SmartScript - A Domain-Specific Language for Appliance Control in Smart Grids

Diego Adolf
Netcetera
Zypressenstrasse 71, 8040 Zurich
Switzerland
diego.adolf@netcetera.com

Ettore Ferranti
ABB Corporate Research - Grid Automation
Segelhofstrasse 1K, 5405 Baden-Dättwil
Switzerland
ettore.ferranti@ch.abb.com

Stephan Koch
ETH Zurich, Power Systems Lab.
Physikstrasse 3, 8092 Zurich
Switzerland
koch@eeh.ee.ethz.ch

***Abstract*—This paper describes an auto-configuring agent based software architecture connecting appliances, smart meters, solar panels, and a KNX building automation system, resulting in a complete demand-side smart grid. The agents are responsible for providing access to all datapoints in the system as well as sending commands to the active components. To control the system, a domain-specific language (DSL) called *SmartScript* was developed, whose benefits are twofold. The first one is to provide users, experts in electrical engineering and/or building automation but not in software systems, with a high level tool which they can use to control a demand-side smart grid. The second benefit is to provide a layer to implement and test quickly and effectively energy-aware algorithms without having to deal with all the underlying connections. Finally, some demo applications created using SmartScript (i.e., smartphone interface, voice-controlled building automation system) are presented in this work, in order to give an example of how SmartScript can be used.**

I. Introduction

This paper describes the work done as part of the Smart Grid Demo Lab project being executed at the Swiss ABB Corporate Research Center. The global aim of the project is to build a software infrastructure integrating both ABB Smart Grid Solutions and third-party products, in order to provide a common platform to support R&D activities in the area of smart grids and demonstrate advanced and emerging technologies to selected customers. Although the umbrella project focuses on both utility and demand-side (building-level) smart grids, the scope of the current work is restricted to the latter, leaving the application to utility smart grids for future work. An infrastructure consisting of different appliances such as smart meters, solar panels, batteries, etc. interconnected through gateways has been put in place.

Figure 1 shows a schematic drawing of the infrastructure at its final stage, i.e. when all appliances have been integrated. In the center of the figure, the *agents platform* is depicted. The network of these interconnected agents, along with their attached appliances, represents the actual smart grid. Each agent runs an instance of the Smart Grid Demo Lab software architecture on a gateway, usually a low-end computer.

At the bottom of the figure, the connection between agents and appliances is shown. Appliances are connected to the agents using a software component of the agent called *Protocol Adapter*. The Protocol Adapter handles the physical connection between one or more appliances and the gateway running the agent.

The upper part of the illustration shows applications accessing the smart grid, e.g. a visualization tool, with 3D graphics and multitouch capabilities [1], to graphically represent energy flows (left) or a normal desktop computer (right). Application access for the smart grid is provided internally through the corporate network, or externally through a virtual private network. Also, a database is depicted representing a central repository collecting historical data for future research projects.

The software architecture running on the gateways connected to the appliances is an auto-configuring, agent based software which is able to distribute appliance information using the publish/subscribe paradigm and implement commands directed towards different smart appliances.

In this paper, we introduce both the software architecture and a domain-specific language (DSL) developed to provide a high-level tool for the implementation of automation algorithms or to develop applications for the smart grid.

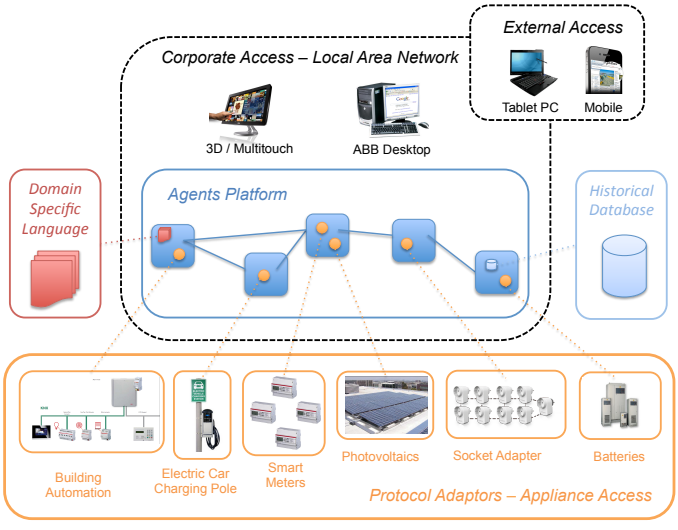

Fig. 1. Smart Grid Demo Lab infrastructure at final stage.

978-1-4673-0911-0/12/$31.00 ©2012 IEEE

II. Related Work

A. Smart Grids – General Overview

Smart grids are electric grids which have been extended with information and communication technology (ICT), providing grid members with a certain level of *smartness* and the ability to exchange information amongst each other. The term *smartness* is ambiguous and often not explained, though, in the context of smart grids, it could be summarized as the grid's ability to sense the state of the grid and act automatically and independently upon this state based on rules or algorithms previously defined. A rough classification of smart grids can be made into smart grids installed on the level of the utility grid [2], [3], [4] and smart grids installed on a building level (demand-side smart grids). The present work focuses mainly on the latter category, for which several contributions are presented below.

The current trend is to reduce, as much as possible, the burning of fossil fuels in order to achieve heating and cooling [5]. However, this will increase even more the consumption of electric energy, which will need to be aligned with the available electrical energy produced by renewable resources (e.g., PV, Wind, Water) and lead in turn to "electricity only" buildings. Thus, an optimal management of electricity consumption becomes more and more relevant. Kupzog et al. [6] summarize the three most important measures to optimize the use of electricity in a building.

The state of the art of building automation technologies currently offers many different solutions. One of them is the KNX building automation system [7]. It is a standard (ISO/IEC 14543) which allows communication of KNX compatible devices over a bus system. It can be used to automate certain processes in buildings, such as illumination and heating control.

A completely wireless solutions can be achieved by using ZigBee [8] enabled devices. ZigBee is a low-power wireless protocol designed for transmission ranges from 1 to 100 meters. With ZigBee, adaptors for almost any appliance can be used for smart metering and remote activation and deactivation. Examples of such adaptors are those produced by Plugwise [9].

Another system which is mainly used for smart metering is MBUS [10]. It allows communication over twisted pair cable and newer versions also implement a wireless connection.

In 2007, a non-profit organization called digitalSTROM.org [11] was created at ETH Zürich to promote an appliance integration technology. The core of the technology they promote consists of a chip which can be mounted directly on a regular appliance or can be connected through a socket adapter.

OptiControl [12] is a project in which several institutions of ETH Zürich, industry partners and the Swiss Federal Office of Meteorology collaborate in order to exploit the latest advances in sensorics, building technology, model predictive control (MPC) and weather forecasting for applications in building automation. Their goal is to combine these technologies to reduce energy consumption of buildings while maintaing high user comfort with a moderate initial investment and operating costs.

Project SAMBA [13] (Self-Adapting Monitoring for Building Automation) explains how smart buildings can be used to the advantage of the grid operators, mainly in terms of grid stabilization and peak shaving as well as for cost reduction and energy consumption minimization in buildings.

Even though these systems serve as enablers, most of them provide few cooperation mechanisms, or even none at all. As a consequence, a building employing more than one building automation system cannot take advantage of the full potential building automation has to offer. Thus, we propose an architecture capable of accommodating different building automation technologies allowing information exchange as well as interaction amongst different building automation technologies.

Some systems have been proposed in order to overcome the problem encountered when integrating different types of building automation technologies. One of these is BACnet [14]. It specifies a standard with which compatible devices have to comply in order to use one common communication mechanism. However, currently there are already systems installed which are not compliant with a standard enabling cooperation of systems. For this reason, new smart grid systems should be able to integrate a variety of technologies, even if not all of them comply with a specific standard, e.g. by providing adapters and abstraction layers. This type of design is reflected in the modularity and flexibility of the Smart Grid Demo Lab software architecture.

While integration is one obstacle in exploiting the full potential of building automation, there is still another main difficulty to overcome. As different systems are integrated it becomes more and more difficult to provide a single control interface that is able to control all devices in a smart grid regardless of their technology while still providing a single user interface for control. From the above mentioned demand-side smart grid technologies, one can deduce that, due to the fact that none of them focuses on the integration of other smart grid technologies their design does not foresee a generic control mechanism. The importance of a single interface for smart grid technologies becomes clear when the necessity of a centralized management console for a building is considered, or when applications shall be built on top of the smart grid (e.g. voice control). Also, as soon as interactions with the grid come into play, a standard for communication between buildings and grid is necessary. In the Smart Grid Demo Lab a first attempt to provide a common interface for an heterogeneous smart grid has been made with SmartScript.

III. Software Architecture

In Section II, several solutions for smart grids have been presented. Although the presented solutions are viable in certain scenarios, they still do not fulfill all needs in the areas of, e.g., seamless integration of different appliance technology or appropriate management tools. A software architecture was developed as part of the Smart Grid Demo Lab project to

address these issues. Moreover, it is the foundation for the integration of KNX into the smart grid as well as for the DSL used for appliance control (SmartScript).

A. Smart Grid Demo Lab – Software Architecture

The core element of the Smart Grid Demo Lab is the underlying software architecture. It is the bridge between appliances and the smart grid and enables communication amongst members. In our smart grid infrastructure, each agent is an independent entity running an instance of our software architecture and is attached to a set of appliances. The agents can receive commands directed towards the appliances and exchange information about them. Appliance data is exchanged using the publish/subscribe paradigm explained in the next section. Creation of new agents, e.g. to add support for a certain appliance technology, is strongly facilitated for developers by providing reusable software components in the form of modules. Any static vendor-specific or technology-specific elements have been avoided in the architecture to allow accommodation of many types of appliances in a generic way. Additionally, the software design targets platform independence for higher portability. The main programming language used to develop the architecture is Python [15].

1) Publish/Subscribe Paradigm: The publish/subscribe paradigm describes a concept for information distribution. There are two roles which can be incorporated by an entity involved in the information exchange: *publisher* and *subscriber*. A publisher informs a group of potential subscribers about the availability of a certain data feed, also called *service*. This action is referred to as *publishing*. Any potential subscriber interested in the data feed can notify the publisher that it wishes to *subscribe* to the service. From then on, every time the publisher obtains new data related to the previously published service, it will forward this update to all subscribers of the service. If a subscriber wishes not to receive service data anymore, it informs the publisher that it wishes to *unsubscribe* from the service. It should be noted that an entity may act as publisher and subscriber at the same time.

In the case of the smart grid architecture, a service data feed should only contain updates, i.e. subscribers will only receive service data when changes occur. This way, resources (e.g. bandwidth) can be saved, as a constant value can be assumed as long as no service data is received.

2) Software Components: The architecture currently consists of eight main modules, each independently handling a dedicated task. The element connecting all modules is the *Smart Agent* or *Aggregator* module. Figure 2 depicts all modules and illustrates their relation to each other. A detailed description of each module's design and implementation can be found in [16]. The following list summarizes each module's purpose:

- *Smart Agent or Aggregator:* Delegates incoming data to the corresponding module for processing and handles their output. Also, it implements the most basic commands used to exchange data between agents, the so-called *primitives*. These are, the publish/subscribe primitives (*publish*, *subscribe*, *unsubscribe* and *unpublish*) and the primitive *send* used to send service data to subscribers.
- *Publish/Subscribe:* Stores information of published services and subscribers of those services. It also stores the subscription to services provided by other agents.
- *Publisher:* Implements functionality to publish a specific service in the local network.
- *Browser:* Can be used to find services published in the local network.
- *Protocol Adapter:* Represents the component linking one or more attached devices to the agent. It is the only part of the architecture which needs to be designed for a specific appliance technology. A common interface is provided through the Protocol Adapter such that other modules can access the appliance data without the need to know which type of device is actually attached. Also, Protocol Adapters can provide functions to operate the device, such as turning it on or off or access to measurement data, e.g. power consumption. In this paper, a special Protocol Adapter for the KNX building automation system was developed.
- *Communication:* Exposes all commands and primitives which an agent supports to the network using remote procedure calls (RPC). These are mainly the publish/-subscribe primitives implemented in the Smart Agent module as well as appliance specific commands implemented in the Protocol Adapter. Communication is secured by encrypting the communication channel using Transport Layer Security (TLS) and by authenticating agents through certificates.
- *DB Proxy:* Is used in conjunction with service data storage, as an interface to the DB Connection module, which performs the actual storage operation. This module is used, for instance, to run an agent connected to the

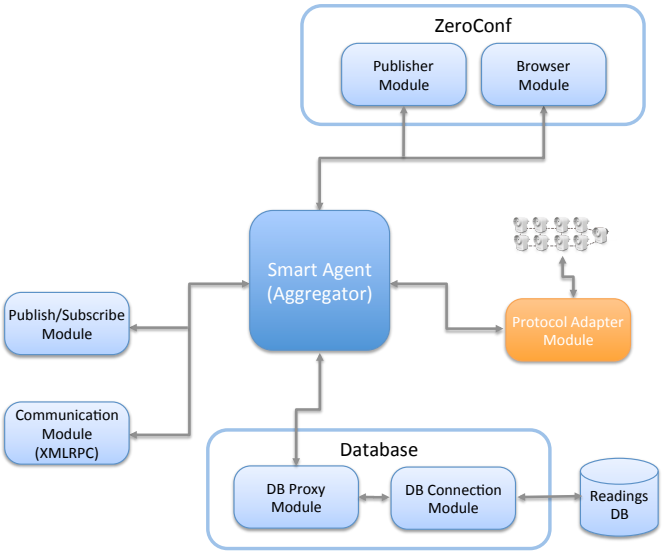

Fig. 2. Smart Grid Demo Lab software architecture with all modules.

database which collects all service data in the smart grid and stores it in a central database.

- *DB Connection:* Receives data from the DB Proxy module and stores the data. By having a second module to actually perform the storage operation, the container used to store the data can be implemented with more flexibility. For example, an FTP server, an SQL database or several storage methods simultaneously could be used without the Smart Agent module having to deal with the details of the data container as it only interacts with it through the interface provided by the DB Proxy.

3) Unique ID: In the smart grid every device and its properties have to be addressable unambiguously. Each device in the smart grid is labeled with an identifier which exists only once in the smart grid. This identifier is referred to as *unique ID*. A unique ID is normally composed of a physical address of the device (e.g. MAC address), a serial number or another unambiguous identifier.

With this labeling system it is possible to assure that commands directed to a certain device will reach that device and no other. Also, data collected from them, e.g. for storage in the historical database, can be associated with the correct device.

Example of unique IDs used in the smart grid are "ABB Electricity Meter 00411278" (smart meter) or "000D6F00000469A9F" (MAC address of a Plugwise socket adapter). It should be noted that properties of these devices, such as power consumption or power state, can also be clearly identified by concatenating the name of the property with the unique ID. For example, the power state of a Plugwise socket adapter can be referred to as "000D6F00000469A9F.PowerState".

IV. KNX Agent & SmartScript

One of the main topics of this paper is the integration of KNX into the Smart Grid Demo Lab infrastructure. For this purpose, a KNX agent connecting the KNX system with the rest of the agent infrastructure was developed. In addition to the KNX agent, SmartScript, a domain-specific language for appliance control in smart grids was developed.

The sections below present the software design used for KNX and SmartScript software components as well as the design criteria and grammar definition of the domain-specific language.

A. KNX Agent

In order to integrate the KNX system into the Smart Grid Demo Lab infrastructure, an agent bridging the smart grid and the KNX system is necessary. This agent, referred to as KNX agent, is based on the software architecture presented in Section III-A2. It implements the KNXnet/IP tunneling protocol. Its task is to provide reliable communication with the KNX bus. It will act as a client for a KNXnet/IP gateway which has physical access to the bus. Other agents are then able to forward data to the bus and read data from the bus through the connection established by the KNX agent with the KNXnet/IP gateway.

Figure 2 shows all modules, i.e. software components, of the architecture. Each agent in the smart grid is bound to one or more appliances. The module in charge of handling communication with those appliances is the Protocol Adapter. Protocol Adapters are customized for each type of appliance technology. In the case of the KNX agent, the Protocol Adapter must implement KNXnet/IP in order to communicate with the KNX bus. Figure 3 shows schematically how the KNX agent connects to the KNX bus. The database modules have been excluded, as the KNX agent does not directly interact with a database, and thus, does not require them.

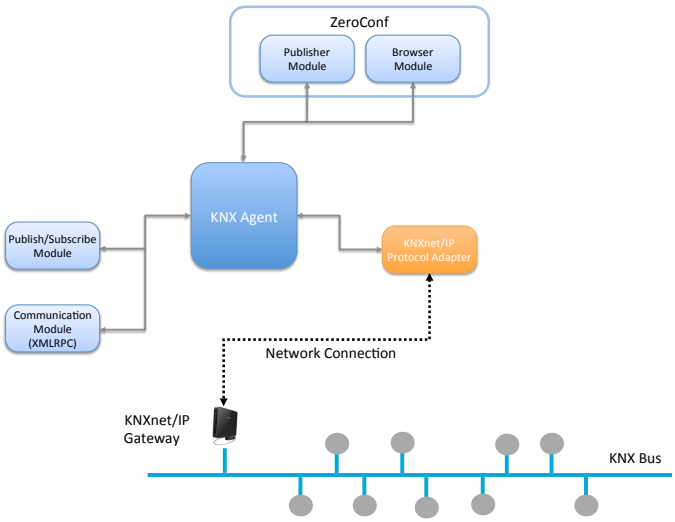

Fig. 3. KNX agent with the KNXnet/IP Interface integrated in the Protocol Adapter.

The main functionality provided by the KNX agent is:

- *Smart Agent Primitives:* Support of all primitives implemented by smart grid agents (see Section III-A2).
- *ZeroConf Support:* Discovery of the agent after publishing its service in the smart grid.
- *Publish/Subscribe:* Distribution of KNX data using the agent infrastructure with the publish/subscribe paradigm.
- *KNX Bus Access:* Support of read and write operations for KNX datapoints on the KNX bus.

Detailed information about the design and implementation of the KNX agent can be found in [17].

B. SmartScript

SmartScript is a domain-specific language intended for appliance control in smart grids. The design of SmartScript focuses strongly on the ease of use for its users.

We assume a user that does not have in-depth knowledge of KNX or any other smart grid technology currently integrated in the Smart Grid Demo Lab. Access to appliances should be intuitive and addressable using very high-level commands. For example, SmartScript shall have a syntax and wording as similar as possible to the English language and devices

and device states shall be addressable using intuitive and descriptive terms. For instance, a user shall be able to address lights, shutters or heating controllers by their name and not by their physical address in the smart grid (e.g. MAC address or KNX address). The state of a light shall be *ON* or *OFF* and not *1* or *0*.

Also, the user should be able to address functions related to each other without knowing if these are executed by different devices on a lower level. For example, if the user wants to turn on the light and afterwards dim it, she should be able to direct both commands to the *light* and not one to the light *switch* and the other to the *dimmer*.

In order to provide this service to users of SmartScript, we propose to encapsulate one or more devices in one identifier. An identifier can have several properties which can belong to one device or more. This, however, is hidden from the user, as she will only address the identifier and its properties and not the devices themselves.

A full definition of the SmartScript grammar can be found in [17].

1) Explanation of SmartScript Language Elements: SmartScript supports three types of statements, *Action*, *If* and *Loop*. *Action* is a supertype used to group language elements which interact with devices. The *If* type is used for flow control. *Loop* is used to group a set of instructions which are executed repeatedly. These statements and those derived from them are described below.

- *Device:* A device has two components, an *identifier* and a *property*. This allows to group several devices which relate to each other. The relation between the devices can be given by various means, for instance similar functionality, e.g., group a light switch and a light dimmer together or because of their location. An example for grouping devices with a common location would be an office. It should be pointed out that specifying a property after an identifier is not mandatory. This is used in order to allow even more flexibility when addressing devices. If a property is not specified, a default property will be assumed for that identifier. Exemplary devices could be: *Office21 Light*, *Office21 Shutters*, *Office21 Heating*, or *Light Switch* and *Light Dimmer*.
- *valueType:* This element is used to specify a state of a device in SmartScript commands, e.g. *ON*, *OFF*, *95%*, *1723.45*, etc.
- *SetAction:* Statement used to set a device to a given state.
- *GetAction:* Statement used to retrieve the current state of a given device.
- *ExportAction:* Statement used to export the state of a given device to a spreadsheet format. It is intended to enable evaluation of device information afterwards, e.g. visualize power consumption.
- *Loop:* Used to group several statements inside a loop which is executed periodically. This statement is specially useful in combination with algorithms that perform ongoing evaluation of the state of a building. To give an example, consider a heating control loop which deactivates the heating when a certain upper temperature level has been reached and reactivates the heating as soon as a lower temperature boundary is surpassed.
- *StopAction:* Statement to terminate the execution of a *Loop* element. Only valid inside a loop.
- *If:* Conditional statement used to control program flow, i.e. if a specified condition is fulfilled certain statements are executed, if not, other statements are executed.
- *Condition:* Represents a condition which is evaluated in an *If* statement by comparing the current state of a device and a given state.
- *OPERATOR:* Logical operators which can be used for comparison in *Condition* elements. A typical use of *OPERATOR* could be in an If-statement like *"if Light Switch* ***is*** *'ON' ..."* or *"if Temperature* $>$ *22.5 ..."*.

2) Requirements for Agent Infrastructure: Making access to devices possible via SmartScript requires the implementation of two extensions in the agent infrastructure. These are listed below:

- *Device Library:* The device library is a list containing all devices which shall be accessible via SmartScript. This list will map each device's SmartScript name to its corresponding unique ID (for unique ID see Section III-A3). It will also specify functions used to convert device states to a human-readable form (e.g. 1 is translated to *ON* and *0* is translated to *OFF* for a light switch). Further, the agent through which this device is reachable will be specified. The device library shall only be implemented by agents that process SmartScript directly.
- *RPCs for Device Access:* Every agent providing access to devices with SmartScript support shall implement a *set* and a *get* remote procedure call. The *set* command will be used to set the state of a device and *get* to retrieve it. The devices will be addressed with their unique IDs.

3) Examples: To illustrate the usage of SmartScript, exemplary SmartScript commands are shown below:

```
1  /* Read total power of tract G. Obviate property, as
2      default is 'Total'. */
3  get TractG_Power
4
5  /* Check if windows are open in office G0137. */
6  get G0137_Window
7
8  /* Move shutters up in office G0137. */
9  set G0137_Shutter 'UP'
10
11 /* Dim lights to 60% in office G0137. */
12 set G0137_Light Dimmer to '60%'
```

The commands make use of several devices which were previously defined using the device library. In this example, shutters, lights and window sensors were defined for office number *G0137*. Also, the KNX power meter for tract G of a building was defined. The name of the instances, i.e. *TractG_Power*, *G0137_Shutter*, *G0137_Light* and *G0137_Window*, map to the *identifier* keyword of the *Device* language element. The examples below use the default property of each device and thus do not have to explicitly specify the property. One

exception to this is the command in line 12. Here the property *Dimmer* is explicitly stated. This way, the command is not sent to the default property but to the *Dimmer* property of the corresponding device, i.e. G0137_Light.

C. Applications

To demonstrate the new features added to the Smart Grid Demo Lab infrastructure and the advantages of having a DSL to control smart grid appliances, two demonstration applications have been created. The applications developed take advantage of the online SmartScript processor accessible through the KNX agent as well as the RPCs to read and write to KNX datapoints. The developed applications are described in the following paragraphs.

1) Voice Control for KNX: Voice control for KNX was implemented to demonstrate the advantages of having high-level access to write and read operation on KNX datapoints over RPC. This application was implemented using Microsoft Windows' built-in speech recognition software. An additional library called *Dragonfly* [18] was used as a Python interface to the speech recognition system. Voice commands are mapped to write requests controlling lights and shutters in one office. In the future, the application could be extended to support a broad spectrum of devices or to directly interpret SmartScript input through speech.

2) KNX Control for Android: An exemplary application was created to demonstrate the advantages of a DSL for appliance control. A smartphone application was created for the Android platform [19], enabling control of KNX light and shutter actuators for all offices in tract G of the ABB Research Center.

A web based user interface was used in conjuction with a Python backend. User interactions trigger events which are processed in the background and converted to SmartScript and then executed through the KNX agent. Outsourcing of device interaction via SmartScript using RPC reduced the development time significantly. Application development focused mainly on the creation of an appropriate user interface. This demonstrates that the abstraction provided by SmartScript is of significant use when developing applications. It allows the developer to focus on the core service provided by his application instead of the interactions with the devices.

V. Conclusion

In this work, we showed how the Smart Grid Demo Lab modular and agent based architecture can provide a significant advantage for the integration of appliances into a smart grid and the creation of agents controlling them. The introduction of the publish/subscribe paradigm in conjunction with auto configuration of agents demonstrated a good solution for service data exchange within a smart grid.

Additionally, the proposed SmartScript language can provide several benefits to the design of the future smart grid. It provides a single interface for any device in our testbed, regardless of its technology, presenting an otherwise heterogeneous system as a homogeneous one. Further, the DSL's syntax drastically simplifies device access. Demo applications created to test it (e.g., smartphone KNX interface, voice-controlled building automation system) give a good example of how SmartScript can be used to facilitate the development of applications close to the end user.

By integrating the KNX building automation system and implementing SmartScript, the Smart Grid Demo Lab project has extended its ground for smart grid development and continues to provide many research opportunities in this area.

With SmartScript, an easy way of controlling devices in the smart grid has been provided. It was outside the scope of paper to implement and test automation algorithms properly. Thus, future projects may focus on algorithms for energy consumption minimization or user comfort maximization using SmartScript or directly the underlying agent infrastructure.

References

[1] M. Naef, E. Ferranti, "Multi-touch 3D navigation for a building energy management system," in *Proceedings of IEEE Symposium on 3D User Interfaces*, 2011.

[2] Kok, K., S. Karnouskos, J. Ringelstein, A. Dimeas, A. Weidlich, C. Warmer, S. Drenkard, N. Hatziargyriou, V. Lioliou, "Field-testing smart houses for a smart grid," in *Proceedings of the 21st International Conference on Electricity Distribution CIRED*, Jun. 2011.

[3] "Xcel Energy SmartGridCity - Benefits Hypothesis Summary," Jul. 2008.

[4] Press Release ABB, "ABB and Fortum to develop large-scale smart grid for sustainable city project," Nov. 2009.

[5] Baker N., Steemers K., *Energy and Environment in Architecture: A Technical Design Guide*. Taylor & Francis, 2002.

[6] Kupzog, F., Palensky, P., "Wide-Area Control System for Balance-Energy Provision by Energy Consumers," *proc. of 7th IFAC International Conference on Field- buses & Networks in Industrial & Embedded Systems, Toulouse, France*, pp. 337–345, 2007.

[7] K. Association, "Official website," February 2011, http://www.knx.org/. [Online]. Available: http://www.knx.org/

[8] Paolo Baronti, Prashant Pillai, Vince W.C. Chook, Stefano Chessa, Alberto Gotta, Y. Fun Hu, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards," *Computer Communications*, vol. 30, pp. 1655–1695, May 2007.

[9] Plugwise, "Official website," June 2011, http://www.plugwise.com.

[10] M-Bus, "Official website," June 2011, http://www.m-bus.com.

[11] Georg Dickmann, "DigitalSTROM: A centralized PLC topology for home automation and energy management ," *IEEE International Symposium on Power Line Communications and Its Applications*, pp. 352–357, Apr. 2011.

[12] Frauke Oldewurtel, Dimitrios Gyalistras, Markus Gwerder, Colin N. Jones, Alessandra Parisio, Vanessa Stauch, Beat Lehmann and Manfred Morari, "Increasing energy efficiency in building climate control using weather forecasts and model predictive control," *10th REHVA World Congress Clima 2010*, May 2010.

[13] D. L. Guzzella, "SAMBA (Self-Adapting Monitoring for Building Automation) - ETH Research Database Project Summary," July 2011.

[14] Steven T. Bushby, "BACnet: a standard communication infrastructure for intelligent buildings," *Automation in Construction*, vol. 6, pp. 529–540, Sep. 1997.

[15] "Python Programming Language. Official Website," May 2011, http://www.python.org. [Online]. Available: http://www.python.org

[16] D. Adolf, "Agent Based Architecture for Smart Grids," Semester Thesis, ETH Zürich, 2010.

[17] D. Adolf, "SmartScript - A Domain-Specific Language for Appliance Control in Smart Grids," Master Thesis, ETH Zürich, 2011.

[18] "Dragonfly. Official website," August 2011. [Online]. Available: http://code.google.com/p/dragonfly/

[19] "Android. Official website," August 2011. [Online]. Available: http://www.android.com