Fault detection and diagnosis in building energy systems: A tool chain for the automated generation of training data

Christine van Stiphoudt, Florian Stinner, Gerrit Bode, Alexander Kümpel, Dirk Müller
RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Mathieustraße 10, 52074 Aachen, Germany
E-mail: van.stiphoudt.christine@rwth-aachen.de

Abstract. The application of fault detection and diagnosis (FDD) algorithms in building energy management systems (BEMS) has great potential to increase the efficiency of building energy systems (BES). The usage of supervised learning algorithms requires time series depicting both nominal and component faulty behaviour for their training. In this paper, we introduce a method that automates Modelica code extension of BES models in Python with fault models to approximate real component faults. The application shows two orders of magnitude faster implementation compared to manual modelling, while no errors occur in the connections between fault and component models.

1. Introduction
With 36% of the final energy consumption worldwide in 2017 [1], the building sector shows potential for reduction of greenhouse gas emissions. Building energy systems (BES) are one domain for improvement. To ensure a preferably optimal and energy-efficient operation, computer-aided systems are used. Building energy management systems (BEMS) monitor and control the energy-related technical equipment of buildings. The increasing complexity of these systems and their devices makes them susceptible to faults [2]. Studies show that faults in BES can increase the respective energy consumption by up to 50% [2, 3, 4] depending on the type of building and energy system, the way they are maintained and the fault types considered. Early detection of faults can reduce these rising consumptions and further prevent total system failure.

Performed manually, the detection of errors is very time-consuming. Algorithms based on artificial intelligence could help here.

To achieve good results, training data is required that includes time series of both nominal and component faulty behaviour [5]. However, historical data does not always exist, is incomplete or contains uncertainties. Especially a distinct allocation of time series to particular faults is difficult since several faults may cause similar variations in the data and one fault may result in several symptoms [6]. To overcome this problem and to generate faulty data in a targeted manner that is clearly assignable to activated faults, models and simulations can be used. Further, the generation of variable data sets is less time-consuming and less expensive compared to experiments.
In this paper, we present an automated process of generating training data for BES, since editing of models manually is time-consuming and can be error-prone. The implementation addresses models written in Modelica and uses component-based fault models within an existing fault model library. Unlike [7], we do not integrate faults directly into a complete library. Instead, we extend existing model code with references to or adjustments of connections to fault models within a fault model library. This is advantageous because changes within a library do not necessarily require modifications of the already extended model as long as they do not affect the interfaces. In addition to the adaptation of Modelica models, we explain how fault activation and simulations can be automated.

2. Model analysis and model extension

In this section, we first introduce the applied fault models. In the second part, an overview on the analysing method for the models’ syntax is given. Lastly, we explain the model extension.

2.1. Applied fault models

The applied fault models are part of a fault model library that has been developed for simulations with the Modelica model library AixLib [8]. In this paper, we consider fault models for the following components: sensors, dampers, valves, pumps and fans. Instead of replacing a component model, the fault models change the components’ input or output values. Thus, the integration of fault models into existing BES models requires the generation or adjustment of connections. For visualisation, two component models before and after a fault model integration are shown in figures 1 and 2. Some component models require additional type conversion blocks (cf. pump component model in figure 2) for connections with the corresponding fault model. The fault models will be published as part of a pending PhD thesis.

![Figure 1: Component models before integration of fault models.](image)

![Figure 2: Component models after integration of fault models. This pump component model requires additional type conversion blocks from Boolean to Real and Real to Boolean.](image)

2.2. Syntax analysis of Modelica code

To adjust existing Modelica models in a way that elements, e.g. fault models, can be subsequently integrated, the models’ code needs to be modified. We implemented this modification in Python. To be able to access particular parts within the Modelica code, we use a language recognition tool, named ANTLR [9]. ANTLR generates language specific parsers that are able to parse structured text (like Modelica models). The code of the models is thereby transformed into parse trees (i.e. a data structure) that enable access to specific parts within the models’ code. While visiting these parse trees, we adjust the code of the models.

2.3. Automated fault model integration and code extension

In figure 3, a schematic overview of the Modelica code extension process is shown. Each visitor is a class that implements an interface of ANTLR and performs various operations on the generated parse trees. Whereas initialVisitor and modelVisitor traverse the BES models, faultVisitor...
traverses the fault model library. Besides the generated parse trees (i.e. the Modelica models’ code), all visitors need additional information on component models of AixLib and Modelica standard library to perform their assigned tasks. This information is stored in a manually generated text file (config file) and contains allocations of suitable fault models, suitable ports that the fault models should be connected to and equation-templates and type-conversion-templates. The config file can be accessed by each visitor.

After generating the parse trees, \textit{initialVisitor} compares the component models of the BES model with the component models of the config file. For each matching component model, \textit{faultVisitor} selects a corresponding fault model from the fault model library. Adapting the fault model names by adding a number makes them uniquely identifiable, even if a fault model of the same type is integrated several times. In addition, \textit{faultVisitor} saves parameters of each selected fault model that are required for activating the faults before starting simulations (see chapter 3). The \textit{modelVisitor} receives information from \textit{initialVisitor} and \textit{faultVisitor}. It combines the passed information with the BES model. This comprises placing the adjusted fault models correctly within the model’s code, placing conversion blocks if necessary and adding or adjusting the model’s code in order to establish valid connections to the placed fault models and conversion blocks. The output is a valid BES model with all possible fault models integrated.

3. Fault activation, simulation and data generation

In order to generate training data, simulations need to be performed with the newly extended BES models. As the integrated faults are inactive by default, single or multiple faults must be activated by two fault model parameters (i.e. Boolean and time for activation) before starting a simulation. Once a fault is triggered, it remains active throughout the remaining simulation.

The simulation of the extended BES models is intended to be performed automatically. We use a DymolaPython interface [10] to control the execution of simulations in the simulation environment Dymola via Python. Our fault activation setting activates one fault per simulation run (model by model). In addition, a fixed number of multiple faults are activated simultaneously once per model.

The automated fault activation and simulation process works for a high share of Modelica models. Nevertheless, there are two exceptions in which manual adaption is still required. First, when fault models are automatically integrated into a sub-model, the automated simulation start
requires the allocation to its corresponding compilable model. Second, integrated fan faults still require nominal values for mass flow rate and pressure difference. Since both the superordinate model and the fan values cannot be read out in the models’ code, this information needs to be passed to the simulation script (applies for superordinate models) or can be integrated directly into the extended models (applies for fan values).

4. Application and evaluation
We apply the automated fault model integration and simulation tool to ten parametrised models of the AixLib. Different degrees of complexity regarding the total number of elements, the number of potential fault models and the type of component elements are considered. Five of these models are listed in table 1 and considered in this paper. The model GenericAHU is a sub-model and not directly compilable.

Table 1: Test models with number of elements and elements with corresponding fault model.

| Test model name       | Element count: total (connected) | Elements with corresponding fault model and number, if element exists several times |
|-----------------------|----------------------------------|----------------------------------------------------------------------------------|
| Pump                  | 8 (7)                            | pump                                                                             |
| HeatPump              | 21 (20)                          | pump, sensor                                                                     |
| OneRoomRadiator       | 36 (26)                          | pump (2), sensor (3)                                                             |
| SimpleHouse           | 39 (27)                          | pump (w/ type conversion), damper, fan, sensor                                   |
| GenericAHU            | 53 (39)                          | sensor (7), damper (4), fan (2)                                                  |

4.1. Assessment of functionality
The functionality of the automated fault model integration and simulation is assessed simultaneously. For each test model, a manual version with integrated fault models is built that serves as comparison version. Since Modelica code does not need to be identical and different positions do not necessarily lead to different functionalities, we use the simulation results of the augmented models to compare and analyse the functionality of both the fault integration and fault simulation. Manually and automatically extended test models are simulated with the same parameter settings and the same simulation time.

4.2. Assessment of advantages
To evaluate the automated fault model integration in terms of time and accuracy compared to manual fault model integration, we conduct an experiment. Participants are asked to perform the fault integration on the selected test models manually and to measure their required time. They can use the graphic interface of Dymola or code-based editing. Before the start of the task, participants are introduced to the fault model library and connection examples.

The measured times include deciding for which components a fault model exists, which fault model corresponds to which component model, and implementing the connections, i.e. extending or changing the model’s code. For the evaluation of time, the times measured by the participants are compared with the measured times of the automated fault model integration. To evaluate the accuracy, the models processed by the participants are compared with the corresponding manual comparison versions (cf. chapter 4.1). Deviations are categorised into four error categories:

- **E1** - False component model (e.g. sensor faults model connected to temperature model)
- **E2** - False fault model (e.g. valve faults model connected to damper model)
- **E3** - Incorrect connection (e.g. old connection not removed or faulty drawn new connection)
- **E4** - Missing fault model (e.g. no sensor faults model integrated for sensor model)
5. Results
In the following, we present the results obtained for assessing the developed automated fault model integration and simulation tool with methods described in chapter 4.

5.1. Functionality of automated fault model integration and simulation
For each test model, the integration and simulation process is performed both manually and using the automated tools. Each calculated variable of the manually edited and simulated models is compared with the equivalent variable of the automatically edited and simulated models. In total, we run 30 simulations each manually and with the automated tools. All simulation results of the automated augmented and simulated test models match with those calculated for the manual augmented test models.

5.2. Advantages of automated fault model integration
The experiment described in chapter 4.2 is conducted with seven participants. Their working experience with Modelica ranges from 0.5 to 1.5 years for six participants to 4 years for one participant. The comprehensibility of the experiment’s task is rated with an average of 5.29/6 by the participants.

The results of the measured fault model integration times is shown in figure 4. For each test model (listed on the x-axis) the corresponding measured times are allocated on the logarithmic scaled y-axis. The mean values of the operating times per model measured by the participants range between 124 and 910 seconds (i.e. 2.06 and 15.17 minutes). The times measured for the automated fault model integration range between 0.8 and 2.4 seconds. Hence, a time advantage for the automated fault model integration of two orders of magnitude can be derived for these test models. In addition to the time measurements, the analysis of the participants’ augmented models shows deviations from the desired comparison version in 20 of the 35 analysed test models. A classification of all detected deviations into four error categories (cf. chapter 4.2) can be found in figure 5.

6. Discussion
The automated fault model integration is the first sub-step of the training data generation tool. It shows its advantages through the significantly reduced time required for fault model integration and the absence of errors compared to manual fault model integration. The second
sub-step of the training data generation tool is the automated fault activation and simulation. This sub-step is inherently automated for the activation of single and a fixed number of multiple faults. While the fault model integration is implemented independently of a Modelica simulation environment, the automated conducting of simulations uses the DymolaPython interface and is tied to executions of simulations with the simulation environment Dymola. However, it is possible to replace this interface and to adjust it to a simulation environment of choice. In order to perform sub-step 1 together with sub-step 2, additional information is required for particular models. This refers to models that are not directly compilable and models containing fan faults (cf. chapter 3).

7. Conclusion
In this paper, we present a framework for generating training data for FDD algorithms in BEMS in an automated way using BES models. The sub-steps comprise the integration of fault models, the activation of selected faults and the execution of simulations that have been successfully implemented. For this purpose, syntax analyses of the models’ code are carried out, models’ code is extended after comparison with stored data and a simulation interface is used. The automated integration of fault models as first sub-step shows large time savings and is two orders of magnitude faster than the manual fault model integration. Furthermore, no errors occurred in the connections of component models to integrated fault models. The implementation considers component models of the AixLib for sensors, dampers, valves, pumps and fans. Overall, the fault model integration and simulation process is extendable for additional fault models and adaptable to other libraries. Future research is required to investigate the relationship between fault activation and training of algorithms to find out which data sets need to be generated to achieve the best possible results in FDD. Furthermore, it is planned to make the project code available as open source.

Acknowledgments
We gratefully acknowledge the financial support provided by the German Federal Ministry for Economic Affairs and Energy (promotional reference 03SBE0006A).

References
[1] Abergel T, Dean B, Dulac J and Hamilton I 2018 Global Status Report. Towards a zero-emission, efficient and resilient buildings and construction sector (Int. Energy Agency and UN Environment Programme)
[2] Narayanaswamy B, Balaji B, Gupta R and Agarwal Y 2014 Data driven investigation of faults in HVAC systems with model cluster and compare (MCC) BuildSys ’14 vol 1 (Memphis TN, USA: Association for Computing Machinery) pp 50–59
[3] Roth K W, Westphalen D, Llana P and Feng M 2004 The Energy Impact of Faults in U.S. Commercial Buildings Int. Refrigeration and Air Conditioning Conf. at Purdue
[4] Yu Y, Woradechjumroen D and Yu D 2014 Energy and Buildings 82 550–562 ISSN 0378-7788
[5] Benndorf G A, Wystriel D and Réhault N 2018 Applied Physics Reviews 5
[6] Zhao Y, Li T, Zhang X and Zhang C 2019 Renewable and Sustainable Energy Reviews 109 85–101 ISSN 1364-0321
[7] de Kleer J, Jannsen B, Bobrow D G, Kurtoglu T, Saha B, Moore N R and Sutharshana S 2013 Fault Augmented Modelica Models Proc. of the 24th Int. Workshop on Principles of Diagnosis pp 71 – 78
[8] Müller D, Lauster M, Constantin A, Fuchs M and Remmen P 2016 Aixlib – an open-source modelica library within the ieac-ebc annex 60 framework BauSIM 2016 pp 3–9
[9] Parr T ANTLR 4 Documentation, 2019. URL accessed on: October 16, 2020 URL https://github.com/antlr/antlr4/blob/master/doc/index.md
[10] Fuchs M How to: Dymola Python Interface. Aixlib Wiki. URL accessed on: November 27, 2020 URL https://github.com/RWTH-EBC/AixLib/wiki/How-to:-Dymola-Python-Interface