DEMKit: A Flexible Smart Grid Simulation and Demonstration Platform Written in Python

Gerwin Hoogsteen, University of Twente, Department of EEMCS, g.hoogsteen@utwente.nl

Smart Grids and Distributed Energy Management (DEM) methodologies are heavily studied within academia and theoretical models and control methodologies have been developed over the last years. To validate these models and control methodologies, often simulations are carried out in several connected domain specific tools and environments. The complexity of such a setup often hinders a simple transition from models to a simple proof-of-concept for demonstration and real-world experiments. On the other hand, for demonstration, large field tests arise that require validated and optimised implementations of control algorithms. These specialised implementations often lack flexibility to change the implementations after observing inconvenient behaviour or even fundamental errors in the concept itself. However, due to the multidisciplinary nature of the energy system, it is often hard to see effects of implementation decisions on all subsystems on beforehand.

To tackle this problem, a new and agile DEM platform for simulations, co-simulations and simple demonstrators is presented in this paper. We call this platform DEMKit and has a focus on energy management within smart microgrids and households. The main goal of this platform is to provide a package of components for efficient and effective research on DEM methodologies that should be usable to any researcher with some scripting knowledge. On the other hand, the platform should be able to bring simple simulation models to life on real appliances as a proof-of-concept within a day. The latter is made possible by keeping the simulations as close as possible to the real environment, such that most of the code base can be re-used.

The software is written in Python to benefit from dozens of scientific packages and object oriented programming structures. On the other hand, this allows researchers to run the software on multiple operation systems and embedded platforms with little experience in computer science. Support for hundreds of domestic appliances and an user interface are available through the use of the open-source home automation software OpenHAB [1]. With such rapid prototyping, feedback and changing requirements based on the proof-of concept can be incorporated in an early stage of development. Since simulation and demonstration share the same code-base, measurements from a test can be imported to re-simulate and resolve potential issues for a new prototype.

The following section present the general features and structure of the software. Subsequently, the interface for simulations and demonstration is given, followed by a short conclusion and recommendations.

This research is conducted within the EASI project (#12700) supported by STW and Alliander.

I. Platform Overview

To enable the rapid prototyping, a cyber-physical systems approach is taken during the development. Different components representing physical devices and control logic are available. These components include devices, metering equipment, controllers and physical infrastructure. In addition a central object, referred to as the *host*, is used to carry out tasks such as simulation logic for simulations or an application programming interface (API) for demonstration purposes, which are discussed in Section II. The platform itself provides functions such as printing information, initialisation functionality and load models. Such load models, also written in python, consist of the instantiation of multiple components and the links between them.

One common concept throughout the platform is the use of discrete time intervals for simulation and logging. An interval can be as small as one second and multiple timebases can be used throughout the model. All components implement the *timeTick*-function to advance in time and calculate energy consumption, update predictions and perform state changes if applicable. Data is stored in the timeseries database InfluxDB [2]. Tools as Grafana [3] can be used for plotting and analysis of the data. A system overview is depicted in Fig. 1.

Devices are the key components within the platform . Within simulations, generic *device* components are available that model the default behaviour of a device. For DEM methodologies, the flexibility offered by devices is of interest. Therefore, the generic flexibility classes as defined in [4] are implemented, making it straightforward to generate and use flexible load models within the platform. These device classes include *static loads*, *buffers*, *whitegoods* and *electric vehicles*. In addition to the behaviour and flexibility description, each device also implements default functionality such as storage for schedules and variables expressing the current power consumption for multiple commodities such as active power (for the three phases separately), reactive power or heat. In case of a proof-of-concept, the generic device components have to be replaced by a connector that links to a real device, such that control logic from the model can read the state of the device and change its behaviour.

These schedule variables can be filled by the optional *controller* components, which separates control algorithms from devices in order to test different control strategies. The controllers are divided in two groups, namely device controllers and fleet controllers. Device controllers implement device

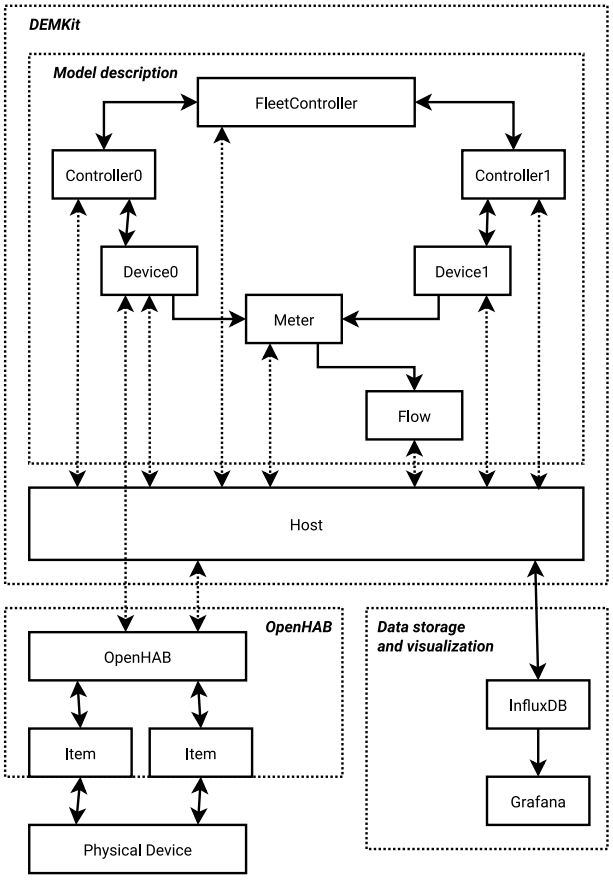

Fig. 1. Diagram of the DEMKit platform with a model and external software.

specific prediction, scheduling and online control algorithms and communicate with a fleet controller. Fleet controllers ensures that a fleet of devices is steered towards the objective power profile or value. Currently two control methodologies are implemented: Profile Steering [5] (single and multi-commodity support) and auction-based control mechanism [6] (single commodity only).

Power consumption of multiple devices can be aggregated into one value through a *meter* component. This meter component can be linked to a node in the *flow* model. Such a flow model is consists of a graph, which represents the physical energy distribution network. Connected to this graph is a load-flow solver to calculate the steady state within this network. Currently a load-flow solver for three-phase four-wire unbalanced low-voltage electricity distribution networks is implemented. This solver reads out the consumption data of the meter components to determine the voltages and currents in the network.

II. Simulation and Demonstration

Within a simulation, the *simulationHost* provides the simulation logic. All components defined within a model link to this host and the simulation is started using the *startSimulation* command from the model description. The simulation host first initialises each component within the model. After this, a for-loop is started to simulate time progression based on the selected simulation timestep size. Within this loop, the *timeTick* function of all components is called. Since there is a dependency between the flow simulation, meter components and devices, the order of simulation is important. Therefore, timeticks are distributed first to *controllers* and *devices* in randomized order. After all these components received the timetick, the *meters* receive a timetick, followed by the *flow* simulators. The randomized order of controllers and devices simulated the real-world situation where ordering in events may not be given either.

These choices make it easy to transfer a pure simulation model to a co-simulation or demonstration environment. In order to do so, the *simulationHost* needs to be replaced by a *restHost*. Device models that are to be replaced by their physical counterparts need to be exchanged for device connector components. In the current implementation, this *restHost* provides an REST API to provide access through HTTP calls and JSON data objects. The API exposes functionality for external timeticks, event messages to device and controller objects. The control structure itself can also be distributed over multiple embedded systems as control signals are included in the current API. A set of device connectors is implemented which interfaces with a set of items in the OpenHAB [1] home automation platform. This allows bi-directional communication between the DEMKit platform and OpenHAB items. Subsequently, these OpenHAB items are linked to properties of supported devices such as, among others, Tesla electric vehicles or Miele smart home appliances.

III. Evaluation, Conclusion and Future Work

The first stable version of DEMKit provides tools for feature rich simulations with a holistic view on smart grids. Within the six months since its development started, the DEMKit platform is already embraced within the Computer Architecture for Embedded System research group of the University of Twente. Currently, simulations for several research papers are ran on it and new control concepts are tested. Furthermore, a real-world co-simulation currently runs stable within a household. In this co-simultion, measurement data of household loads and a solar panel are included, together with a washing machine that can be delayed by the control system, and a virtual battery with peak-shaving as objective for this household.

Further improvements are more realistic time management, such as the supertime concept used in Ptolemy [7]. Also adding an event based message bus, as used in OpenHAB, would improve the flexibility of the platform.

References

[1] OpenHAB, [Online] Available: http://openhab.org.
[2] InfluxDB, [Online] Available: http://influxdata.com.
[3] Grafana [Online] Available: http://grafana.org.
[4] G. Hoogsteen, A. Molderink, J.L. Hurink and G.J.M. Smit, "Generation of flexible domestic load profiles to evaluate Demand Side Management approaches", 2016 IEEE International Energy Conference (ENERGYCON), Leuven, 2016.
[5] M.E.T. Gerards, H.A. Toersche, G. Hoogsteen, T. van der Klauw, J.L. Hurink and G.J.M. Smit, "Demand side management using profile steering", 2015 IEEE Eindhoven PowerTech, Eindhoven, 2015.
[6] K. Kok, "the PowerMatcher: Smart Coordination for the Smart Electricity Grid", PhD dissertation, Vrije Unviversiteit Amsterdam, 2013.
[7] J.T. Buck, et al. "Ptolemy: A framework for simulating and prototyping heterogeneous systems", 1994.