.1.

[10] OpenModelica, “OMEdit,” 2020. https://openmodelica.org/?id=92:omeditintegration&catid=10:main-category (accessed Jul. 24, 2020).

[11] W3C, “Web Ontology Language (OWL),” 2013. https://www.w3.org/OWL/ (accessed Jul. 24, 2020).

[12] W3C, “Semantic Web — Main Page,” 2019. https://www.w3.org/2001/sw/wiki/Main_Page (accessed Nov. 04, 2020).

[13] J. Kortelainen, “Semantic Data Model for Multibody System Modelling,” 2011.

[14] Protégé, “Protégé — A free, open-source ontology editor and framework for building intelligent systems,” 2020. https://protege.stanford.edu/ (accessed Jul. 24, 2020).

[15] Modelica Association, “Modelica Language,” 2020. https://www.modelica.org/modelicalanguage (accessed Jul. 24, 2020).

[16] Modelica Association, “The Modelica Association,” 2020. https://www.modelica.org/ (accessed Jul. 24, 2020).

[17] W3C, “Resource Description Framework (RDF),” 2014. https://www.w3.org/RDF/ (accessed Nov. 04, 2020).

[18] W3C, “SPARQL 1.1 Overview,” 2013. https://www.w3.org/TR/sparql11-overview/ (accessed Nov. 04, 2020).

[19] W3C, “SWRL: A Semantic Web Rule Language Combining OWL and RuleML,” 2004. https://www.w3.org/Submission/SWRL/ (accessed Nov. 04, 2020).

[20] ThermoSysPro, “Information — ThermoSysPro version 3.2,” 2020. https://build.openmodelica.org/Documentation/ThermoSysPro.html (accessed Oct. 30, 2020).

[21] B. El Hefni, D. Bouskela, and G. Lebreton, “Dynamic modelling of a combined cycle power plant with ThermoSysPro.”

[22] B. El Hefni and D. Bouskela, *Modeling and Simulation of Thermal Power Plants with ThermoSysPro*. Springer International Publishing, 2019.