Research online Publishing www.ronpub.com

Open Access

Open Journal of Big Data (OJBD)
Volume 4, Issue 1, 2018

http://www.ronpub.com/ojbd
ISSN 2365-029X

Operation of Modular Smart Grid Applications Interacting through a Distributed Middleware

Stephan Cejka[A], Albin Frischenschlager[A], Mario Faschang[B], Mark Stefan[B], Konrad Diwold[A]

[A] Siemens AG, Corporate Technology, Research in Digitalization and Automation,
Siemensstraße 90, 1210 Vienna, Austria, {stephan.cejka, albin.frischenschlager, konrad.diwold}@siemens.com
[B] AIT Austrian Institute of Technology GmbH, Energy Department,
Donau-City-Straße 1, 1220 Vienna, Austria, {mario.faschang, mark.stefan}@ait.ac.at

Abstract

IoT-functionality can broaden the scope of distribution system automation in terms of functionality and communication. However, it also poses risks regarding resource consumption and security. This article presents a field approved IoT-enabled smart grid middleware, which allows for flexible deployment and management of applications within smart grid operation. In the first part of the work, the resource consumption of the middleware is analyzed and current memory bottlenecks are identified. The bottlenecks can be resolved by introducing a new entity that allows to dynamically load multiple applications within one JVM. The performance was experimentally tested and the results suggest that its application can significantly reduce the applications' memory footprint on the physical device. The second part of the study identifies and discusses potential security threats, with a focus on attacks stemming from malicious software applications within the framework. In order to prevent such attacks a proxy based prevention mechanism is developed and demonstrated.

Type of Paper and Keywords

Regular research paper: *IoT application management, distributed Smart Grid applications, Java virtual machine, memory optimization*

1 Introduction

The increased integration of renewable energy in the European energy system has led to a paradigm shift regarding the operation of medium and low voltage distribution grids [2]. These grids were designed for the distribution of energy among consumers, which requires very little control. As renewable energy sources (e.g., PV-systems) are usually integrated at a medium low voltage level, the role of distribution grids has changed. In order to facilitate this new role new means of control and monitoring mechanisms/solutions are required to manage renewable energy installed on the distribution system level and maintain the required operation quality within distribution grids. Mechanisms discussed in the context of active distribution system control and monitoring are usually coined under the term "smart grid operation". The term *Smart Grid* can be defined as *"an electric system that uses information, two-way, cyber-secure communication technologies, and computational intelligence in an integrated fashion across the entire spectrum of the energy system from the generation to the end points of consumption of the electricity"* [20].

In order to facilitate new means of monitoring and control distribution system operation has started to adopt Internet of Things (IoT) concepts. IoT constitutes the efforts of integrating information technology seamlessly with real world things [28]. In the context of distribution

system operation (DSO) this has led to a transformation of formerly passively/manually operated devices such as transformers, breakers or switches into devices, which are actively integrated in the system operation process [29]. Using IoT-concepts in the DSO domain has led to a wide range of novel applications [27] as well as business cases [21], which aim for increasing the efficiency of operation and fostering the decarbonisation of power supply systems.

This article extends the work presented in [6], focusing on the memory optimization of a distributed middleware by introducing a mechanism for dynamically loading multiple applications into single run-time environments. In addition this article analyses security threats to which this middleware is susceptible and demonstrates how the application of proxies can be used to prevent threats emanating from malicious third party applications hosted on the middleware. The memory and security issues show-cased in this article are based on a system developed in the context of an IoT-enabled secondary substation. In a first step the architecture underlying the system is presented and the role that applications hosted on such a substation play is outlined.

1.1 IoT-compliant Power Distribution Grids and the Role of Applications in Such Systems

The application of IoT devices in the context of smart grid operation as well as the role such devices could take within new forms of distribution system operation has been extensively discussed in the context of fog/edge computing (see e.g. [1]). The reason is that a growing number of devices such as smart meters, smart breakers, electric vehicles or smart storage systems which are active in distribution grids already display this functionality. These devices can effect several domains of the smart grid (cf., domains of the Smart Grid Architecture Model [7]). Such devices are integrated into smart grid operation since their application increases the economical and ecological effectiveness and allows the active operation of power distribution grids [15].

In order to fully exploit IoT functionality within a distribution grid novel services are required which utilize ICT-connected power grid components and external services. Especially, in the context of low voltage grid automation novel solutions based on IoT-functionality are actively developed (e.g., [10, 23]). Possible examples of such a utilization are forecasting mechanisms and the integration of weather information into system operation [15].

There are different means how IoT-enabled power grid devices can be integrated into grid operation from an architectural point of view. One option is to shift computation into the cloud or a dedicated data warehouse. In such a scenario field devices are used to measure and aggregate grid information, while computation happens elsewhere. Such an approach has been investigated in the context of distribution system operation [11]. The architecture underlying such an approach can be scalable [22] and it has been shown that such cloud based approaches yield a better execution time than their non-cloud equivalent [9].

Within this study an edge-based approach is presented, where data storage and computation are performed on a dedicated field devices. The reason for choosing such an approach in contrast to a cloud approach lies in the fact that edge-based computation constitutes less strict infrastructure requirements (in terms of the availability of an uplink and downlink) as computation and aggregation are performed in the field, while guaranteeing the execution and performance of system critical applications.

It should be noted that the presented approach does not exclude a hybrid approach where different means of computation are used both in the field as well as in the cloud, as the system can be easily extended to interact and utilize higher level systems such as a data warehouse or a dedicated cloud environment. The integration of such services in the context of the presented framework (given that the required infrastructure is at hand) would further boost the scope of operation of the presented system.

An important component regarding the operation of low voltage distribution grids is the secondary substation. It can be seen as a link between the medium (MV) and the low voltage (LV) grid. Under a passive operation scheme the substation constitutes an isolated device, i.e. all functionality within the substation is pre-installed and every interaction with the substation requires staff on site. As outlined such an approach is no longer feasible given the increased integration of renewables into the energy system. For such scenarios a proactive substation is required, which allows to adjust its functional scope by the deployment of applications and active integration of information in these applications.

An intelligent secondary substation (iSSN) as presented in [15] can be seen as the result of the evolution from a passive towards an active substation allowing for a novel function scope. In order to achieve this functionality increased computational power and means of communication (both north and southbound) are required. In contrast to passive systems where the functional scope is fixed, the iSSN allows to extend the functional scope. Examples for such new

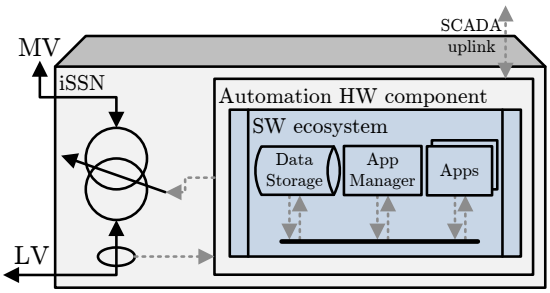


Figure 1: Architectural overview of the iSSN, containing power and communication links, a switchable medium (MV) to low voltage (LV) transformer and an automation hardware (HW) component – typically an industry-grade computer – which operates the smart grid applications [15].

functionality are voltage and (re-)active power control, the optimization of distributed generation, virtualization, or even decentralized market interaction. The functions are realized as distributed software applications – so called Smart Grid applications – which can be deployed on the iSSN. Figure 1 gives an architectural overview of an iSSN. A key requirement is the ability of applications to communicate with each other. This allows to bundle functionality in applications (in terms of services) and compose complex functionality via the interaction of applications. Within this study this is achieved via a proprietary distributed middleware system, the Gridlink, which has been previously introduced [15, 4].

1.2 Similarity between IoT and Interacting Distributed Power Applications

In many industrial applications built on top of a distributed middleware framework, applications can either be hosted and distributed on several automation components or on the same machine. Gridlink was built with a focus on distributed computation. The ratio was the ability to establish iSSN functionality across device boundaries – i.e., a substation with several dedicated automation devices (e.g., one device being responsible for the interaction with the transformer and other devices being responsible for data aggregation and/or southbound and northbound communication).

A special case arises if all applications are hosted on one automation device. This is a likely scenario regarding distribution system operation as the number of automation devices required for power system operation increases exponentially with decreasing voltage level. A DSO typically has to manage a few MV grids to which several hundreds of LV-grids are connected. For example, in 2014 Wiener Netze GmbH, the DSO for Vienna and surrounding areas, was responsible for 46 substations between high and medium voltage level, and 10.718 secondary substations between medium and low voltage level [26]. Therefore, the cost effectiveness of the automation component is an important aspect.

In the case of Gridlink this leads to a large number of industrial applications (i.e., interlinked Java applications) running on one constraint automation component (i.e., an industry grade computer). Each application demands for a specific amount of computation and memory resources. Besides the resources required for the operation of the iSSN additional computational power, memory and bandwidth needs to be reserved for the provisioning features (deploying, updating and installing applications).

This situation maps well to the domain of IoT, where distributed functionality has to be achieved by low priced and resource constraint components via the interaction of functional modules. In addition, the number of applications (in IoT terminology: functional modules) running on a device should have a limited effect on the performance. Another requirement is that functionality is not coupled to a specific hardware platform in order to facilitate a heterogeneous world of devices.

1.3 Outline

This article shows that the desired modularity of iSSN application modules running on a constraint device is restricted by limited computation resource available on typical automation hardware. In order to overcome this obstacle the framework of the distributed middleware has to be extended. In addition the article investigates security threats that can arise in IoT-like application environments via malicious third party applications, and presents a potential solution that allows to minimize these threats.

Section 2 summarizes our previous work on the iSSN application framework, including the middleware Gridlink, its provisioning features as well as previously developed smart grid applications. In Section 3 a typical use case scenario with several iSSN applications is presented which is used to evaluate the system's performance. In Section 4 the performance of the use case scenario is evaluated. It is shown that modularity jars with limited resources in the presented setup. In order to overcome this problem a framework enhancement is proposed. The old solution is compared with the new one, showing that significant savings in resource consumption can be achieved.

The second part of the article deals with potential security threats which can arise in a distributed middleware. Section 5 identifies potential threats for the proposed system. A specific threat which can arise in

the context of distributed automation is the unauthorized access of control/information by malicious applications. To overcome this problem a proxy mechanism is suggested and its application is demonstrated. Section 6 concludes this article and outlines planned future work.

2 Application Framework for Distribution System Operation

The application framework for distribution system operation used in this article has been previously proposed [14, 15, 4]. It was developed to establish a flexible and modular software ecosystem in the context of intelligent secondary substation automation. The resulting distributed middleware – the Gridlink (Section 2.1) – comprises a runtime environment for distributed applications, a communication infrastructure for the interaction of applications as well as means to manage (deploy/update/remove) applications during runtime. During the development of Gridlink a number of non-functional requirements were taken into consideration which include the modularity, resilience, and scalability of the resulting system. A key requirement was to achieve an easy integration of Smart Grid applications as well as the management of running applications, while keeping the influence of an application on others minimal.

Figure 1 sketches a potential scenario regarding the deployed components on an iSSN. Besides a number of applications the iSSN hosts an AppManager with a remote uplink to establish provisioning features (Section 2.2). Thus, the application framework allows for the use of a Plug & Automate paradigm, i.e., applications can be activated and deactivated during the runtime of other applications. In addition a data storage is deployed, which can be used by applications for the local aggregation of information received from the grid. Previously a number of Smart Grid applications have been introduced, including services and functions for

- acquisition, processing, and analyzing of field component measurement data (e.g., Smart Meter measurements) [14, 4],
- linking the iSSN to the energy market [18, 17], and
- making decisions, e.g., by a voltage controller application able to switch the transformer's tap-changer based on historical and/or current data [12, 4].

2.1 Gridlink

Gridlink was introduced as a middleware solution for intelligent secondary substations, based on vert.x and Hazelcast [4, 15]. Gridlink as distributed middleware solution (in contrast to many typical IoT middleware solutions, as e.g. listed in [25]) has a key advantage in not creating a single-point-of-failure within the system.

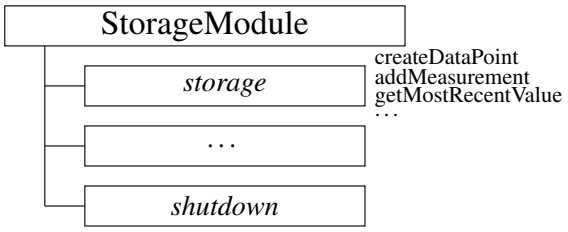


Figure 2: Modules, roles and services [4]
A module (in this example the *StorageModule*) is registered for the roles *storage* and *shutdown*. The role *storage* provides several services (e.g., *createDataPoint*).

A Gridlink-based secondary substation is composed of several application modules (written in Java), which interact via a message bus. The event bus underlying the system is based on an asynchronous communication model, which allows to couple modules with management functions such as a service registry and application update, upgrade and provisioning. Application modules run on a single Java virtual machine (JVM) that dynamically form a cluster of known instances during execution using multicast discovery. In addition, modules are able to enter and leave the system at any time without influencing other modules' execution or communication. While the failure of modules can impact the overall operational reliability of applications, the impact of a failing module on the overall applications can be limited by introducing redundancy and increasing timeouts to react on failed transmissions.

Message Exchange: The communication infrastructure provided by Gridlink for the application modules includes the concepts of roles, topics and services. An example is outlined in Figure 2.

Messages (i.e., requests or events) are identified by a type (e.g., `createDataPoint`), contain a destination address (e.g., the role `storage`) and may optionally include a payload (e.g., the data point to be created). They are either sent to a designated module role address or published to a topic address reaching all registered modules, by default being marshalled into a JSON representation for transmission. Dedicated proxies are executed after the module called the send or publish command but before the message is really transmitted (Figure 3). This allows for prior executing additional message processing steps, including the modification of a message (e.g., for encryption). The recipient module,

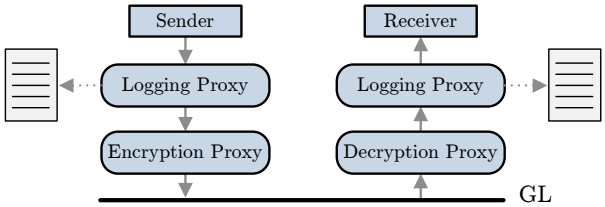


Figure 3: Example that utilizes the Gridlink proxy concept for logging and encryption of message payloads [15]

registered to the role or topic to which the message was issued to, is the data sink. After the respective proxies are executed on the recipient side, e.g., for decryption, the received message is handed over to the module's registered service handler for the message type and processed, which may include issuing a reply message to the sender.

Whether a proxy is in use, is defined only by the operator and stays completely transparent to sender and receiver module. These proxies are attached via the module's configuration file and can be plugged in and out during runtime of the affected module. According to their appearance in the configuration file, multiple proxies are executed in sequence.

Gridlink Registry: A module can access a distributed list of all currently attached and active modules – the Gridlink Registry. It furthermore includes all roles and topics the modules are registered to, the list of requests they are able to handle and the associated replies, as well as their proper JSON format specification. Thus, using the Gridlink Registry each module knows at any time which modules are currently present to behave accordingly. By using the defined message schemata even components outside the Gridlink system could establish communication to Gridlink modules.

2.2 Provisioning Tasks

At some time it is necessary to install new features, update applications, reconfigure them or uninstall them. These steps are grouped under the term provisioning, requiring – for cost efficiency – a functionality to initiate these steps directly from the remote operator control center without requiring staff on site.

A designated core module – the *AppManager* module – is responsible for such tasks. Figure 4 shows the simplified steps that are necessary:

1. The provisioning task is initiated from the remote operator site, for example a web-based dashboard of the DSO.

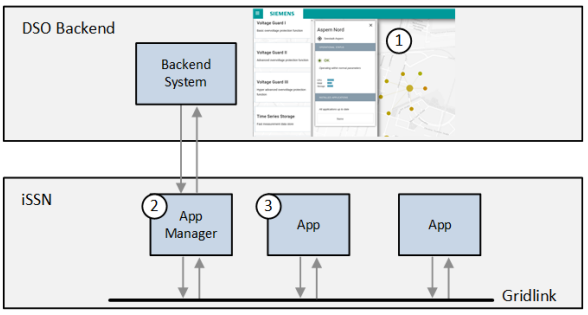


Figure 4: Provisioning tasks are initiated from the remote site, e.g., by an operator's dashboard, communicated to and executed by the local *AppManager* module. Status information is communicated back to the operator.

2. The *AppManagaer* receives these provisioning commands. In case of an installation it downloads the respective module.

3. The *AppManager* manages installation, configuration, updating, and removal of the other modules – hence, they are termed managed modules.

As the *AppManager* is a normal Gridlink module itself, it is able to communicate with any other module by sending and receiving Gridlink messages. The *AppManager* does not introduce a single point of failure to the functionality of the iSSN, as its fail crashes the remote provisioning features only. Interferences of modules with the operation of other modules can thus be kept to a minimum.

Since starting and stopping modules is tightly connected with the *AppManager*, selected design choices of this software will be highlighted, to better understand the later proposed solution.

Start-Up: During the *AppManager*'s start-up, all Gridlink modules in its `managed` directory are started – currently as a new Java process (cf., Figure 5). These modules are the *AppManager*'s *managed* modules.

Module Installation: The Module Installation task allows a functionality upgrade of the iSSN. First, the *AppManager* receives an install command triggered from the remote side. Afterwards the respective application is downloaded, the content of the archive is extracted and started.

Module Configuration Update: The Module Configuration Update task allows a modification of

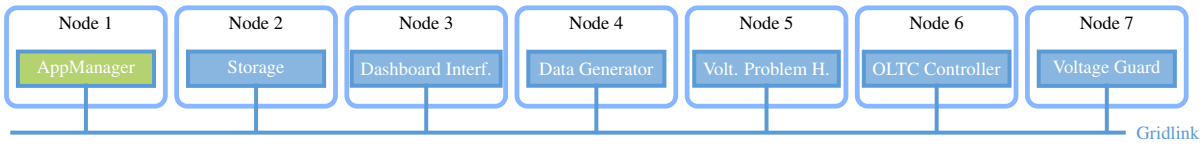


Figure 5: The current solution with one module per JVM/node

operation parameters during the module's execution. Once the configuration file is changed, the corresponding module is automatically notified. By design, the configuration update shall not require a restart. In the same way, Gridlink proxies can be attached and detached from a module dynamically during its runtime.

Module Uninstallation: When a module shall be removed, the *AppManager* sends a shutdown request to the module and waits for the module to disappear from the Gridlink Registry. Therefore, each module implicitly implements a shutdown role (cf. Figure 2). On success, the module's directory is deleted from the file system. Otherwise, the *AppManager* can be requested to kill the process over its remote link.

Further Provisioning Tasks: Information of the running modules, such as their state and the currently used configurations can be requested. This may also include information that bases on the Gridlink Registry.

During the runtime, it may be required to replace a module in execution with another version. This procedure is basically a combination of the module uninstallation and installation procedures, including – however – to keep the current module's state for (near to) continuous work of the module.

3 iSSN Use Case Scenario: Active Voltage Regulation

Secondary substations are typically used to transform and distribute electric energy to connected customers within a restricted voltage band. Active voltage regulation measures are required to counteract voltage fluctuation caused by renewable generation units and high loads. Such active regulation can be achieved, e.g., by using on-load-tap-changer transformers [12]. Subsequently we demonstrate a use case dealing with the detection and handling of voltage band violations [4]. All seven Gridlink modules that are required for this use case are running on one host machine in the iSSN (cf., Figure 5).

1. AppManager Module: The *AppManager* module handles the software provisioning tasks described in Section 2.2. This module has a remote communication connection, where the provisioning commands are received and downloads of artifacts take place.

2. Storage Module: The applications typically require access to some historical and current data. The *Storage* module is used for the permanent storage of the accumulated data and to make these data available to other modules on request. In this use case, the module is responsible for measurement data (i.e., simulated voltage value time series) and meta data of the simulated data points [5].

3. Dashboard Interface Module: The *Dashboard Interface* module provides a web-browser based view on current and historical values of sensors and the transformer.

4. Data Generator Module: The *Data Generator* module generates measurement values by simulating some houses' power consumption during the day as well as their power production in case they are equipped with a photovoltaic (PV) installation using predefined profiles.

5. Voltage Problem Handler Module: The *Voltage Problem Handler* module is a monitoring module that detects problematic voltages in the grid. The permitted voltage band in the use case is defined to be between 220 V and 240 V and is thus tighter than power quality criteria limits specified in EN 50160, that requires 95 % of all ten minutes-average values within a week to be within 207 V and 253 V ($U_n \pm 10\,\%$) [8]. Upon detection of a violation the operator is notified such that proper countermeasures can be taken.

6. On-Load-Tap-Changer (OLTC) Controller Module: The *OLTC Controller* module links the message bus and the transformer in the substation. Tap change requests transmitted to the transformer result in an increase of the voltage level by 2.5 V on tap up; a decrease on tap down, respectively.

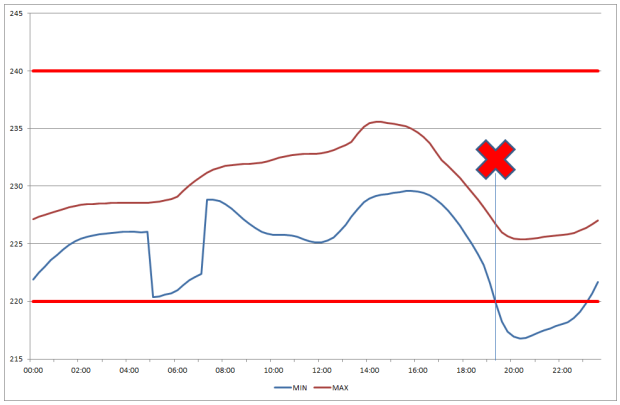


Figure 6: Minimum and maximum voltage level in the simulated low voltage grid before installation of the *Voltage Guard* Module.

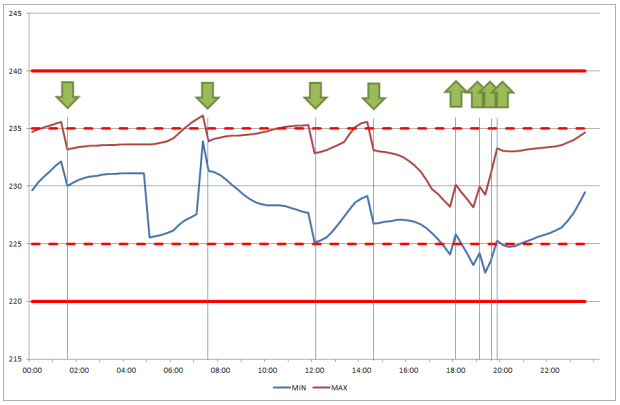


Figure 7: Minimum and maximum voltage level in the simulated low voltage grid with the *Voltage Guard* installed. Arrows show the submitted OLTC tap changes during the day.

7. Voltage Guard Module: The *Voltage Guard* module analyses the voltage levels in the grid and reacts when approaching the voltage band limits. Once a voltage value below 225 V or above 235 V is detected at one of the monitored data points, the module may decide to request the OLTC controller to change the transformer's tap position. The algorithm in use tries to keep the number of tap changes at a minimum level by not reacting immediately when hitting the barrier but only after a certain defined threshold level of cumulative voltage violations over time is exceeded [24, 30].

Scenario Execution: The simulated low voltage grid scenario consists of a group of apartment houses yielding a voltage level over the day submitted by the *Data Generator* module (Figure 6). During the day, a voltage level violation is detected by the *Voltage Problem Handler* module and communicated to the operator. Note that at this time all modules except the *Voltage Guard* Module are installed and running.

As countermeasure to the limit violations the operator decides to install the *Voltage Guard* Module (7) during the runtime of the Gridlink middleware and the modules 1–6. Due to its installation, tap position changes are communicated to the OLTC and thus the values all remain within the allowed voltage band (Figure 7).

Further Modules and Related Work: For presenting the impacts of node management it is reasonable to utilize the shown use case scenario. Some of the described applications are – however – only of basic functionality and need to be replaced or extended for real world use cases. The data generator – for example – necessarily needs to be replaced by a Data Concentrator module connected to the sensors and smart meters installed in the grid. Other publications that focus more on the module functions than on the framework, have introduced more sophisticated modules, e.g., for data processing, analysis, and grid operation [14, 13]. These applications enable novel functions on the power distribution grid level.

4 Middleware Optimization

The usage of the presented iSSN Application Framework in the Smart Grid testbed of the Aspern Smart City Research (ASCR) revealed two shortcomings. First, the necessity to specify roles over which messages are exchanged on sender and receiver side resulted in a complicated and error prone configuration process (Section 4.1). The second problem was a high main memory demand created by the modular approach (Section 4.2).

4.1 Service Discovery Optimization

As described in Section 2.1, Gridlink modules register handlers for various message types on roles to receive messages. On the recipient side, the pseudo-code to register a handler looks as follows:

```
registerHandler(<role-name>,
<message-type>, <handler>)
```

To send a message, a module needs to supply a message of the appropriate type and the role to which the message shall be delivered:

```
send(<role-name>, <message>)
```

Thus, the sender needs to know the role name on which the receiver awaits messages and the message

type the receiver can handle. In systems consisting of high numbers of modules, this resulted in complicated configuration files for each module; as requiring to specify on which roles the module has to listen for which message-types and to which roles the module has to send messages of which message type. Errors in these configurations inhibited the communication between those modules.

Field experiences show that in most cases, the sender does not care who the recipients of its messages are. The information, to which modules messages shall be sent to – however – is required by mentioned send command. This information which message-types are handled by which module is already located in the registry. Hence, the client module sending a message of a specified type could search in the registry for modules handling those message types and more importantly over which roles those messages have to be exchanged. With this information the module is able to send the message once for each found role. Since a new module could have registered a handler, this registry search has to be performed every time a message is sent. With the help of this mechanism, the cumbersome and error prone configuration process is reduced significantly.

The second step is to remove the trouble for module developers to search the registry and thus to remove duplicated code. Thus, Gridlink API was extended to include the described behavior, such that the role address is no longer required to be specified:

```
send(<message>)
```

If a module uses the send method without supplying a role, the message is sent to all modules with a registered handler appropriate to the message-type, by implicitly querying the registry. As the registry is distributed to all modules and all role registrations by modules are observed by the local registries, the search for that role requires local available information only.

4.2 Memory Optimization

The modularity of the approach requires running a high number of various modules for small independent tasks simultaneously. Previously, one Gridlink node executed exactly one module (cf., Figure 5), i.e., the *AppManager* initializes a new process for each module it is managing, introducing a high overhead in terms of main memory consumption. In consequence, the number of required modules for a typical Smart Grid use case is typically higher than the RAM-constrained devices can host simultaneously, before undesirable effects like trashing occur.

The use case consisting of seven modules being executed concurrently on the same machine, requires already more than 800 MB of main memory, not available on the target hardware platform. Thus, a mechanism to execute multiple Gridlink modules in one node and thus in one JVM process had to be developed ("scale up"). The proposed extensions shall however not invalidate any of the Gridlink functions. It shall be possible to run all modules as previously intended, therefore decisions like the start-up of modules, communication etc., should remain.

To solve the described problem of excessive memory consumption, a new Gridlink module – the *NodeManager* – is proposed. It introduces the ability of managing multiple modules on the same node, and thus, in the same JVM instance (Figure 8). To avoid ambiguous use of the term "managed", we will now distinguish between *app-managed* for management by the *AppManager*, and *node-managed* for management by the *NodeManager*, respectively. Therefore, the *NodeManager* is responsible for the module's start-up, provides a command line interface to node-managed modules, and is able to undeploy them. The process of application provisioning is notably influenced by the introduction of a *NodeManager*:

Start-Up: The *AppManager* was introduced as an unmanaged module [4]. However, besides installing it on its own node (cf., Node 1 in Figure 5), the *AppManager* now can – just like any other module – be started parallel to and by a NodeManager – as a node-managed module (cf., Node 1 in Figure 8). As in the old setup, all Gridlink modules in the *AppManager*'s `managed` folder are started during its start-up by creating a new JVM process, including a *NodeManager* module if it is contained in this folder (cf., *NodeManager* on Node 2 in Figure 8). No special handling is necessary, as the *NodeManager* complies with the module structure conventions. On the *NodeManager*'s start-up, all Gridlink modules nested inside its own `managed` subfolder are started within the *NodeManager*'s node/process (cf., the remaining modules on Node 2 in Figure 8). Special attention has to be paid to libraries of these node-managed modules: Java does not allow for including two classes with the same fully qualified name to the classpath. By using an own class loader for each module, possible problems of influences between libraries are kept to a minimum (cf., isolated bundles in OSGi [19, 16]).

Module Installation: The module's configuration needs to include whether it shall be started traditionally as its own process via the *AppManager* or on an existing node next to a *NodeManager*. Based on this decision, the artifact is extracted to the respective directory. In the

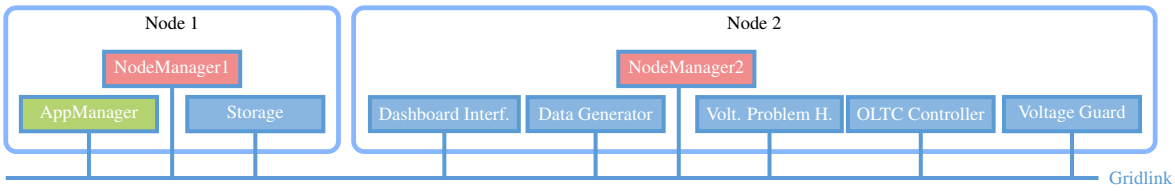


Figure 8: Solution with multiple modules on one JVM/node ("scale up").
In this example it is decided to execute the AppManager and the Storage module on one node; all other modules together on one other node, respectively.

latter case the request to start the module needs to be communicated to the respective *NodeManager* module.

Module Configuration Update: Configuration changes of a module always reach the *AppManager*, which is responsible for the modification of the JSON configuration file. The location of a module's configuration file depends on whether it is node-managed or not, as it either is located in the *NodeManager*'s or in the *AppManager*'s `managed` directory. The *AppManager* internally knows all the modules it has started, but not necessarily all modules started by other node managers (e.g., the *Storage* module in Figure 8). However, the *AppManager* is only responsible for its app-managed modules and thus, all modules of which he is capable of configuration changes are known.

Module Uninstallation: In case of the node-managed module's refusal to shut down, the *AppManager* cannot kill the process as earlier, as this would also shut down the *NodeManager* and every other module on that node. It can however request the managing *NodeManager* to stop the module, shifting the responsibility. As for configuration, for removal of the module's files the *AppManager* has to know the directory location.

Summary: In consequence, both *AppManager* and *NodeManager* module are responsible for their ⋆-managed modules. The difference is that app-managed modules are started in their own JVM and thus, have their own process; node-managed modules are started beside the *NodeManager* in the already existing process. Figure 8 shows both options:

1. *NodeManager1* on Node 1 is an unmanaged module, initially started.

2. It has started the *Storage* and the *AppManager*. These two modules are node-managed modules – managed by *NodeManager1*.

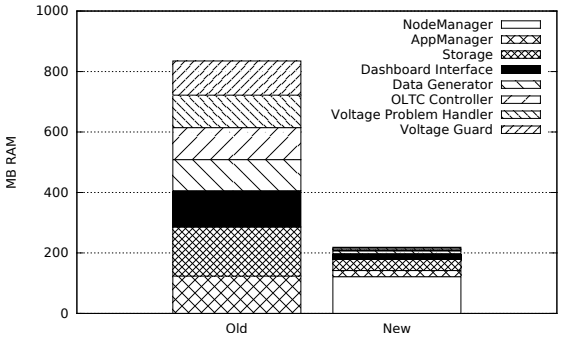


Figure 9: Evaluation results

3. The *AppManager* on Node 1 has traditionally started *NodeManager2* as its own process on Node 2. Therefore, *NodeManager2* is an app-managed module – managed by the *AppManager* on Node 1.

4. The five modules running within Node 2 have either been started (node-managed) by *NodeManager2* or by request of the *AppManager* to *NodeManager2* (in result being app-managed by the *AppManager* and node-managed by *NodeManager2*).

   In the scenario of Section 3, the *AppManager* would thus issue an request to *NodeManager2* to start the *Voltage Guard* module after its download.

As communication of all modules is done via the Gridlink – and so are the start commands from the *AppManager* to the *NodeManager* – the *AppManager* is also able to request modules to be started on Node 1 by issuing requests to its own *NodeManager1*.

Evaluation Results: We conducted experiments to show the impact of the new solution. Thus, we faced the memory consumption of the old solution – all seven modules of the iSSN use case (cf. Section 3) being started within their own node – with the new solution using the proposed *NodeManager* concept to start the modules in one JVM.

Figure 9 shows the results of both experiments. It is shown that each JVM requires at least 100 MB in main memory. Thus, the start of the *NodeManager* module shows a comparable memory consumption to each module in the first experiment. More specifically, for n modules with their memory consumption for additional required libraries, its logic, and its data structures m_i and a (for simplification assumed) constant amount of Gridlink dependencies' memory consumption c, the memory consumption for the old solution was:

$$M_{old} = \sum_{i=0}^{n} (c + m_i) = nc + \sum_{i=0}^{n} m_i$$

In contrast, the memory consumption of the new – *NodeManager*-based – solution requires Gridlink dependencies to be load only once, resulting in a significant reduction of the total memory consumption:

$$M_{new} = c + \sum_{i=0}^{n+1} m_i$$

Therefore, the higher the number of modules, the greater the difference of the required memory between old and new Smart Grid application management solutions becomes. However, when only one module should be started on a host, the old system requires less memory. This results from the *NodeManager*'s memory requirements for its own logic and a small Gridlink middleware overhead.

5 Security Threats

Potential security threats of Gridlink communication have been first identified and analyzed in [3]. Countermeasures against potential attacks relating node management consist of trusted applications and the middleware's security layer.

5.1 Event Bus Blocking Attack / Trusted Gridlink Apps

The event bus thread is shared between all modules running on one vert.x node. This aspect gets important – and security relevant – with the introduction of node management. If the thread is blocked by one module all other modules on the node are affected, e.g. by getting stuck or showing other unexpected behavior (Event Bus Blocking Attack) [3]. A malicious module could thus deny all other modules on the node from their proper execution.

As modules developed by other parties are possible, a mechanism for certified modules needs to be introduced (Trusted Gridlink Apps). An execution of uncertified and thus untrusted modules beside trusted ones on the same physical node may be prohibited. This partly solves the Event Bus Blocking Attack, as the influence of untrusted to trusted modules' execution remains limited.

To further improve the availability, fundamental modules (e.g., the storage), modules with higher processor consumption or modules that are required to reply fast could be moved to their own nodes, such that they are not required to share the event bus with other modules.

5.2 Communication Issues / Gridlink Security Layer

We earlier introduced Gridlink proxies that allow the normal execution to be interrupted to run user-defined code for interception or modification of messages or to execute additional tasks when messages are transmitted over the component (cf. Figure 3). A number of proxies can be defined for an application allowing a proxy side customization of the in- and outgoing data.

Message Encryption Proxies: The use of Gridlink proxies for encrypting messages before sending and decrypting them again at the receiver denies that malicious modules can read the communication. These proxies can either use symmetric or asymmetric cryptography. By use of proxies only features that are already included in the Gridlink are utilized; cryptography proxies are thus termed the Gridlink Security Layer establishing end-to-end encryption [3].

Message Filter Proxies: This subsection will demonstrate the application of proxies in the context of the secure integration of third party Gridlink applications. In such cases, the communication between Gridlink modules needs to be observed. The need for message filter proxies is shown by the following three motivating examples:

1. Assume that the *Voltage Problem Handler* module is developed by a third party. Such analytics modules pre-process sensor information. While this module should be able to request the *Storage* module for measurement data, other information and services available on the message bus should not be accessible by the module.

2. The *OLTC Controller* Module module provides a link between the message bus and the transformer in the substation. Applications are able to access current information on the tap position and can issue service requests regarding tap position changes, which are forwarded to the substation

controller via the module. The operation of the transformer is critical due to active interference with grid operation. If the voltage is too high at any point in the grid, machines of industries and devices of households can get damaged; if the voltage is too low, they may cease to operate. Thus, it must be ensured that only authorized modules are able to issue tap change requests to the *OLTC Controller* module.

3. The *AppManager* module issues shutdown requests to all modules using normal Gridlink messages. It needs to be ensured that only this module can send such requests, furthermore that every module accepts such requests only from that sender.

The intended behavior can be achieved using the proxy concept outlined previously. Modules are equipped with proxies which monitor and control the incoming and outgoing messages of a module, such that messages identified as prohibited can be filtered out.

Message Filter Security Proxy for incoming messages
On receiving a message check message type and sender; if permitted hand message over to module, drop it otherwise.

Message Filter Security Proxy for outgoing messages
Before sending a message check message and receiver; if permitted send message, drop it otherwise.

Both proxies work likewise: if the type of the message is not among the allowed types and/or communication with the other party (i.e., sender or receiver) is not permitted the message is dropped. Otherwise, the incoming message is handed over to the designated module; the outgoing message is sent over the communication network, correspondingly. Note that all proxies in use are specified by the module's user (i.e., the operator) and not by its developer. Thus, this mechanism allows to specify the allowed communication of a module without requiring in-depth inspection of the third party module.

For the motivating examples – besides encryption of messages – the following proxies are reasonable:

1. a proxy filtering outgoing messages of the *Voltage Problem Handler* module, that let pass only messages of permitted type and destination, e.g., this module is not permitted to issue tap change requests to the *OLTC Controller* module. In this case, the proxy is defined to drop all outgoing messages that are not data retrieval requests to the *Storage* module.

2. a proxy filtering incoming messages at the *OLTC Controller* module, that let pass only messages of permitted type and source, e.g., this module must not accept tap change requests from the *Voltage Problem Handler* module. In this case, the proxy is defined to drop all incoming messages that are not tap change requests by the *Voltage Guard* module.

3. a proxy for all modules except the *AppManager* that filters outgoing shutdown request messages; furthermore a proxy for all modules that filter incoming shutdown requests not originating from the *AppManager*.

Proxy Security Concept Demonstration To demonstrate the concept a malicious analytics module was generated. Besides its designated task (i.e., receiving current sensor information from the data generator module, preprocessing them and storing them in the Storage module) the module will also try to subscribe information from the OLTC module in order to receive information of the current tap position as well as send tap position change requests and therefore try to actively interfere into the grid operation process.

To demonstrate the application of proxies the scenario was tested under two setups. In the first setup no proxies were used, while the second second setup equipped each module with a module specific proxy defining the allowed in- and output of each module. The exemplary proxies for incoming and outgoing messages for the analytics module are shown in Listing 1.

The proxy of Listing 1 checks if incoming messages are sent from the specified sender as well as correspond to the data model which is required from the sender. If an incoming message does not correspond to an expected payload from an expected sender it is not forwarded to the module.

The proxy of Listing 2 works in a similar manner for outgoing message, filtering them according to data type and receiving module. Messages which do not correspond to the specification are not forwarded and will thus not reach their destination module.

When running the scenario without proxies each module is able to access information as well as services from all the other modules. Therefore the malicious module can access information on the current tap position as well as request the tap position to be changed. This behavior can be prevented using application specific proxies which manage the incoming and outgoing communication and service scope of the modules. Using proxies the analytics module is only

Listing 1: Example incoming proxy for an analytics module

```
public class AnalyticsInProxy extends LayerInProxy {

  private final Logger logger = ...;

  ...

  public Envelope apply(Envelope message) {
    if (message instanceof GridEvent && message.getFrom() == "Data
        Generator") {
      logger.info("Received GridEvent message from DG. This message type is
          allowed and will be forwarded to the module");
      return message;
    } else {
      logger.info("Received message type or sender which was not specified.
          This message type is not allowed and will not be forwarded to the
          module");
       return null;
    }
  }
}
```

Listing 2: Example outgoing proxy for an analytics module

```
public class AnalyticsOutProxy extends LayerOutProxy {

  private final Logger logger = ...;

  ...

  public Envelope apply(Envelope message) {
    if (message instanceof AddEntryRequest && message.getTo() ==
        "Storage")) {
      logger.info("Outgoing AddEntryRequest to Storage. This message type
          is allowed and will be delivered");
      return message;
    } else {
     logger.info("Outgoing message of forbidden type or forbidden receiver.
         Message will not be forwarded");
         return null;
    }
  }
}
```

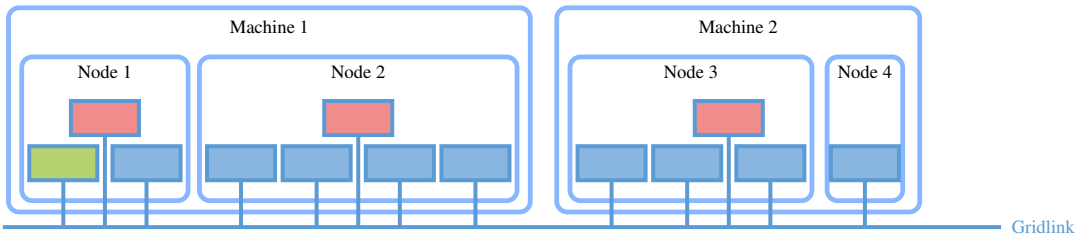


Figure 10: Solution with modules on two machines ("scale out")

able to receive information from the data generator (of the type *GridEvent*) and store information in the Storage module using the *AddEntryRequest* service. All other means of communication are blocked. While the provided example is very simple it demonstrates the application of proxies in the context of security.

As mentioned, the numbers of proxies used in the context of a module is not limited. Therefore, a number of proxies can be defined for an application allowing a proxy side customization of the incoming and outgoing data.

6 Conclusion and Outlook

To counteract the high memory demand of multiple Smart Grid application modules running each on its own JVM instance, we introduced the new *NodeManager* entity. Using node management, multiple Gridlink modules can be executed concurrently within one JVM without influencing installation, update and uninstallation during the runtime of the node and other modules. Memory usage was thus reduced by a huge extent as dependency libraries need to be loaded only once and no additional JVM caused memory overhead is introduced. Additionally required main memory just results from the module's implementation.

The evaluation shows that the described extension allows for modular applications plugged in (installed/started) and out (stopped/uninstalled) during the runtime of the node (Plug & Automate paradigm) and the concurrent execution of a high number of modules. With this solution, we raised a potential problem by a commonly used event bus which might lead modules to mutually block each other on the same node. We thus introduced some security measures (Trusted Gridlink Apps, Gridlink Security Layer) that allow to specify the allowed modules on one node and the allowed communication of modules without requiring changes in the (possibly third party) module's implementation. The proxy concept was demonstrated in a scenario where a malicious third party application tries to access information and services. This behavior can be prevented using application specific proxies which limit the information scope of each application.

In future work, it may be necessary to run modules on different machines (Scale Out, cf. Figure 10). While Gridlink has always been able to communicate beyond the barriers of physical machines, application provisioning is more complicated in this setup requiring file transfer to and process execution on the other machine. This extension would allow various use cases, such as load balancing and the use of fault tolerant and standby modules. Those functionalities are future work, once needed in a concrete use case.

Acknowledgements

The presented work is developed in the Smart Grid testbed of the Aspern Smart City Research (ASCR) and conducted

(i) in the "iNIS" project (849902), funded and supported by the Austrian Ministry for Transport, Innovation and Technology (BMVIT) and the Austrian Research Promotion Agency (FFG), and

(ii) in the "SCDA-Smart City Demo Aspern" project (846141), funded and supported by the Austrian Climate and Energy Fund (KLIEN).

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[2] R. E. Brown, "Impact of smart grid on distribution system design," in *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*. IEEE, 2008, pp. 1–4.

[3] S. Cejka, A. Frischenschlager, M. Faschang, and M. Stefan, "Security concepts in a distributed middleware for smart grid applications," in *Symposium on Innovative Smart Grid*

Cybersecurity Solutions 2017, Mar 2017, pp. 104–108.

[4] S. Cejka, A. Hanzlik, and A. Plank, "A framework for communication and provisioning in an intelligent secondary substation," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2016.

[5] S. Cejka, R. Mosshammer, and A. Einfalt, "Java embedded storage for time series and meta data in Smart Grids," in *2015 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Nov 2015, pp. 434–439.

[6] S. Cejka, A. Frischenschlager, M. Faschang, and M. Stefan, "Memory optimization of a distributed middleware for smart grid applications," in *Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security, IoTBDS 2017, Porto, Portugal, April 24-26, 2017*, 2017, pp. 331–337.

[7] CEN-CENELEC-ETSI, "Smart Grid Reference Architecture," CEN-CENELEC-ETSI Smart Grid Coordination Group, Technical Report, Nov 2012. [Online]. Available: http://ec.europa.eu/energy/sites/ener/files/documents/xpert_group1_reference_architecture.pdf

[8] CENELEC, "EN 50160:2010 - Voltage characteristics of electricity supplied by public electricity networks," Mar 2011.

[9] V. Chang and G. Wills, "A model to compare cloud and non-cloud storage of big data," *Future Generation Computer Systems*, vol. 57, pp. 56–76, 2016.

[10] Y. Chollot, P. Deschamps, A. Jourdan, and S. Mishra, "New approach to regulate low voltage distribution network," in *23rd International Conference on Electricity Distribution (CIRED)*, Jun 2015, paper 1145.

[11] G. L. De Alvaro, G. A. Taylor, D. C. Wallom, G. Gershinsky, A. Y. Huete, and K. Diwold, "High performance computing and communications technology solutions for future smart distribution network operation," *IET Conference Proceedings*, pp. 0730–0730(1), 2013.

[12] A. Einfalt, F. Zeilinger, R. Schwalbe, B. Bletterie, and S. Kadam, "Controlling active low voltage distribution grids with minimum efforts on costs and engineering," in *39th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, Nov 2013, pp. 7456–7461.

[13] A. Einfalt, S. Cejka, K. Diwold, A. Frischenschlager, M. Faschang, M. Stefan, and F. Kupzog, "Interaction of smart grid applications supporting plug & automate for intelligent secondary substations," *CIRED, Open Access Proceedings Journal*, vol. 2017, no. 1, pp. 1257–1260, Oct. 2017.

[14] M. Faschang, M. Stefan, F. Kupzog, A. Einfalt, and S. Cejka, "'iSSN Application Frame' – A flexible and performant framework hosting smart grid applications," in *CIRED Workshop 2016*, Jun 2016, paper 255.

[15] M. Faschang, S. Cejka, M. Stefan, A. Frischenschlager, A. Einfalt, K. Diwold, F. Pröstl Andrén, T. Strasser, and F. Kupzog, "Provisioning, deployment, and operation of smart grid applications on substation level," *Computer Science - Research and Development*, vol. 32, no. 1, pp. 117–130, 2017.

[16] K. Gama and D. Donsez, *Towards Dynamic Component Isolation in a Service Oriented Platform*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 104–120.

[17] T. Gawron-Deutsch, S. Cejka, A. Einfalt, and D. Lechner, "Proof-of-Concept for market based grid quality assurance," in *23rd International Conference on Electricity Distribution (CIRED)*, Jun 2015, paper 1495.

[18] T. Gawron-Deutsch, F. Kupzog, and A. Einfalt, "Integration of energy market and distribution grid operation by means of a flexibility operator," *e & i Elektrotechnik und Informationstechnik*, vol. 131, no. 3, pp. 91–98, 2014.

[19] N. Geoffray, G. Thomas, B. Folliot, and C. Clément, "Towards a new isolation abstraction for osgi," in *Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems*, ser. IIES '08. New York, NY, USA: ACM, 2008, pp. 41–45.

[20] H. Gharavi and R. Ghafurian, "Smart grid: The electric energy system of the future," *Proceedings of the IEEE*, 2011.

[21] V. Giordano and G. Fulli, "A business case for smart grid technologies: A systemic perspective," *Energy Policy*, vol. 40, pp. 252–259, 2012.

[22] C.-S. Li, H. Franke, C. Parris, B. Abali, M. Kesavan, and V. Chang, "Composable architecture for rack scale big data computing," *Future Generation Computer Systems*, vol. 67, pp. 180–193, 2017.

[23] M. Mangani, F. Kienzle, M. Eisenreich, Y. Farhat Quinones, R. Bacher, and A. Brenzikofer, "GridBox: An Open Platform for Monitoring and Active Control of Distribution Grids," in *23rd International Conference on Electricity Distribution (CIRED)*, Jun 2015, paper 1070.

[24] A. Plank, F. Zeilinger, and A. Einfalt, "Untersuchung der Effektivität von Regelkonzepten in Verteilnetzen," in *14. Symposium Energieinnovation, Graz, Austria*, Feb 2016, in german.

[25] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb 2016.

[26] T. Schuster, "Auswirkungen der unsymmetrischen Belastung im Niederspannungsnetz für dezentrale Energieeinspeiser," in *Tagungsunterlagen Eninnov 2014*, 2014, in german.

[27] M. L. Tuballa and M. L. Abundo, "A review of the development of smart grid technologies," *Renewable and Sustainable Energy Reviews*, vol. 59, pp. 710–725, 2016.

[28] D. Uckelmann, M. Harrison, and F. Michahelles, "An architectural approach towards the future internet of things," in *Architecting the internet of things*. Springer, 2011, pp. 1–24.

[29] X. Yu and Y. Xue, "Smart grids: A cyber-physical systems perspective," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1058–1070, May 2016.

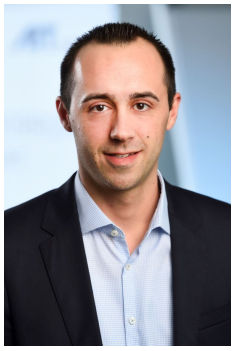
[30] F. Zeilinger, A. Einfalt, K. Diwold, A. Plank, and A. Lugmaier, "Influence of Different Framework Conditions on the Effectiveness of Control Concepts in Distribution Grids," in *CIRED Workshop 2016*, Jun 2016, paper 259.

AUTHOR BIOGRAPHIES

Stephan Cejka joined the department for Corporate Technology of Siemens Austria in 2014. There he works as Research Scientist and Junior Expert in a research group for Smart Grids and Smart Buildings. He received his bachelors degree in Software & Information Engineering from Vienna University of Technology in 2013 and his magister degree in Laws from University of Vienna in 2016. Currently, he is enrolled in the master study "Software Engineering & Internet Computing" at Vienna University of Technology and in the PhD study in Laws at University of Vienna. Research interests include distributed systems, data storage and privacy in the Smart Grid area.

Albin Frischenschlager was born in 1988 in Vienna. He received a master degree in Computer Engineering from the Vienna University of Technology. His diploma thesis was about: "Autonomous path planning using probabilistic maps". During his studies, Albin Frischenschlager was employed as system-administrator and software developer for different companies. Since October 2014, Albin Frischenschlager is working in a Smart Grid research group of Corporate Technology at Siemens AG Austria as a project manager and software developer.

Mario Faschang was research engineer and project manager at the Energy Department of AIT Austrian Institute of Technology GmbH in Vienna until August 2017. Faschang holds a M.Sc. (Dipl.-Ing.) degree (2011) in electrical engineering and information technology from TU Vienna and a Ph.D. in engineering science (2015). Before joining AIT, he was with Siemens AG Austria and research assistant at TU Vienna. Faschang's research focuses on future power grids and voltage control in low voltage distribution grids with high share of distributed, alternative power generation and electro mobility. Faschang was awarded the Austrian INiTS award, the "green tech" award, and the Austrian Smart Grids Pioneer Award together with his colleagues in 2012. He is member of Austrian Electrotechnical Association (OVE), IEEE Austria, and officer of IEEE Austria Young Professionals Affinity Group.



Mark Stefan studied Computer Science (Bachelor and Master) at the Vienna University of Technology. He started his professional career at Robert Bosch AG in Vienna (software and function development, project management) where he was working for about 2.5 years. In 2012, he joined the Institute of Computer Aided Automation at the Vienna University of Technology, working as project assistant and doing his PhD-studies. He developed an algorithm for optimizing railway systems in terms of deadlock detection and avoidance as well as the minimization of the traction energy consumption. Since June 2014, he is working as Research Engineer and Project Manager at the AIT Austrian Institute of Technology GmbH. Since 2014, Dr. Stefan holds lectures at St.Pölten University of Applied Sciences (Application of Graphs in the Railway Sector).

Konrad Diwold received a master degree in Artificial Intelligence from the Free University Amsterdam (Netherlands) in 2007. From 2008 until 2011 Konrad Diwold was a research associate at the parallel and complex systems workgroup at the University of Leipzig (Germany). His work concerned biological inspired algorithms. He received a Ph.D. in computer science from the University of Leipzig in 2012. From 2011 until 2014 Konrad Diwold was a research associate at the Fraunhofer Institute for Wind Energy and Energy System Technology (Kassel, Germany) in the department of distribution system operation. His work concerned the smart integration of renewable energy resources in distribution system operation. Since January 2015, he is working in a research group of Corporate Technology with focus on Smart Grid technologies at Siemens AG Österreich.