Automated Design and Model Generation for a District Heating Network from OpenStreetMap Data

Marcus Fuchs[1], Dirk Müller[1]
[1] RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Aachen, Germany

Abstract

Despite the context of fast-changing energy systems that require flexibility and dynamic operation, state-of-the-art district heating models often rely on static analyses. To contribute towards advancing the capabilities of district heating design for current and future challenges, this paper presents a framework for automating the generation of static as well as dynamic district heating system models. To illustrate the application of this framework, we present a case study in which the developed framework automatically designs a district heating network for 250 buildings using data input from OpenStreetMap and generates a static system model that is simulated in Python as well as a dynamic system model that is simulated in Modelica.

Introduction

Many energy systems in urban environments are currently facing challenges that result from recent changes in energy markets, technologies, regulatory frameworks and expectations of society. One of the major drivers for these changes is the goal of decarbonizing the energy supply chain. The substitution of large central fossil-fired power plants by more distributed and renewable energy sources leads to a more complex system architecture and a more dynamic system operation. This makes managing energy flows to match a fluctuating supply with varying demand patterns a major task. At the same time, market competition and customer expectations create economic pressures to utilize storage capacities, generation flexibility, waste heat and synergies between different energy subsystems to operate energy systems not only efficiently, but also profitably. To address the challenges of this emerging paradigm in energy system design and operation, new requirements and boundary conditions also challenge traditional methods for planning system design and operation. At the same time, new technologies leading to increased computational capabilities as well as advanced programming and modeling frameworks offer potential for the development of novel tools for system planning and operation control that focus specifically on the new challenges.

For district heating, such new challenges could include the simulation of a more dynamic operation. On the one hand, district heating networks are often supplied by combined heat and power units. Through that link to the electric grid, they could serve as aggregators of buildings' demand side management potential as well as for their thermal energy storage potential. On the other hand, increasingly competetive energy markets create stronger incentives for optimized operation of district heating networks, which may be facitilited by more flexible and dynamic operation patterns. While steady-state thermal network models are still often used and considered adequate to predict the thermal performance of district heating systems (see e.g. (Wang et al., 2016; Liu et al., 2016)), they may not be suited to adequately represent all of the transient effects of such dynamic operation patterns.

One approach to address these novel challenges for district heating modeling that has recently gained interest is to model district heating systems using the modeling language Modelica. As district heating models largely depend on the representation of the pipe networks, modeling the pipe component has been a main focus of these developments. In one example, del Hoyo Arce et al. (2015) describe the development of the *DH Systems* model library with a focus on simplified medium and a buried pipe model. This pipe model uses a finite volume approach with simulation results compared for 1, 10, and 100 nodes per pipe, respectively. Tugores et al. (2015) also use a discretized pipe model for integrated modeling of heat generation, storage, distribution and building demands for a district combining district heating and an aquifer storage system. In the optimization model presented by Runvik et al. (2015), the pipe network of a district heating system is modeled with a fixed time delay and a heat loss model. As an alternative to discretization of pipe models or fixing the time delay for fluid transport in the network, Giraud et al. (2015) utilize the `spatialDistribution` operator recently introduced to the Modelica language. This approach allows to model a plug flow, which offers potential to more accurately model temperature wave propagations through the network with reduced computational effort. In a similar way, the participants of IEA Annex 60 have also started an initiative to include a plug flow pipe model for district heating simulation in the open-source *Annex 60 Library* (Wetter et al., 2015).

In addition to current advances in district heating modeling, Modelica's clear-text model format and high re-usability of component models also has the advantage of facilitating automated model generation. This is especially beneficial for use cases with large numbers of individual component models that assemble a system model, as is the case for district heating systems. As an early example of using Modelica for automated model generation for the application of modeling state machines, Dressler (2004) presents a method to automatically generate Modelica code from JGrafchart state machine models. Another case of automating the generation of Modelica state machine models is described by Pohlmann et al. (2014). They present a method to generate Modelica code from a markup language called MechatronicUML. Closer to energy system modeling, Thorade et al. (2015) present a tool chain to automatically generate Modelica model code from Building Information Model (BIM) input. In this process, the actual Modelica code generation is done by a Python tool named *CoTeTo* supporting the templating via the packages *Mako* (Bayer, 2016) and *Jinja2* (Ronacher, 2016). This approach is similar to that of the software tool *TEASER*, a Python package for automated generation of building models based on German guideline VDI 6007-1.

Combining the advantages of Modelica as a modeling language to describe the dynamic behavior of district heating networks and as a target language for automated model generation, this paper presents a graph framework written in Python to represent district heating networks in a model-neutral graph format and automate the generation of dynamic system models with Modelica code output. As a benchmark for comparison, we also implemented a static district heating model based on the same system graph to analyze the effects of dynamic system modeling compared to state-of-the-art static district heating models. In this context, the presented methods for automated model generation based on model templates with Modelica as a target modeling language offer potential for significant reductions of manual modeling effort.

To further reduce manual efforts, the presented methods make use of the freely available crowd-sourced data of OpenStreetMap (Haklay and Weber, 2008). So far, some of the main research topics regarding OpenStreetMap have been the data quality of OpenStreetMap itself (see e.g. (Haklay, 2010; Fan et al., 2014)), the creation of 3D city models from OpenStreetMap data (see e.g. (Over et al., 2010; Goetz, 2013)), and transport-related analyses (see e.g. (Huber and Rust, 2016; Fan et al., 2016)). To the best of the authors' knowledge, this paper is one of the first studies to utilize OpenStreetMap data in the field of urban energy systems modeling.

In the following section, we present the graph framework, followed by a description of the methods for automated model generation applied to district heating modeling. Subsequently, we present verification results for both models and describe a use case with a generic district heating network for 250 buildings automatically generated based on OpenStreetMap data of a town in Germany. This is followed by a presentation of simulation results with both modeling approaches before the final section for drawing the conclusions from the presented work.

Graph Framework uesgraphs

The work on district heating modeling presented in this paper is part of a broader effort to develop tools for urban energy systems analyses and optimization. In order to translate the structure of an urban energy system with buildings and various energy networks into a system model and make it accessible for analyses and model generation, a graph representation is a widely used modeling approach. In such a graph model, nodes can be used to represent the entities formed by specific subsystems or components such as individual buildings or network junctions, while edges can account for the network connections between these entities.

In order to allow for an integral representation of the urban energy system meeting a variety of energy service demands, we developed the Python package uesgraphs, for which we plan an open-source release in the future. While the main application of this package within the scope of this paper will concentrate on the thermal part of the urban energy system including the buildings and district heating networks, the developed graph framework aims at enabling a common representation of the entire urban energy system including networks like the electric grid as well as energy carrier grids like those for natural gas.

Even though a comprehensive graph model for urban energy systems should include a variety of different networks for integral analyses, it may nevertheless be useful to allow for the extraction of subgraphs that represent individual networks or a limited selection of specific networks and/or buildings. Such a flexible approach combines the possibilities for integral analyses, representation of links between different network types as well as focused analyses of a subgraph without the overhead of processing unused subsystems. Ideally, it should be possible to combine and split different subgraphs dynamically according to the requirements of specific analyses.

A further requirement for the capabilities of the graph model is the need to assign data to individual nodes, edges, subgraphs, and the entire graph. In addition, there should be fast and robust methods to interact with the graphs, e.g. to add or remove elements, traverse the graph, find shortest paths, and evaluate degrees of connectedness. These requirements are similar to those of graph models for other domains, such as the analyses of social networks or of mapping rules in software development. More specific to the application of a graph model to urban energy systems is the need to reference the position of nodes and edges to a coordinate system. While in many other applications the graphical representation of a graph is independent of given node positions and can be chosen freely according to the specific goals of the visualization, in urban energy systems it is often useful to map the positions of the nodes according to given coordinates of building positions. Thus, a graph model of an urban energy system should allow for the storage, processing and visualization of such node positions.

With several widely used implementations of general graph methods available in various programming languages, the chosen approach for uesgraphs was to extend an existing graph implementation towards a framework suited to the specific requirements of urban energy system modeling rather than to develop basic functions for graph modeling from scratch. One well tested graph implementation in the programing language Python is *networkX*. Choosing a Python implementation has the advantages of an object-oriented, open-source environment that is widely used and highly accessible to automate workflows in energy system analyses and to interface with models and result files from the modeling language Modelica. To manage node positions and perform geometric operations and analyses, the Python package *shapely* is used, as it provides better performance than other comparable packages especially for larger number of nodes.

Automated Model Generation

As it is one of the key objectives of the *uesgraphs* framework to support analyses of urban energy systems with a high degree of workflow automation to reduce manual effort, the package includes the `uesgenerator` module to assist users with quickly generating example networks. On the one hand, readily providing use cases for the package makes it more easily accessible. On the other hand, since detailed information on existing energy networks is usually not freely available to researchers and practitioners seeking to analyze and demonstrate urban energy management concepts, the capability to quickly generate use cases may also be a useful feature for further research. To generate such a use case, any given district or city with enough information on OpenStreetMap can be imported to a `UESGraph` instance. This `UESGraph` can then be used by a `UESGenerator` instance to add a supply node and automatically design a heating network for a given number of buildings. For the scope of this paper, `UESGenerator` does not aim at fully replacing the necessary engineering work to plan energy networks but is rather a proof of concept to quickly generate heating network use cases.

As a foundation for designing a heating network, the `UESGenerator` class needs a `UESGraph` instance containing building nodes and a subgraph representing the area's street network. This graph data can be imported from OpenStreetMap, but the algorithm also works for graphs created manually or from data generated by e.g. a Geographic Information System (GIS). In addition, the user should provide the position of a supply node, assumptions for the maximum heat demands of the buildings, and design temperatures for the supply and return of the heating network as well as for the specific pressure drop in the pipe network in Pa/m. Finally, the user can specify the number of buildings to be connected to the new heating network and a probability for buildings to connect to the network (connection probability p_{con}). If set to $p_{con} = 1$, all the buildings nearest to the supply will be connected to the network until the specified number of buildings to be connected is reached. For probabilities smaller than 1 each building will randomly choose to be connected to the network with the given probability.

The automated addition of a heating network takes two major steps. First, the network topology is created by adding edges that connect the chosen buildings to the supply node following the street layout. The street layout is chosen as the environment for possible pipe connections, as in many urban areas, this is the only available pathway to install a pipe network for district heating. In a separate second step, this network topology is used to estimate pressure losses in the network at the heat demand's peak load and size the pipe diameters accordingly. Dividing the algorithm into these two parts is done with the prospect that the topology generation may also be useful for other kinds of networks. While e.g. electric grids may require different methods for dimensioning of the power lines, it may be useful to utilize and enhance `UESGenerator`'s topology generation algorithm to quickly generate network use cases.

Based on the graph description of a district heating network in *uesgraphs* (either created with the *uesgenerator* methods described above or in some other way), we implemented further Python packages to automatically generate district heating network models ready for simulation. On the one hand, we created a simple implementation of a static district heating network model including its solver for simulation in Python to have a verified benchmark. On the other hand, we utilize the graph representation of the network in Python and the Python templating package *Mako* to generate valid Modelica code representing the district heating network in a dynamic system model. This model can then be simulated in a Modelica tool, for which we use the Software Dymola.

Static District Heating Model

In many cases, the main goal of modeling district heating and cooling networks is to calculate flow rates, pressures and temperatures across all nodes and edges of the network. To calculate the unknown values, the supply temperatures and pressure heads at the supply nodes, and the flow rates and return temperatures at the demand nodes are usually given. In many cases, the hydronic problem is decomposed into two separate models, one for the hydraulic circuit and one for the thermal behavior of the system, respectively (Liu et al., 2016). For the static model, we implemented the common approach of the Newton method as described by Larock et al. (2000) to solve the hydraulic network problem.

To calculate flow rates and pressures in a pipe network with the Newton method, Larock et al. (2000) suggest two sets of equations. The set of *Q-equations* is used to calculate flow rates for all pipes, i.e. for all edges in the network graph. Based on these flow rates, the pressure drops across all pipes and subsequently the pressures at all nodes in the graph can be calculated using the set of *H-equations.* Both sets of equations can be used in matrix form to formulate linear systems of equations that can be efficiently solved. For the *Q-equations*, the implementation in our Python package *dhcstatic* follows this approach, using the package *numpy* (van der Walt et al., 2011) to solve the system of equations. As in *dhcstatic* the resulting flows for all edges from the *Q-Equation* calculations are already available in a graph, *dhcstatic* uses a recursive graph traversal method instead of the *H-equations* in matrix form.

The hydraulic calculation is performed before the thermal calculation, so that the temperatures can be arrived at based on the previously calculated flow rates and the known pipe properties. To solve the thermal problem, Liu (2014) describes a method for the thermal modeling that calculates the temperatures at all nodes in the network by solving a linear system of equations in matrix form, similar to the calculation of flows with the set of *Q-equations* according to Larock et al. (2000). The model in *dhcstatic* applies the same basic equations to calculate the temperatures at all nodes as Liu (2014). This is a common approach, modeling the heat loss from fluid to ambient as an exponential decay as a function of the time the fluid takes to flow through the pipe and in dependence of the pipe's thermal characteristics. The basic temperature drop due to heat loss in the pipe is described as

$$T_{\mathrm{out}} = (T_{\mathrm{in}} - T_{\mathrm{amb}}) \cdot \mathrm{e}^{-\frac{\gamma L}{c_{\mathrm{p}} \dot{m}}} + T_{\mathrm{amb}} \tag{1}$$

where T_{in} and T_{out} are the pipe's inlet and outlet temperatures in K, T_{amb} is the ambient temperature of the pipe's surroundings in K, γ is the pipe's overall heat transfer coefficient per unit length in W/(m·K), c_{p} is the medium's specific heat capacity in J/(kg·K), and $\dot{m}$ is the mass flow rate in the pipe in m/s. In this case, the overall heat transfer coefficient per unit length γ can be calculated by

$$\gamma = U \cdot 2 \cdot \pi \cdot r \tag{2}$$

where U is the overall heat transfer coefficient of the pipe in W/(m^2·K), r is the pipe radius and thus $2\pi r$ is the pipe's circumference in m.

With this approach, the outlet temperature of a pipe can be calculated straightforwardly, if the mass flow rate and the inlet temperature are known. If this pipe is the only incoming connection to a node, T_{out} will also be this node's temperature T_{node}. If more than one pipe transport an incoming flow to the node, the node temperature can be calculated as the mixing temperature of all incoming flows IF. Under the assumption of perfect mixing, the node temperature can be calculated with (3).

$$T_{\mathrm{node}} = \frac{\sum_j^{IF} (\dot{m}_{\mathrm{in},j} T_{\mathrm{in},j})}{\sum_j^{IF} \dot{m}_{\mathrm{in},j}} \tag{3}$$

Given the network structure, mass flow rates in all pipes, and the temperatures at the supply nodes, (1) and (3) are sufficient to calculate the temperatures at all nodes. As mentioned above, this can be done using a single system of equations in matrix notation. Yet, this requires to formulate any incoming temperatures $T_{\mathrm{in},j}$ for a given node in dependence of the known supply node temperatures. In larger networks with flow paths of serial pipes and multiple, possibly nested loops, this leads to rather complicated matrix entries and causes overhead in keeping track of all these flow paths. Thus, as with the pressure calculation outlined above, *dhcstatic* again uses a recursive graph traversal method instead of matrix notation to calculate the temperatures.

Dynamic District Heating Model

In contrast to the static district heating model of *dhcstatic*, which directly implements all model equations and the algorithms for solving the simulation in Python, we use the modeling language Modelica to define component models of supplies, demand stations, pipes and network junctions. The Python methods bundled in the package *uesmodels* then process the network's *uesgraph* description to generate Modelica code for a system model assembled by these component models. This model can then be used inside of a Modelica simulation environment to simulate the dynamic system behavior.

The approach followed by *uesmodels* aims at enabling a full representation of the system model in the form of a graph in Python that can be directly written to Modelica code with a clear one-to-one mapping of graph nodes to component models and graph edges

to Modelica `connect` equations. To arrive at this one-to-one mapping, the original abstracted system graph from *uesgraphs* needs to be manipulated with regard to its structure of nodes and edges. In its description of district heating systems, *uesgraphs* follows the common convention to represent supplies, buildings and junctions as nodes while pipe connections are represented as graph edges. This representation works well for graph analysis, visualization by graph drawing and for applications like in the `UESGenerator` and `DHCStatic` classes.
In contrast, the target design of the Modelica system model consists of the component models. The ports of these models are connected by `connect` equations of the form

```
connect(modelOne.port_b, modelTwo.port_a);
```

Thus, the graph representation of the model structure should use supplies, buildings, pipes and junctions as nodes while the edges should contain information on which of the models' ports should be connected. To manage the different kinds of nodes, the `UESModel` class inherits nodelists for supplies and buildings as well as for junctions (i.e. network nodes) from `UESGraph` and adds a new nodelist for pipe nodes. Furthermore, the `UESModel` class reads the graph representation of a heating network from an `UESGraph` instance with its `import_from_uesgraph()` method and converts it to the new format. During this conversion, the method places a pipe node in the middle between the two previously connected nodes, copies all attribute data about the pipe form the original edge of the `UESGraph` to the new pipe node in the `UESModel`, and creates new edges to build the graph connection through the newly introduced pipe model.
After import and conversion of the graph structure, the method `UESModel.set_connections()` can be called to automatically assign the new edges with attributes about the specific ports they connect in the models. To this end, the method iterates over all edges and classifies the connected nodes by node type. For each node type, a corresponding Modelica model and information about its ports can be assigned. For pipes, as the model is designed to allow for flow reversal, the connections to `port_a` and `port_b` can be assigned randomly, as long as it is done in a consistent way.
For the individual component models, the implementation for a given use case can be chosen by utilizing different model libraries and adjusted templates for the model generation. For this paper, we will focus on the dynamic behavior of the distribution network. Therefore, the supply and demand nodes are each modeled in a basic model including ideal sinks and sources. The demand nodes' sinks prescribe a given mass flow rate for each timestep while the supply node's source prescribes the fluid properties at feed-in for supply pressure and temperature. For the pipe model, we utilize a model from the open source *Annex 60 Library*. This pipe model's structure is shown

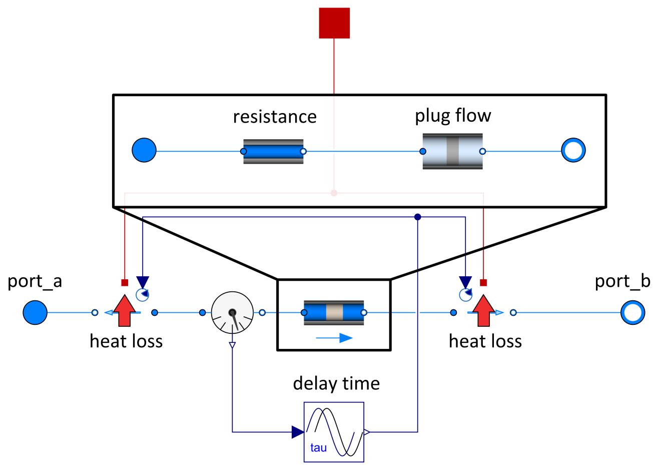

Figure 1: Structure of the pipe model from the Annnex 60 Library

in Fig. 1. It utilizes submodels for the hydraulic resistance and a plug flow implementation accounting for temperature wave propagations. This plug flow model applies a variable delay in dependence of the current flow velocites to the fluid properties including its enthalpy. Thus, temperature waves in the fluid are propageted at the speed of the flowing medium. As the flow parameters can not be manipulated between the start and end point of the plug flow, the models uses two submodels for the heat loss. These models apply a heat loss calculation to the fluid in design direction and let the fluid pass without any changes against their desing direction. This ensures that the heat loss is calculated using the actual outlet temperature of the pipe and not the incoming temperature before being processed by the delaying plug flow model.

Model Verification

In order to verify the model implementations of the district heating modeling packages *dhcstatic* and *uesmodels* for static conditions, we model test networks given by Larock et al. (2000) and Liu (2014) and compare the results with given values in the references. Fig. 2 shows a network given in Larock et al. (2000) with 2 supply nodes, 3 demand nodes and a central pipe loop. With the inclusion of multiple supplies and a pipe loop, this example is a suitable test to verify the developed models work in a wide range of network setups. While our implementations are usually based on the metric units system, for this comparison, flow rates for the comparison will be given in ft^3/s as used by Larock et al. (2000).
Table 1 compares the results reached with *dhcstatic* with the reference results. When rounded to the 4 decimal points that the reference in Larock et al. provides, the results are the same for both models. To arrive at this solution, *dhcstatic* takes 5 iterations. Based on these volume flow rates, *dhcstatic* goes on to calculate the pressure losses through all pipes and

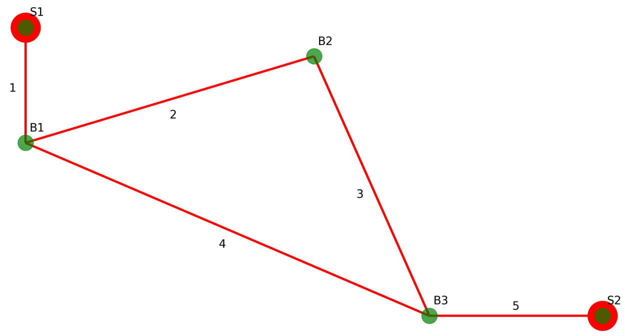

Figure 2: UESGraph of verification example from Fig. 4.24 in Larock et al. (2000)

thus the pressures at all nodes. As in this example the pressures at both supply nodes are given, there are 3 unknowns for the pressure levels at the 3 demand nodes. Traversing the graph while calculating the pressure drops, this results in the values summarized in Table 2. For the comparisons, the reference values of Larock et al. originally given as pressure head in ft have been converted to pressure in Pa.

Table 1: Comparison of dhcstatic volume flow results with reference from Larock et al. (2000) for hydraulic example.

Pipe	$\dot{V}_{\text{dhcstatic}}$ in ft^3/s	$\dot{V}_{\text{Larock}}$ in ft^3/s
1	2.11906	2.1191
2	1.05825	1.0583
3	0.44174	0.4417
4	0.06080	0.0608
5	1.18094	1.1809

Table 2: Comparison of dhcstatic pressure results with reference from Larock et al. (2000) for hydraulic example.

Demand node	$p_{\text{dhcstatic}}$ in Pa	p_{Larock} in Pa
B1	201452.0	201450.8
B2	169453.9	169453.5
B3	200615.1	200612.4

Like Larock et al. (2000) gave a detailed example network by which it is possible to verify the implementation, Liu (2014) presents a simple network for calculating the temperatures at all nodes as a result of temperature drops across the pipe edges. Fig. 3 shows the example network consisting of one supply (S1) and two buildings (B1 and B2) and three pipes (1 - 3). In Liu's work, the pipe lengths are given with 400 m each, the supply temperature at S1 is 100 °C and the thermal resistances per unit length are 5 (m·K)/W for each pipe. The given mass flow rates are 3 kg/s for pipe 1, 1 kg/s for pipe 2 and 2 kg/s for pipe 3. This leads to a flow from B1 to B2 through pipe 2,

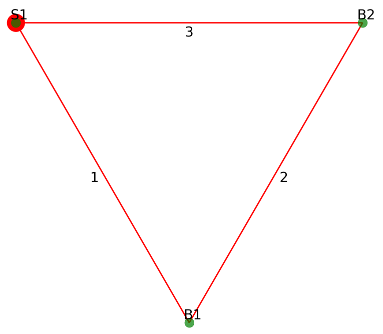

Figure 3: Visualization of the reference network from Liu (2014)

with mixing of flows at B2. Passing the `UESGraph` with this network layout and data to `DHCStatic`, the thermal model can be simulated for the one given time step by calling `DHCStatic.calc_temperatures()`. As described above, this method traverses the graph and assigns temperatures to all nodes by calculating the temperature drops over each edge. Table 3 shows the temperatures at B1 and B2 calculated with *dhcstatic* and compares them to the reference values given in Liu (2014). For B1, there is no significant difference between the model results and for B2 the difference is 0.001 K for the given reference accuracy.

Table 3: Comparison of dhcstatic temperature results with reference from Liu (2014) for thermal example

Demand node	$T_{\text{dhcstatic}}$ in °C	T_{Liu} in °C
B1	99.4283	99.428
B2	98.6742	98.673

Based on this comparison between *dhcstatic* and the reference results, we conclude that this static district heating model has been implemented correctly with only minor deviations to the reference results. When comparing the results from the *dhcstatic* simulation in Python with the Modelica model generated with *uesmodels* for the test network of Larock et al., the largest relative differences in mass flow rates are 0.4 % and 0.12 %, respectively. All other deviations are in a range between 0.01 % and 0.06 %. The supply temperatures arriving at the demand nodes in this case differ by 0.01 - 0.02 K between both models. The differences in calculated supply pressures from the pressure losses in the pipe path leading to the demand nodes vary between 44 and 70 Pa for the demand nodes B1 and B3 which are located nearer to the supplies, and around 230 Pa for B2 which is located farther away from both supply nodes. Taking into account the fundamentally different simulation environments, the deviations between the two models can be considered acceptable. In addition, this example illustrates how the automated work flow for model generation and simulation in the Python and

Modelica environments can be utilized for model comparisons based on a common graph representation of a network.

District Heating Use Case

To demonstrate the presented developments in a use case, we generate a heating network based on OpenStreetMap data of the town of Röttenbach in Germany. In order to include all of the administrative boundaries of the town, we exported the OpenStreetMap data for the area between 49.6486 and 49.6880 latitude, and 10.8678 and 10.9751 longitude. After filtering all elements to ensure they are located within the administrative boundary of the town, the resulting *UESGraph* contains 1590 building nodes and 948 street nodes with around 34 km of street network. While the buildings are only represented as a single point node in the graph, the building outline information from OpenStreetMap is used to calculate each building's ground area and assign this information as a node attribute. For the graph of Röttenbach, all buildings' total ground area calculated in this way sums up to around 320,000 m^2, with individual building ground areas varying between 8 m^2 and 23,900 m^2. If we assume that we can neglect all buildings with an area less than 50 m^2 for energy analyses and remove those nodes from the *UESGraph*, we are left with 1558 building nodes.

In this simplified example, we assign each building with the same heat demand profile. To arrive at this profile, we simulated a generic example building using a reduced order building model from the Modelica library AixLib (http://github.com/RWTH-EBC/AixLib) based on German guideline VDI 6007-1. To parametrize the model, we utilized the open-source Python package *TEASER* (http://github.com/RWTH-EBC/TEASER) that enriches a minimum required data set describing year of construction, usage type and size of a building with statistical data to provide a full input data set for the building model mentioned above.

Specifying the number of buildings to be connected to the network to 250 and setting the connection probability to 0.5, a run of the *UESGenerator* algorithm adds a heating network to the *UESGraph* with a total pipe length of about 14 km. At this stage, the graph representation contains building nodes, street nodes and district heating network nodes as well as street and heating network edges in one common graph representation. For the following analysis we extract the heating network to a separate *UESGraph*. Fig. 4 shows a plot of the generated network that displays the assigned design diameters of the pipe network by varying the line thickness of the edges. In this figure, a thicker red line indicates a larger pipe diameter. This example illustrates how the algorithm for sizing the hydronic network creates main supply lines of larger diameter branching into smaller diameters to approximate the design criterion of a specific pressure loss per unit length for all pipes.

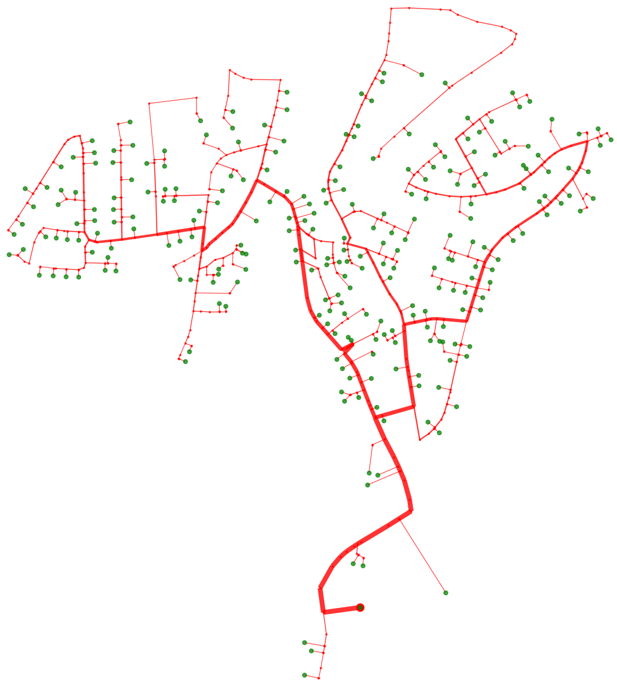
Figure 4: Visualization of the reference network from Liu (2014)

Results

The graph representation of the network described in the previous section can be directly used for a static simulation in Python using *dhcstatic* as well as for generating a dynamic system model in Modelica using *uesmodels*. Both simulations calculate the mass flow rates, pressures and temperatures in the entire network. While the static model considers every time step as a separate quasi-static calculation with instant propagation of temperature change information through the entire network, the dynamic model integrates each timestep's results from the previous state of the system. In this model, temperature change information is propagated in the form of temperature waves traveling through the network at the current flow velocity.

Fig. 5 visualizes the simulated supply temperatures in the network for one timestep. Under the considered flow conditions, the main temperature drops caused by heat losses from the network to the environment occurr in the line far from the supply node. The line thickness in Fig. 5 scales with the simulated mass flow rate for each pipe, showing the flow in the main supply line being fed in at the supply node and then branching off through the network to the 250 connected buildings. While most buildings receive a supply temperature close to the temperature at feed-in, the temperature drop for the farthest buildings over

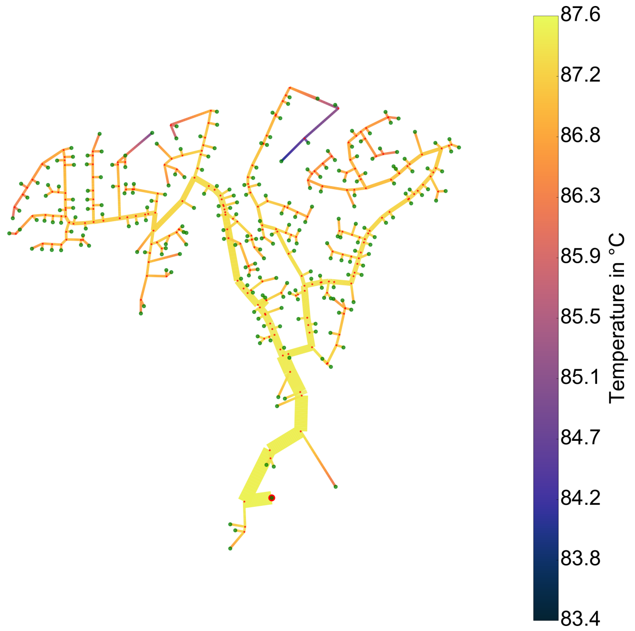

Figure 5: Supply temperatures in the network

the pipe network is around 4 K.

To investigate the differences between the static and the dynamic model, we introduce a dynamic system operation by controlling the supply temperature to vary in dependence on the outdoor air temperature. We set the supply temperature to 100 °C for outdoor air temperatures below -20 °C and a supply temperature to 80 °C for outdoor air temperatures above 20 °C. A linear interpolation between these extreme values results in a linear heating curve for the set supply temperature. Fig. 6 shows the effects of this varying supply temperature as simulated with both models. In the static model of *dhcstatic*, changes at the supply node immediately influence this timesteps temperatures at all places in the network, which only depend on the instant heat losses calculated along the flow paths. In the Modelica model, temperatures are calculated to propagate at flow velocity, leading to a time delay until a temperature wave reaches a given point in the network.

In Fig. 6, the comparison shows the supply temperature reaching the building farthest from the supply node. In this case, temperature waves take 2-3 hours to travel through the entire network. Another effect visible in this comparison is a slight damping of temperature changes in the Modelica model. This is mainly caused by differences in the heat loss calculation. While the static model uses the current flow velocity to estimate the time the fluid spends in the pipe, the dynamic model uses the integral of the flow velocity to better account for the actual travel time of a fluid parcel through the pipe. Thus, high frequency changes get damped in the dynamic model, possibly better representing the behavior of real pipes.

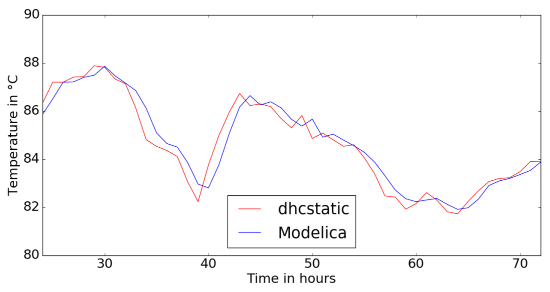

Figure 6: Comparison of simulated supply temperature for building farthest from supply

Conclusion

This paper presents developments to automate the generation of district heating models in Python and Modelica. As demonstrated by the presented use case, the underlying graph framework *uesgraphs* can be utilized to import building positions and street layouts from freely available OpenStreetMap data. With implemented algorithms in the *UESGenerator* module, users can quickly generate district heating use cases described as graph instances in *uesgraphs*. This can be be used as a foundation to generate district heating models. We implemented a static district heating model for direct simulation in Python and compared this benchmark with a dynamic district heating model in Modelica.

Two model verification examples showed that both models yield plausible results that agree well with reference values from literature for the standard case of static district heating calculation. In the use case for a network of 250 buildings, the comparison of both models showed different behavior regarding the propagation of temperature waves through the network. In addition, the different heat loss calculations in both models lead to a slight damping of high frequency dynamics in the dynamic model. Overall, the results suggest that the dynamic model can better account for the behavior of the network under dynamic operation conditions.

In addition, the use case illustrated how the presented workflow automations can facilitate the generation of use cases and direct model comparisons based on a model-neutral graph representation of the system. This approach offers potential for further model comparisons and model integration in an urban energy system analysis context. Ongoing research for efficient modeling approaches of district heating networks in Modelica could help to enhance the presented modeling approach to account for more detail including supply units and building substation behavior. This can lead to valuable insights on how district heating networks should be operated in the context of changing urban energy systems.

References

Bayer, M. (2016). Mako Templates for Python. http://www.makotemplates.org/. Accessed: 2016-09-30.

del Hoyo Arce, I., S. L. Perez, S. H. López, and I. M. Dávila (2015, sep). Lessons learnt from network modelling of a low heat density district heating system. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linkoping University Electronic Press.

Dressler, I. (2004). *Code Generation from JGrafchart to Modelica*. Ph. D. thesis, Lund Institute of Technology, Lund.

Fan, H., B. Yang, A. Zipf, and A. Rousell (2016). A polygon-based approach for matching openstreetmap road networks with regional transit authority data. *International Journal of Geographical Information Science 30*(4), 748–764.

Fan, H., A. Zipf, Q. Fu, and P. Neis (2014). Quality assessment for building footprints data on OpenStreetMap. *International Journal of Geographical Information Science 28*(4), 700–719.

Giraud, L., R. Baviere, M. Vallée, and C. Paulus (2015, sep). Presentation, validation and application of the DistrictHeating modelica library. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linkoping University Electronic Press.

Goetz, M. (2013). Towards generating highly detailed 3D CityGML models from OpenStreetMap. *International Journal of Geographical Information Science 27*(5), 845–865.

Haklay, M. (2010). How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. *Environment and planning B: Planning and design 37*(4), 682–703.

Haklay, M. and P. Weber (2008). OpenStreetMap: User-generated street maps. *IEEE Pervasive Computing 7*(4), 12–18.

Huber, S. and C. Rust (2016). osrmtime: calculate travel time and distance with OpenStreetMap data using the Open Source Routing Machine (OSRM). *Stata Journal 16*(2), 416–423.

Larock, B. E., R. W. Jeppson, and G. Z. Watters (2000). *Hydraulics of pipeline systems*. Boca Raton, FL: CRC Press.

Liu, X. (2014). *Combined Analysis of Electricity and Heat Networks*. PhD, Cardiff University, Cardiff.

Liu, X., J. Wu, N. Jenkins, and A. Bagdanavicius (2016). Combined analysis of electricity and heat networks. *Applied Energy 162*, 1238–1250.

Over, M., A. Schilling, S. Neubauer, and A. Zipf (2010). Generating web-based 3D City Models from OpenStreetMap: The current situation in Germany. *Computers, Environment and Urban Systems 34*(6), 496–507.

Pohlmann, U., J. Holtmann, M. Meyer, and C. Gerking (2014). Generating Modelica Models from Software Specifications for the Simulation of Cyber-Physical Systems. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 191–198.

Ronacher, A. (2016). Jinja2 (The Python Template Engine). http://jinja.pocoo.org/. Accessed: 2016-09-30.

Runvik, H., P.-O. Larsson, S. Velut, J. Funkquist, M. Bohlin, A. Nilsson, and S. M. Razavi (2015, sep). Production planning for distributed district heating networks with JModelica.org. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linkoping University Electronic Press.

Thorade, M., J. Rädler, P. Remmen, T. Maile, R. Wimmer, J. Cao, M. Lauster, C. Nytsch-Geusen, D. Müller, and C. van Treeck (2015, sep). An open toolchain for generating modelica code from building information models. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linkoping University Electronic Press.

Tugores, C. R., H. Francke, F. Cudok, A. Inderfurth, S. Kranz, and C. Nytsch-Geusen (2015, sep). Coupled modeling of a district heating system with aquifer thermal energy storage and absorption heat transformer. In *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. Linkoping University Electronic Press.

van der Walt, S., S. C. Colbert, and G. Varoquaux (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering 13*(2), 22–30.

Wang, J., Z. Zhou, and J. Zhao (2016, jul). A method for the steady-state thermal simulation of district heating systems and model parameters calibration. *Energy Conversion and Management 120*, 294–305.

Wetter, M., M. Fuchs, P. Grozman, L. Helsen, F. Jorissen, M. Lauster, D. Müller, C. Nytsch-Geussen, D. Picard, P. Sahlin, and M. Thorade (2015, dec). IEA EBC annex 60 modelica library an international collaboration to develop a free open-source model library for buildings and community energy systems. In *Proceedings of BS2015: 14th Conference of International Building Performance Simulation Association, Hyderabad, India, Dec. 7-9, 2015*. International Building Performance Simulation Association.