



VRIJE
UNIVERSITEIT
BRUSSEL



Graduation thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in Applied Sciences and Engineering: Applied Computer Science

A MIDDLEWARE FOR SMART INTERACTIONS ACROSS INTERNET OF THINGS PLATFORMS

YEVHEN DUKHNO
Academic year 2017–2018

Promoter: Prof. Dr. Beat Signer
Advisor: Payam Ebrahimi
Faculty of Engineering

Abstract

During the recent years, the market of the Internet of Things (hereinafter referred to as IoT) has been growing tremendously together with the number of different IoT platforms and heterogeneous smart devices. The lack of standards, as well as the variety of protocols and patents has led to so-called data silos, where isolated IoT products are not able to exchange information. In addition, in the current state of art, the ability to handle the complex rule behaviour of those products is not advanced enough.

In this thesis we introduce a solution in the form of an IoT middleware to enable smart interactions with a user across IoT platforms. During the research we reviewed the literature and works in the related domains and designed a middleware architecture. Moreover, we have defined a communication grammar which IoT platforms are required to be compliant with in order to support the suggested functionality of the middleware. We also provide some guidelines regarding the interactions with existing rule engines of IoT platforms. Furthermore, we have implemented a prototype of the middleware in order to justify the operability of the proposed solution.

Acknowledgements

First of all, I would like to thank my promoter Prof. Dr. Beat Signer to give me the opportunity to write a thesis at the WISE lab and to provide feedback on my thesis and presentations. The next person I want to thank is Payam Ebrahimi for his guidance and priceless support during the year. In addition, I want to thank my friends and classmates for their support. I wish also to thank my family for their help and support. Finally, I want to thank my girlfriend, Karina for her endless inspiration, support, help and patience during my whole master program at VUB.

Contents

1	Introduction	
1.1	Problem Statement	9
1.2	Contributions	9
1.3	Requirements for the Solution	10
1.3.1	Functional Requirements	10
1.3.2	Non-Functional Requirements	10
1.4	Thesis Outline	11
2	Related Work	
2.1	Literature Review	13
3	Design	
3.1	Components	23
3.2	Rules	24
3.3	User Interface	25
4	Implementation	
4.1	Architecture	27
4.2	Communication Convention	28
4.3	Data Management	34
4.3.1	Database Description	34
4.3.2	Database Schema	34
4.4	Interaction	36
4.5	Use Cases	38
5	Future Work and Conclusions	
5.1	Future Work	43
5.2	Conclusions	44
A	Appendix	

1

Introduction

The IoT nowadays is quite a hot topic as well as an increasing trend in the digital world. Gartner, Inc provides the following definition of the IoT: “*it is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment*”¹. A “boost” of the IoT trend was accelerated by a massive expansion of wireless networks starting around 2010.

The area of applications for the IoT is quite broad. Based on experiments with focus groups a research team from the University of Stuttgart [23] in 2016 came to the conclusion that smart devices (elements of the IoT system) in the near future would at least partially substitute a smartphone as the main notification device for incoming messages or upcoming appointments. Based on Figure 1.1, where one of the worksheets used in the focus groups is shown, we can see the opinion of a participant concerning the desired “smart” setting of a living-room which might be achieved by using different kinds of IoT devices. As an example and a proof of concept, Scheible et al. introduce the SMARTKITCHEN project [18]. The idea is to supply the cooking area with the media content display system (i.e. a projector) presenting context-relevant information based on current cooking activities. A quick overview is shown in the drawing in Figure 1.2.

Due to the growth of the IoT trend, a demand on satisfaction of user’s

¹<http://www.gartner.com/it-glossary/internet-of-things>

Figure 1.1: Living room setting with participant's drawings [23]



Figure 1.2: Media-enhanced cooking environment [18]

needs is also increasing. In their paper Mennicken et al. [12] are discussing the possible future of home automation research. The studies discovered that people did not consider their homes to be “smart” if they were themselves better or more efficient in carrying out the tasks the home was supposed to automate. The development and implementation of “translating” and conveying to the machine how inhabitants define their homes in their natural, less technical understanding is considered as one of the current challenges in order to substitute existing approaches for controlling and accessing individual devices and creating connections between them using the vocabulary and metaphors that were traditionally developed for people with a technical background. Analysing the work, we can come to a conclusion that the one of general goals for developers in terms of Home Automation is to simplify the workflow/interaction with the system for an average not really technical-savvy user.

There are number of research projects aiming to improve and enrich the ways of interaction between a user and an IoT system based on a mixed reality approach. According to Microsoft², *Mixed Reality*³. is the result of blending the physical world with the digital world; this definition is illustrated in Figure 1.3. Among those research projects is a research team led by Ullah [20] that published an augmented reality smart home control concept illustrated in Figure 1.4, Rashid and Co [17] who had offered a prototype of improved IoT system based on using radio-frequency identification tag technology and a smartphone as a scanner in order to improve user’s experience while doing book shopping in the offline stores.

Another research team [8] suggests to use the *Global Public Inclusive Infrastructure* (GPII)⁴ in order to better perform the context aware tasks as well as to adapt the environment to multi-user case.

The authors highlight a problem of intelligibility and control in smart environments outlining a number of challenges and opportunities [22]. Not the least of them are issues of multi-user intelligibility: accountability (some user’s action might impact other inhabitants) and privacy with security (sensitive information of the user should not be shared with other inhabitants or guests).

So what does “Things” mean in the IoT? In this context when we say “a thing” we mainly understand tags (i.e. RFID⁵, QR codes⁶), devices (i.e. Ar-

²<https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality>

³<https://docs.microsoft.com/en-us/windows/mixed-reality/mixed-reality>

⁴<https://gpil.net>

⁵<https://en.wikipedia.org/wiki/rfid>

⁶<https://web.archive.org/web/20130129064920/http://www.qrcode.com/en/qrfeature.html>

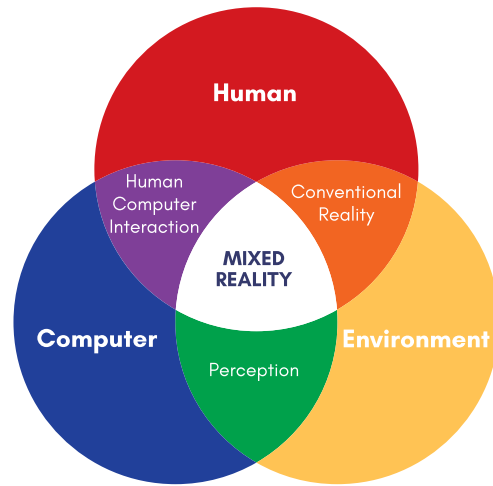


Figure 1.3: What is Mixed Reality? (source: docs.microsoft.com)

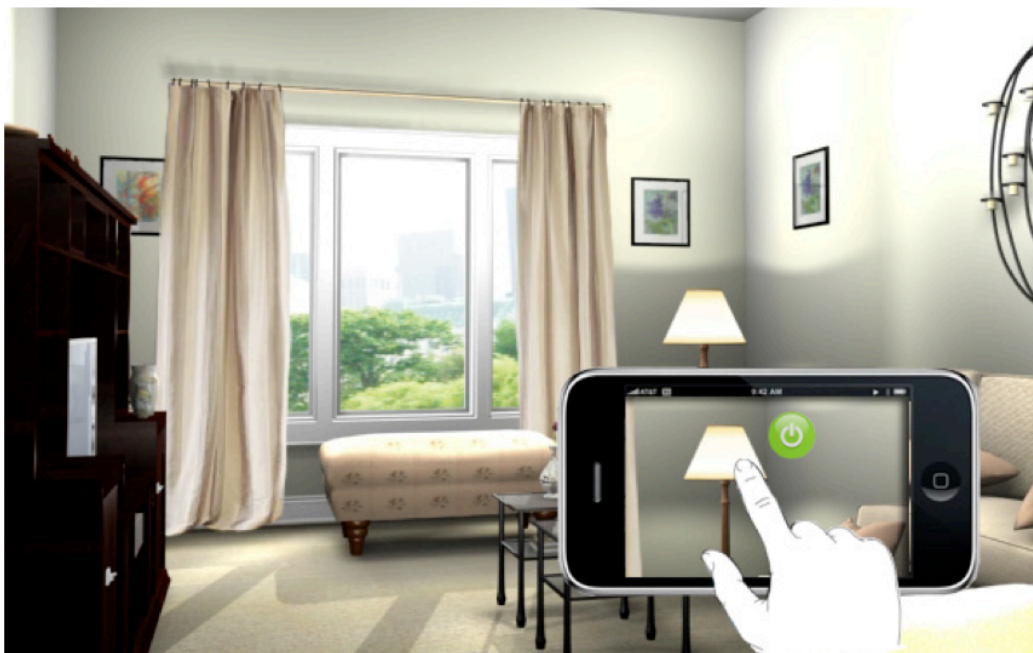


Figure 1.4: Smartphone as a remote touch controller [20]

duino⁷, Raspberry Pi⁸), machines (i.e. Philips Hue⁹, smart cars) and environments (i.e. smart buildings, smart cities). Focusing on the networking protocols and communication of the “things”, the IoT approach implies the existence of a large number of different solutions also known as platforms with different sets of the protocols. This, according to World Wide Web Consortium¹⁰, leads to so-called data silos, high costs and limited market potential of the IoT since coming to machines and environments, complexity and power of computation increases dramatically. Therefore, the World Wide Web Consortium has introduced the concept of the Web of Things illustrated in Figure 1.5¹¹. The idea is to consider each “thing” as a separate autonomous web server providing resources to other elements of a system [11].

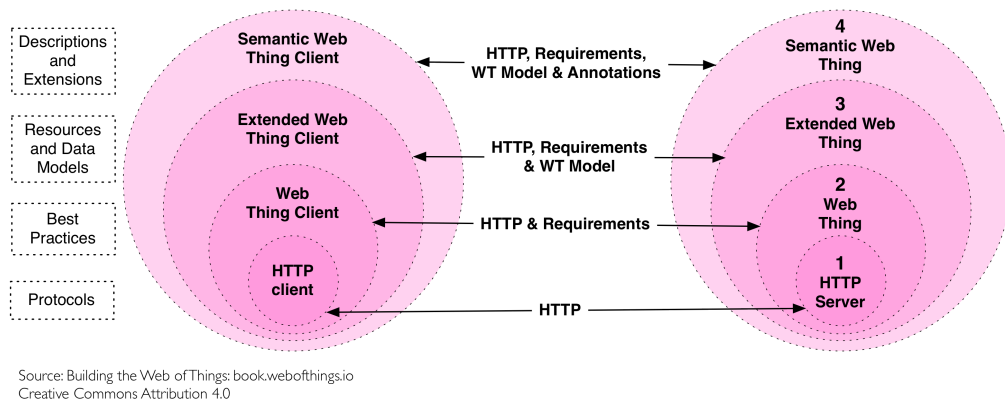


Figure 1.5: Web of Things model (source: w3.org)

The founders of the WebofThings.org community¹², Guinard and Trifa [7], define the Web of Things as a subdivision of Application Level of the TCP/IP

⁷<https://www.arduino.cc>

⁸<https://www.raspberrypi.org>

⁹<https://www2.meethue.com/en-us>

¹⁰<https://www.w3.org/WoT>

¹¹<https://www.w3.org/WoT/IG>

¹²<https://webofthings.org>

model and come up with the structure of the Web of Things shown in Figure 1.6. Its architecture consists of Networked Things, an Access layer, a Find layer, a Share layer and a Compose layer. Our interest will be focused on the IoT Application Level meaning Web of Things, therefore we will discuss each layer of it in more details.

Networking Things are in charge of all low-level functionality. A great number of work has been accomplished in this domain researching transport and network protocols, including the research of Mizutani and Mitsugi [14] where scientists from Keio University, Japan present an RFID emulator providing a universal Low Level Reader Protocol compatible testing environment as automatic identification system. The problem of all research projects limited to this layer is a variety of protocols which results in a fragmentation for higher layers of the IoT.

The *Layer 1* (access) manages the connectivity of things to the Internet. A very important point is to transform a thing to the web-service providing its application programming interface (API) to other things. Compliance of the API with REpresentational State Transfer (REST)¹³ allows a thing to be seamlessly integrated to the World Wide Web, having a personal unique Uniform Resource Locator (URL) [6].

The *Layer 2* (find) helps to make things findable and automatically usable by another things and not only reachable by HTTP clients [7]. This is achieved based on a semantic web approach¹⁴. The term Semantic Web was introduced by Tim Berners-Lee in 1998 and is supported by the World Wide Web Consortium. For instance, a team at the University of Aberdeen [16] introduced a semantic framework together with the semantic model in Figure 1.7 for reasoning about the capabilities of IoT devices based on original information collected from things and their associated services.

The idea of the *Layer 3* (share) is to supply the IoT with the secure and effective way to transfer data among things. Hereby different protocols and technologies such as various authorisation standards like OAuth¹⁵, OpenID¹⁶ and others, API tokens or different encryption techniques can be used. Even social networks can be used for sharing the data.

However, the domain we would like to focus on is the *Layer 4* (compose). The reason of existence for this layer is a need to build large-scale, meaningful applications for the Web of Things¹⁷. These applications, also known as physical mashups, are developed in order to provide users with little or no

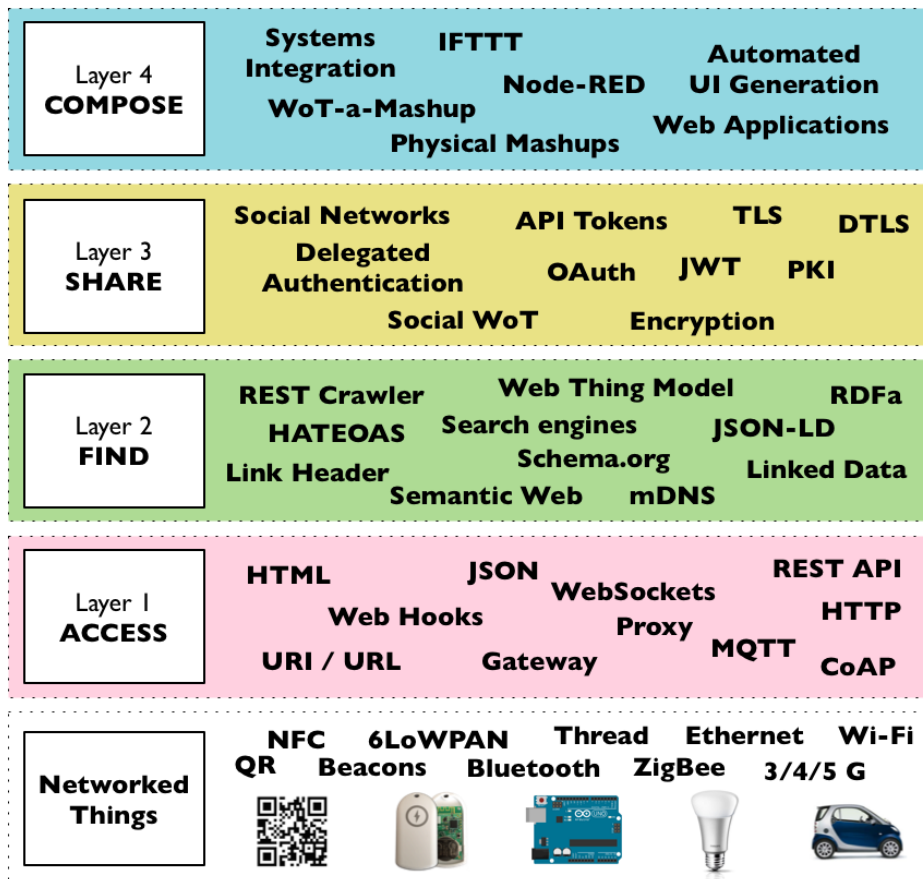
¹³<https://www.w3.org/TR/2004/NOTE-ws-arch-20040211>

¹⁴<https://www.w3.org/DesignIssues/Semantic.html>

¹⁵<https://oauth.net>

¹⁶<https://openid.net>

¹⁷<https://webofthings.org/2017/04/08/what-is-the-web-of-things>



Source: Building the Web of Things: book.webofthings.io
 Creative Commons Attribution 4.0

Figure 1.6: Web of Things architecture [7]

technical background with the opportunity to develop mini web applications, flexibly automatise their everyday routines. Also they are meant to allow more intelligent context awareness for smart environments and to manage the complex behaviour of future IoT systems.

One of the most famous projects in this domain is IFTTT¹⁸. It is a mashup-service which helps to combine different web applications in one tool. The working principle is quite simple. IFTTT allows to create and manage tasks. Each task consists of two parts: “Trigger” and “Action”. The trigger is a certain condition, when the action should be executed. Generally, this model can be described as *If This Then That* (the name of the service is an acronym of this sentence). A task can, for example, be to set the thermostat to day temperature when the alarm is stopped¹⁹. Nevertheless, IFTTT as well as the other services that have similar functionality (i.e. Zapier²⁰, SmartThings²¹) do not provide an option to create more complex IoT behaviour.

1.1 Problem Statement

A number of IoT platforms is constantly growing and this results in the increase of complexity and time spent for users that have to manage various devices. At the same time, there is no strategy on unification in order to adopt devices with smart behaviour by multiple IoT platforms. This means, that the issue of the data silos²² in the IoT is still not solved. In addition, rule engines in IoT platforms are not mature enough to support the authoring of the complex context rules. According to Ur et al. [21], a user nowadays has a need in a context-aware automated environment for their smart home. Hence, there is a necessity in a context-aware unifying system which will facilitate the seamless experience of the user interaction in the IoT world.

1.2 Contributions

Addressing the problem that is mentioned in Section 1.1, we offer a solution which integrates heterogeneous IoT platforms. It allows a unified intelligent seamless rule transition involving devices from different platforms. This thesis has the following contributions:

¹⁸<https://ifttt.com>

¹⁹<https://ifttt.com/collections/iot>

²²<https://www.w3.org/WoT>

- *The middleware.* In order to achieve the interoperability for IoT platforms, we provide an IoT middleware. This software is supposed to serve as a hub for IoT platforms and to allow a user to author rules between smart devices from those platforms.
- *Smart interactions.* This feature is achieved by embedding a declarative rule-based approach in the middleware. It will help the middleware to be aware of a user's context and to trigger implicit interactions across various IoT platforms.
- *Communication standard.* Along with the middleware we propose a standard for grammar, which has been inspired by the Web of Things, that is required to be adopted by IoT platforms to communicate with our middleware.
- *Approach on how to deal with individual platform features.* As already mentioned above, some IoT platforms might have their own rule engines. Our solution provides a strategy for handling their activity in order to avoid possible ambiguities.

1.3 Requirements for the Solution

We need to define a set of requirements for the proposed solution in order to achieve the mentioned goal. Functional, as well as non-functional requirements are listed in the sections as follows.

1.3.1 Functional Requirements

- A solution should provide a universal communication convention between IoT platforms and the middleware.
- A solution should be compliant with the Web of Things standard.
- The middleware should be interoperable. By “interoperability” we mean allowing a user to author rules for connected platforms using the middleware interface.

1.3.2 Non-Functional Requirements

- A solution should support the suggested communication convention.
- A system should be able to process and retain all needed data for a proper functioning.

1.4 Thesis Outline

Here is the proposed framework for the rest of the thesis. Observations of the performed literature review are described in Section 2. Section 3 contains a discussion about the design of the introduced middleware. Moreover, we provide details of the user interface together with the suggestions on the overcoming challenges related to the rule handling. Technical details of the proposed solution along with the communication grammar are explained in Section 4 and cases are discussed in Section 4.5. Finally, Section 5 consists of conclusions and future work.

2

Related Work

2.1 Literature Review

As mentioned before, the focus of this research is to investigate efforts made in the domain of the *Layer 4* (compose) of the Web of Things architecture.

From year to year more and more researchers in the field of Computer Science independently come to the conclusion that there is a gap between users' demands and the current state of art. According to Jara et al. [10], a rapid development of the IoT market is happening now. However, the current IoT market is concentrated on the vertical integration of the services. Most of the present frameworks are using Machine to Machine (M2M) approach¹. Meanwhile, as shown in Figure 2.1, currently there is substantially more demand on the global interoperability of solutions that cannot be supplied by the modern IoT appearance. Authors state, that building a semantic Web of Things is the next big task of the IT industry.

Blackstock and Lea [2] in their paper investigate problems and challenges concerning interoperability across the Web of Things. Among the objectives of their research, the authors set a goal of addressing the following questions:

- *What are the different “levels” of interoperability relevant to the Web of Things and what are their pros and cons for the community?*

¹https://en.wikipedia.org/wiki/Machine_to_machine

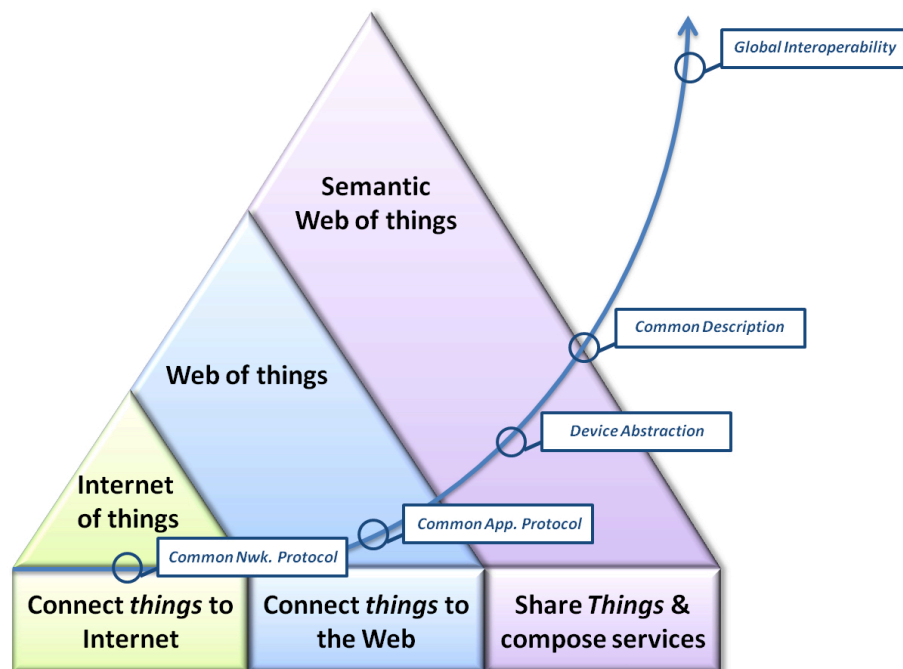


Figure 2.1: Evolution of the IoT market [10]

- *Is there value in exploring an interoperability approach toward standardisation and if so what interest, and appetite exists in the community?*

According to them, nowadays there is a growing trend of aggregating the web presence of Internet of Things components in the web instead of providing web servers to each particular thing separately. The authors call this these aggregation units “Web of Things hubs”. The authors divide WoT hubs into the categories which include Web-enabled IoT products, Web-centric IoT development platforms and Sensor Webs. They say, that different Mashup Platforms and Tools may be used in order to incorporate hubs for the purpose of the creation of various applications. Event though these mashup tools can simplify the application implementation process, they only supply a limited degree of interoperability across the Internet of Things components. Improved interoperability might cause the rise of usage of the things with a smart behaviour, profiting a IoT solution provider, a hub developer and an end user. Blackstock and Lea note, that in order to do so, since the Web of Things is still in its early stage of development, there is a need of the collaboration between different parties including the acceptance of common standards, models and practices.

The authors offer to establish four steps toward the Web of Things hub interoperability. The first one they call *Web of Things Core*. As shown in Figure 2.2, at this level minimal interoperability is present allowing the application developers to interact with hubs via mashup tools using RESTful APIs. Though, either application or mashup platforms developers are supposed to implement some hub-specific adapters in order to allow the proper interaction.

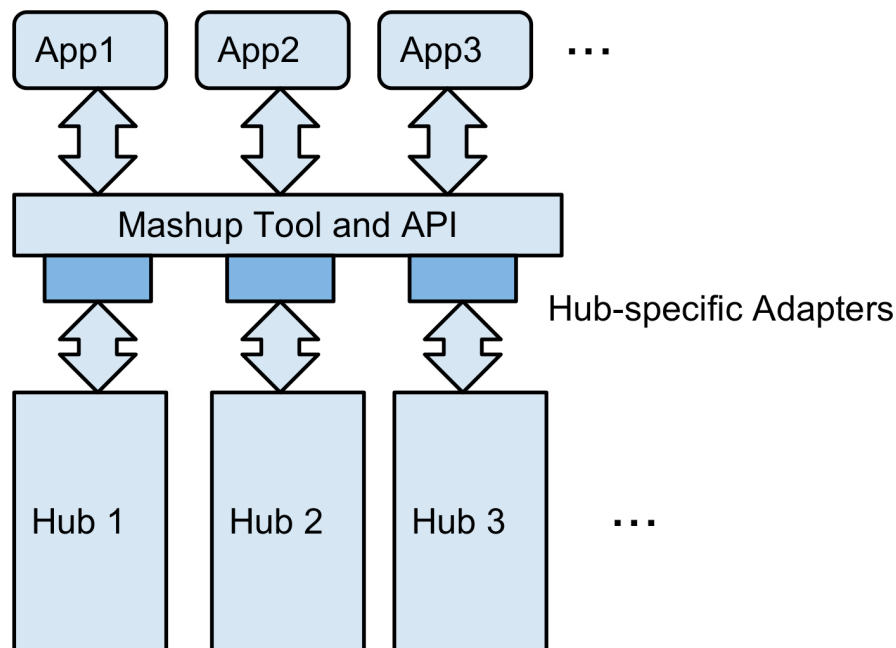


Figure 2.2: Mashup platforms used to address WoT hub interoperability [2]

The second stage is called *Web of Things Model*. At this level authors desire to reach the consensus on a general group of models and mechanisms to regulate what components and data will be processed to a hub. In the next *Web of Things Hub* level the authors require to agree on some more technical details of the implementation, such as URLs and standard schemes of the data transporting. Security measures are also meant to be decided at the same stage. Also this level allows more automatic interoperability. The last level is called *Web of Things Profiles*, where Internet of Things parties are suggested to reach the consensus on thing profiles. It means, that for example, a humidity sensor in one hub should be consistent with a humidity sensor from another hub. The data model is also considered to be

unified for devices of the same profiles. Once the Web of Things community is compliant with this interoperability level, authors foresee the possibility of the interaction between applications and hubs as shown in Figure 2.3. Though this research neither provides any proof of the concept nor highlights the problem of the interoperable authoring of rules, the concept itself can be considered as promising and some ideas of it might be kept for our project.

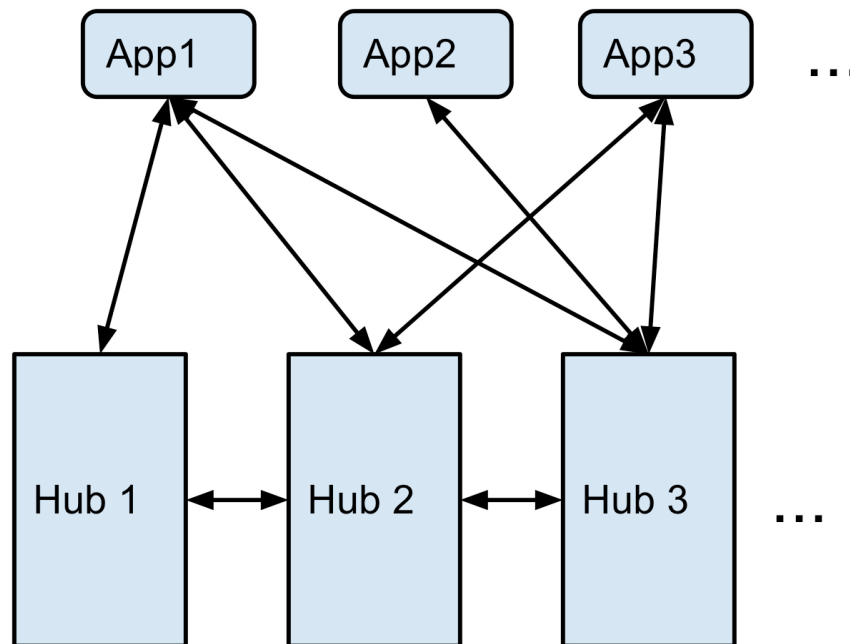


Figure 2.3: Desired WoT interoperability [2]

Mineraud and Tarkoma [13] suggest that IoT parties should adopt their generic IoT architecture in order to be compliant with the following ground rules:

- interoperability with other systems;
- ownership of data and device functionalities;
- scoping for managing data and functionality visibility;
- loose coupling and late binding.

The authors introduce an extension called “IoT hub” to be adopted by current IoT platforms. This approach is made in order to allow platforms

being compliant with the architecture shown in Figure 2.4. A structure of the software consists of a common bundle of protocols to be adjusted to a specific platform or a device. The hub is based on a RESTful API. It is supposed to deal with connections with the web for an IoT platform sharing its data.

Apparently, Mineraud and Tarkoma provide some general guidelines on how to build Web of Things adapters for IoT platforms. It might be quite beneficial from the scope of our research, since we are interested in bringing IoT platforms to the Web of Things level if they are not there yet. However, their recommendations are quite broad. It is clear that the proposed solution does not support managing any rule-based interaction. Thus, this project cannot be claimed as a fully interoperable middleware that supplies issues of the current state of art with the solution.

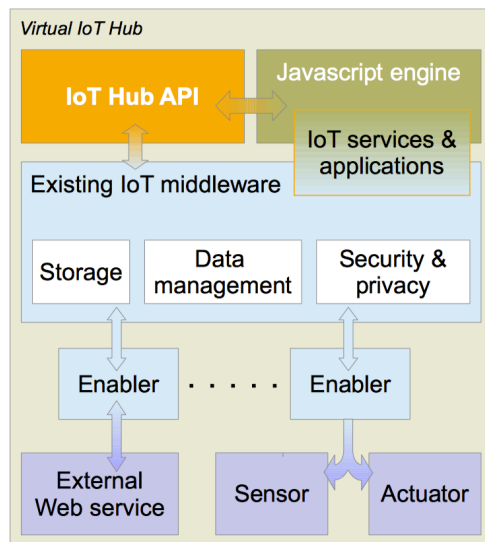


Figure 2.4: IoT hub architecture [13]

Doukas and Antonelli [4] make an investigation on the paper of Hong and Landay [9] that concludes challenges of developing the context-aware applications, including a need to build and deploy some back-end infrastructure in order to encounter data management and exchange. Furthermore, they consider a lack of unifying interfaces to interconnect smart things from

the IoT domain to be an issue. In order to address these problems, the authors decided to make a use of the COMPOSE (Collaborative Open Market to Place Objects at your Service)² framework. The underlying idea of this cloud-based framework is to provide an open source infrastructure together with a bundle of tools in order to develop intelligent applications which are able to perform the exchange of information across different smart objects. Communication is supposed to be settled using a RESTful API.

In the scope of their research, authors provide a mobile-based application which is context-aware. This application consists of four update services: Location, Weather, Social and Activity. It composes measurements from different sensors related to these services in a dashboard-like menu of a smartphone. Its goal is mainly to notify user about different events. Though the idea approaches towards an interoperable solution, the application does not provide any control over neither the definition of dependencies between events nor the creation of rules. Thus, it cannot be considered as a fully functional interoperable middleware.

The importance of the problem that is stated in Section 1.1 is also supported by the existence of the IoT-EPI³, a European Initiative for Internet of Things platform development, which was created to increase the opportunities for platform development, interoperability and information sharing.

The project called INTER-IoT⁴ is one of the projects from this European Platforms Initiative. Its goal is to make a full implementation cycle for a framework to enable the interoperability for various Internet of Things platforms. Their approach is to build blocks in different application domains in order to present a framework, methodology, APIs and tools allowing interoperability as illustrated in Figure 2.5. Though all advantages of the INTER-IoT project, the research is still ongoing. So far, there is neither a proof of concept nor any demo available.

Looking further for projects related to smart interactios, we got an inspiration partially from the research of Beneventi et al. [1] from the University of Parma. They provided an idea on how to make intelligent applications combining a FIPA compliant⁵ agent-based system, scripting and a rule engines. This way they wanted to achieve a higher level of adaptivity and complexity in different applications.

They have chosen Drools⁶ as a rule reasoning mechanism. Drools is an

²<http://www.compose-project.eu>

³<https://iot-epi.eu>

⁴<http://www.inter-iot-project.eu>

⁵<http://www.fipa.org>

⁶<http://www.drools.org>

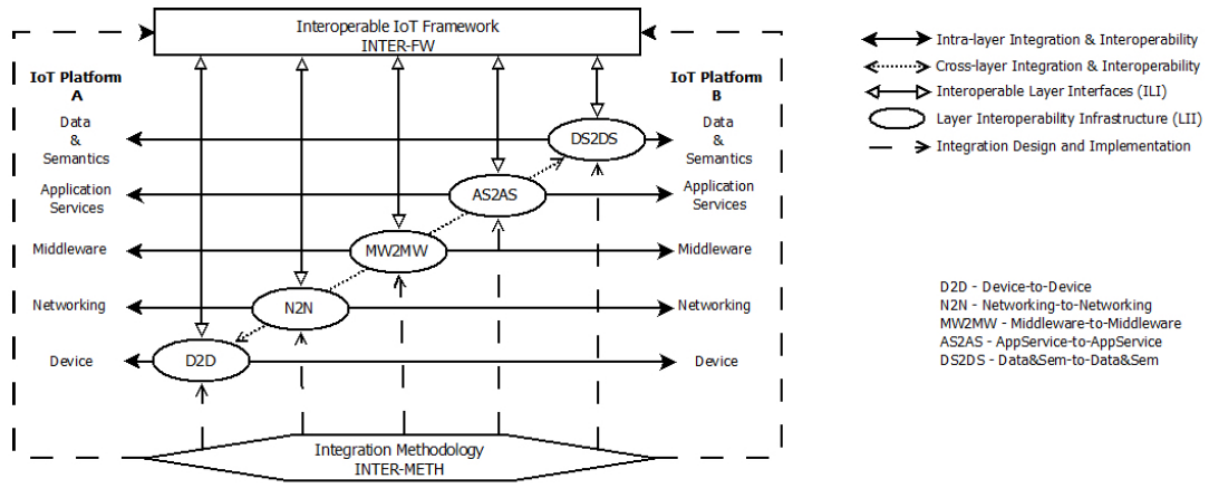


Figure 2.5: INTER-IoT approach

implementation of the Rete algorithm [5] done in object-oriented manner. The JAVA Agent DEvelopment Framework⁷ (JADE) was chosen as an agent-based platform for the project. Since the rules in the system are in the form of scripts, the team used BeanShell⁸ in order to process and execute them. Overall, this research project presents an interesting strategy on building a rule-reasoning system. At the same time, a lack of an interface to connect the presented tool to the web can be considered as a disadvantage.

The Context Modelling Toolkit (CMT) [19] is an approach to go beyond the simple context in “*IF this THEN that*”-like systems. The CMT is claimed to be a software product which allows end users to model the situations (i.e. “*a projector is in use in the meeting room*”) in order to define more advanced context rules in a user-friendly manner. The Context Modelling

⁷<http://jade.tilab.com>

⁸<http://www.beanshell.org>

Toolkit also takes into consideration a user's expertise level providing suitable interface and flexibility. Three levels are available (descending experience level): *programmer*, *expert user* and *end user* as shown in Figure 2.6.

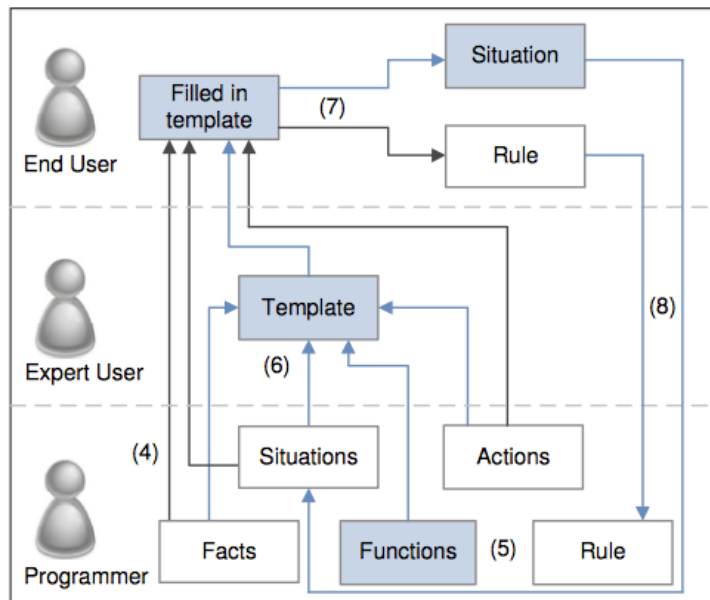


Figure 2.6: Multi-layered CMT approach [19]

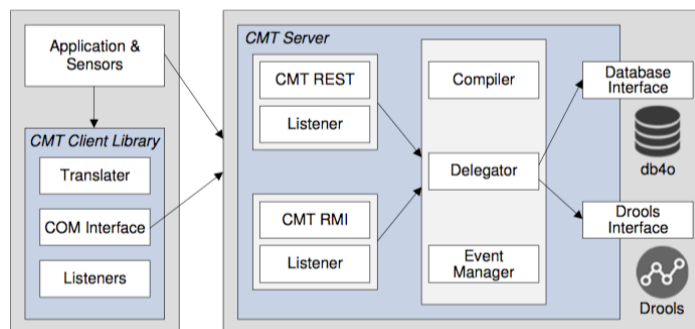


Figure 2.7: CMT architecture [19]

A main benefit of using the Context Modelling Toolkit is a powerful rule reasoning system since the *...implementation of the rule compilation is a*

complex mechanism which takes into account the full first order logic and provides in depth error handling for client applications... [19].

Concerning the architecture, it is as the client-server rule-based framework as shown in Figure 2.7. Nevertheless, the CMT is still under development. Moreover, it does not provide a clear RESTful API for the communication. It still has a need for an additional middleware layer in order to achieve a goal of being the desired solution mentioned in Section 1.1.

Summing up the literature review, the Computer Science community has come to a conclusion, that there is a need in a context-aware interoperable middleware that could supply the IoT world with the cross-platform smart interaction, at the same time being compliant with the current Web of Things standards.

3

Design

In order to tackle the problem described in Section 1.1 and to supply the requirements defined in Section 1.3 as much as possible, we propose a solution in the form of an IoT middleware. We use the iterative and incremental methodology of the software development in order to optimise this process. Being a RESTful server, the IoT middleware is a context-aware unifying tool that improves a seamless experience of the smart interaction across IoT platforms. IoT platforms can connect to the middleware, export their data to it and handle the command requests coming from the middleware. Our solution is fully compliant with the Web of Things standard. Hence, for compatibility, the IoT platforms should only follow the rules of communication and general conceptual agreements defined in Section 3 and Section 4. It means that developers of those platforms will be fully responsible for the support of the corresponding functionality in their systems. Designing the solution we came to the model shown in Figure 3.1.

3.1 Components

Each platform itself is a hub for different objects with a smart behaviour. All those objects we call from now on *devices*. A device can be any separate object which can perform any task and be controlled by the platform. In our middleware we generalise devices to two types: state devices and value

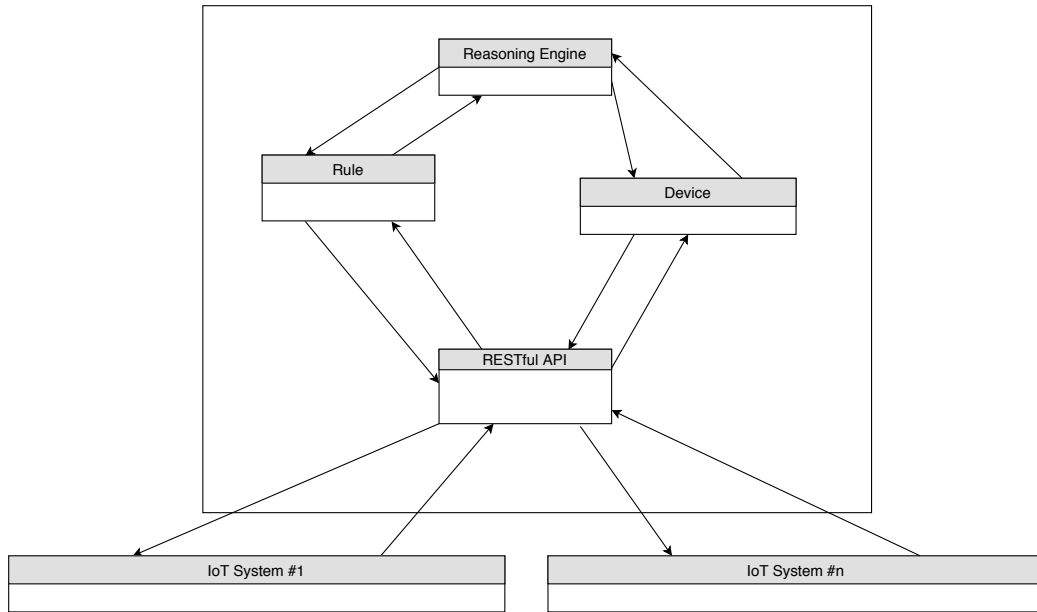


Figure 3.1: Model of the middleware

devices. A state device is a device that for one moment of time can be in no more than one state. A list of acceptable states for this device should be shared by the platform. A value device is a device, which current state can be any alphanumeric sequence.

3.2 Rules

Smart behaviour of devices in IoT platforms is initiated with some rules. Our middleware will support importing rules from platforms in a format similar to *If This Then That*. Semantically, our format can be explained as *IF THIS device is in THIS state/value THEN THAT device has to be set to THAT state/value*. Further, it is possible to define new rules between different platforms using only a user interface discussed in Section 3.3. The proposed system is designed to support the authoring of complex rules making use of the Context Modelling Toolkit [19]. However, during our investigation we realised that a standalone Python solution can be a lot more beneficial in the IoT community, since Python is used widely in the IoT community. Therefore, in our design, we have foreseen the ability to support first order logic. Described in Section 4 prototype of our solution demonstrates the ability to deal with rules with logic operators **and** and **or**, that is the first step towards the reaching an agent-based implementation.

Though, an import of the rules from IoT platforms to the middleware still is a not solved problem itself. How a rule continues its existing inside the platform after sharing it with the middleware? Here we made an analysis of possible solutions.

The first option can be just leaving a rule untouched inside a platform after exporting. Advantages are the ease of implementation and the opportunity to keep local control over the platform behaviour. However, one of the goals of the middleware is to provide a universal authoring tool in one place for all connected platforms. Thus, this solution might create ambiguity in the interaction between users and the middleware. For instance, if a user decides to make a change to the rule via the middleware, only that instance of the rule will be modified. In addition, leaving a rule as it might cause conflicts at execution time since two separate copies of the rule will be stored and triggered from different places simultaneously.

Another option could be to permanently remove a rule from platform after it has been successfully exported to the middleware. As a benefit we can prevent any possible conflicts with the rule execution. The user experience is also improved due to increasing obviousness in the interaction with the middleware. Concerning disadvantages, a rule will be needed to become set up back on the platform after a user decides to disconnect it from the middleware. Furthermore, a user will lose an opportunity to author rules via direct interaction with the platform if for some reason they wish to go with this idea.

Finally, we can mute exported rules in a platform. It will save a positive point of the removing. At the same time, there is no more need to set up the rule after disconnection from the middleware again since it can be just unmuted.

Though, we have foreseen that not all the IoT platforms do support the muting of rule functionality. Therefore, in our opinion, the most ultimate solution would be to mute platform rules whenever it is possible and delete them from platforms which do not offer such an option.

3.3 User Interface

In order to make our solution a convenient tool for the end user, it has to be supplied with a user friendly interface (UI). It is supposed to make interactions of users and the middleware as intuitive as possible.

First of all, the user should be able to get the relevant information about platforms connected to the middleware. Rules, that are already defined, should be also represented in an informative way. But the most important

challenge is to provide a user with the understandable mechanism of authoring new rules. Since the number of devices, that are available for automation might become enormous, the perception of possible rules can easily stop being obvious. In order to address this challenge, we suggest to tag smart devices based on their properties (i.e. device's function, name, states, etc.). Moreover, the idea is to allow a device to be included in different tag groups, so that it can be found in the most natural way. For instance, a TV can be marked by a **Sound** tag as a device that emits sound and **Video** as a device that can be used to watch videos and **Living room** as a device that is located in the corresponding area.

When a user wants to initialise the rule creation process by choosing a trigger device, they will get tag names to choose from. Then after selecting a tag, a user will get a list of the corresponding available devices. Using this strategy in the UI implementation, we can really profit in the optimisation of the user-middleware interaction.

Since the implementation of the UI is kept as one of the future works, along with a prototype of the solution we present a pseudo command-line interface (CLI) for advanced users and developers. An advantage of such interface is the ease of interaction. The proposed pseudo CLI can be reached both locally and remotely, for instance, using SSH¹-based tools in the latter case.

¹<https://www.ietf.org/rfc/rfc4251.txt>

4

Implementation

4.1 Architecture

A prototype of the middleware is implemented as a web server, that can be fired up on any operating system where Python is installed. It is based on the micro-framework Flask¹. Flask is in charge of generating the IP address of the middleware that is used in order to reach the proposed system on the Internet. This host address followed by a corresponding route should be used in order to access different functionalities of our software. A full list of available routes is shown in Listing 4.1. Consequently, the introduced solution can be deployed to be used in a local network and as a remote server.

```
1 <host_address>/new_connection  
  <host_address>/sync_devices  
3 <host_address>/sync_rules  
  <host_address>/update  
5 <host_address>/ask_control  
  <host_address>/ui
```

Listing 4.1: Routes of the middleware

There is a number of tasks that might cause deadlocks in the runtime if they are executed in one thread (i.e. tasks that require a response from a user

¹<http://flask.pocoo.org>

or that might take a considerable amount of time for some calculations. We do not want them to block the whole software activity. Therefore, in order to enable parallel execution, we use an asynchronous task/job queue Celery². Since Celery is based on distributed message passing, we use the most widely deployed open source message broker RabbitMQ³. Though RabbitMQ supports various standards (including MQTT⁴ and STOMP⁵), it was originally developed⁶ to support AMQP [15], an open protocol for transporting messages between components of a system, which is used in our software. The architectural schema is shown in Figure 4.1.

4.2 Communication Convention

Here we define the schema and grammar of the communication between platforms and the middleware. Hence, all the communication is done using JSON of a special format. This convention is our contribution because so far there is not any standard to solve the problem of interoperability stated in Section 1.1. We suggest the following convention.

At the first time connection to the middleware happens, an IoT platform should make an HTTP request to the address `<host_address>/new_connection` and follow the format shown in Listing 4.2. In this request only a name of the platform needs to be mentioned.

```
{
  2  "type": "object",
     "properties": {
     4    "platform_name": {
         "type": "string"
     6    }
     },
     "required": [
     8   "platform_name"
    10 ]
}
```

Listing 4.2: JSON schema for a new connection request

If the mentioned platform wants to export its devices to the middleware, it should follow the format shown in Listing 4.3 while making an HTTP request to `<host_address>/sync_devices`. A platform can have a number

²<http://www.celeryproject.org>

³<http://www.rabbitmq.com>

⁴<http://mqtt.org>

⁵<http://stomp.github.io>

⁶<http://www.rabbitmq.com/protocols.html>

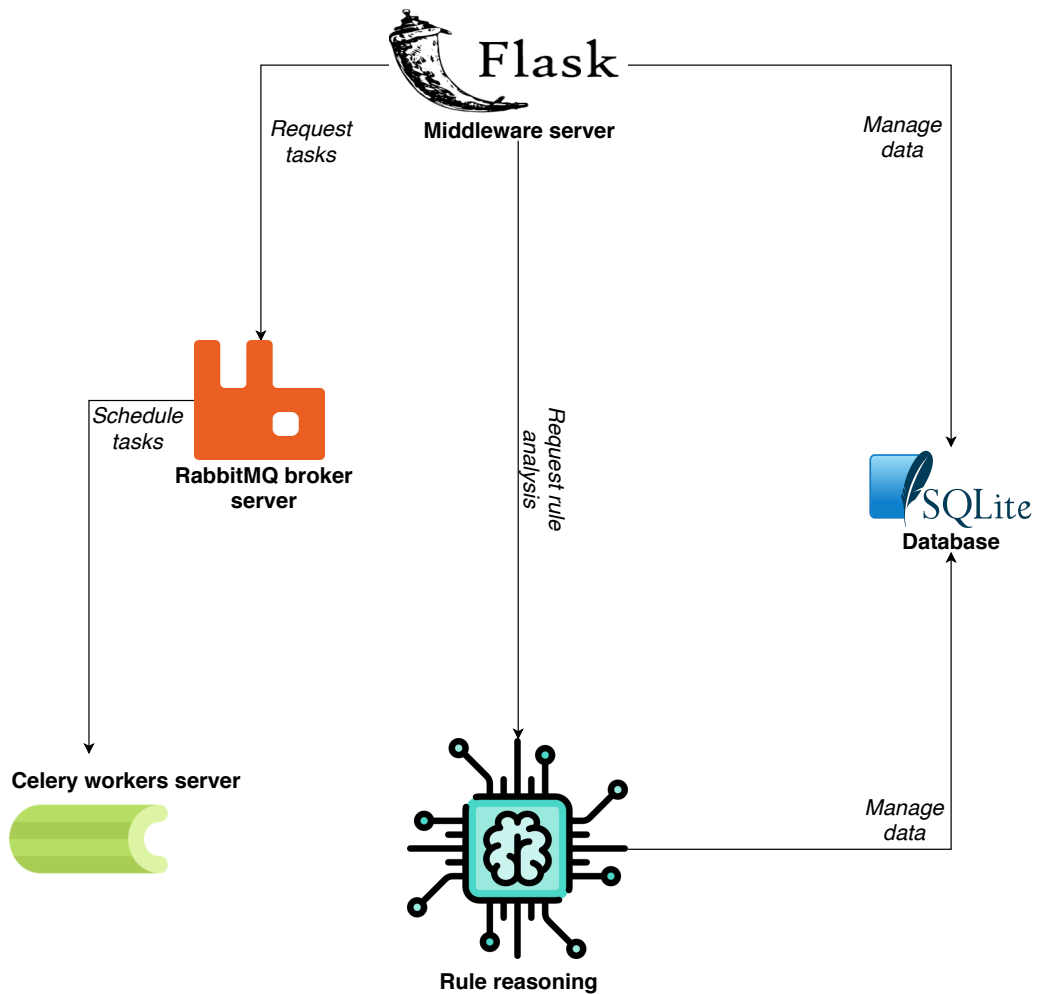


Figure 4.1: Architecture of the middleware

of devices. For each device to be shared with the middleware, the platform should mention its name and its type (state or value, as discussed before). Depending on the device type, the platform should also mention either a list

of discrete states or optionally a range of acceptable numeric values. Finally, the current condition of the component should also be shared.

```
1 {
2   "type": "object",
3   "properties": {
4     "platform_name": {
5       "type": "string"
6     },
7     "devices": {
8       "type": "array",
9       "items": {
10        "type": "object",
11        "properties": {
12          "device_name": {
13            "type": "string"
14          },
15          "device_type": {
16            "type": "string"
17          },
18          "discrete_states": {
19            "type": "array",
20            "items": {
21              "type": "object",
22              "properties": {
23                "state_name": {
24                  "type": "string"
25                }
26              },
27              "required": [
28                "state_name"
29              ]
30            }
31          },
32          "num_values": {
33            "type": "array",
34            "items": {
35              "type": "object",
36              "properties": {
37                "value_min": {
38                  "type": "number"
39                },
40                "value_max": {
41                  "type": "number"
42                }
43              },
44              "required": [
45                "value_min",
46                "value_max"
47              ]
48            }
49          }
50        }
51      }
52    }
53  }
```

```

47         ]
48     },
49     },
50     "currently": {
51         "type": "string"
52     },
53 },
54 "required": [
55     "device_name",
56     "device_type",
57     "currently"
58 ],
59 }
60 },
61 },
62 "required": [
63     "platform_name",
64     "devices"
65 ]
66 }

```

Listing 4.3: JSON format for sharing the components

In order to export existing rules to the middleware, platforms are assumed to make an HTTP request to the address `<host_address>/sync_rules` compliant with the format shown in Listing 4.4. Platforms can share multiple rules with the middleware per request. Internally in the platform, rules can have an identifier, which is encouraged to be mentioned in the request. However, the most important details of a rule are a trigger and an actuator. The trigger is a device, a switch of which to a certain state causes a change for the actuator device's state. Both the trigger and the actuator together with corresponding states or their values should be also mentioned in the rule sharing request.

Upon a change of a component's current measurement, a platform has to notify the middleware making an HTTP request to `<host_address>/update` using a request format shown in Listing 4.5. From this request the middleware needs to get a name of the updated component and a new measurement of it. Multiple updates can be submitted in one request.

The periodically incoming HTTP requests to the address `<host_address>/ask_control` from platforms asking for new commands from the middleware should follow the format shown in Listing 4.6. As a response, platforms might get the update of device's current state (or value) or an order to mute the rule which was imported previously. If the muting option is not supported, platforms should remove the rule marked to be muted. We have foreseen a possibility to use web sockets for achieving the

```
{
  2  "type": "object",
    "properties": {
      4    "platform_name": {
          "type": "string"
        6    },
      "rules": {
          8    "type": "array",
            "items": {
              10    "type": "object",
                "properties": {
                  12    "rule_identifier": {
                      "type": "string"
                    14    },
                    "if_device_name": {
                      16    "type": "string"
                    18    },
                    "if_device_currently_condition": {
                      20    "type": "string"
                    22    },
                    "then_device_name": {
                      24    "type": "string"
                    26    },
                    "then_device_currently_reaction": {
                      28    "type": "string"
                    30    }
                  32    }
                },
              34    "required": [
                  "rule_identifier",
                  36    "if_device_name",
                  "if_device_currently_condition",
                  38    "then_device_name",
                  "then_device_currently_reaction"
                40    ]
            }
          42  ]
        }
      }
    }
  }
}
```

Listing 4.4: JSON format for sharing the rules

```
{
  2  "type": "object",
    "properties": {
      4    "platform_name": {
          "type": "string"
        6    },
      "updates": {
        8    "type": "array",
          "items": {
            10    "type": "object",
              "properties": {
                12    "device_name": {
                    "type": "string"
                  14    },
                  "new_currently": {
                    16    "type": "string"
                  },
                  "timestamp": {
                    18    "type": "string"
                  }
                20    }
              },
            22    "required": [
                "device_name",
                24    "new_currently",
              ]
            26    }
          },
        28    "required": [
          "platform_name",
          30    "updates"
        ]
      32    }
    }
```

Listing 4.5: JSON format for making updates

```
1 {  
2   "type": "object",  
3   "properties": {  
4     "platform_name": {  
5       "type": "string"  
6     }  
7   },  
8   "required": [  
9     "platform_name"  
10  ]  
11 }
```

Listing 4.6: JSON format for making an update request

same functionality.

4.3 Data Management

4.3.1 Database Description

In order to supply the middleware with a sufficient data storage option, we defined some requirements for the data management system. First of all, a database should be durable and have a reasonable performance. It has to be able to store structured data efficiently. And the last but not the least: a database should provide an enforcement of declarative constraints. Thus, since the data for our solution has some relations, we have given a preference to a database based on the relational data model [3] over the non-relational one (also known as NoSQL⁷). The SQLite⁸ is chosen as a database engine for the proof of concept since it is a full-featured but in the same time self-contained and serverless database-management system.

In our database tables `Platforms`, `Devices`, `States`, `Rules` and `Commands` are presented. Each table contains an integer primary key (the name varies depending on a table). In addition, tables consist of several other fields and constraints. A schematic illustration of the database is shown in Figure 4.2.

4.3.2 Database Schema

Table `Platforms` has a mandatory unique text field `platform_name`. The field name is self-explanatory.

⁷<http://nosql-database.org>

⁸<https://sqlite.org>

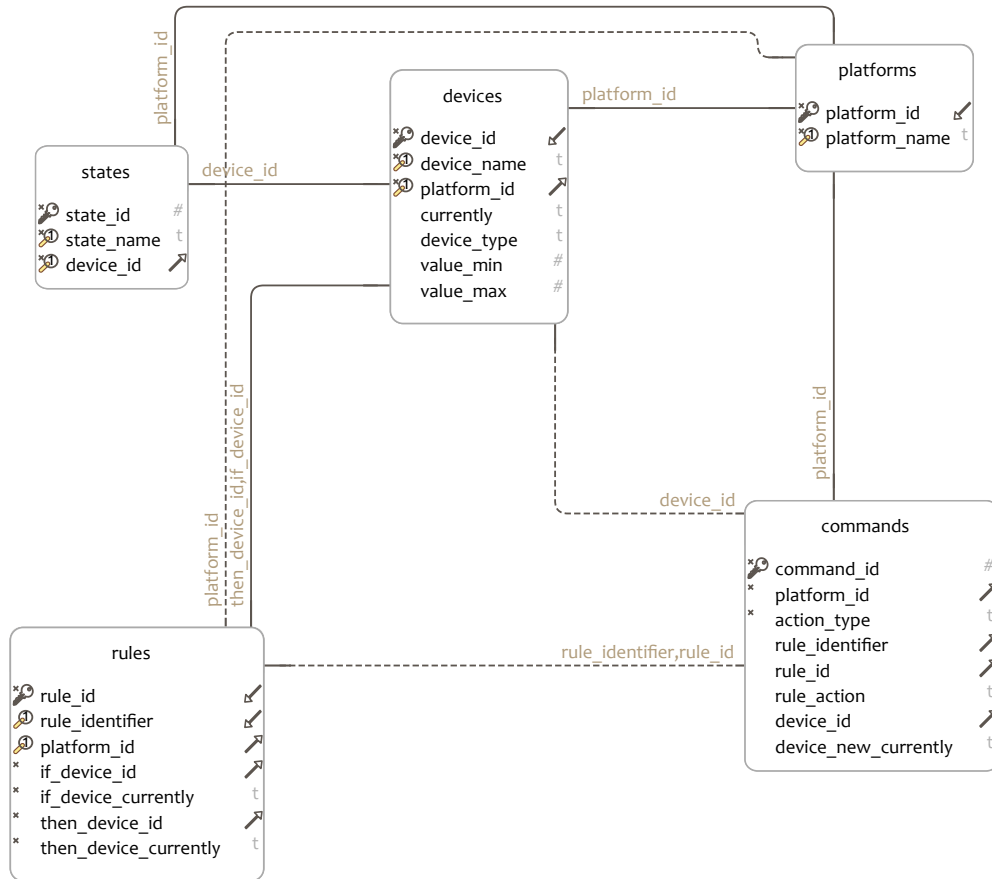


Figure 4.2: Database schema

Table **Devices** contains a mandatory text field `device_name`; an integer field `platform_id`, which is a foreign key for the table **Platforms**; a text field `currently`, which stays for the current state or value of device; a text field `device_type` which, as mentioned before, should be either “STATE” or “VALUE”; and, finally, numeric fields `value_min` and `value_max` which determine minimal and maximal value limits for devices the “VALUE” type.

Table **States** has a mandatory text field `state_name`; a mandatory inte-

ger field `device_id`, which is a foreign key for the table `Devices`. Additionally, a constraint on the uniqueness for the threes `state_name` and `device_id` insures correctness of the input data.

Table `Rules` consists of a text field `rule_identificator` and an integer one `platform_id`, a foreign key for the table `Platforms`, to be filled in case this is an imported existing rule from the platform; `if_device_id`, `then_device_id` are mandatory integer fields and foreign keys for the table `Devices`; `if_device_currently_id`, `then_device_currently_id` are mandatory integer fields and foreign keys for the table `States`. Furthermore, a constraint on the uniqueness for the pairs `rule_identificator` and `platform_id` insures correctness of the input data.

Table `Commands` contains a mandatory integer field `platform_id` —a foreign key for the table `Platforms`; a mandatory text field `action_type` which is supposed to be either “rule” or “device”; foreign keys `rule_id` and `rule_identificator` for the table `Rules` of corresponding data types; foreign key `device_id` for the table `Devices` of an integer type and text field `device_new_currently`.

4.4 Interaction

In this section we provide details on different aspects of the interaction with the proposed system. After deploying the middleware, the initial stage is the connection of IoT platforms with it. Supported platforms are required to consume the IP address of the middleware and perform the following connection steps automatically. User can simply feed the corresponding IP address of the middleware to platforms via their own user interfaces.

After the initial stage described in the previous paragraph, the pseudo CLI of our solution prototype provides all the important information about the current status of the system. Connected IoT platforms, available devices and defined rules are displayed in the pseudo CLI. An example of the pseudo CLI is shown in Listing 4.7.

In the presented pseudo CLI lines with `#` in the beginning are ignored by the system. Every other entry is considered as a command. Our system understands symbol `?` as a separator for commands. By default, we assume using the separator after each logical part of a command. A user can remove rules, handing a command starting with `-` and followed by `rule_id` to the middleware. In order to create a rule, a command should begin with `+` and then contain desired conditions and consequences of the rule execution. If for some reason there is a need to remove the platform and devices related to it, a user can hand a removing command as well. Concrete examples are

```

1 # PLATFORMS:
2 #####
3 # platform_id: 121
4 # name: garden_platform
5 #####
6 # platform_id: 5
7 # name: home_assistant_platform
8 #####
9
10
11 # DEVICES:
12 #####
13 # lamp_01 from home_assistant_platform
14 # with possible states: on off
15 #####
16 # humidity_sensor from home_assistant_platform
17 #####
18 # watering from garden_platform
19 # with possible states: off low high
20 #####
21 #####
22
23
24
25 # RULES:
26 #####
27 # rule_id: 21
28 # trigger: humidity_sensor,home_assistant_platform -
29 #           drought_alert
30 # actuator: watering,garden_platform - high
31 #####

```

Listing 4.7: Pseudo command-line interface of the middleware prototype

shown in Section 4.5.

4.5 Use Cases

In this section we describe some examples of application of the introduced solution.

In the first scenario the proposed solution is deployed in the smart home environment. A wise home owner, who wants to decrease the energy expenses, wants now to use their Nest thermostat⁹ in his private house, because they want to get a remote control over the temperature in their accommodation. In addition, the home owner wants to automate a behaviour of the heating system, so that if the temperature in the bedroom goes below a certain degree the thermostat can activate a warming mode. It is quite a trivial problem in case there is an available Nest Temperature Sensor¹⁰ to be paired with the Nest thermostat. At the same time, a Nest thermostat does not support any third-party temperature sensor. Unfortunately for the home owner, they have only a Xiaomi Aqara Temperature Humidity Sensor¹¹ and do not want to purchase any additional hardware. Once Nest and Xiaomi adopt their platforms to be compliant with our middleware, this home owner can make their wish come true. First they need to run the middleware. Then its IP address has to be provided to Nest and Xiaomi IoT platforms. After this initial step platforms start communicating with the middleware. Possible steps of the interaction are described below:

- Nest and Xiaomi IoT platforms make the first connection request.
- The information about components of these IoT platforms is shared with the middleware.
- Since none of IoT platforms have any rules previously defined by the home owner, the step of sharing these rules with the middleware is skipped.
- Using the middleware UI (pseudo CLI in case of the prototype), the home owner now can now get an overview of their smart environment as shown in Listing 4.8.

⁹<https://nest.com/thermostats/nest-thermostat-e>

¹⁰<https://nest.com/thermostats/nest-temperature-sensor>

¹¹<https://item.mi.com/1164900031.html>

```

# PLATFORMS:
2 #####
# platform_id: 1 name: nest_platform
4 #####
# platform_id: 2 name: xiaomi_platform
6 #####
8
10 # DEVICES:
#####
12 # thermostat from nest_platform with possible states: off heat
    cool
#####
14 # temperature_sensor from xiaomi_platform
#####
16
18
# RULES:
20 #####

```

Listing 4.8: Smart environment overview

```

+?xiaomi_platform?temperature_sensor?16?nest_platform?thermostat
?heat

```

Listing 4.9: Example of a rule definition

- In order to reach the desired automation, the home owner creates a rule for turning on the heating once it gets quite cold inside. The rule example is shown in Listing 4.9.
- After the rule is created, the pseudo CLI section of rules updates correspondingly as shown in Listing 4.10.
- Now every time the temperature decreases till 16 degree the thermostat switches to a heating mode, the desired interoperability is achieved.

In the next scenario the proposed IoT middleware can be used to improve a user experience of watching TV. A user wants to adjust the colour tone of their Samsung Smart TV¹² based on the room luminosity measured by Hue lamp from the Phillips IoT platform¹³.

¹²<https://www.samsung.com/be/tvs>

¹³<https://www2.meethue.com>

```
1 # RULES:
  #####
3 # rule_id: 1
  # trigger: temperature_sensor ,xiaomi_platform - 16
5 # actuator: watering ,garden_platform - heat
  #####
```

Listing 4.10: Updated rule section in the pseudo CLI

- At first, the initial connection step should be done.
- After platforms share the required information with the middleware, the pseudo CLI of if the middleware will look as shown in Listing 4.11.
- Then the rules of adjusting the colour tone of the smart tv should be defined as shown in Listing 4.12.
- Once the settings process is completed, the middleware is ready to change colour tone of the smart TV based on the room luminosity level.

```

# PLATFORMS:
#####
# platform_id: 1 name: samsung_tv_platform
#####
# platform_id: 2 name: phillips_platform
#####

# DEVICES:
#####
# smart_tv from samsung_tv_platform with states warm1 warm2
    standard cool1
    cool2
#####
# hue_lumin_sensor from phillips_platform with states low mid
    high
#####
# hue_lamp from phillips_platform with states on off
#####

# RULES:
#####

```

Listing 4.11: Smart environment overview

```

1 +?phillips_platform?hue_lumin_sensor?low?samsung_tv_platform?
    smart_tv?warm1
  +?phillips_platform?hue_lumin_sensor?mid?samsung_tv_platform?
    smart_tv?warm2
3 +?phillips_platform?hue_lumin_sensor?high?samsung_tv_platform?
    smart_tv?standard

```

Listing 4.12: Example of a rule definition

5

Future Work and Conclusions

5.1 Future Work

Presented middleware can still be improved. The second iteration of the development might be focused on the amelioration of the rule reasoning engine of the software. Even though the Context Modelling Toolkit is a cross-platform software, it has serious limitations. Since it is being implemented in Java, it requires a virtual machine to run on top of the system in order to compile fact classes. This issue can be overcome by achieving an agent-based implementation in Python. The main benefit of this goal is an opportunity to make evaluations on the fly due to the interpretable nature of Python. Another benefit of using Python is a high popularity of this language in the IoT community that potentially opens doors in the research — anyone can play around a Python implementation easily. At the same time, an agent-based system requires a full support of the first order logic. Achieving the operability for the first order logic might be the first priority task for the future. Achieving the UI described in Section 3.3 can be the next development iteration, so that both the developer interaction mode and the end user interaction mode would be available. Finally, once the proposed IoT middleware becomes a completely operable agent-based system implemented in Python with the advanced user interface, a comprehensive user study can be performed.

5.2 Conclusions

In this thesis we explored topics of the context-aware interoperability in smart environments. The lack of interaction across IoT platforms is one of the main questions in this domain. Furthermore, current abilities of IoT platforms in operating the rule engines is thoroughly not mature enough for complex context-aware automation.

Addressing these challenges, we have introduced an architecture of the possible software solution in the form of an IoT middleware, supplying it with the prototype. The interaction with the prototype can be performed using the introduced file-based pseudo command-line interface. Along with the middleware, we have defined a grammar for cross-platform communication, based on the Web of Things standard. In addition, we have discussed the issue of operating existing rules in IoT platforms while transferring them to a middleware and come up with some suggestions.

The area of application for our solution is quite wide. It can be used in smart homes, enterprises or other automated environments. Developers of different IoT platforms can easily make use of benefits of the context-aware smart interactions making their products compliant with the provided unifying standards. Finally, vendors who make their IoT solutions compliant with our universal middleware can potentially boost their sales which will consequently lead to an increase in their share price at the IoT market.



Appendix

```
1 class Connection:
2     """
3     Connection with the database
4     """
5
6     def __init__(self):
7         """
8         Open database connection
9         """
10        self.dbc = sqlite3.connect("database.db")
11        self.dbc.row_factory = sqlite3.Row
12        self.c = self.dbc.cursor()
13
14    def do(self, sql, *args):
15        """
16        Execute SQL query
17        """
18        try:
19            result = self.c.execute(sql, tuple(args)).fetchall()
20            self.dbc.commit()
21            logging.debug('Transaction successful!\nQuery: {0}'
22                          '\nvalues: {1}'.format(sql, args))
23            return [dict(row) for row in result]
24        except Exception:
25            logging.debug('Transaction FAILED!\nSQL: {0}'
```

```

27         '\nvalues: {1}'.format(sql, args))
        self.dbc.rollback()

29     def close(self):
        """
31         Close database connection
        """
33         self.dbc.close()

```

Listing A.1: **Connection** class made for a database connection

```

1 DROP TABLE platforms;
  CREATE TABLE platforms
3 (
4     platform_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
5     platform_name TEXT NOT NULL
6 );
7 CREATE UNIQUE INDEX states_platform_id_uindex ON platforms (
8     platform_id);
9 CREATE UNIQUE INDEX states_platform_name_uindex ON platforms (
10    platform_name);
11 DROP TABLE devices;
12 CREATE TABLE devices
13 (
14     device_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
15     device_name TEXT NOT NULL,
16     platform_id INTEGER NOT NULL,
17     currently TEXT,
18     device_type TEXT,
19     value_min REAL,
20     value_max REAL,
21     FOREIGN KEY (platform_id) REFERENCES platforms(platform_id),
22     CONSTRAINT device_name_platform_id_unique UNIQUE (
23     device_name, platform_id)
24 );
25 CREATE UNIQUE INDEX devices_device_id_uindex ON devices (
26     device_id);
27 DROP TABLE states;
28 CREATE TABLE states
29 (
30     state_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
31     state_name TEXT NOT NULL,
32     device_id INTEGER NOT NULL,
33     FOREIGN KEY (device_id) REFERENCES devices(device_id),
34     CONSTRAINT state_name_device_name_platform_id_unique UNIQUE

```

```
(state_name, device_id)
35 );
CREATE UNIQUE INDEX states_state_id_uindex ON states (state_id);
37
39 DROP TABLE rules;
CREATE TABLE rules
41 (
    rule_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
43    rule_identifier TEXT,
    platform_id INTEGER,
45    if_device_id INTEGER NOT NULL,
    if_device_currently_id INTEGER NOT NULL,
47    if_device_currently TEXT NOT NULL,
    then_device_id INTEGER NOT NULL,
49    then_device_currently_id INTEGER NOT NULL,
    then_device_currently TEXT NOT NULL,
51    FOREIGN KEY (platform_id) REFERENCES platforms(platform_id),
    FOREIGN KEY (if_device_id) REFERENCES devices(device_id),
53    FOREIGN KEY (if_device_currently_id) REFERENCES states(
state_id),
    FOREIGN KEY (then_device_id) REFERENCES devices(device_id),
55    FOREIGN KEY (then_device_currently_id) REFERENCES states(
state_id),
    CONSTRAINT rule_identifier_platform_id_unique UNIQUE (
rule_identifier, platform_id)
57 );
CREATE UNIQUE INDEX rules_rule_id_uindex ON rules (rule_id);
59
61 DROP TABLE commands;
CREATE TABLE commands
63 (
    command_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
65    platform_id INTEGER NOT NULL,
    action_type TEXT NOT NULL,
67    rule_identifier TEXT,
    rule_id INTEGER,
69    rule_action TEXT,
    device_id INTEGER,
71    device_new_currently TEXT,
    FOREIGN KEY (platform_id) REFERENCES platforms(platform_id),
73    FOREIGN KEY (device_id) REFERENCES devices(device_id),
    FOREIGN KEY (rule_id) REFERENCES rules(rule_id),
75    FOREIGN KEY (rule_identifier) REFERENCES rules(
rule_identifier)
);
77 CREATE UNIQUE INDEX commands_command_id_uindex ON commands (
command_id);
```

Listing A.2: DDL for the prototype database

```

1 @app.route('/update', methods=['GET', 'POST'])
  def update():
3     if request.get_json():
        dbc = Connection()
5         try:
            platform_id = Queries.get_id(
7                dbc=dbc,
                table='platforms',
9                platform_name=request.get_json()['platform_name']
                ) [0][
                    'platform_id']
11            logging.debug('PLATFORM_ID: {}'.format(platform_id))
            loop_num = 1
13            for update in request.get_json()['updates']:
                device_id = Queries.get_id(
15                    dbc=dbc,
                    table='devices',
17                    platform_id=platform_id,
                    device_name=update['device_name']) [0][ '
                device_id' ]
19                logging.debug('DEVICE_ID: {}'.format(device_id))
                logging.debug('LOOP #{}'.format(loop_num))
21                Queries.change_smth(
                    dbc=dbc,
23                    table='devices',
                    target_id=device_id,
25                    currently=update['new_currently'])
                check_rules.delay(
27                    device_id=device_id, currently=update['
                    new_currently'])
                logging.debug('END LOOP #{}'.format(loop_num))
29                loop_num += 1
                dbc.close()
31                update_ui.delay()
                return 'Updates are done'
33            except:
                logging.debug('Oops!')
35                dbc.close()
                return 'Oops!'

```

Listing A.3: Method for handling an update request

```

1 @app.route('/sync_devices', methods=['GET', 'POST'])
2 def sync_devices():
    """

```



```

states}

48         ask_user = {
50             "1_device_id":
52             fresh_device['device_id'],
54             "1_device_name":
56             fresh_device['device_name'],
58             "1_device_states":
60             state_dict,
62             "1_platform_name":
64             request.get_json()['
66             platform_name'],
68             "1_platform_id":
70             platform_id,
72             "2_device_id":
74             suggested_device_id,
76             "2_device_name":
78             old_device['device_name'],
80             "2_device_states":
82             state_dict,
84             "2_platform_id":
86             old_platform['platform_id'],
88             "2_platform_name":
            old_platform['platform_name'],
            "rule_status":
            "pending"
        }

        filename = 'smart_rules/suggestion_
        {}.json'.format(
            str(datetime.now().timestamp()).
        replace(
            '.', ', '))
        with open(filename, 'w') as
        to_approve:
            json.dump(ask_user, to_approve)
            approve_rules.delay(filename)

            if device['device_type'].lower() == 'value':
                try:
                    value_range = (
                        float(device['num_values']['
        value_min']),
                        float(device['num_values']['
        value_max']) + 1)
                except:
                    value_range = (None, None)
            Queries.add_device(
                dbc=dbc,
                platform_id=platform_id,

```

```

    device_name=device['device_name'].lower
    ),
    device_type='value',
    currently=device['currently'],
    value_min=value_range[0],
    value_max=value_range[1])
    dbc.close()
    update_ui.delay()
    logging.info('Devices have been imported to the
middleware')
    return 'Devices have been imported to the middleware
.'
except:
    logging.debug('Oops!')
    dbc.close()
    return 'Oops!'
```

Listing A.4: Method for handling a request of sharing the components

Bibliography

- [1] Alessandro Beneventi, Agostino Poggi, Michele Tomaiuolo, and Paola Turci. Integrating Rule and Agent-based Programming to Realize Complex Systems. *WSEAS Trans. on Information Science and Applications*, 2004.
- [2] Michael Blackstock and Rodger Lea. Toward Interoperability in a Web of Things. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, pages 1565–1574, Zurich, Switzerland, 2013. ACM.
- [3] Edgar Frank Codd. A Relational Model of Data for Large Shared Data Banks. 13(6):377–387, Jun 1970.
- [4] Charalampos Doukas and Fabio Antonelli. COMPOSE: Building Smart & Context-aware Mobile Applications Utilizing IoT Technologies. In *Global Information Infrastructure Symposium*, pages 1–6, Trento, Italy, October 2013.
- [5] Charles L. Forgy. Rete: A Fast Algorithm for the Many Pattern/-Many Object Pattern Match Problem. *Artificial Intelligence*, 19(1):17–37, 1982.
- [6] Dominique Guinard and Vlad Trifa. Towards the Web of Things: Web Mashups for Embedded Devices. 2009.
- [7] Dominique Guinard and Vlad Trifa. *Building the Web of Things: With Examples in Node.Js and Raspberry Pi*. Manning Publications Co., Greenwich, USA, 1st edition, 2016.
- [8] Alexander Henka, Lukas Smirek, and Gottfried Zimmermann. Personalizing Smart Environments. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 159–160, Stuttgart, Germany, 2016.

-
- [9] Jason I. Hong and James Landay. An Infrastructure Approach to Context-aware Computing. *Human-Computer Interaction*, 16(2):287–303, December 2001.
- [10] Antonio J. Jara, Alex Olivieri, Yann Bocchi, Markus Jung, Wolfgang Kastner, and Antonio Skarmeta. Semantic Web of Things: an Analysis of the Application Semantics for the IoT Moving Towards the IoT Convergence. 10:244–272, 04 2014.
- [11] Andreas Kamilaris. Enabling Smart Homes Using Web Technologies, 2012.
- [12] Sarah Mennicken, Jo Vermeulen, and Elaine M. Huang. From Today’s Augmented Houses to Tomorrow’s Smart Homes: New Directions for Home Automation Research. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 105–115, Seattle, USA, 2014. ACM.
- [13] Julien Mineraud and Sasu Tarkoma. Toward Interoperability For The Internet of Things With Meta-hubs. *Computing Research Repository*, abs/1511.08063, 2015.
- [14] Iori Mizutani and Jin Mitsugi. A Multicode and Portable RFID Tag Events Emulator for RFID Information System. In *Proceedings of the 6th International Conference on the Internet of Things, IoT 2016*, pages 187–188, Stuttgart, Germany, 2016. ACM.
- [15] John O’Hara. Toward a Commodity Enterprise Middleware. *Queue*, 5(4):48–55, May 2007.
- [16] Edoardo Pignotti, Stanislav Beran, and Peter Edwards. What Does This Device Do? In *Proceedings of the First International Conference on IoT in Urban Space*, pages 56–61, Rome, Italy, 2014.
- [17] Zulqarnain Rashid, Enric Peig, and Rafael Pous. Bringing Online Shopping Experience to Offline Retail Through Augmented Reality and RFID. In *5th International Conference on the Internet of Things*, pages 45–51, October 2015.
- [18] Jürgen Scheible, Arnd Engeln, Michael Burmester, Gottfried Zimmermann, Tobias Keber, Uwe Schulz, Sabine Palm, Markus Funk, and Uwe Schaumann. SMARTKITCHEN Media Enhanced Cooking Environment. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 169–170, Stuttgart, Germany, 2016.

-
- [19] Sandra Trullemans, Lars Van Holsbeeke, and Beat Signer. The Context Modelling Toolkit: A Unified Multi-layered Context Modelling Approach. *Proceedings of the ACM on Human-Computer Interaction*, 1:8:1–8:16, June 2017.
- [20] Ahmed Mohammad Ullah, Md. Rashedul Islam, Sayeda Farzana Aktar, and SK Alamgir Hossain. Remote-touch: Augmented Reality Based Marker Tracking for Smart Home Control. In *15th International Conference on Computer and Information Technology*. IEEE, December 2012.
- [21] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. Practical Trigger-action Programming in the Smart Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 803–812, Toronto, Canada, 2014.
- [22] Jo Vermeulen and Russell Beale. Challenges and Opportunities for Intelligibility and Control in Smart Homes. In *Workshop Smart for Life: Designing Smart Home Technologies that Evolve with Users*, Seoul, Republic of Korea, 2015.
- [23] Alexandra Voit, Dominik Weber, and Stefan Schneegass. Towards Notifications in the Era of the Internet of Things. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 173–174, Stuttgart, Germany, 2016.