Honda Research Institute EU
Honda Research Institute Europe GmbH
https://www.honda-ri.de/

Smart Company Digital Twin - Supporting Controller Development and Testing Using FMI

Tobias Rodemann, René Unger

2018

Preprint:

This is an accepted article published in Japanese Society of Automotive Engineers Spring Meeting. The final authenticated version is available online at: https://doi.org/[DOI not available]

Smart Company Digital Twin

- Supporting Controller Development and Testing Using FMI-

Tobias Rodemann [1)] René Unger [2)]

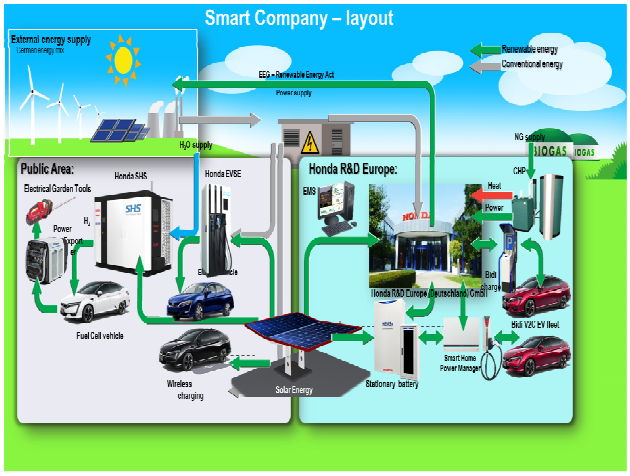
1) Honda Research Institute Europe,
Carl-Legien-Str. 30, 63067 Offenbach, Germany
(E-mail: tobias.rodemann@honda-ri.de)
2) EA Systems Dresden,
Würzburger Str. 14, 01187 Dresden , Germany
(Email: rene.unger@ea-energie.de)

ABSTRACT: This article describes a simulation system for a smart building complex that integrates local energy production, flexible loads, and energy storage with electric mobility. Our target is to increase overall efficiency to reduce costs and greenhouse gas emissions. We developed a simulator that uses the Modelica-standard based software SimulationX which covers a variety of physical domains. The simulator is employed for testing future system configurations, smart maintenance, and development of energy management systems. In order to be able to test advanced controller concepts we use the Functional Mockup Interface (FMI) technology to link the simulator with external controller software written in Python.

KEY WORDS: **Standardized: Control, Smart Grid,** Free: FMI, Energy Management

1. INTRODUCTION

Optimizing the energy efficiency of complex industry building complexes is a difficult task. Advanced modeling tools like those based on Modelica [(1)] can support this process and lead to faster and more reliable assessments of the current situation and potential extensions. We developed a building system model for the future layout of our facility campus, called smart company, that will consist of a variety of energy production (e.g. Photo Voltaics), storage (stationary battery), and consumption units (e.g. computing cluster). One of the main research questions is the integration of electric mobility in the form of battery electric vehicles (BEVs) or Fuel Cell Vehicles (FCVs) into the building energy management.



A proper analysis of such a system requires an optimal control of all flexible units, for example co-generators for heat and power (CHPs) or charging stations. While simple controllers can be implemented within the Modelica framework, more complex approaches like Model-Predictive Control [(4)] are better handled in high-level languages like Python or Matlab.

We therefore implemented a Functional Mockup Interface (FMI, [(6)]) – based solution to couple the main building simulation running in SimulationX [(2)] with an external Python-based controller implementation. This document outlines some of the technical challenges and solutions for the FMI connection between a Modelica-based simulation system and a high-level scripting language.

2. APPLICATION SYSTEM

2.1. Smart Company Philosophy

The main concept behind Smart Company is to provide a test environment for various energy-related modules for the integration of electric mobility and smart electricity grids. The effects of different modules need to be considered within a system concept. An example is a battery storage system that can be used to store excess power from a PV (photo voltaic) system, so that optimal battery and PV system size are linked. A related question is smart charging of BEVs depending on the amount of available power.

2.2. Main Elements

To simplify the description, we sort the elements of our smart company system into three groups – those that produce energy, those that store it and (flexible) loads that consume energy.

2.2.1. Energy Production

The main source of electricity is the local energy grid. In addition to that we employ a local co-generator for heat and power (CHP) that can cover most of the system's base load (almost complete coverage at nights and during weekends) and also provides a substantial part of the heat production on most days. Additional heating power is delivered by three additional conventional boilers which, like the CHP, run on natural gas. Additional electricity will be produced by a large-scale photovoltaic (PV) system in the range of 400 kW peak. PV is well suited for our system, since on sunny days it covers most of our additional (non-base) loads which currently are mainly office, workshops, test benches and cooling.

2.2.2. Storage Elements

We employ a conventional heat storage which adds extra flexibility for our CHP system. For the storage of electric power, we plan to use a large stationary battery as well as dynamic capacity in the batteries of connected BEVs. We also consider to convert electricity to hydrogen for fueling FCVs and possibly to convert the hydrogen back to electricity to be used in the system in peak power situations.

2.3. Consumers

Our facility has the usual standard consumers for a medium-sized office and research facility, like heating/ventilation/air-conditioning (HVAC), lighting, IT systems, and others. Of special interest is a diverse set of different charging stations with individual power consumption levels between 2 kW and 210 kW in peak. We are also investigating if HVAC systems can be modulated to shift their power consumption if other consumers (e.g. charging stations) have a high demand (peak shaving).

2.4. Data Management

In order to better understand energy flows and to detect inefficiencies we employ an advanced data monitoring system for most energy related system parameters, a data collection and archiving solution, and various tools for data analysis and visualization. Besides a large number (50+) of sensors for heat, cold and electricity consumption (including reactive power and harmonics), a local weather station and vehicle charging parameters are monitored as well in a high time resolution.

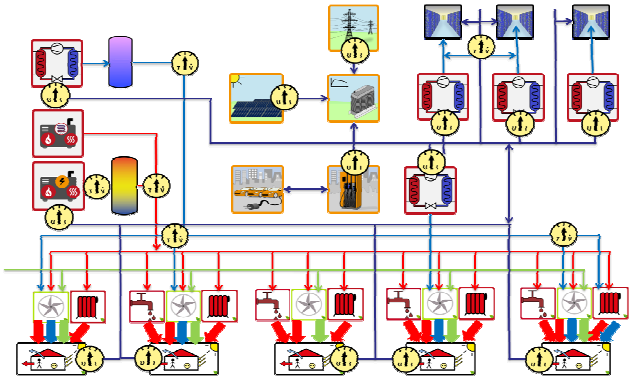


Fig. 2 Data Monitoring System

A key feature of our monitoring server is the ability to detect anomalies in sensor readings that point to either device or sensor failures. Currently we resort to basic checks defined by an expert in the field (e.g., 'plausible range of values'), but in the future we plan to build complex statistical models and detect deviations from these models. These models can be built based on historical sensor data but could also be computed by our digital twin model.

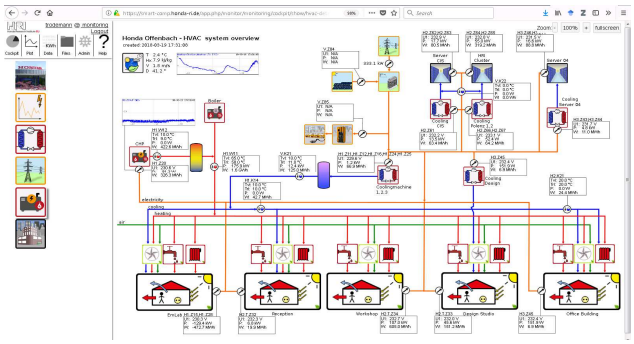


Fig. 3 Monitoring Data Visualization System

3. SIMULATION APPROACH

From our experience we believe that there many reasons for using a simulation approach in addition to the real-world hardware implementation:

- The simulation system can be assembled much quicker than the real system. Therefore our digital twin is actually older than its real-world brother. Adding new modules can be a matter of minutes instead of weeks or months.
- Despite some licensing costs, simulation-based tests are much cheaper than any real-world system test.

- Simulation results can be derived quickly – simulating a complete year only requires a few minutes. This allows a fast assessment of for example new technologies or regulations.
- Data collection is easy. The simulation provides all necessary information for business case and technical analysis (consumption, room temperature, battery usage, etc.).
- Since buildings are stationary they can't be taken on a test run, for example we can't control climate conditions or usage patterns. In the simulation all factors can be controlled and varied as desired.
- Climatic conditions (365 days) or usage scenarios (days to weeks) are slow in real time but can be simulated in minutes.
- Different variations of the system, like size of the PV system, can be compared easily. Using our computing cluster we can run 10,000s of different configurations in a day in order to find the best suited one considering a number of criteria [3,7,8].
- In the simulation it is possible to test extreme (e.g. coldest week, failure of CHP, all drivers want to charge at same time, etc.) and failure scenarios. These tests could never be done with a real building that is currently in use.
- One of the major reasons for using a simulation is that it is essential to develop and test different controller concepts. Any validation of a controller requires careful tests with a large variety of scenarios that would be extremely difficult to execute on the real system. In a simulation different situations can be covered. For some controller concepts like Model Predictive Control (MPC [4]) a (simplified) simulation model is required to optimize the predictive control.

It has often been noted that simulations are not reliable in contrast to real-world tests with the real hardware. We believe that results from a simulation study using many different configurations and external conditions are more reliable than results of a single real-world system test with very limited variability. Simulation-based development and business model assessment should therefore be seen at the same level as conventional approaches.

3.1. Modelica

There is a variety of simulation tools for buildings or PV systems, but we require an approach that can cover a larger number of physical domains. Modelica [1] was specifically developed for this purpose as a free modeling standard. Several commercial and free tools have been developed that implement the Modelica framework and provide a modeling approach similar to the well-known Matlab/Simulink without some the latter's drawback like unidirectional information flow which often leads to loops in the system design. Our simulation approach allows us to investigate specific physical properties of some modules in detail (e.g. battery voltage) without the excessive computing demand of many 3D modeling approaches. Where this is necessary, for example to estimate the heat demand of rooms depending on weather conditions, results are pre-computed and loaded either as data tables or as calibration parameters for simplified models into the simulation.

3.2. SimulationX + Green City Library

Our system is implemented using the SimulationX [2] software with the Green City library extension, which was developed for smart building energy management and e-mobility problems. This allowed us to use most modules off-the-shelf and only rarely required custom adaptations. An example of a system model is shown in Figure 4.

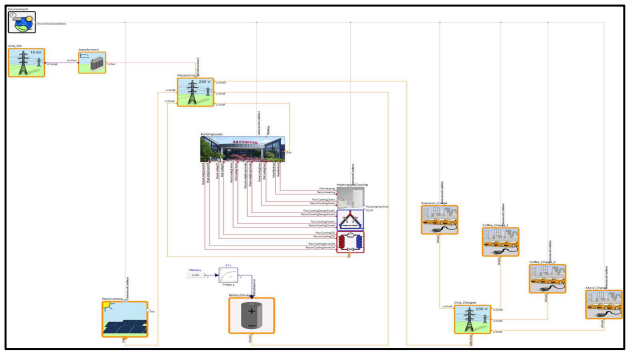


Fig. 4 SimulationX model of system

3.3. Performance

The basic simulation without an external (FMI-based) controller can complete a whole year of simulated time in 60 – 300 s (depending on specific system configuration). The duration increases if lots of data has to be saved to file, which is rarely necessary (normally only a few items are of interest). Memory demand is modest so that the simulation can be run on most standard office PCs and even laptops. The simulation model can be exported as C-code that is compatible to a variety of platforms including Windows and Linux.

3.4. System Optimization

With the simulation system we have performed an automatic configuration optimization using state-of-art many-objective

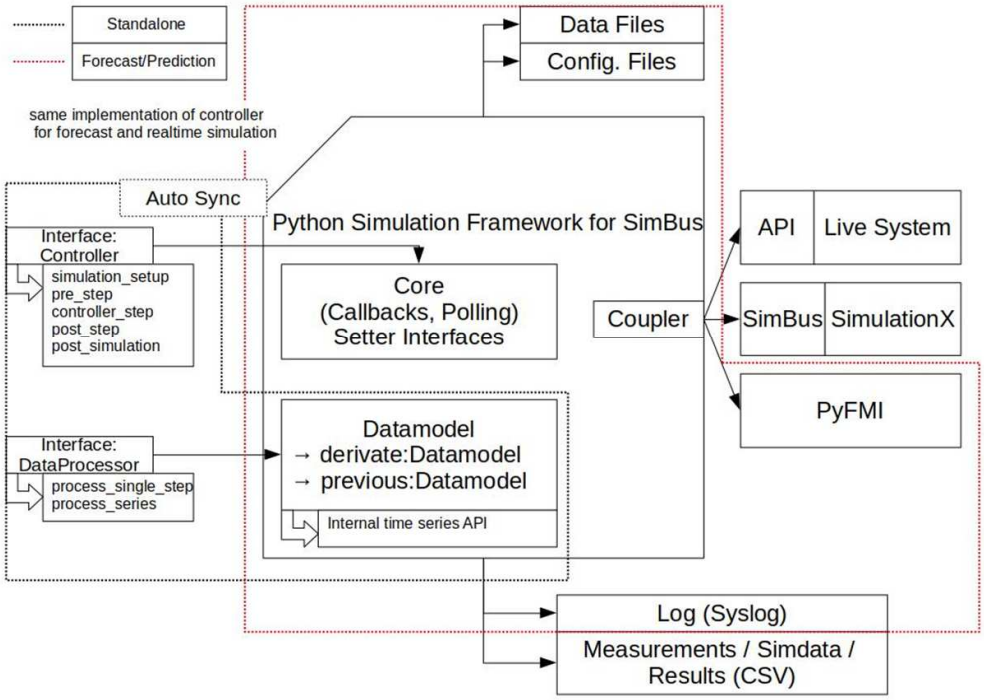


Fig. 5 Basic Modules of SIMBUS universal coupling framework

optimization methods to find best possible compromise configurations that minimize additional investment costs, yearly electricity bills, CO2 emissions, and maximize resilience and battery lifetime [7,8]. We got a better understanding about interdependencies between different elements and optimal parameter settings. An interesting finding [7] was that a sub-optimal control could make even substantial investments worthless. An example was a battery used for peak load reduction, where a simple control approach could not reach the desired savings. We conclude that the control algorithms are key elements for any energy system and need to be investigated in detail.

3.5. Controller in the Model

The overall system will require a number of controllers for components provided by different suppliers (like HVAC, BEV charging, and CHP). Due to the system design these controllers will influence each other, leading to a challenging development problem. One needs to develop an overall controller architecture (type of controllers and how they interact) and test this concept under many different conditions. Controllers can be implemented directly in Modelica using simple rules like e.g. “charge battery if SOC is below 30%” (SOC = battery state of charge). These types of rules can be written explicitly as Modelica code or by using simple control blocks from the standard library. The latter is more effort but provides a visual representation of the control rules. For example one can directly see which inputs are used by the controller. There are limits to the implementation of more elaborate controller approaches in Modelica, though. For these cases we want to use high level programming languages like Python or Matlab that are more flexible and come with a variety of tool packages from the area of Machine Learning and Data Mining. Our final target is an advanced software environment for “Controller-in-the-loop” testing.

3.6. Python Controller

Python [5] is a high-level interpreter language with a large number of free libraries that can provide important functionality without much extra programming effort. As an interpreter language it is also easy to debug and modify.

The next section will now outline how we connected a controller written in Python with our SimulationX model. The solution is to use the Functional Mockup Interface (FMI) standard that is well integrated in SimulationX (the developers of SimulationX are one of the key players in the FMI community).

4. COUPLING SIMULATOR AND CONTROLLER

While Modelica is ideal for modeling the physical and usage characteristics of the Smart Company energy system it is less suited to model complex controller and machine learning algorithms. Especially for the latter, many sophisticated

scientific libraries based on the Python programming language are available.

To connect these two worlds, a suitable solution was needed:

- Easy to use coupling to concentrate on controller functions, not numerics and system setup
- Performance
- Flexibility regarding test platform
- Use of the same controller code in development (co-simulation), regression testing and live control

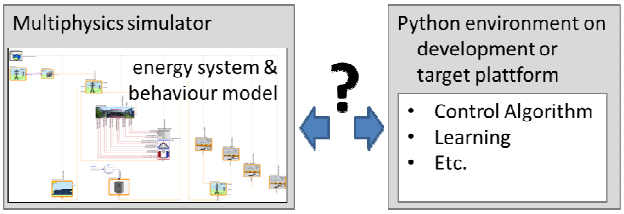


Fig. 6 Co-simulation tasks

Another important issue for us was reusability of models and code throughout the development process phases:

- Initial multiphysics model and mockup controller in Modelica
- Sophisticated model and conceptual controller (Python) in development environment
- Model optimization and detailed controller implementation on target platform
- Fast regression testing of controller algorithms and parameters against the simulation model

4.1. Coupling Options

For coupling the Modelica and Python worlds, several options are available. The first and most robust way is a loose 'coupling' where only independently simulated data are exchanged between energy system simulation and controller. This solution is suitable to condition learning algorithms based on simulation data but not suitable to test controllers in a closed loop.

Another option is to use one of the APIs for databases (SQL), the OPC (Process Control) or the IP Socket integrated into SimulationX. All are flexible and fast for their respective applications, yet the definition and implementation of the actual interface puts a considerable effort on the developer. This is especially true if simulator coupling is used in an early stage of development, when the required interfaces and exchanged variable are modified often.

A third option is in using Modelica external functions or the newer FMI standard for simulator coupling.

4.2 Functional Mockup Units (FMUs)

The Functional Mockup Interface Standard (FMI) defining FMUs was developed to exchange simulation models and use them as a 'gray' box within other simulation environments. Each FMU is a ZIP file which contains the necessary API definition as XML files, controller code as binary files and helpers as well as documentation.

FMI comes in two flavors, model exchange where the binary only contains the model DAEs (Differential Algebraic Equations) and co-simulation where the binary integrates the solver code as well. For the controller test application, co-simulation is most suitable as explained later.

While FMI defines all the basic coupling functions and variable naming scheme, the developer still has to implement them on the controller side. Another major disadvantage during development is the deployment process to the testing environment. The developer needs to compile and package, deploy and import the FMU to the simulation environment after each change. Additionally, since binary code is exchanged, controller under test as well as simulator needs to run on the

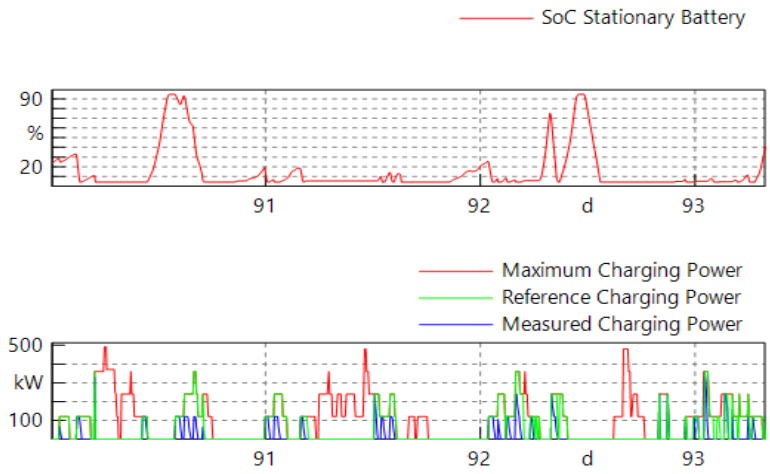


Fig. 7: Example feasibility study: state of charge of simulated large stationary battery for the smart company complex; peak shaving in combination with the demands and overall set points for controlled charging of a large EV fleet

same hardware platform. This would be a problem, since we need to feed different platforms from the same code repository: development system (Windows), optimization cluster (Linux), target controller (embedded Linux).

4.3. Using SIMBUS

The SIMBUS framework was created to combine the advantages of FMI with automatic code generation and a workflow to allow the developer to concentrate on the controller and test model while having co-simulation as a transparent layer.

Using an existing FMU or a text file containing the interface definition with the exchanged control and measurement signals, SIMBUS generates a new FMU and Python library.

The FMU can be imported into SimulationX. Its input and output parameters are connected to the respective model references.

The Python library needs to be added to the controller software project. It provides an object oriented data model, controller and data processor interfaces and the required connectivity functionality.

Afterwards the SIMBUS application (Java) is started. Now the Python controller program and SimulationX can be launched. The simulation behaves as if the python code was directly integrated into the Modelica model. The controller code can also run in a debugging environment, so that all changes to the controller logic are instantly affecting the running simulation. Control signals and their reaction can be analyzed in detail using the various toolboxes integrated in SimulationX.

4.4. Real World Controller

The Model is a digital twin of the real Smart Company environment. It consists of environment, building energy system, user behavior and a significant number of charging stations. All sensors and actuators relevant for the overall controller need to be mapped, which results in a significant number of inputs (200+) and outputs (35+) of the controller under test.

Currently as of version 2.0 the FMI standard does not support arrays or structured variables which will become available in Version 2.1+. This results in a significant connection effort during modelling. To improve this we encapsulated the actual FMU into a separate type which basically does the connector mapping (see figure 6) from the FMI scalars to complex Modelica connectors. The actual controller coupling implementations are inherited from this type.

During development the implementing type contains a pure Modelica implementation of a basic controller. During initial python controller development the implementing type was switched to the generated SIMBUS FMU. For the optimization and regression testing phase, an implementation with inputs and outputs internally inverted (code export connector) is used. This way the whole model can be used as single block within the Python controller.

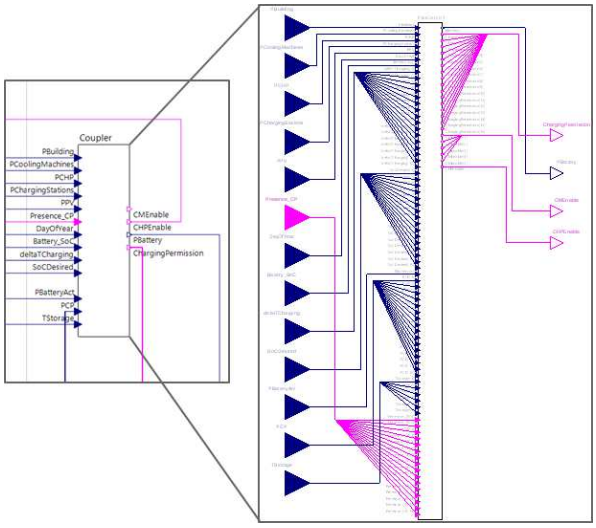


Fig. 8: Connector mapping for improved usability

4.5. Flexibility and Performance

While the coupler uses FMI as standardized interface to the simulator, all FMI requests are internally mapped to a TCP layer. The Simulator as well as the Python framework are connecting via TCP/IP to the SIMBUS application which acts as a shared memory and synchronization instance. This application is also the point, where other tools can dock on, if interaction between several controllers needs to be tested.

The TCP coupling has advantages and disadvantages. A major advantage is the flexibility which allows us to switch between python controller in development environment with debugger (IDE) and implementation on the embedded platform within an instant, even without restarting the simulation. Additionally the developer is free to use the operating system of choice, independent of the Windows platform for the simulator.

On the other hand, the TCP layer adds a significant performance bottleneck. We measured between 5 ms (local network) and 200 ms (several VPNs) delay in each synchronization step, depending on network load and the number of exchanged values.

For most of our tests under real world conditions we measure a ping delay of 10-20 ms. Therefore we used a synchronization step size of 1 minute for simulation runs up to one month simulated time and a size of 5-10 minutes for longer running

simulations. This way, synchronization overhead to simulation for coupling was kept within limits.

The setup was a developer laptop with Windows, running SimulationX with Green City Library as well as our own library extensions, the SIMBUS application. Python was running locally or on an embedded PC with Linux.

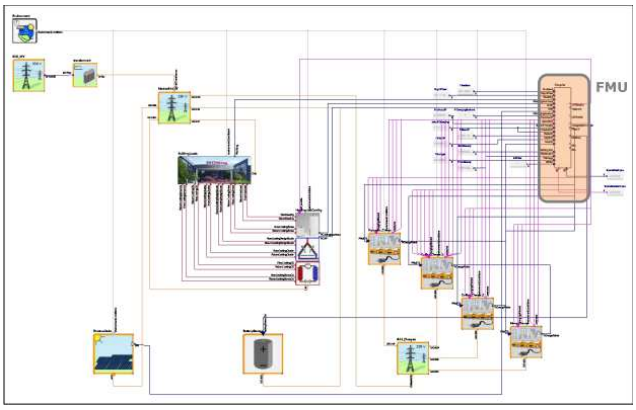


Fig. 9 Simulation model with integrated FMU

4.6. Faster Testing Cycles

During model development and initial controller implementation, the whole analyzation toolset of SimulationX was extremely helpful. With the maturing of the controller software, a faster and simpler testing environment is needed. The basic idea is to turn the whole setup around. Instead of integrating the controller into the model, the model is integrated into the python environment.

The Python PyFMI library makes it possible to integrate Models exported as FMUs into Python programs. We used the code export functions of SimulationX to create an FMU with integrated solver of the model.

The SIMBUS framework also provides a wrapper for PyFMI, so the controller code can run within the same environment, but connects directly to the FMU of the smart company model instead to the Simulator. Without the need for TCP/IP, the communication overhead is about one order of magnitude less. Additionally, several instances can be run in parallel on our computation cluster which makes it possible to test thousands of controller optimizations and parameter sets as well as different weather and usages scenarios in a short time.

By using such ‘synthetic statistics’, automatic adaption and machine learning algorithms within our controller can efficiently start into operation.

4.7. Online Models

The integration of FMUs into the Python code offers another interesting option currently researched. Subsystem models can be simulated online within the controller to evaluate the best operational strategies for our Smart Company Controller (10). For example the controller can test different potential operational strategies for the day. The forecasts on weather, renewable energy and prices as well as on vehicle charging demand and building usage are fed into the online model. The model is then executed with different controller parameter sets (strategies). After simulation, the strategy with the best results is used until external conditions change.

This concept integrates with the previously described development workflow as models and much of the code base can be re-used. It also perfectly integrates with our approaches to machine learning.

4.8. Controller test environment

One problem for thoroughly testing a controller is that for complex systems to be controlled, certain controller input constellations don’t appear in the test data and are difficult to produce in the simulation without resorting to massive changes of the simulation model. We therefore implemented a separate Python-based controller test environment in which the developer can produce any combinations of inputs and observe the controller output in isolation.

Inputs can be fed manually through user inputs or automatically by triggering separate Python scripts. This enables us detailed testing and debugging of isolated functions as well as system effects of the whole controller.

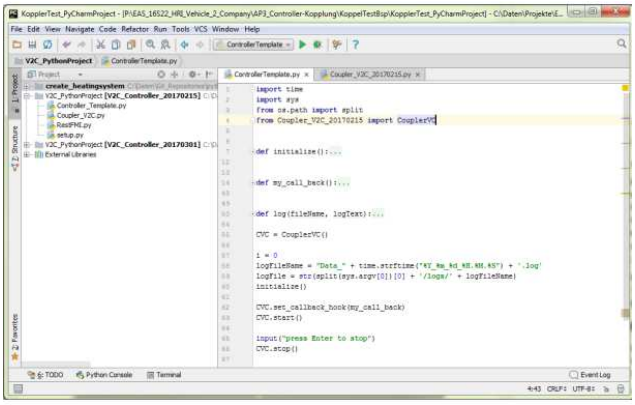


Fig. 10: Basic python template for FMI controller

5. CONCLUSION

We have described a detailed simulation environment that uses models developed on the basis of detailed smart meter measurements. We have applied this tool for a better understanding of our targeted system and to assess the benefits

(for example expressed as Return-Of-Investment) of various investment options.

Key elements for high energy efficiency are optimized controllers. For the development of these controllers it is necessary to link the simulation model to a high-level programming language where complex control architectures and for example Machine Learning (ML) methods can be implemented easily.

The connection between simulation and external controller software was realized using the FMI interface. Our tests showed that a new controller FMU can be set-up in less than a day of work and provides a flexible and easy to manage development environment. We found that the implemented tool-chain speeds up work considerably especially for novice users.

We currently plan to evaluate more advanced controller concepts like Model Predictive Control for the overall (top-level) energy management.

An additional planned usage of the FMI interface is to connect Multi-Agent Systems to the simulation, for example to evaluate the impact of user behavior on the overall energy flows. This could be for example a simulated community of EV drivers who individually respond to price signals or waiting times of the charging stations, by adapting their charging requests.

Finally we note, that the developed methods are not limited to the building domain, but could easily be applied in other domains with similar challenges as for example energy management in Hybrid Electric Vehicles (HEVs).

REFERENCES

(1) P.Fritzson & P. Bunus: Modelica – a general object oriented language for continuous and discrete-event system modeling, Proceedings of the 35th annual simulation symposium, p. 14-18 (2002)

(2) ESI-UTI, SimulationX 3.8, Online: www.simulationx.com

(3) R. Unger, B. Mikoleit, T. Schwan, B. Baeker, C. Kehrer, T. Rodemann, Green Building - Modeling renewable building energy system with eMobility using Modelica, Proceedings of Modelica conference, 2012

(4) Camacho, Eduardo F., and Carlos Bordons Alba. Model predictive control. Springer Science & Business Media, 2013

(5) Python: https://www.python.org/

(6) Blochwitz, Torsten, et al. "The functional mockup interface for tool independent exchange of simulation models." Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical University; Dresden; Germany. No. 063. Linköping University Electronic Press, 2011.

(7) T. Rodemann, A Many-Objective Configuration Optimization for Building Energy Management, accepted for IEEE WCCI 2018

(8) T. Rodemann, A Comparison of Different Many-Objective Optimization Algorithms for Energy System Optimization, submitted to PPSN 2018

(9) C. Reinhardt, Diploma Thesis on Model Predictive Control for optimized BEV charging at ‘Ladecampus’, TU Dresden, 2018