Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

MOSAIK – SCALABLE SMART GRID SCENARIO SPECIFICATION

Steffen Schütte
Michael Sonnenschein

OFFIS – Institute for Information Technology
Escherweg 2
26121 Oldenburg, GERMANY

ABSTRACT

The development of control strategies for the Smart Grid, the future electricity grid, relies heavily on modeling and simulation for being able to evaluate and optimize these strategies in a cost efficient, secure and timely way. To generate sound simulation results, validated and established simulation models have to be used. If these models are not implemented using the same technology, the composition of simulation models is an interesting approach. We developed a composition framework called mosaik, which allows to specify, compose and simulate Smart Grid scenarios based on the reuse of such heterogeneous simulation models. In its current version, it is suitable for the analysis of Smart Grid issues that can be observed using discrete-time or discrete-event based models. In this paper we focus on the presentation of a scalable (in terms of simulated objects) scenario definition concept based on a formal simulator description presented in earlier publications.

1 INTRODUCTION

To create a sustainable power grid, Germany and many other countries have started efforts to increase the share of electricity generated by renewable sources, such as wind turbines and photovoltaics (PV), i.e. electrical power from sun light. To keep the power system stable and reliable, these fluctuating resources need to be accompanied by a number of measures to keep electricity supply and demand at equilibrium. These measures include the usage of storages as well as the exploitation of consumer flexibility (Demand-Side-Management, DSM). As a consequence, the future power grid needs to be able to collect information about the ever growing number of distributed energy resources, storages and controllable consumers and has to manage these resources in a smart way, hence the name Smart Grid. Control strategies for this complex and new task still need to be developed and in particular evaluated and tested, for example with respect to grid stability or other objectives. To ensure that this testing can be done as economically as possible and especially without jeopardizing the reliability of today's grid, these control strategies need to be tested in simulated Smart Grid scenarios first.

In order to yield sound and scientifically reliable results, simulations have to rely on valid and (ideally) established models. As a consequence, a lot of effort is put into the modeling and validation of both single system components such as PV or wind energy converters and composite sub-systems, e.g. entire low or medium voltage power grids. Therefore, it is desirable to reuse existing models in new projects and simulation studies as much as possible. If the existing models are implemented on different technological platforms, for example because each model uses a platform that is ideal for the specific problem (e.g. load flow estimation) or because models are provided by different project partners, simulation composition is an interesting approach. Composition is the "capability to select and assemble simulation components in various combinations into simulation systems" (Petty and Weisel 2003). The challenges for composing Smart Grid scenarios lie in the large number of objects (in the following called entities) that a scenario can be comprised of. Existing composition approaches (see Section 2) usually require a manual specification

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

of the connections between different models or simulators. This is not feasible for large-scale Smart Grid scenarios. Therefore, we developed a domain specific framework called *mosaik* (Schütte, Scherfke, and Tröschel 2011; Schütte, Scherfke, and Sonnenschein 2012) which allows the automatic composition of (time-stepped) large-scale Smart Grid simulation models as a test bed for control strategies based on the reuse of existing simulators. This also includes the use of available commercial simulation packages. Depending on the connected power grid simulator, mosaik can be used for the simulation across different voltage levels. By now, mosaik is designed for the analysis of issues that can be observed using discrete-time or discrete-event based models which have a resolution of 1 second or more. The concept is based on six different layers shown in Figure 1, inspired by the M&S architecture (Zeigler, Kim, and Praehofer 2000, p. 496).

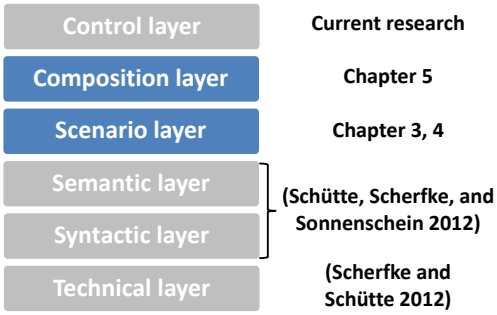

Figure 1: Layers of the mosaik concept.

The technical layer provides a mechanism to find, initialize and manage the available simulators at runtime. The syntactical layer, as presented in Schütte, Scherfke, and Sonnenschein 2012, offers a generic simulator API to integrate simulators into mosaik. On the semantic layer, parameter, models and entities of each simulator are formally described such that the meaning of the data exchanged via the generic API is unambiguous. These formal simulator descriptions can then be used on the scenario level to formally describe Smart Grid scenarios. Finally, the composition layer performs the actual simulator composition based on the formal scenario and simulator descriptions and the control layer allows to interact with the simulated entities at runtime. The applicability of mosaik for the analysis of issues on sub-second level is subject to future work. This paper focuses on scenario and composition layer. The semantic and syntactic layers have been presented in Schütte, Scherfke, and Sonnenschein 2012 and the technical level is described in Scherfke and Schütte 2012. The rest of the paper is organized as follows. In Section 2 we discuss related work with particular focus on the Smart Grid domain. After domain specific requirements for the scenario definition have been presented in Section 3, Section 4 introduces our approach for a formal, domain specific scenario description. Subsequent, Section 5 shows how we use the formal scenario definition to automatically compose the Smart Grid simulation. Finally, in Section 6 we conclude by discussing the current results and future enhancements.

2 RELATED WORK

Different tools and approaches for simulating Smart Grid scenarios exist. Examples are Smart Cities (Karnouskos and Holanda 2009), a purely agent based simulator, GridLAB-D (Chassin and Widergren 2009), a powerful simulation tool for power systems developed by the Pacific Northwest National Laboratory (PNNL), or HOMER (LLC 2012), an easy to use simulator for designing and analyzing hybrid power systems, which contain a mix of conventional generators and renewable power sources. Although being partly extensible, these tools were designed to be used as single, monolithic simulator, i.e. they are not designed for composing other simulators. In other domains different composition approaches exist. In the military domain these are usually built around the High-Level-Architecture (HLA), such as Hemingway, Neema,

Nine, Sztipanovits, and Karsai 2011; Moradi 2008; Benali and Ben Saoud 2011. In the environmental domain OpenMI (Gijsbers and Gregersen 2005) is a widely used standard for simulation model exchange and composition. However, all these approaches are not ideal for Smart Grid simulation where a large number of entities has to be placed in a power grid in a scenario specific way. The reason for this is that usually the existing composition approaches are made for composing a relatively small number of model/simulator instances. In OpenMI, for example, every single model relation has to be specified manually. Also, in HLA based scenario management the interplay between the entities of the simulators usually requires to include scenario specific logic in the participating simulators (federates) (Löfstrand, Ericsson, Johansson, Strand, and Lepp 2004). Mosaik follows a different approach, aiming to provide a compact and scalable scenario description on entity level and requiring no scenario logic within the simulators.

3 SCENARIO REQUIREMENTS

As "new approaches [for simulation interoperability] are unlikely to be accepted by the M&S industry if they are connected with tremendous migration costs due to reprogramming efforts" (Tolk and Muguira 2004) mosaik aims to be as lightweight as possible by focusing only on domain specific requirements for composing Smart Grid scenarios. Figure 2 shows how such a scenario may look like.

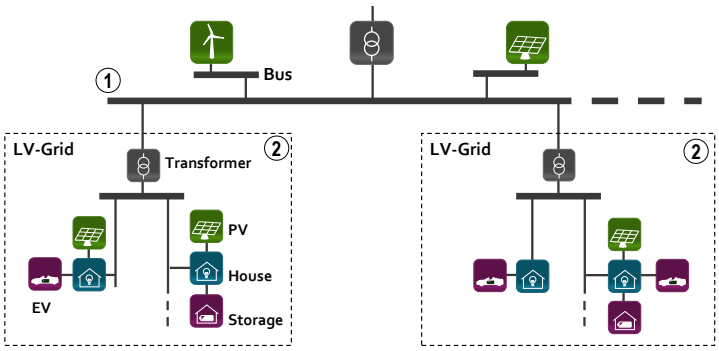

Figure 2: Example of a simple Smart Grid scenario.

The electricity grid, as the critical part of the Smart Grid, plays a central role in each scenario. Other entities are arranged within this grid in a (mostly) static topological fashion. In this example, renewable sources as well as two low voltage (LV) grids (2) are connected to a medium voltage (MV) grid (1). The low voltage grids are comprised of different types of loads, storages and energy sources. For power flow analysis, the grid is usually simplified into a Bus-Branch model, i.e. other elements such as breakers or protection equipment are not modeled explicitly. Different requirements can be derived from this scenario.

Besides the power grid, entities of consumers, producers and renewable energy sources are the building blocks of the scenario, which need to be connected to different nodes of the grid topology. EVs are the only entities that move from one point of the grid to another. This means that the connection point of these entities depends on their state (e.g. their location). It also has to be considered that some entities may depend on other entities. For example, in a residential scenario, PV systems may only be installed on top of a house and not stand-alone. Therefore, PV systems may only be connected to those busses of the grid that have a house connected to it, as well. The major requirement, however, is related to the size of the scenarios which may well involve several thousand entities. To handle such large scenarios, the mosaik scenario definition must allow to distribute a large number of entities within a grid topology in a random fashion so that no manual specification of the connections is required. To obtain realistic scenarios for this procedure, the scenario definition must allow to define constraints for the distribution as well as the dependencies between entities. For example, to specify the aforementioned constraint that the PV modules must only be connected to those busses that have a connection to a house (no stand-alone PV systems) and that there must not be more than one PV module per bus. Finally, certain patterns occur often in larger

scenarios (e.g. use LV-grids to build a MV-grid scenario). For such cases it shall be possible to capture these patterns as scenarios and (re)use them hierarchically, for example to use compositions of low voltage grids as building blocks on the higher voltage levels, as shown in Figure 2.

4 SCENARIO DEFINITION

In Schütte, Scherfke, and Sonnenschein 2012 we presented a generic Smart Grid simulator API, called SimAPI, for the syntactic level and a domain specific language (DSL) called MOSL (MOsaik Specification Language) that allowed to formally capture the semantics and structure of the simulators implementing the SimAPI on the semantic level. We implemented MOSL with the powerful Xtext framework (Xtext 2012). Heart of the semantic level is a reference data model that defines dynamic and static data items for the entities that are part of the simulation models. Figure 3 depicts this structure.

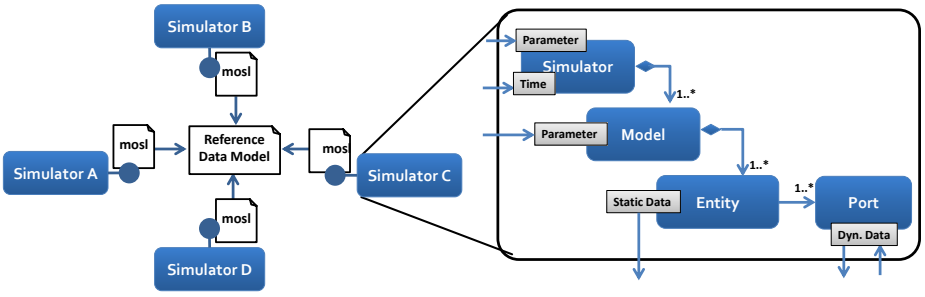

Figure 3: Reference data model and simulator structure as presented in (Schütte, Scherfke, and Sonnenschein 2012).

For the entities, the dynamic input and output data defined in the reference model are grouped into ports and two entities can be connected when all incoming and outgoing flows can be mutually satisfied. To compose a Smart Grid scenario, the scenario level must allow to specify which entities to connect. In this paper we therefore extend MOSL by several concepts that allows to describe these connections in such a way that the requirements found in the last section are met. One of these concepts is a *composition*. It can be split into three sections. First, parameter sets are defined. A parameter set defines parameters for a simulator, its models or for other compositions, in case of hierarchical compositions (see 4.4). Second, so called "entity sets" are created by specifying the number of model instances that are to be created and used within the scenario. Finally, rules for establishing the connections between the entities of two entity sets are defined. The following sections describe these three steps in detail. It is important to understand that these steps are performed during scenario design-time, i.e. before any simulator is actually instantiated and used. This is possible, because the formal simulator description provides sufficient semantics about the models and the entities it can provide.

4.1 Parameter Sets

As depicted in Figure 3, the simulator as well as its models may offer one or more configuration parameters. For being able to use a simulator and initialize its models, values for these parameters have to be provided. The scenario definition allows to specify these parameter values. As we cannot preclude that the choice of parameter values for a simulator does not influence the behavior of a model, the model parameter values are defined subordinate to the simulator parameter values.

Definition 1 A *simulator parameter set* contains a number of (non redundant) simulator parameter values and a number of subordinate model parameter sets.

Definition 2 A *model parameter set* contains a number of (non redundant) model parameters and specific values for these.

Definition 3 A *composite parameter set* contains a number of (non redundant) composition parameter values.

For both, simulator and model parameter sets, values have to be assigned to all parameters of the corresponding semantic description (simulator or model), except for those having default values defined. Listing 1 shows how the definition of a parameter set for an EV simulation model may look like. Due to space limitations the examples in this paper only show snippets of the scenario definitions. The complete code for all examples as well as the simulator definitions from the semantic level are available at http://mosaik.offis.de/downloads/mosaik_wintersim2012.zip. The keyword *sim* starts the simulator parameter set followed by a name for the parameter set and the simulator name (from the simulator description). The model parameter set has a similar structure. The composite parameter set is shown in Section 4.4. The DSL editor provides consistency checks such that the user can neither specify wrong parameter names nor values with a wrong data type.

Listing 1: Parameter sets for an EV simulation.

```
composition example_scenario
  ev_battery_capacity:float in Wh //A composition variable of type float
  //Simulator parameter set
  sim evSim_params:EVSim with stepsize 1 //unit as in simulator description
      start_date = 2010-01-01 00:00:00
      end_date = 2010-01-31 23:59:59
      //Model parameter set
      model config ewe_e3:EVModel
        init_soc = 0.6 //Overwrite default value
        c_bat = ev_battery_capacity //optional: * <offset> + <scale>
        chargingPower = 11.0
      end
      //More model parameter sets, e.g. for other EV types
    end
end composition
```

Listing 1 also shows how parameters for a composition (here: *ev_battery_capacity*) can be defined which can be used instead of constant values within the composition. When using such a parameter, an optional scale and offset can be specified. Using scale and offset to transform a value is an accepted concept in automotive bus system, as well as in other simulation standards, for example to convert between units (e.g. from Celsius to Fahrenheit (MODELISAR 2011, p.33)). Section 4.4 shows an example where this feature is used.

4.2 Entity Sets

Now being able to parametrize the building blocks of our composition, we need to specify how many instances of a model (or composition) are required for the scenario. However, we are not interested in the models themselves but rather in the entities the models contain. We therefore introduce the notion of an entity set.

Definition 4 An *entity set* is a named container for all entities that result from the instantiation of a number of models or compositions with the same parameter set.

Listing 2 shows how entity sets are created by referencing a specific model parameter set. In the example an entity set called *evs* is created which represents all entities of 4 model instances of the *EVModel* with the configuration values of the referenced parameter set. The specific simulator and model do not need to be specified as this information can be derived from the referenced parameter set. In case of the EV model the entity set simply contains 4 *Vehicle* entities. However, the example also shows the definition of an

entity set called *grid* representing the entities of a power grid model. This entity set may contain a number of entities of different types, e.g., Bus, Branch and Transformer entities. It is important to understand that at this point in time (design-time) there are no entities in the entity sets that are actually simulated. Rather the possible entity types within the set can be derived from the semantics of the simulator description. At run-time (see Section 5) we use the term *entity set instance* to avoid any confusions.

Listing 2: Creation of entity sets.

```
//<entity set name> = <cardinality> new <parameter set name>
evs  = 4 new ewe_e3 //4 instances of the EV model
grid = 1 new grid_parameter_set_name //e.g. 1 a low voltage grid model
```

4.3 Connection Rules

Connection rules are the heart of the mosaik scenario concept. As mentioned in the introduction, the mosaik scenario definition shall allow the definition of large-scale scenarios. Besides the possibility of hierarchically reusing composition descriptions (4.4), this scalability is achieved by connection rules which can operate on a potentially large number of entities. They allow the user to specify rules that are followed by the simulation composition engine for establishing n:1 connections between the entities. Each rule operates on two entity sets. For each entity set a type selector has to be specified such that only a single type of entities is used from each set. Further, the multiplicity for the first entity set has to be specified. For example, 1..3 entities of entity set A shall be related to each entity of entity set B.

Definition 5 A connection rule is a 8-tuple $< es_1, et_1, es_2, et_2, min, max, cond_{stat}, cond_{dyn} >$ with $es_1, es_2 \in EntitySets; et_1 \in types(es_1); et_2 \in types(es_2); lower, upper \in \mathbb{N}; lower \leq upper)$ where $EntitySets$ are the defined entity sets and $type$ is a function that returns the entity types occurring in each set (based on the semantic simulator description).

For describing complex scenarios, a connection rule allows to specify different conditions ($cond_{stat}, cond_{dyn}$) that will be introduced in the next sections. Listing 3 shows a basic connection rule connecting up to 3 *EV* entities of the *evs* entity set to *Bus* entities of the *grid* entity set defined in the last section. The DSL editor supports the user by allowing to choose only those entity types on the right hand side of the rule that have compatible data flows to the left hand entity type. This can be done based on the information provided by the simulator description and the reference data model.

Listing 3: A basic entity connection rule.

```
//connect <multiplicity> <set 1>.<type 1> to <set 2>.<type 2>
connect 0..3 evs.Vehicle to grid.Bus //0..3 EVs to 1 Bus
```

To achieve scalability (the code in Listing 3 would be the same for 1.000 vehicles in a larger grid, for example) a random selection of entities from the entity sets in the connection rules is performed. However, it is also possible to select specific entities based on their IDs, to allow the modeling of very specific scenarios. How the connection rules are interpreted at composition-time will be shown in Section 5.

4.3.1 Topological Conditions

In many cases, whether two entities shall be connected or not depends on other connections that have already been established. For example, in a scenario where PV modules are connected to certain busses of the grid (e.g. at EV charging poles) one wants to limit the connection of EVs to only those busses that have PV modules connected as well. For such cases, the mosaik scenario concept offers a function to count the number of entities of a specific type that are connected to an entity occurring in an entity set of the connection rule. Listing 4 shows how such a condition can be described using the DSL. Heart of this count function is a concept called *EntityPath*. An entity path allows the user to specify a path in the entity

topology via entities of specific types and (optionally) from a specific entity set. At composition time, when evaluating the connection rule, such a path definition results in a set of all entities that are reachable via this path. In this example we count the entities of type *PVModule* related to (--) the potential *Bus* to which the *Vehicle* can be connected. The count function syntax uses vertical bars known from set theory. Again, the DSL editor supports the user by only allowing to specify those relations that have been established before by other connection rules. While this is a simple example for illustration purposes, the DSL allows to specify deeper entity paths with multiple relations, e.g. *where |typeUsedInRelation--set1.type1--set2.type2--...| > intValue*. Also, multiple count functions can be combined arbitrarily using AND and OR operators.

Listing 4: Connection rule with topological condition.

```
//Connect PVs to busses first
connect 0..1 pvs.PVModule to grid.Bus
//Now connect EVs to busses having a PV connection
connect 0..3 evs.Vehicle to grid.Bus where |Bus--pvs.PVModule| > 0
```

4.3.2 Static Data & User Defined Functions

For providing even more flexibility, the conditions of a connection rule can be extended by comparisons of the entities static data. Figure 4 shows a scenario where such functionality can be used to keep the scenario specification short. It includes three different models, a power grid model, a PV model (both known from the former examples) and a weather model that models the solar irradiation in a cell based fashion for different geographical regions. The cells have to be connected as input to the PV modules to refine the overall scenario.

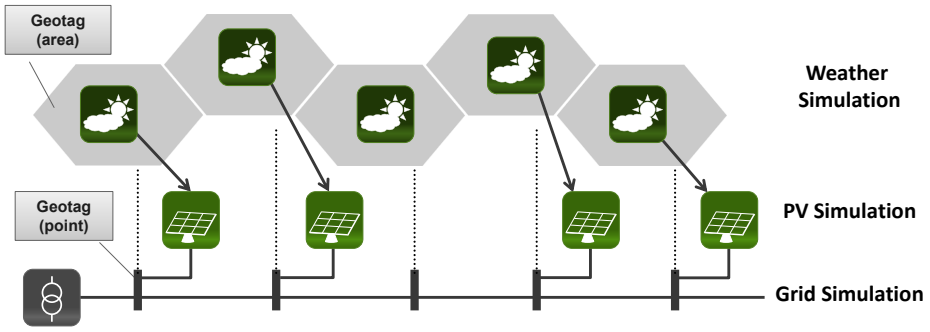

Figure 4: Creating relations based on derived static data.

We assume that our grid model offers static data about the geolocation of each bus. The weather model has *WeatherCell* entities representing the sun irradiation in a certain area. Each of these entities provides its covered area as static data, as well. For large scale scenarios it would be cumbersome and error prone to connect weather cells and PV modules manually.

Listing 5: Connection rule with user defined function for comparing static entity data.

```
//Connect PVs to busses first
connect 0..1 pvs.PVModule to grid.Bus
//Now connect WeatherCells to PVModules having a Bus inside the cells area
//PVModules are on left side of the statement due to n..m:1 constraint
connect 0..* pvs.PVModule to weather.WeatherCell
    where contains(PVModule--grid.Bus.positionPoint, WeatherCell.area) == True
```

Listing 5 shows how these connections can be specified easily by applying a user defined function called *contains* that checks if a point is within a given area. It has to be defined in the reference data model

and an implementation has to be provided to the mosaik engine as a plug-in. In this case it is checked if the position of the *Bus* connected to (--) the *PVModule* is within the area of a *WeatherCell*. At composition time, this will result in a connection of zero or more PV modules to 1 weather cell which matches the PV modules location. The latter is derived from the static location attribute of the Bus it is connected to. Again, the concept of entity paths (see 4.3.1) is used.

4.3.3 Dynamic Connections

EVs are a very special type of entity in Smart Grid scenarios, as these are the only type of equipment that can move between different nodes (charging locations) of the power grid. So far, to support this functionality, mosaik allows to specify dynamic relation conditions. In contrast to the count and user function conditions presented above, which are evaluated once at composition time and may only use static data items, the dynamic conditions are reevaluated every simulation step and operate on the dynamic data flows of the entities. Listing 6 shows how the EV connection rule of the scenario shown in Listing 4 can be extended in such a way that the vehicles are only connected to the PV nodes when having a certain location (assuming *location* is a dynamic data flow of type *string* defined in the EV entity definition of the simulator description).

Listing 6: Dynamic connection conditions.

```
//EV.location can be one of {"charge_pole", "other", ...}
connect 0..3 evs.Vehicle to grid.Bus where |Bus--pvs.PVModule| > 0
                                    when Vehicle.location == "charge_pole"
```

4.4 Hierarchical Modeling

The descriptions of the compositions can be nested hierarchically, i.e. used in other composition descriptions. This can be useful to keep the overall scenario definition small. As seen in Figure 2, for example, one may define a certain number of low voltage distribution grids (e.g. for the different agglomeration types city, rural, commercial) and then reuse these multiple times in a larger scaled medium voltage grid scenario (maybe with different parameters influencing the number of entities deployed in each LV-grid). In such a case the hierarchy of the compositions is directly related to the topology of the desired simulation scenario. Another use case for using hierarchical compositions is presented in this section: The grouping of entities that have dependent parameters. Figure 5 shows such a scenario.

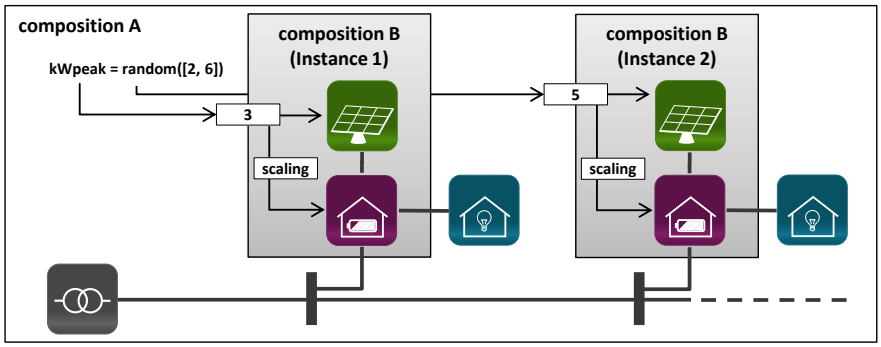

Figure 5: Example for hierarchical scenario modeling.

It consists of a power grid simulation with n nodes (only 2 are shown), each representing a connection point of a house with a PV system. To be realistic, the different PV systems have different sizes. Let us assume the goal of the scenario is the evaluation of a concept for local, battery based storages placed in each house. More precisely: For saving costs, the battery buffer shall be kept as small as possible by scaling it

dependent on the peak power of PV system of each house. Further, we assume that we have independent simulation models (grid, house, PV system, battery buffer) which we want to use as building blocks for the scenario. As it is inconvenient to specifying the exact size of the PV systems for a large number of houses, we use a mosaik feature that has not been introduced, yet. Instead of using constants as parameter values it is possible to specify random distribution functions. These can be parametrized to operate on a range of values (using square brackets) or to select from discrete values (using curly brackets) and are evaluated for each time a specific value is required (e.g. when passing the parameters for instantiating a model to the simulator). In this case we can parametrize it to achieve an equal distribution of PV system peak power configurations ranging from 2 to 6 kW_{peak}. As the battery buffer shall be scaled accordingly, we need to use this randomly generated value for configuring it as well. Therefore we place both, battery and PV model in a composition B defining the PV peak power as parameter for this composition. This parameter can then be used directly for parameterizing the PV model and, using the scale/factor feature (see 4.1), we can scale it for parameterizing the battery buffer as well. Now one can use this composition B in the composition A where its peak power parameter is assigned to the desired random distribution function. By placing 1 PV and 1 battery buffer in a separate composition we achieve different random based values for the PV peak power (and thus the battery size) which would not be possible when placing all in the same composition, as the parameter of PV and battery model are not dependent on each other. Another example for such dependent entity parameters would be the deployment of differently sized combined heat and power (CHP) plants and an appropriate thermal storage for these. Furthermore, please note that the hierarchy vanishes at run-time, i.e. after the composition has been accomplished. So for the control strategies integrated on the top layer of the mosaik framework, only the resulting entity topology is visible/relevant. Listing 7 shows how compositions can be used in connection rules, using the -> operator.

Listing 7: Hierarchical composition.

```
composition bufferAndPv_parameter:compositionB
  p_peak = random([2.0, 6.0])
end
grid = 1 new grid_parameter_set_name
houses = 80 new house_parameter_set_name
bufferAndPVs = 80 new bufferAndPv_parameter
connect 1 houses.H0 to bufferAndPVs->buffers.Battery
connect 1 bufferAndPVs->buffers.Battery to grid.Bus
```

4.5 Simulation Study

Finally, for being able to actually simulate a scenario, the root composition has to be specified and potential parameters have to be provided with specific values. Therefore we introduced the concept of a simulation study, implying that a study may include several simulation runs with different parameters. For example, to simulate different seasons of the year. MOSL therefore allows to specify a list of values for each parameter of the root composition and can automatically combine these in a linear or Cartesian fashion.

5 COMPOSITION

The concepts presented so far are used to formally describe a Smart Grid scenario and do not involve the execution of a simulator. For actually simulating the scenario, a *Simulation Study* is given to the mosaik simulation engine. As the engine is implemented in Python, we developed a code generator (available at https://bitbucket.org/sschuette/xtextdsl2python) for Xtext that serializes the DSL files to the Python readable YAML format. First, the engine collects information about all entity sets (*new* statements) so that the overall number of simulation model instances is known. This is required as mosaik tries to minimize the number of simulator instances by executing as many model instances as possible on a single simulator, depending on the specified simulator capabilities. Next, the simulator processes are started and the different

models are initialized with the parameters referenced in the entity sets. If this was successful, the actual entity instances (grouped into entity set instances) and their static data are available and the connection rules can be applied. Finally, based on the connections and the resulting data flows, the simulators need to be scheduled such that they are stepped in the correct order. However, in the following we like to elaborate on the application of the connection rules, as this is specific to our approach (as compared to the scheduling problem).

Basis for applying a connection rule are two entity set instances. As shown in Section 4.3, the connection rule provides a type selector that is applied to the entity set instances first such that two new, homogeneous sets result. Figure 6 (A) shows the resulting two sets (in the following called $left$ and *right*) for the entity sets defined in Listing 2, which consist of EV and Bus entities.

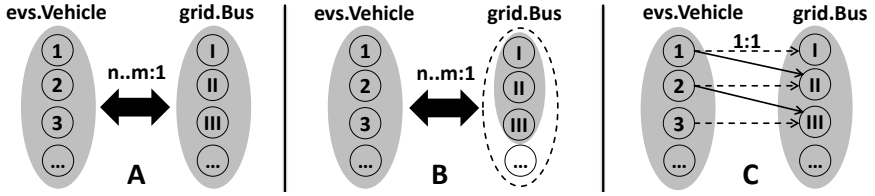

Figure 6: Applying connection rules to entity sets.

Mosaik now relates these entities in a random fashion (if no specific IDs are given for the connection rule) such that lower and upper bound of the multiplicity given in the connection rule definition are satisfied. Currently we have implemented uniform distribution. However, we plan to add other distribution functions which is required to model certain scenarios in a realistic way. For example the number of vehicles per household is not distributed equally (most have 2, few have 3, ...). Although this approach sounds trivial, there are a number of problems that arise which we like to show in the remainder of this section.

In general, three cases have to be distinguished. First, a connection statement may have no static condition (see 4.3.1, 4.3.2) at all. In this case, shown in Figure 6 (A), any two entities can be randomly chosen and connected if the lower and upper bound is not violated. Second, a condition is specified but only involves operations on one of the two sets (6 B). In such cases, the entities that do not meet the condition are removed from the set (here all but entity I/II/II) and again, any two of the remaining sets (gray) can be related. For these variants (A and B) it can easily be checked if the lower and upper bound can be met even before the entities are actually connected, just by taking a look at the number of entities in the sets. The following constraints must be satisfied:

$$\begin{aligned} WithinLowerBound(left, right) &: |left| \geq lower \cdot |right| \\ WithinUpperBound(left, right) &: |left| \leq upper \cdot |right| \end{aligned}$$

Finally, a condition may operate on entities from both entity sets, e.g. the user function shown in Listing 5. In such a case, the condition has to be checked for all entity tuples of the Cartesian product of the entity sets. As this can be very time intensive (O(n^2)), mosaik analyzes the connection rule beforehand and selects the required strategy. However, there are also methods to reduce this complexity by clustering entities of one set into groups with identical attributes, thus reducing complexity to O($n \cdot |cluster|$). Other optimization techniques known from the database domain, such as predicate migration (Chaudhuri and Shim 1999) or function caching, are currently being investigated and beyond the scope of this paper. More interesting is the fact that there can be situations where the lower and upper bound for connection rules with conditions involving both entity sets cannot be guaranteed. If they can be met may vary from one run to another based on the random seed. Figure 6 (C) shows such a situation with dashed lines showing possible combinations that fulfill the connection rules condition and solid lines showing selected connections. Entities 1 and 2 were connected to II and III, respectively. However, entity 3 cannot be connected to any other (only III is possible) without violating the 1:1 multiplicity constraint although another combination (i.e., 1-I, 2-II, 3-III) would be possible. Although this may only occur in rare cases it has to be accounted for.

This situation can be formulated and solved as classical discrete, finite domain Constraint Satisfaction Problem (CSP). Formally, a CSP is defined by a set of variables $V = \{X_1, ..., X_n\}$, a non-empty domain $D_i = \{v_1, ...v_k\}$ of possible values for each variable X_i and a set of constraints $C = \{C_1, C_2, ..., C_m\}$ (Russel and Norvig 2009). A possible solution for the CSP is any variable-value assignment $A : V \rightarrow \bigcup_i D_i$ that does not violate any of the constraints. In our case we can treat the entities of the left set as variables. For each variable, those entities of the right set that fulfill the conditions of the connection rule constitute the domain of possible values. Furthermore we have a single global constraint (a constraint involving all variables) C_{Mult} that is True, if the multiplicity lower and upper bound is met when considering all variable assignments. A standard CSP solver can now be used to obtain valid connections.

$$\begin{aligned} V &= \{l_1, ...l_n | l_i \in left\} \\ D_i &= \{r_1, ...r_n | r_j \in right; cond_{stat}(l_i, r_j) = True\} \\ C_{mult} &: lower \leq |\{l | l \in A^{-1}(r)\}| \leq upper \ \ \forall\, r \in \bigcup_i D_i \end{aligned}$$

6 CONCLUSION

In this paper we presented a scalable and flexible scenario definition approach for topological domains, such as the Smart Grid domain. Scalability is achieved by allowing composition descriptions to be reused in a hierarchical fashion and by defining connection rules operating on arbitrary large sets of entities instead of having to connect single entities manually. The scenario definition is based on the formal simulator description presented in Schütte, Scherfke, and Sonnenschein 2012. This will form the basis of our simulation platform mosaik that allows to compose heterogeneous simulation models into large-scale Smart Grid scenarios. The concepts presented in this paper have been implemented and are currently being applied and evaluated in different research projects. Due to space limitations, not all details of the scenario definition could be shown. The application of the connection rules during the composition process was presented briefly. Its optimization is subject to current research. Future work includes the integration of an energy standard compliant API to access the simulated entities as well as a synchronization mechanism (e.g. such as described by Gehrke, Schuldt, and Werner 2008) for the integration of multi agent based control strategies. Furthermore, the integration of simulators based on the FMI standard (MODELISAR 2011) is being planned.

REFERENCES

Benali, H., and N. B. Ben Saoud. 2011, July. "Towards a component-based framework for interoperability and composability in Modeling and Simulation". *Simulation* 87 (1-2): 133–148.

Chassin, D. P., and S. E. Widergren. 2009. "Market Operations". *Power & Energy Society General Meeting, 2009. PES '09*:1–5.

Chaudhuri, S., and K. Shim. 1999, June. "Optimization of queries with user-defined predicates". *ACM Transactions on Database Systems* 24 (2): 177–228.

Gehrke, J. D., A. Schuldt, and S. Werner. 2008, December. "Designing a Simulation Middleware for FIPA Multiagent Systems". In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence*, edited by L. Jain, P. Lingras, M. Klusch, J. Lu, C. Zhang, N. Cercone, and L. Cao, 109–113: IEEE.

Gijsbers, Peter J. A. and Gregersen, Jan B. 2005. "The OpenMI Standard in a nutshell". http://www.openmi-life.org/downloads/documentation/the_openmi_standard_in_a_nutshell.pdf [accessed September 10, 2012].

Hemingway, G., H. Neema, H. Nine, J. Sztipanovits, and G. Karsai. 2011, March. "Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach". *Simulation* 88 (2): 217–232.

Karnouskos, S., and T. N. D. Holanda. 2009. "Simulation of a Smart Grid City with Software Agents". *2009 Third UKSim European Symposium on Computer Modeling and Simulation*:424–429.

HOMER Energy LLC 2012. "HOMER Energy - Hybrid Renewable and Distributed Power Design Support". http://homerenergy.com [accessed September 10, 2012].

Löfstrand, B., M. Ericsson, M. Johansson, J. Strand, and H. Lepp. 2004. "Scenario Management - Common Design Principles and Data Interchange Formats". In *European Simulation Interoperability Workshop (SIW), 04E-SIW-070*, Volume 46.

MODELISAR 2011. "Functional Mock-up Interface for Model Exchange and Co-Simulation 2.0 Beta 3". http://www.modelisar.com/specifications/FMI_for_ModelExchange_and_CoSimulation_v2.0_Beta3.pdf [accessed September 10, 2012].

Moradi, F. 2008. *A Framework for Component Based Modelling and Simulation using BOMs and Semantic Web Technology*. Ph.d. thesis, KTH Stockholm.

Petty, M. D., and E. W. Weisel. 2003. "A Formal Basis For a Theory of Semantic Composability". In *Proceedings of the Spring 2003 Simulation Interoperability Workshop, Orlando, FL, April*, 416–423.

Russel, S. J., and P. Norvig. 2009. *Artificial Intelligence: A Modern Approach*. 3rd ed. Prentice Hall.

Scherfke, Stefan and Schütte, Steffen 2012. "mosaik - simulation engine architecture". http://mosaik.offis.de/downloads/mosaik_architecture_2012.pdf [accessed September 10, 2012].

Schütte, S., S. Scherfke, and M. Sonnenschein. 2012. "mosaik - Smart Grid Simulation API". In *Proceedings of SMARTGREENS 2012 - International Conference on Smart Grids and Green IT Systems*, edited by B. Donnellan, J. P. Lopes, J. Martins, and J. Filipe, 14–24: SciTePress.

Schütte, S., S. Scherfke, and M. Tröschel. 2011. "Mosaik: A framework for modular simulation of active components in Smart Grids". In *1st International Workshop on Smart Grid Modeling and Simulation (SGMS)*, 55–60: IEEE.

Tolk, A., and J. A. Muguira. 2004. "M&S within the Model Driven Architecture". In *Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC)*, 1–13.

Xtext 2012. "Xtext". http://www.xtext.org [accessed September 10, 2012].

Zeigler, B. P., T. G. Kim, and H. Praehofer. 2000. *Theory of Modeling and Simulation*. 2nd ed. New York: Academic Press.

AUTHOR BIOGRAPHIES

STEFFEN SCHÜTTE studied Software Technology in Hannover, Lüneburg and Wolverhampton. He received his Master degree in 2007 at the Leuphana University of Lüneburg. After working as a software developer in the automotive industry for two and a half years he decided to join the Smart Grid research team at OFFIS in Oldenburg, Germany as scientific research assistant. Since then he has been working on a concept for modular Smart Grid simulation as a basis for other research projects of the institute. His email address is schuette@offis.de and his web page is http://www.offis.de/en/offis_in_portrait/structure/persons/details/profile/msc-steffen-schuette.html.

MICHAEL SONNENSCHEIN studied Computer Science and Mathematics at the Aachen University of Technology (Diploma 1979); PhD in computer 1983, Habilitation' in Computer Science 1991, both at Aachen University of Technology. Since 1991 he is professor for Computer Science at Oldenburg University, first as head of a working group on programming languages and systems, since 2001 as head of a working group on Environmental Informatics. As a member of the executive board 'energy' of the OFFIS Institute for Information Technology he headed several projects on energy management in smart grids. His research interests include techniques for modeling, simulation, and heuristic optimization in environmental applications. His email address is sonnenschein@offis.de and his web page is http://www-ui.informatik.uni-oldenburg.de/1409.html.