An Open Data-Based Discrete Simulation Environment for Testing Smart Grid Messaging

Eric MSP Veith[1], Bernd Steinbach[2] and Jürgen Otten[3]

[1,3]Institute of Computer Science
Wilhelm Büchner Hochschule
Pfungstadt, Germany
e-mail: eric.veith@wb-fernstudium.de

[1,2]Institute of Computer Science
Freiberg University of Mining and Technology
Freiberg, Germany
e-mail: veith@mailserver.tu-freiberg.de

***Abstract*—Including more renewable energy sources in the energy mix will increase the necessity for a finer grained, automatic control of changes in the energy level. Any such software needs extensive testing before it can be released for general availability. Simulation environments will be a part in these testing stacks, but need realistic input data in order to yield expressive and therefore useful results. However, comprehensive input data is often not available or fragmented. We therefore propose a simulation environment that can use open data sources, switch them dynamically, and attribute the testing results with a possible data quality impact.**

***Keywords*—smart grid; simulation; messaging; open data; renewable energy sources**

I. Introduction

The shift from traditional, fossil energy sources to renewable ones such as photovoltaic or wind poses challenges on the existing energy grid. The inclusion of these renewable energy sources becomes more and more difficult with higher number of deployments. A multi-agent based approach has been suggested, for example, in [1], in order to counter the increased complexity in grid management.

Simulation runs have been widely accepted as a means to test software, especially network-based, distributed software architectures. While unit testing can assert the correct functional behavior of a particular module, or unit, of the software being tested, the interaction of several instances of the complete software architecture cannot be asserted by unit testing alone. Several simulation environments have been proposed, both generic network simulators, such as OMNeT++ [2], and smart grid-specific solutions, such as MASGriP [3].

Descriptive and therefore successful testing always depends on the right choice of test data. In the case of renewable energy sources, a large portion of these simulation data is formed by weather measurements. Even source data regarding a single topic such as the aforementioned weather measurements can be heterogeneous, for example, when a portion of the data has a high resolution while another portion of it only offers a low resolution. But a testing environment usually does not explicitly care about the actual source of data; the task of asserting data quality therefore falls to the scientist. He, in turn, typically handles data quality differences by partitioning the source data and, therefore, doing different test runs.

Data quality is especially important when using open data. Using open data can be useful or even necessary for different reasons, like, e.g., budget limits. In this paper, we propose a simulation environment for smart grids that utilizes open data and annotates simulation runs based on these sources and can continue even if the data source is fragmented or incomplete.

II. Motivation

In [4], we have proposed a lightweight smart grid messaging protocol for a distributed agent-based environment that, by design, treats all items within the energy grid equal, but forms microgrids automatically. Using this protocol, consumers and producers interact with each other and calculate demand and supply in a distributed manner. The nodes in the grid, which are represented by agents, commit themselves to deliver or consume a certain amount of energy based on the preceding message exchange. The goal of this system of distributed agents, which exhibit these messages, is to use renewable energy sources more effectively. In order to do so, short-term forecasts are used in a grid-wide, distributed planning phase that leads to an automated, more dynamic grid management.

Since the location of some types of renewable energy sources, such as wind farms, are dictated by the source itself—a wind farm must be built on sites where the wind currents are strong and steady—, we expect the general behavior of smart grid agent messaging to show patterns based on locations.

As such, a simulation environment for this kind of smart grid distributed agent approach must focus on map positions. Hence, it becomes tightly coupled to spatial information, which, in turn, allow us to treat simulated items more effectively. Consider, for example, wind speed measurements. These are valid for a certain area and thus can be applied to any number of items within this area. Without this spatial relationship, each simulated item would have to look up measurements individually, thereby causing more queries and calculations being made. In fact, each participant in our simulation environment needs to be locatable and thus automatically becomes an item on the map when entering the environment.

Having spatial data, i.e., a map, as the common basis also allow us to use realistic weather data. Although [5] suggests that many typical weather conditions can be synthetically modeled, the use of real measurements allow us to test agent behavior for location-specific weather conditions and phenomena.

 ISBN: 978-1-61208-332-2

The most precise information source for weather data is typically a national weather service. Often, however, data needs to be bought. Supplying data to a long-running simulation for a whole country can therefore imply a financial impact that is not desirable. The same can hold true for spatial data such as the position of wind parks or other power plants.

We therefore propose the use of open data within our simulation environment. But since open data can be less exact, care must be taken. Although this could simply mean setting up a separate simulation run, one would loose the internal state of the simulation environment. A combination of both the more exact but expensive national weather service's and the open data source can therefore be desirable. This combination, however, must be carefully augmented in order to try to assert the impact of using different data sources throughout one single simulation run.

Transparently switching between different data sources allows us to take advantage of the more exact data whenever possible and still enables us to have a long-running simulation, thereby observing our agent software's behavior and the message exchange caused by it over a longer time, which potentially yields more diverse data. The assessment additionally shows the impact this switching to a less exact data source had on a particular simulation run.

The remainder of this paper is structured as follows. We briefly outline the general architecture of the simulator in the following Section III, where we outline how the spatial indexing helps us to express the locality of certain events. Afterwards, in Section IV, we show how our proposed environment can be used for larger-scale simulations spanning multiple computing nodes. The extensive use of open data is described in Section V, followed by a discussion of the implications of using open data sources in Section VI. Finally, we conclude and show pointers to future work in the final Section VII.

III. Architecture of the Spatially-Indexed Simulator

Each simulation is controlled by a `Controller` class instance that forms the core of this time-discrete simulation environment. It is responsible for tracking the current time within the simulation and issuing events. `Event` objects are issued during a *tick* and reach all relevant participants within the simulation, which process them and finally return them to the controller. As soon as all have returned, the controller advances to the next time in the simulation at which events are scheduled.

The `Controller` also contains a list of all items participating in the simulation. These items are wrapped in `MapItem` classes. Since every simulation participant is ultimately a `MapItem`, this provides an unified interface between simulation controller and simulated item. By acting as adapters, these wrapper classes feed the native agent interfaces with the input data generated within the simulation environment.

When agents communicate with each other, the same wrapper technique is applied. The simulation provides virtual data connections, which facilitate the transition to the virtual simulation environment. The original agent software can therefore remain unchanged: testing is done as black-box testing. Consequently, agent code is not part of this paper.

Mainly the targets of events such as a simple time change or a simulated sensor reading are `MapItem` objects. This group is constituted of nodes within the power grid that are subject to the simulation and thus our agents that are being tested, namely, power lines, substations, consumers or power generators. However, all data sources that deliver input like, for example, weather data, are also a part of this group.

The simulation replaces actual hardware sensors of an agent by artificial stimuli. These stimuli are modeled as discrete events within the simulation. They also have coordinates attached to them: A wind speed measurement, for example, is valid for a particular region; the same applies to a load profile. The sources for artificial sensor data provided by the simulation are thus instances of the `MapItem` class, called `ValueGeneratingMapItem`. Their coordinates are constituted by the area they provide valid data for.

Consequently, each `Event` instance created by such a map item also has an *area of effect* for which it is valid. Any event is thus delivered to those map items that reside inside this area. Thus, events can be seen as tuples (A, V), containing the area of effect and a value. Typically, an event's area of effect equals the polygon forming the position of the map item. For example, for wind speed changes, this is the region for which the measured data are valid.

Finding a particular item on the map is done by using a R*-Tree structure [6]. For each item, its coordinates are reprojected to WGS84 [7], if necessary. WGS84 is the shorthand symbol for the Spherical Mercator projection, used, e.g., by the Global Positioning System (GPS). It uses latitude/longitude values instead of metric units and thus forms a common basis for all coordinates stored in the database.

After reprojecting, a bounding box for the new coordinates is computed. The bounding box is the rectangle used by the R*-Tree structure, into which it is then inserted.

Thus, the proposed simulation environment is able to integrate the artificial stimuli in the same manner as it integrates the actual agents being tested, which allows for greater flexibility and simplicity in the overall design of the underlying software architecture.

Prior to running, the simulation environment is configured via an object of the class `Description`. This class' attributes contain the information required in order to set up the simulation itself, which are the simulation's start and possibly stop time—the latter only if the simulation does not run endlessly—, a polygon designating the area within which the simulation takes place (the "bounding box"), and the definition of a factory class responsible for populating the environment. The latter one is accompanied by an enum set that specifies, for the three categories consumers, power generators and power grid, whether items from the respective category should be

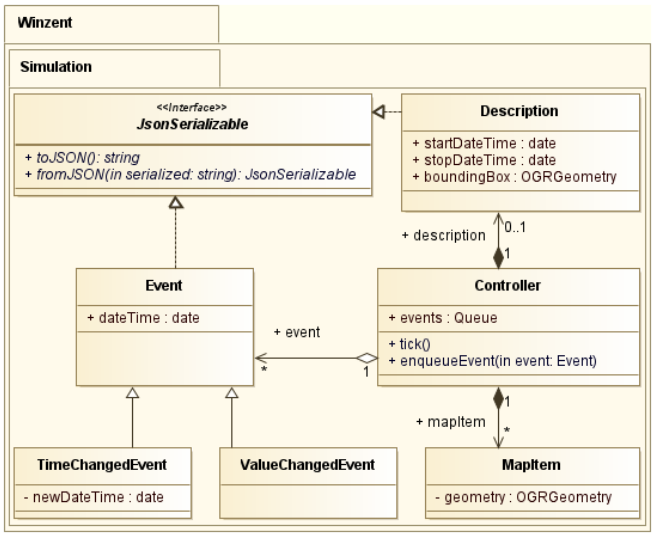


Figure 1. The `Simulation` namespace

created or not.

The description can be serialized to and from JSON [8], allowing an user to configure a simulation run by just creating a description in the text editor and feeding it to the simulator upon start. Thus, "simulation recipes" can be created which allow for repeated simulation runs in a defined manner. This is important for asserting the correct working of a new agent implementation and measuring possible improvements of such a new software version.

This basic layout of the relevant classes is depicted in the UML diagram of Figure 1.

The simulation controller offers another interface based on Qt's signal-slot principle [9], which can be used for other programs to link to the controller and observe the events within the simulation. All classes within the `Winzent::Simulation` namespace form one library; any frontend can link to it and display the simulation and its results in the way it chooses. Thus, simulation logic and view are separated which also enables the simulation to run headless. A headless simulation service can be used for running a clustered simulation as we outline in the next Section.

IV. Distributed Simulation

The simulation environment is multi-threaded; it keeps the actual event processing in separated worker threads to speed up the duration of one tick. When running standalone, the number of worker threads spawned equals the number of CPU cores. Each worker thread has an incoming event queue filled by the controller, as well as an outgoing event queue, wherein processed events are stored.

On local execution, these worker threads also store the map items themselves, i.e., event processing takes place on the local machine within these worker threads. For remote execution, the class `TcpSimulationThread` serializes and deserializes events and transmits them to remote slave workers instead of feeding a calculation.

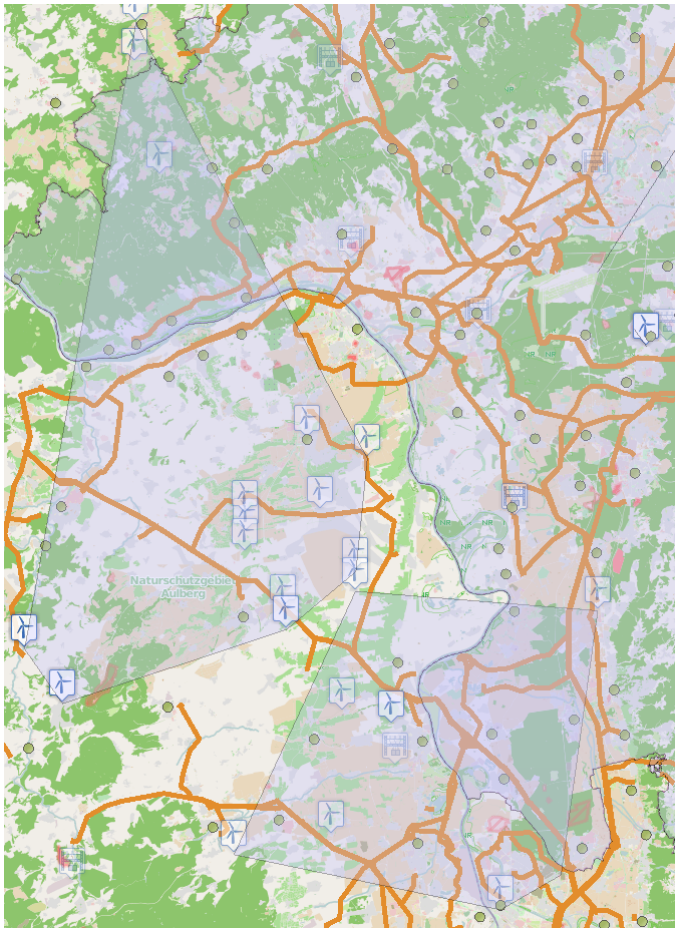


Figure 2. The simulator showing a part of Hesse, Germany, with areas of effect highlighted

These remote slave workers use the same classes as the master machine, i.e., the ones described in Section III. Such a slave machine uses one instance of the `TcpSimulationThread` to receive new events, which are enqueued into the slave `Controller`'s own event queue. The events then get processed on this slave machine via worker threads; processed events are serialized and transmitted back to the master controller.

This way, the simulation test bed can easily span multiple machines although the same classes are used. Since there is per definition one master controller that issues a new tick only once all other events for a simulation time has returned, all machines are in sync at the very beginning of a new tick.

Configuration is done via the `Description` class which also supplies all other parameters as per Section III. The simulation description additionally allows to specify a fixed mapping of a node or a set of nodes to a particular machine. This allows to break out specified nodes and run them on an embedded board, thereby additionally simulating hardware and memory constraints of a real smart grid node.

 ISBN: 978-1-61208-332-2

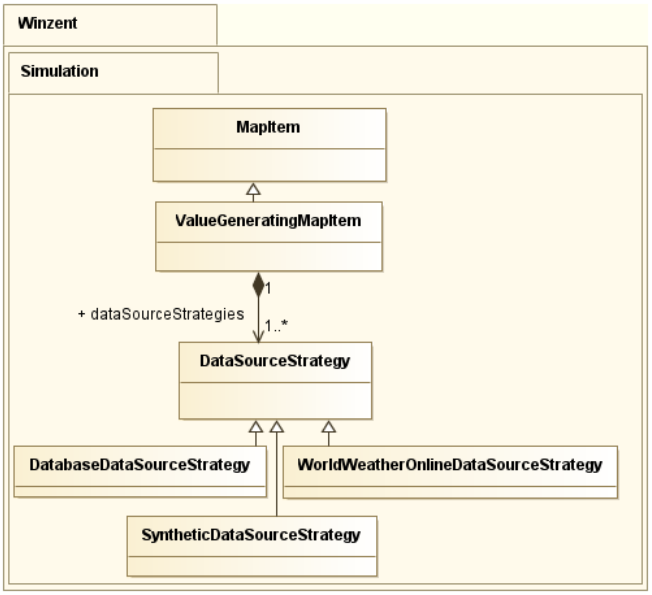


Figure 3. UML class diagram of the data source strategy pattern

V. Use of Open Data

The Winzent simulator does not rely on a single source of data for one particular input. Given, for example, weather data, it can query a national weather service's database or an online data provider such as OpenWeatherMap [10] or WorldWeatherOnline [11]. It is also possible to switch to completely synthetic input value generation.

However, not all sources have the same accuracy. Continuing with the weather data example, a dataset bought from the national weather service is likely to have a higher number of measurements available than an online service, which might serve data at three-hour intervals. As such, the user is likely to prefer the national weather service's dataset to other sources.

The simulation environment allows to express this by simple ordering through priorities. Only if a data source cannot offer any more data, the simulation environment will switch to one with a lower priority. These data sources are an implementation of the strategy pattern [12] used in object-oriented design.

Every time a `ValueGeneratingMapItem` receives a `TimeChangedEvent`, it queries the attached data source with the highest priority for data. If it cannot offer any, the `DataSourceStrategy` instance throws a `DataSourceDepletedException`. This prompts the `ValueGeneratingMapItem` instance to query the next one with a lower priority. This is done until the last data source has thrown an exception, which is then escalated to the `Controller`, which finally stops the simulation: A simulation which is not completely covered by valid data has no use in going on. Figure 3 shows the class architecture accomplishing this behavior.

While this simple mechanism allows the simulation to proceed even if one data source is depleted, the potential change in data quality obviously has a huge influence on the final results of the simulation run. An assessment of the data quality has to be done; at the least, the user has to be notified of the change.

Therefore, each data source has a derivation from the optimal data source attached to it. The optimal data source is, per definition, the first data source in the queue of possible data sources. Each other data source therefore is compared against the optimal source as long as this first source has data available. We discuss the metrics offered by this simulation environment in Section VI.

Another public source for data is the OpenStreetMap project (OSM) [13], which offers geospatial data free according to their licence [14]. OSM works like a wiki for geospatial data, i.e., everybody can add GPS traces to the database, tag ways and nodes on this map and supply additional data for existing items on the map.

This database can be imported into one's own PostgreSQL/PostGIS server [15]. Together with a Mapnik/mod_tile stack, this database can be used to render map tiles, such as the one which forms the background in Figure 2.

We also extract the locations of elements participating in the electricity grid from OSM where they are not covered by more exact databases.

The OSM database does not only contain spatial data, but also additional information such as the voltages a specific part of the power transmission system carries. This information is made available via hstore-based attributes called tags, which can be queried and extracted as a standard PostgreSQL feature. These tags, in OSM terms *Map Features*, are documented in the project's wiki [16].

VI. Discussion

The proposed simulation environment offers two benefits.

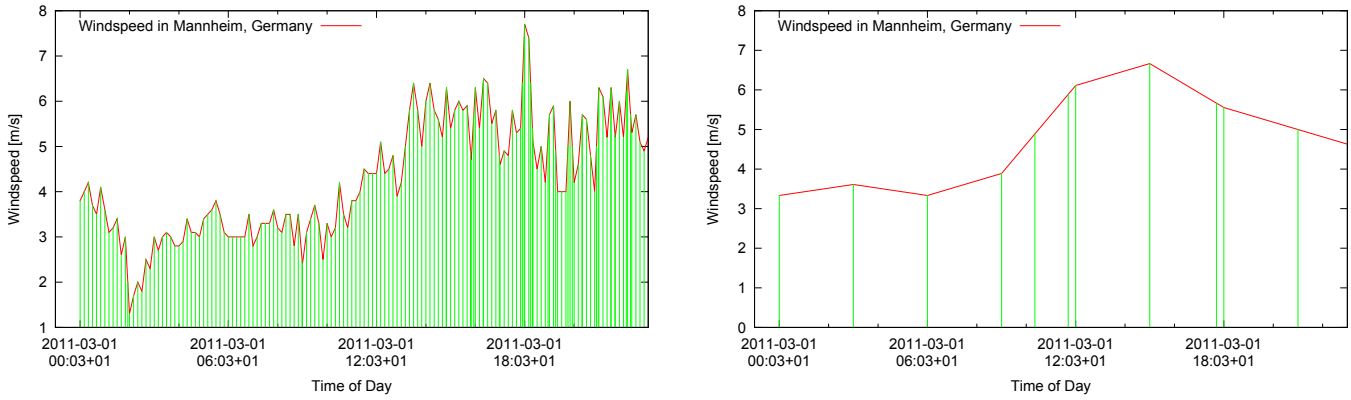
First, it treats both tested agents as well as data sources as items on the map. The attachment of spatial data to map items and events alike allows us to model realistic scenarios using real data sets. Together with the discrete event design, a simulation run can easily scale to run on a cluster.

Second, the simulator can switch data source strategies on the fly. However, with a change in the data source, there is also a possible change in the result of the simulation run. The simulator therefore must record that a switch has occurred, at which point this happened, and what data source strategy was subject to the change. This allows the simulation run to continue even if the preferred data source is unavailable, but also gives the user an indication that the quality of the result data may also have changed.

However, simply stating the fact that such a change occurred does not yet help in assessing the impact of the change. Therefore, the simulation environment provides two additional metrics to aid the user. We discuss these using the example of the German national weather service, *Deutscher Wetterdienst* (DWD) and World Weather Online (WWO).

For the simulation, the number of issued events is of foremost importance. An event triggers actions in the agents which participate in the simulation, ergo without events, there

 ISBN: 978-1-61208-332-2



(a) DWD (b) WorldWeatherOnline.com

Figure 4. Wind speed at March 3, 2011 in Mannheim; impulses symbolize triggered events

will be no action. Thus, the number of events issued is the first available measurement for the quality of a data source. Figure 4 shows how different sources trigger a different number of effects: The vertical impulses show an event that would have been fired. One can observe that the open data source leads to a significant lower number of events in the system than the bought, high-resolution data from the DWD.

Using the second data source therefore reduces the expressiveness of the simulation run: The two data sources differ in their entropy. We obtain the per-source entropy using $H(X) = \sum_i P(x_i)I(x_i) = -\sum_i P(x_i)log_{10}P(x_i)$.

For our example day, the DWD source obtains $H(DWD) \approx 1.57\ Sh$ while the open data source has an entropy of $H(WWO) \approx 1.04\ Sh$.

In order to arrive at these numbers, records the number of events that are triggered by the primary data source as long as it is available, and also measures how many events would have been issued by all other, lower-priorized data sources along with the values both sources yield.

Obviously, not only the number of events, but also the values conveyed by particular events are important. Whenever a `ValueChangedEvent` is triggered, it simulates a sensor reading coming from a data source within the simulation test bed. When a change in the data source strategy has occurred, this possibly also changes the values for those types events. The simulator therefore also records all possible event values in the same manner as it records the number of events triggered and calculates a derivation to the primary data source strategy.

This derivation is the source for an error calculation that allows us to express how much the current data source differs from the ideal one. The possible derivations introduced by the error of a data source that is not the reference source allow us to judge where a simulation run may have failed even though the data itself suggests it succeeded.

Figure 5 compares two data sources for errors.

These values are finally summed up to produce an overall result of the data source strategy change. Such a comparison has been done in Table I. The values hold true for one particular day, which was also the basis for the previous Figures 4 and 5.

However, the data source strategies are not the only place where open data are being used by the simulation environment. The map itself and the position data of most items are read from a local copy of the OSM database. There is, however, no fallback as it is employed with the data source strategies.

The trustworthiness of this database is important regarding two requirements: First, because it provides positioning data for the power grid, power substations, producers and consumers, and even map tile images. Second, because the additional attributes attached to these nodes, like voltages carried by a part of the power transmission network or installed types of windmills within a wind farm, form another data set that is used throughout the simulation.

The issue of trusting a community-driven data set has been thoroughly discussed for Wikipedia, for example, in [17], and OSM has also been used to increase efficiency of emergency medical services (EMS) [18]. For our own usage, we've compared the OSM data regarding wind farms to those from TheWindPower.net, an on-line wind turbines and wind farms database [17].

It shows that the overall number of wind farms registered is lower in OSM than in The Windpower's data. However, OpenStreetMap always offers a position since this particular attribute must not be null, whereas The Windpower actully can contain data without a location attached. For the simulation, both cases are not useful: If the wind farm data set has a location, but no other useful information, the simulation

TABLE I. COMPARISON OF DATA SOURCES DWD AND WORLDWEATHERONLINE.COM

	DWD	WorldWeatherOnline.com
Events triggered	166	23
Events derivation	0%	13.9%
Derivation (Avg)	$0\frac{m}{s}$	$0.81\frac{m}{s}$
Date		2011-03-01

 ISBN: 978-1-61208-332-2

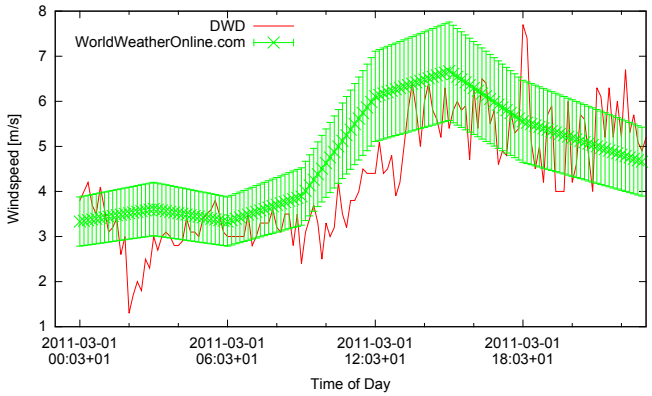


Figure 5. A reference data source (DWD) and an open data source (WWO) with error shown

environment cannot determine the electricity output of the installation; if this data is available, but the position is not, the simulation environment is unable to connect it to the transmission system at the right point.

Table II offers an overview over the most important figures as discussed.

As such, OSM is useful to render map tiles which allow the user to orient himself, but regarding data for simulation purposes, a specialized database should be preferred.

VII. Conclusion and Future Work

In this paper, we have introduced a discrete-event based simulation environment for smart grid messaging with a design adhering to the Don't Repeat Yourself (DRY) principle as much as possible. It is fully spatially indexed, which turns each participant into an item on a map and allows for quickly finding those items which are affected by a change of the environment.

The discrete events and the also discrete, locatable `MapItems` enable the user to run any simulation distributed, while the simulation description provides him with a tool to create reliably repeatable simulation runs.

The simulation environment makes heavy use of open data. It reads location data from OSM and attaches additional information to map items from the same data base. It is able to transparently switch between data sources, mitigating "holes" in one data set. While this allows the simulation to go on, it is also recorded; the simulation environment also provides hints on the possible impact of this change in data sources.

In the future, we will refactor the network code to be based on a Message Passing Interface (MPI), which will allow users to deploy the simulation testbed on any cluster.

TABLE II. COMPARISON OF DATA VOLUME OF OSM AND THE WINDPOWER

Feature	OpenStreetMap	The Windpower
Wind farms registered	508	13649
Wind turbines	1537	20215
Usable datasets	344	10101

VIII. Acknowledgments

This paper has been created as part of a cooperative doctorate program between the TU Bergakademie Freiberg and Wilhelm Büchner Hochschule, Pfungstadt.

The source code of the simulation environment described in this paper will be made available to the public on Bitbucket (*http://www.bitbucket.org/eveith/winzent-simulation*).

References

[1] M. Z. Lu and C. L. P. Chen, "The Design of Multi-agent based Distributed Energy System," *2009 IEEE International Conference on Systems, Man and Cybernetics (Smc 2009), Vols 1-9*, pp. 2001–2006, 2009.

[2] A. Varga, M. Visual, R. Omnet, and S. S. Method, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference ESM'2001*, vol. 42, 2001, pp. 319–324.

[3] P. Oliveira, T. Pinto, H. Morais, and Z. Vale, "MASGriP – A Multi-Agent Smart Grid Simulation Platform," 2012, pp. 1–8.

[4] E. M. Veith, B. Steinbach, and J. Windeln, "A Lightweight Messaging Protocol for Smart Grids," in *EMERGING 2013, The Fifth International Conference on Emerging Network Intelligence*. IARIA XPS Press, 2013, p. 6.

[5] D. S. Wilks and R. L. Wilby, "The weather generation game: a review of stochastic weather models," *Progress in Physical Geography*, vol. 23, no. 3, pp. 329–357, 1999.

[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: an efficient and robust access method for points and rectangles," in *International Conference on Management of Data*. ACM, 1990, pp. 322–331.

[7] B. Decker, "World geodetic system 1984," DTIC Document, Tech. Rep., 1986.

[8] D. Crockford, "RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)," IETF RFC, IETF, Tech. Rep., July 2006.

[9] J. Blanchette and M. Summerfield, *C++ GUI Programming with Qt 4*. Prentice Hall, 2008.

[10] Extreme Electronics Ltd., "Free weather data api for developers," [retrieved: 2013-10-14]. [Online]. Available: http://www.openweathermap.org/API

[11] WorldWeatherOnline, "World weather online," [retrieved: October, 2013]. [Online]. Available: http://www.worldweatheronline.com

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, 1995, vol. 206. [Online]. Available: http://www.cs.up.ac.za/cs/aboake/sws780/references/patternstoarchitecture/Gamma-DesignPatternsIntro.pdf

[13] OpenStreetMap Contributors, "Openstreetmap," [retrieved: October, 2013]. [Online]. Available: http://www.openstreetmap.org

[14] ——, "Openstreetmap copyright and license," [retrieved: October, 2013]. [Online]. Available: http://www.openstreetmap.org/copyright

[15] ——, "Planet.osm," [retrieved: October, 2013]. [Online]. Available: http://wiki.openstreetmap.org/wiki/Planet.osm

[16] ——, "Map features," [retrieved: October, 2013]. [Online]. Available: http://wiki.openstreetmap.org/wiki/Map_Features

[17] S. Javanmardi, Y. Ganjisaffar, C. Lopes, and P. Baldi, "User contribution and trust in wikipedia," in *Collaborative Computing: Networking, Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on*, 2009, pp. 1–6.

[18] M. Azizan, C. S. Lim, W. Hatta, and L. C. Gan, "Application of openstreetmap data in ambulance location problem," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2012 Fourth International Conference on*, 2012, pp. 321–325.

 ISBN: 978-1-61208-332-2