Scaling Home Automation to Public Buildings: A Distributed Multiuser Setup for OpenHAB 2

Florian Heimgaertner, Stefan Hettich, Oliver Kohlbacher, and Michael Menth
University of Tuebingen, Department of Computer Science, Tuebingen, Germany,
Email: {florian.heimgaertner,menth,oliver.kohlbacher}@uni-tuebingen.de, stefanhettich@gmail.com

***Abstract*—Home automation systems can help to reduce energy costs and increase comfort of living by adjusting room temperatures according to schedules, rules, and sensor input.**

OpenHAB 2 is an open-source home automation framework supporting various home automation technologies and devices. While OpenHAB is well suited for single occupancy homes, large public buildings pose additional challenges. The limited range of wireless home automation technologies requires transceivers distributed across the building. Additionally, control permissions need to be restricted to authorized persons.

This work presents OpenHAB-DM, a distributed OpenHAB 2 setup with extensions introducing user authentication, access control, and management tools for decentralized OpenHAB node deployment.

I. Introduction

Open-source home automation systems like OpenHAB [1] and FHEM [2] can help to reduce energy costs and increase living comfort in private homes. Heating, lighting, roller shutters, and various other devices can be controlled according to schedules or based on events and complex rule sets.

Large public buildings could also benefit from open-source building automation. However, home automation solutions targeting single occupancy houses lack several features required for large public buildings with hundreds of rooms.

Most new buildings are designed with building automation in mind and include appropriate wiring and devices. For retrofitting in existing buildings, the most convenient way to connect temperature sensors, radiator valves, presence detectors, and temperature control switches to a home automation controller are wireless technologies like enOcean [3] or z-Wave [4]. The limited range of wireless transceivers is sufficient for most single occupancy houses. Figure 1 compares the situation in small residential buildings to large public buildings. While a single transceiver directly connected to the home automation controller can reach devices in the entire home, in large buildings multiple transceivers are required to achieve sufficient wireless coverage. To manage all devices from a single controller instance, a method to connect the distributed transceivers to the central controller is needed.

The objective of this work was to develop an extensible and user-friendly open-source solution for controlling a large number of devices distributed over a wide area. OpenHAB was selected as base framework because of its support for various home automation technologies through extension modules. To leverage the flexibility of OpenHAB for large public buildings, we propose OpenHAB-DM[1], a distributed multi-user setup of a modified version of OpenHAB 2. OpenHAB-DM uses OpenHAB *slave* controllers on low-power hardware providing connectivity and hardware abstraction for wireless transceivers. Automation features and user interfaces of OpenHAB are exclusively provided by a central *master* controller running on regular server hardware. A publish/subscribe (pub/sub) middleware is used to connect the slave controllers to the master controller.

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-1. The authors alone are responsible for the content of the paper.

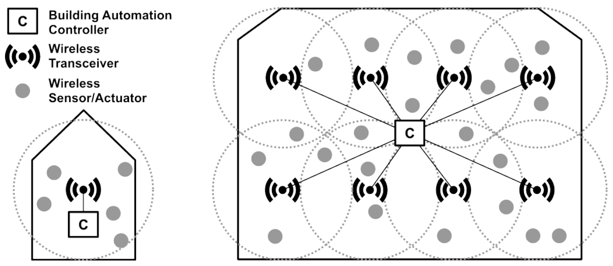


Fig. 1. The range of a single wireless transceiver is sufficient for small residential buildings (left). In large public buildings (right), multiple transceivers are required for full wireless coverage.

A single user interface for building automation in a large public building increases the need for authenticating users and restricting access to room configurations to the actual occupants of a room. Additionally, special permissions are required for administration and facility management staff. We propose a modification to the OpenHAB 2 software adding user authentication and means to restrict access based on group membership of a user.

This work is structured as follows. We give an overview of OpenHAB and the MQTT middleware in Section II. Section III describes the distributed OpenHAB architecture and presents the modifications made to OpenHAB. We present a use case for OpenHAB-DM in Section IV and discuss related work in Section V. Finally, we summarize this work and draw conclusions in Section VI.

II. Description of Base Technologies

In this section, the main building blocks of this work are introduced. We describe the home automation framework OpenHAB and the pub/sub middleware MQTT.

[1]`https://github.com/uni-tue-kn/openhab-dm`

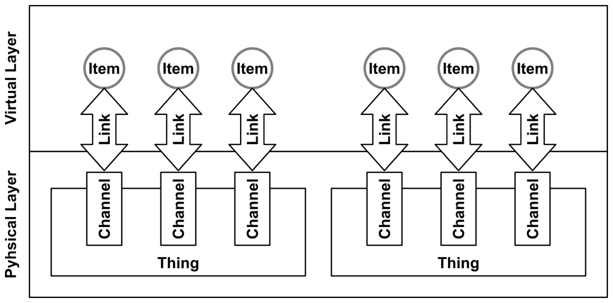


Fig. 2. The virtual layer of OpenHAB consists of items that are linked to the channels of things in the physical layer

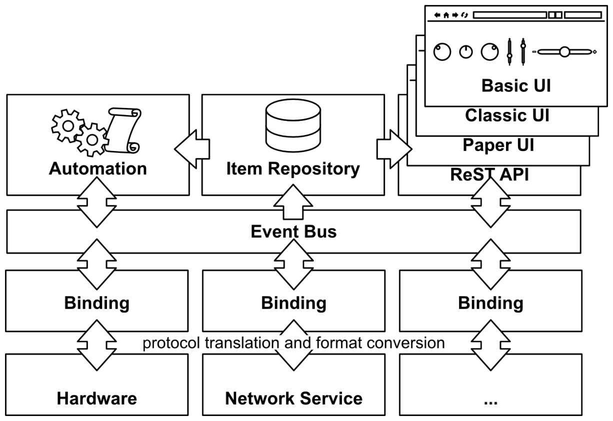


Fig. 3. Architecture of OpenHAB 2. OpenHAB controls sensors, actuators, and network services by sending commands and receiving state updates over an event bus. Bindings connect hardware and network services to the event bus by translating and forwarding events.

A. OpenHAB

The *Open Home Automation Bus* (OpenHAB) is an open-source home automation controller implemented in Java. OpenHAB is a vendor-independent solution for home automation. It supports a large number of home automation technologies such as various types of wireless sensors, switches, and actuators. The current development branch OpenHAB 2 is based on the Eclipse SmartHome (ESH) [5] framework. OpenHAB 2 and ESH are available under the terms of the Eclipse Public License.

Figure 2 illustrates the relationship between *things*, *channels*, and *items*, which are the basic concepts of OpenHAB and ESH. Things are physical devices or services that can be connected to the system. A thing provides functionality through one or multiple channels. Things and channels constitute the *physical layer* of ESH. Items are objects of ESH's *virtual layer*. They are abstract representations of the functions provided by a channel. An item is connected to a channel using a *link*. Items have a state and can receive commands. A wireless radiator valve may be an example for a thing providing two channels. The first channel is a sensor measuring the room temperature. The second channel is a actuator adjusting the valve position. Those channels are represented in OpenHAB as two items, the first giving the actual temperature, the second providing the set-point temperature.

User interaction and automation tasks only operate on item level. Technical details are hidden by the abstraction provided by the *bindings*. Bindings are extension modules connecting devices and external services to OpenHAB. They implement communication protocols and convert data formats to provide an abstract representation of hardware or network services in form of things, channels, and items. As shown in Figure 3, bindings are connected to an *event bus* for asynchronous communication with other OpenHAB components. They receive commands from the event bus, translate them into hardware- or service-specific formats and send them to the devices or services using the respective communication protocols. The state of items is kept in a data structure called *item repository*. The item repository is updated when state changes are received from the event bus and can be queried by automation logic and user interfacess.

OpenHAB automation is based on *events*, *rules*, *scripts*, and *actions*. Events represent state or time changes. Rules execute scripts or actions after being triggered by events. Scripts contain reusable code that can be used by multiple rules. Rules and scripts are written in a scripting language using a Java-like syntax. Actions are predefined Java methods that can be used in rules and scripts.

OpenHAB provides multiple interfaces for interaction with users and applications. System configuration and administration tasks can be performed using the *Paper UI* web interface. For user access to the home automation functionality, OpenHAB provides the web interfaces *Basic UI* and *Classic UI*. While Basic UI and Classic UI provide the same set of features, they are presented in different look-and-feels. A ReSTful [6] web service provides an API for mobile apps. The web interfaces also use the ReST API for AJAX calls.

OpenHAB and ESH are modular systems consisting of OSGi [7] bundles. Apache Karaf [8] and the Eclipse Equinox [9] framework are used as runtime environment.

B. MQTT

The pub/sub communication paradigm decouples communication partners in space, time, and synchronization [10]. Data is sent by *publishers* to a *broker* which forwards data to interested *subscribers*.

The MQ Telemetry Transport (MQTT) [11] is a broker-based, light-weight pub/sub middleware which runs on top of TCP/IP. Communication in MQTT is organized in *topics*. A topic is an abstract representation of a unidirectional information channel and is addressed using its unique name. By subscribing to a topic a subscriber expresses its interest to receive data published to that topic. The MQTT broker keeps track of topic subscriptions and forwards data received from the publishers to the subscribers of corresponding topics. MQTT topics are organized in a hierarchical name space. Like in Unix file system paths, the hierarchy levels are separated by forward slashes. Topics can be subscribed either using fully

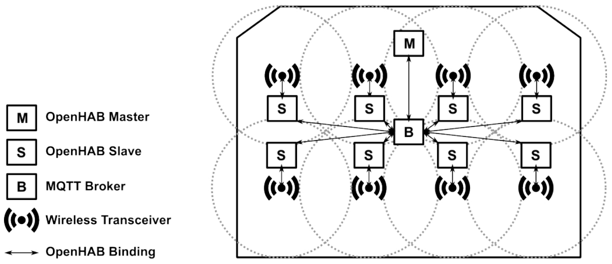


Fig. 4. An OpenHAB master controls sensors and actuators through the wireless transceivers of multiple slaves using broker-based pub/sub communication.

qualified topic names or using wildcards replacing one or multiple hierarchy levels. MQTT is not specifically designed for home automation purposes. However, it is marketed as a pub/sub middleware for the *Internet of Things* (IoT) and has gained a certain popularity in the home automation community. In our work we use the open-source MQTT broker Mosquitto [12].

III. OpenHAB-DM: A Distributed Multiuser OpenHab 2 Setup for Large Public Buildings

We propose OpenHAB-DM, consisting of an OpenHAB 2 master controller and multiple slave controllers interconnected by MQTT. This section describes the distributed system and the extensions and modifications made to OpenHAB 2 for implementing user authentication and access control.

A. Interconnection of Distributed OpenHAB Controllers

In the following, we motivate our work, describe the architecture of the system, and present the extensions to OpenHAB for management of slave controllers.

1) Motivation and Concept: To connect distributed wireless transceivers to a central controller over IP networks, gateways are required. For some wireless systems, technology-specific gateways are available. However, this approach would require the deployment of new gateways to support additional wireless technologies. Instead, we use multiple OpenHAB instances running on low-cost and low-power hardware as technology-independent gateways. Figure 4 illustrates the interconnection of OpenHAB slaves to an OpenHAB master. Slave controllers equipped with wireless transceivers are distributed over the building to ensure full wireless coverage. The slave controllers are connected to the master controller using the MQTT pub/sub middleware.

2) Architecture: We propose an architecture composed of a central *master* OpenHAB controller, an MQTT broker, and multiple distributed *slave* OpenHAB controllers. The master controller provides the user interfaces, runs automation tasks, and authenticates the users. A large number of slave controllers are distributed over the building. The slave controllers connect to sensors and actuators through the attached wireless transceivers using the appropriate bindings. The slaves operate without web user interfaces.

To visualize item states and run automation tasks, representations of the slaves' items need to be present at the master

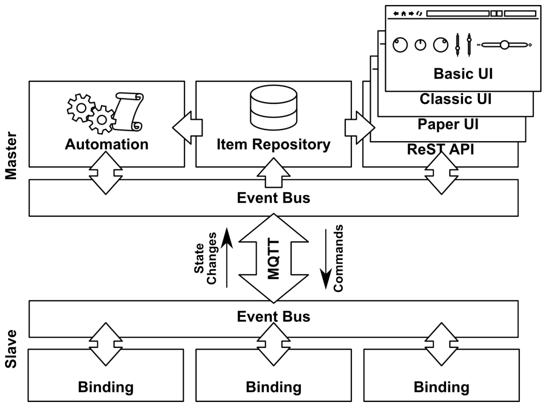


Fig. 5. Architecture of a distributed OpenHAB-DM system. Master and slave controllers are connected by coupling their event buses over MQTT.

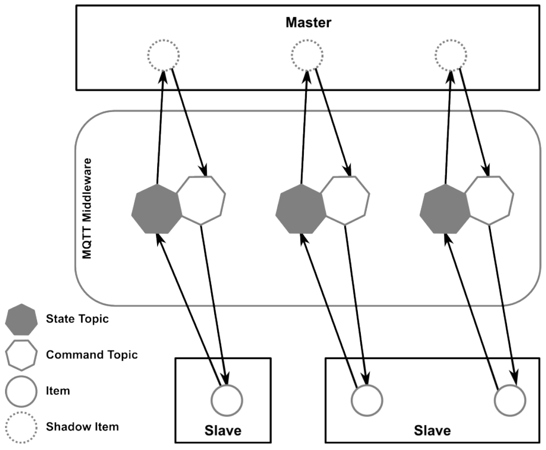
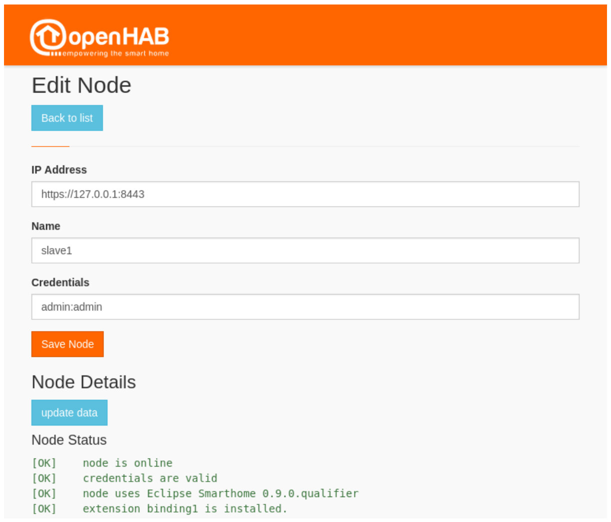


Fig. 6. Master and slave controllers communicate over the MQTT pub/sub middleware. Each item at the slaves is mapped to a pair of MQTT topics for exchanging state updates and commands with a shadow item at the master.

controller. In this work, we use the term *shadow item* to describe the representation of a remote item at the master controller. Figure 6 illustrates the use of the MQTT middleware to synchronize the state of items at the slave controllers and shadow items at the master controller. OpenHAB provides MQTT bindings [13] which can be used to interconnect multiple OpenHAB instances. At the slave controllers, we use the *event bus binding* level of the MQTT binding. This mode of operation exposes the entire event bus of the slave controller to the pub/sub middleware. For each item available at the slave, two MQTT topics are created. One MQTT topic is used for publishing item state updates. A second MQTT topic is used for receiving commands for the item from the master controller. At the master controller, *item binding* level is used. For each slave item managed by the master controller, a shadow item needs to be created at the master. The shadow items are implemented by the MQTT bindings and need to be configured with the correct item types and the MQTT topics of the corresponding remote items.

The overall architecture is shown in Figure 5. The bindings installed at the slaves are connected to the slaves' event buses.



openHAB
empowering the smart home

Edit Node

Back to list

IP Address

https://127.0.0.1:8443

Name

slave1

Credentials

admin:admin

Save Node

Node Details

update data

Node Status

```
[OK]    node is online
[OK]    credentials are valid
[OK]    node uses Eclipse Smarthome 0.9.0.qualifier
[OK]    extension binding1 is installed.
```

Fig. 7. Screenshot of the management UI used for configuration of OpenHAB slave nodes and remote items.

Events on the event bus of a slave are propagated to the event bus of the master via the MQTT middleware. At the master, they are made available to user interfaces and automation logic. Commands are issued by users or automation tasks at the master. The commands are propagated through MQTT to a slave's event bus and are processed by the respective bindings.

We use a Mosquitto broker co-located with the master controller.

3) Management of slave nodes: To configure the OpenHAB slaves via the master's web interface, we implemented a node management component for OpenHAB. Figure 7 shows a screenshot of the node management UI. The web interface can be used to view the slave status or install bindings and other extensions on slaves. Also items can be added to the slaves. For native OpenHAB 2 bindings, the node management component can create the items using the ReST API. For legacy OpenHAB 1 bindings, items require a file-based configuration. We have implemented remote configuration of items for selected OpenHAB 1 bindings. When adding items, the node management component automatically creates the corresponding shadow items at the master.

B. Authentication and Access Control

OpenHAB 2 currently has no concept of users, roles, or permissions. We modified the OpenHAB software to authenticate users and restrict permissions depending on group membership. In the following, we explain our concept of roles and permissions in OpenHAB. We show how this modification was implemented and point out its limitations.

1) Groups and Permissions: Items in OpenHAB can be organized into *groups*. An example of a group hierarchy is shown in the left part of Figure 8. One or multiple parent groups can be assigned to an item. Also, groups can be parent groups of other groups. The group hierarchy forms a directed graph with groups as vertices, parentship as edges, and items as sinks. The edges are directed from parent groups to child groups or items.

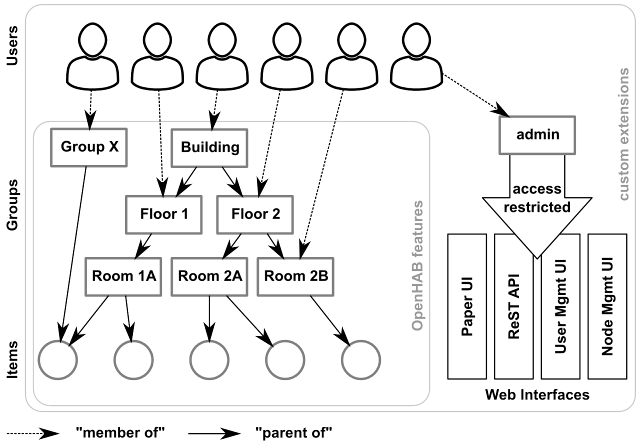


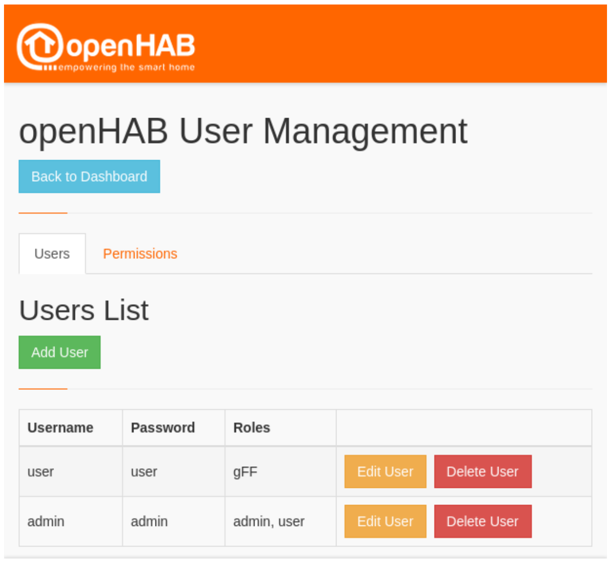
Fig. 8. Example of a group hierarchy. Groups and parents exist in OpenHAB. Users, their membership in groups, and restricted access to web interfaces are introduced by OpenHAB-DM.

While originally intended for visualization, aggregation, and configuration purposes, we adopt this group concept for permission management in OpenHAB-DM. Our implementation uses groups to determine whether a user is allowed to access an item. By mapping users to groups, they are added to the graph as source vertices with membership as edges directed to groups. Figure 8 illustrates the groups and permission concept of OpenHAB-DM. Items are organized in a hierarchy of groups. Users can be members of groups at any level of the hierarchy. If a directed path from a user to an item exists, the user is permitted to access the item.

Roles, e.g., `admin`, are implemented as special groups without parents or children. They are used to grant access to management interfaces or override access restrictions to items.

2) Implementation: As explained in Section II-A, OpenHAB provide several user interfaces. Besides Paper UI for administration, Basic UI and Classic UI are available for regular user operations. In a default OpenHAB 2 deployment, those user interfaces are accessible to anyone inside the local area network without authentication.

OpenHAB-DM implements user authentication for the OpenHAB web user interfaces. Users are required to provide credentials to log in and access the items they are permitted to use. The credentials entered by the user are verified using an internal user database or by asking an external authentication backend (e.g., LDAP). After a user is authenticated, we store user data in a session and the user can access the user interfaces. Session data is managed using the Java Servlet API at the server side, and stored in session cookies at the client. OpenHAB-DM uses a modified `HTTPContext` to test if a session has been established and if a user is permitted to access a resource. If the session is valid but the user is not permitted to access an item, an error message is displayed. If no valid session is found, the user is redirected to a login form to ask for credentials.



openHAB User Management

Users List

Username	Password	Roles	
user	user	gFF	Edit User Delete User
admin	admin	admin, user	Edit User Delete User

Fig. 9. Screenshot of the management UI for configuration of users, roles, and permissions.

To authenticate to the ReST API, ReST clients include an authentication token with each request. The token is generated by the server upon authentication with the user credentials and stored locally by the client. A client submitting a request to the ReSTful web service without a valid authentication token will receive an HTTP 401 (Unauthorized) response. Requests are blocked, if a user does not have sufficient permissions to access a resource. In this case, an HTTP 403 (Forbidden) response is generated. This mechanism is implemented using a `ContainerRequestFilter`.

For use of the ReST API through JavaScript code from inside the web interfaces, a token is generated after successful user authentication and stored in a separate cookie. To protect session cookies and authentication tokens, access to the web interfaces and the ReST API should be possible over HTTPS only.

For both the ReST API and the web interface, only items a user may access should be visible. To achieve this, a filter is added to the methods enumerating the items.

Users only connect to web user interfaces or the ReST API at the master controller. The master enforces access control centrally according to the parent groups of the shadow items. A user database shared between master and slaves is not required, because access to the slaves is only possible via the master. Instead, the slaves only know a single admin user account which is used by the master for accessing the ReST API.

For management of user accounts, assigning group memberships, and setting permissions for the ReST API, a new web interface component was developed. Figure 9 shows a screenshot of this management interface. Access to the user management component requires membership in the `admin` group.

3) Limitations: The authentication and access control implementation for OpenHAB-DM is not compatible with the existing mobile OpenHAB client apps. The mobile apps currently do not implement the functionality to supply user credentials and authentication tokens to the ReST API. Therefore, authentication at the ReSTful web service is not yet supported.

IV. Use Case

Energy costs make up a notable part of the operating costs in large public buildings. Among those energy costs, heating expenses are especially hard to control. In traditional business buildings with fixed working hours, heating costs can be optimized centrally by using timers to adjust the flow temperature and circulation, and per room by tuning thermostatic valves. However, this approach is no longer sufficient with flexible time programs, part-time employment, occasional telecommuting, business travels, shared office space, and multipurpose rooms. Fixed temperature presets no longer match variable room allocation schemes. Additionally, the period of time when flow temperature can be substantially reduced is becoming smaller due to flexible working hours.

To prevent employees from freezing in cold rooms for the first hours after start of work, two solutions are commonly used. Either, radiator valves are left open even if the room is unused, or the flow temperature is raised to reduce the heat-up time. Those solutions are unfavorable both from a financial and an environmental point of view.

To cope with those challenges, we propose using a software system to control valves according to schedules or user configuration.

OpenHAB-DM enables using this technology to adjust room temperatures in large public buildings, e.g. using room schedules in CalDAV calendars. Users can be granted the permissions to control single rooms or groups of rooms using a central web frontend. Office rooms can be configured to preheat before the beginning of individual working hours. Room temperatures can be automatically reduced if the occupants are on vacation, or automatically raised if presence of persons is detected. This approach can reduce energy costs while not interfering with diverging and unusual working hours.

V. Related Work

In this section we give an overview of other open-source home automation solutions.

Freundliche Hausautomatisierung und Energie-Messung (FHEM) [2] is a home automation server implemented in Perl. FHEM is open-source software available under the terms of the GNU General Public License (GPL) v2. It supports various home automation protocols and technologies, e.g., EnOcean, HomeMatic, MAX!, Phillips HUE, etc. Additionally, cameras, online calendars, or weather data can be integrated. Several interfaces are available for interaction with FHEM. Applications can use XML or JSON based web services, or the request/response based telnet interface. Users can connect to one of the various web frontends or use mobile apps. Adding new devices to FHEM is particularly simple. Device representations are automatically created in the system as soon as data is received from the device. For automation purposes, *macros* containing sequences of commands can be executed at defined times or upon reception of an event. FHEM focuses on home automation in private premises. While the system can be protected from unauthorized access using HTTP Basic Authentication, multiple users with different permissions or fine-grained access control schemes are not supported. Connecting distributed FHEM instances is neither supported.

HomeAssistant [14] is a home automation system implemented in Pyhton 3. HomeAssistant is open-source software available under the terms of the MIT license. It is based on a modular architecture. Extension modules provide support for sensors, actuators, weather data, automation services, online calendars, and user interfaces. The system provides an integrated user and administration web interface that can be password protected. Events are used to trigger automation tasks. HomeAssistant includes a concept of multiple persons whose presence or absence can be used as a condition for automation rules. However, for the system persons are like sensors providing input data, e.g., by tracking their mobile phones. Multiple users with different permissions do not exist in HomeAssistant. Controlling remote instances of HomeAssistant from a central instance is not supported.

ioBroker [15] is a home automation system implemented in JavaScript and based on node.js. It is open-source software available under the terms of the MIT license. ioBroker is a modular system that can be extended using *adapters*. Various adapters are available for popular protocols and technologies, such as HomeMatic, MQTT, or EnOcean. ioBroker is the successor of CCU.IO, which was quite popular with the HomeMatic user community previously. Adapters implemented for CCU.IO can be used with ioBroker. ioBroker supports *multihost* mode, a distributed mode of operation. The primary purpose of multihost mode is load balancing but it can also be used to install adapters on distributed ioBroker instances. ioBroker supports multi-user operation and users can also be organized in groups. However, permissions can only be used to restrict administrative functions. Access control on device- or item-level is not supported.

MyController.org [16] is a home automation system implemented in Java. It is open-source software available under the terms of the Apache License 2.0. MyController.org uses JavaScript or Groovy for writing automation rules. MyController.org includes a single web frontend that is used for both administration and user access. It includes user management and supports a sophisticated system of roles and permissions. MyController.org can connect to external services and includes a built-in MQTT broker but distributed operation is not supported.

The Open Building Automation System (OpenBAS) [17] is a building automation platform currently developed at UC Berkeley. It is open-source software available under the terms of the 2-clause BSD License. OpenBAS is focused on control of heating, ventilation, and air conditioning (HVAC) in small and medium business buildings. OpenBAS is based on sMAP [18] which was also developed at UC Berkeley. sMAP uses gateways called *sMAP sources* that provide access to devices over a ReST API. OpenBAS only supports devices connected via sMAP sources so there is no use case for interconnection of multiple OpenBAS instances. OpenBAS includes a web frontend for both administration and user access. The frontend is password-protected, but role-based access control for specific devices is not implemented.

VI. Conclusion

In this paper, we proposed OpenHAB-DM, a distributed multi-user setup of OpenHAB 2. We showed, how OpenHAB slave controllers can be used to connect distributed wireless transceivers to a central OpenHAB master controller over a pub/sub middleware. The presented authentication and access control extensions enable multi-user operation in OpenHAB with fine-grained permissions based on item groups. The node management tool simplifies deployment and operation of the slave controllers and enables configuration of remote devices via the user interface of the central master controller.

Our enhancements to OpenHAB2 enable the reuse of existing technology for private homes in public buildings with large numbers of rooms and users.

Future work includes integration of additional external authentication backends such as Shibboleth or Active Directory, and possibly extensions for mobile apps to support authentication.

References

[1] K. Kreuzer *et al.*, “OpenHAB - empowering the smart home,” 2016. [Online]. Available: https://www.openhab.org/

[2] R. Koenig *et al.*, “FHEM Home Automation Server,” 2015. [Online]. Available: http://fhem.de/fhem.html

[3] EnOcean Alliance, “EnOcean - The World of Energy Harvesting Wireless Technology,” EnOcean Technology Whitepaper, 2015.

[4] Z-Wave Alliance, “Z-Wave,” 2015. [Online]. Available: http://www.z-wave.com

[5] Eclipse Foundation, “Eclipse SmartHome,” 2014. [Online]. Available: https://www.eclipse.org/smarthome/

[6] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Dissertation, University of California, 2000.

[7] OSGi Alliance, “Open Service Gateway initiative,” 2014. [Online]. Available: https://www.osgi.org

[8] Apache Foundation, “Apache Karaf,” 2014. [Online]. Available: http://karaf.apache.org/

[9] Eclipse Foundation, “Eclipse Equinox,” 2016. [Online]. Available: http://www.eclipse.org/equinox

[10] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, “The Many Faces of Publish/Subscribe,” *ACM Computing Surveys*, vol. 35, no. 2, pp. 114 – 131, 2003.

[11] A. Stanford-Clark and A. Nipper, “MQ Telemetry Transport,” 2014. [Online]. Available: http://www.mqtt.org/

[12] Eclipse Foundation, “Mosquitto - an Open Source MQTT Broker,” 2016. [Online]. Available: https://mosquitto.org/

[13] OpenHAB, “OpenHAB MQTT Binding,” 2013. [Online]. Available: https://github.com/openhab/openhab/wiki/MQTT-Binding

[14] P. Schoutsen *et al.*, “HomeAssistant,” 2016. [Online]. Available: https://home-assistant.io/

[15] ioBroker Team, “ioBroker - Automate Your Life,” 2016. [Online]. Available: http://www.iobroker.net/?lang=en

[16] J. Kandasamy *et al.*, “MyController.org - The Open Source Controller,” 2015. [Online]. Available: http://www.mycontroller.org/

[17] D. Culler *et al.*, “OpenBAS,” 2016. [Online]. Available: http://sdb.cs.berkeley.edu/sdb/OpenBAS/

[18] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler, “sMAP - a Simple Measurement and Actuation Profile for Physical Information,” in *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, Nov. 2010.