RBHAS: A SIMPLE RULE-BASED WEB PLATFORM FOR SMART HOME AUTOMATION

Milan Segedinac[1],
Srđan Milaković[1],
Zora Konjović[2]

[1]Faculty of Technical Sciences,
University of Novi Sad,
Novi Sad, Serbia
[2]School of Engineering,
University Singidunum,
Belgrade, Serbia

Abstract:
This paper presents a web software platform aimed at simple intelligent automation in a smart home. The domain logic relies on rule-based and case-based reasoning. The entire software architecture is a client-server architecture relying upon the MEAN stack.

Keywords:
Home automation, IoT, if-then rule, JColibri, MEAN stack.

1. INTRODUCTION

Internet of Things (IoT) can be defined as a network of physical devices with embedded technology for communication and monitoring or interacting with their internal state or environment [1]. The term Internet of Things was coined by Kevin Ashton in 1999 appearing first in his presentation maid for Procter&Gamble [2].

Dave Evans from Cisco predicted in 2011 that, by the end of 2020, the number of devices connected to the internet will reach 50 billion [3], while Morgan Stanley report from 2013 even exceeds Evans by the estimation of 75 billion [4]. The recent predictions are moderate but still remain within the range of 20 to 30 billion by the end of the year 2020. In addition, Gartner expects that the devices with only hardware communication support, requiring a software enabling connections to other devices, will become more frequent [1].

All what is stated above is a serious motivation for scientific research and development of new IoT applications. The results presented in this paper belong just to the latest of those two.

The rest of this paper is organized in four sections. The first one is about related work presenting existing research and results closely connected to the scope of this paper. The second section presents the architecture (software and functional) of the software platform proposed in this paper. Third section brings a brief description of implementation details. The last section contains concluding considerations.

Correspondence:
Zora Konjović

e-mail:
zkonjovic@singidunum.ac.rs

DOI: 10.15308/Sinteza-2017-464-471

2. RELATED WORK

Smart home installations become of high interest during the last decades due to their capacity to control and monitor operation of everyday processes in the house like lighting, heating, security, etc. [5].

There are numerous publications addressing various issues of smart home installations among which the lack of standards [6, 12] and the application of artificial intelligence (AI) techniques [7-11] are relevant for this paper.

As it is common to all technologies experiencing a rush growth, Internet of Things faces a serious problem of standardization. Even the elementary things like basic communication standards are the problem, not to mention those enabling for the higher levels of interoperability.

One solution to that problem could be the independent internet platform used to bridge the gap between connected devices and internet services by means of rules. One such platform is IFTTT[1]. IFTTT rules are very simple IF-THEN constructs with IF part containing a simple condition and acting as a trigger that activates THEN part containing the corresponding action [13]. For the creation and execution of rules one can use a rule-based expert system [14].

Case-based reasoning (CBR) [15, 16, 17] is an AI technique that uses previous experience (cases similar to the current one) to understand and solve a new problem. CBR, which enables adaptation of the old solution to a new problem, is actually a cycle consisting of the “four R” processes (Retrieve, Reuse, Revise, and Retain).

3. RBHAS ARCHITECTURE

This section presents the system architecture, i.e. software architecture and functional structure.

Software architecture

The global architecture of the system is depicted by the Figure 1.

This is a client-server architecture with a web client and a server side consisting of three servers: Recommendation Server, Rule Engine Server, and Proxy Server through which IoT devices are accessing web services.

Recommendation Server consists of Web application, CBR application, and MongoDB database [25, 26, 28].

1 IFTTT – an independent internet platform (if this then that), https://ifttt.com/

Web application serves communication with Rule Engine Servers, CBR application is aimed at rules recommendations, and the MongoDB stores the rules.

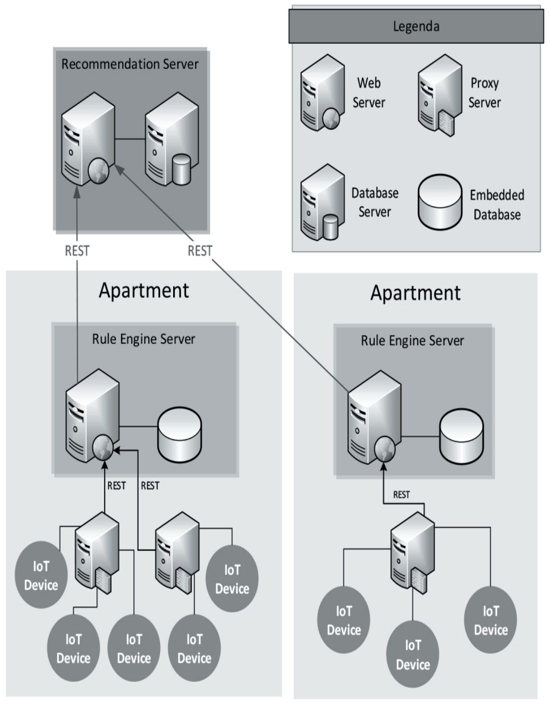


Fig. 1. Architecture of the system

Rule Engine Server is consisted of Web application, a rule- based system, and SQLite database [21]. Here, the Web application supports communication with Recommendation Server and Proxy Servers and, in addition, contains a client application that executes within a web browser. In our application the rule-based system is aimed at managing home devices and SQLite data base stores the rules and the data from IoT devices corresponding to a single home/apartment.

Proxy Server is used to resolve the lack of standards for the communication between IoT devices. Proxy server communicates with IoT devices in a device-specific way, transforms obtained data and sends it to the Rule Engine Server using a negotiated (in advance) protocol.

User interface is a client web application that, as mentioned above, executes within a web browser.

The database model is given on the Figure 2. Green-colored entities describe a rule, while blue-colored ones describe a device together with events and actions.

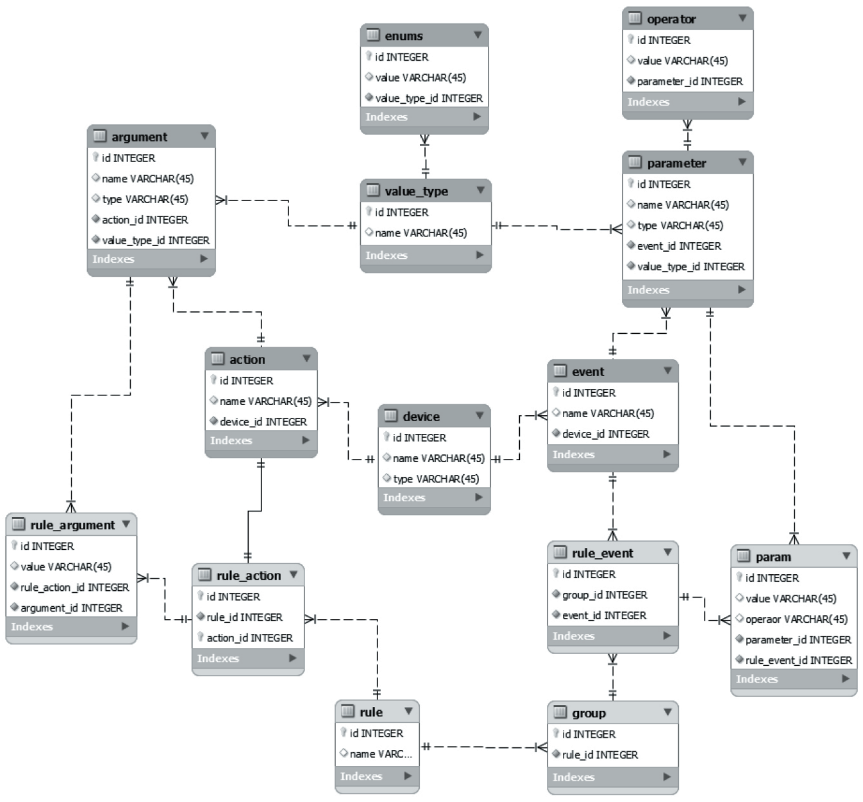


Fig. 2. Database model

Functional structure

Functional structure consists of four groups of functions. The first one comprises functions aimed at devices manipulation (adding new device and displaying existing devices). For adding a new device the REST [24] interface on the Rule Engine Server is used. REST request initiates a Proxy server. Each device is described by its name, type, events and actions that device is capable to execute. Events are described by their names and parameters. Parameter has name, operators, and type. Similar to events, actions are described by their names and arguments. Argument has a name and a type. Listing 1 presents a JSON description of a device, an air-conditioner with two states (transmissions of information about state change and temperature change), and two actions (on/off switching, change of the temperature).

```
{
    name: 'Air Conditioner',
    type: 'air_conditioner',
    events: [{
        name: 'Turned ON/OFF',
        parameters: [{
            name: 'State',
            operators: [{value: '='}],
            valueType: {name: 'enum', enums:
[{value: 'ON'}, {value: 'OFF'}]}
        }]
    }, {
        name: 'Temperature changed',
        parameters: [{
            name: 'Temperature',
            operators: [{value: '='}, {value:
'<'}, {value: '>'}],
            valueType: {name: 'float'}
        }]
    }],
    actions: [{
        name: 'Turn ON/OFF',
        arguments: [{
            name: 'state',
            valueType: {name: 'enum', enums:
[{value: 'ON'}, {value: 'OFF'}]}
        }]
    }, {
        name: 'Change temperature',
        arguments: [{
            name: 'temperature',
            valueType: {name: 'float'}
        }]
    }]
}
```

Listing 1. An example of the JASON description of the device

The second group deals with rules (creating rules, events, and actions; displaying existing rules). Rules are defined by using user interface depicted on Fig. 3. IF part contains events connected by operators AND, OR. THEN part consists of the corresponding set of actions. When defining IF part of the rule, user defines a group firstly. All events from the same group are connected by OR operators and groups are connected by AND operators. User is allowed to edit and delete events and actions. If a user wants to delete a group, she/he only could do it by deleting all events from the group (then the group will be automatically deleted).

The rule shown on Fig. 3 contains two groups and one action. The first group has two events, "*10km Running*" and "*25km Cycling*", while the second one has one event, "*Temperature greater than 30°C*". This rule could be expressed using natural language as:

If (a person) has run for 10 km or cycled for 25km and the temperature is above 30°C, chill the apartment to 26°C.

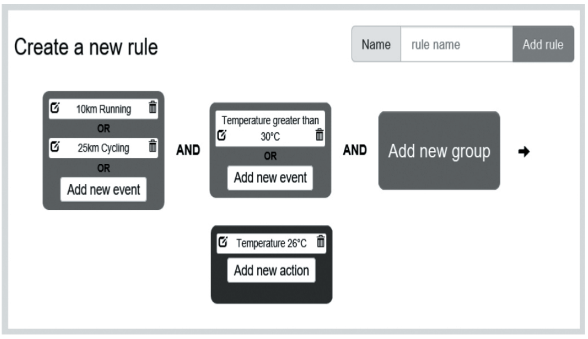
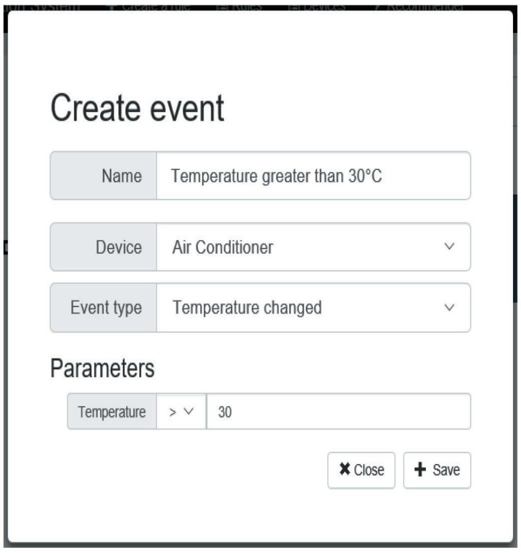


Fig. 3. Defining a new rule

By clicking the button "Add new event" (Fig. 3) within a group, user adds a new event. Afterwards, the modal dialog (Fig. 4) appears requiring the selection of the event name, particular device, event type and values and operator for at least one parameter. The example on Fig. 4 has the event with name "*Temperature greater than 30°C*", selected device "*Air Conditioner*", and event type "*Temperature changed*". This event has only one parameter with a single operator ">", and the value "*30*".



Create event

Name: Temperature greater than 30°C

Device: Air Conditioner

Event type: Temperature changed

Parameters

Temperature > 30

Fig. 4. Modal dialog for adding a new event

Similarly to event defining, user can add a new action. Through the modal dialog the name of action, concrete device, action type, and values and operator for at least one parameter are entered. For example, the action name could be "*Temperature 26*", selected device "*Air Conditioner*", and action type "*Change temperature*". This action has only one argument with value "*26*".

The third group is action execution, which is rather simple. After detecting a change of value on a sensor, the Proxy Server transmits this change to the Rule Engine Server by means of REST. An example of JSON object describing a value change is shown on Listing 2.

```
[{
    "eventId": 2,
    "parameterId": 2,
    "value": "32"
}]
```

Listing 2. A JSON object describing a change of value

The fourth group is aimed at rules recommendation. Rules recommendation is an optional feature, i.e. the system can operate without Recommendation Server. The first thing is selection of events and actions of some registered device that a user wants to use in a new rule. A priority can be assigned to each event and action. Once a selection of events and actions is done, user requests recommendation that causes the Rule Engine Server to send a query to the Recommendation Server, which retrieves rules that match a query. Recommendation Server recommends rules by searching for rules that contain actions and events appearing in a query. The *N-nearest neighbor* algorithm is used for recommendation that operates on a rule base collected from other users. After receiving recommended rules, Rule Engine Server displays a list containing recommended rules. User can select a rule as it is or modify it prior to adding a rule in the rule base.

For example, a user can select two events „TV – Turned ON/OFF“ and „Weather forecast – Weather type“ with priorities 2 and 4 respectively (maximal priority is 5). Additionally, user can select one action „Light bulb – Turn ON/OFF“ with priority 3. As a result three rules are recommended.

The first recommended rule is „Turn off the light if it's sunny“ because this rule contains one event „Weather forecast – Weather type“ and one action „Light bulb – Turn ON/OFF“ from the query. Event has priority 4, and action has priority 3.

The second recommended rule is „Turn off the light if the TV is on“. As the previous one, it contains one event „TV – Turned On/OFF“ and one action „Light bulb – Turn ON/OFF“ from the query, but in this rule the event has priority 2 and the action has priority 3.

Finally, the third recommended rule is „Turn off the light if the room is empty“ because this rule contains only one action, „Light bulb – Turn ON/OFF“, appearing in the query.

4. IMPLEMENTATION DETAILS

So far two servers out of three are implemented, Rule Engine Server and Recommendation Server. This section presents some characteristic details of their implementation.

Rule Engine Server

For the implementation of the Rule Engine Server Node.js [17, 18, and 19], SQLite [21], Sequelize [22], and Nools [29, 30] are used. The application is composed of four modules: Devices Module, Database Module, Rule Module, and Recommendation Module.

Devices Module

This module, which enables (via REST) client application to get a list of all registered devices and to register a new device, has one Express router. This module uses services of the Database Module.

Database Module

Database Module uses object-relational mapping for CRUD operations. For this mapping the framework *Sequelize* is used that requires all entities to be described in JavaScript. An example of the entity Action is shown on Listing 3.

```
sequelize.define('action', {
  name: {
    type: DataTypes.STRING,
    allowNull: false
  }
}, {
  tableName: 'action'
});
```

Listing 3. Entity Action table scheme

Rule Module

Rule Module contains two Express routers, one enabling the Proxy Server to add facts, and another enabling the client application to add/delete/modify rules using an object supporting rules manipulation. Rules are manipulated/defined using Nools, and its DSL that is generated based on corresponding JSON object (Listing 4).

```
{
    name: 'Air Conditioner',
    type: 'air_conditioner',
    events: [{
        name: 'Turned ON/OFF',
        parameters: [{
            name: 'State',
            operators: [{value: '='}],
            valueType: {name: 'enum', enums:
[{value: 'ON'}, {value: 'OFF'}]}
        }]
    }, {
        name: 'Temperature changed',
        parameters: [{
            name: 'Temperature',
            operators: [{value: '='}, {value:
'<'}, {value: '>'}],
            valueType: {name: 'float'}
        }]
    }],
    actions: [{
        name: 'Turn ON/OFF',
        arguments: [{
            name: 'state',
            valueType: {name: 'enum', enums:
[{value: 'ON'}, {value: 'OFF'}]}
        }]
    }, {
        name: 'Change temperature',
        arguments: [{
            name: 'temperature',
            valueType: {name: 'float'}
        }]
    }]
}
```

Listing 4. JSON object describing the rule from Fig. 3

Listing 5 shows the DSL for the rule.

```
rule Rule4 {
    when {
        f1 : Event6;
        f2 : Event6;
        f3 : Event2 (f1.p9 > 10 || f2.p10 >
25) && (f3.p2 > 30);
    }
    then {
        devices.execute([{
                "deviceId": 1,
                "actionId": 2,
                "args": [{"argumentId": "2",
"value": "26"}]
            }
        )
    }
}
```

Listing 5. DSL for the rule from Fig. 3

In addition to rule DSL, Rule Module generates a fact DSL too. An example of event DSL is shown in Listing 6.

```
define Event2 {
    p2 : null,
        constructor : function(p2) {
        this.p2 = p2;
    }
}
```

Listing 6. Event DSL

Recommendation Module

Recommendation Module has one router enabling a client application to request a recommendation from the Recommendation Server.

Client application is implemented using frameworks AngularJS and Bootstrap [20]. AngularUI Router is used for navigation and Angular Resource is used for interaction with REST API. Application consists of several views and modal dialogues each of them having corresponding AngularJS controller.

Recommendation Server

Recommendation Server, which implements CBR approach for recommendations, relies upon jCOLIBRI framework. Because jCOLIBRI supports relational databases through Hibernate ORM, jCOLIBRI was extended in order to support use of MongoDB that is a document-centric database. For that purpose the Java interface Connector is implemented using Hibernate OGM [27].

Interface Connector comprises methods required by jCOLIBRI framework to obtain access to the cases stored as database records, XML documents, or text files. Connector is initialized by means of an XML document that contains path to the Hibernate OGM configuration file, full name of the class containing the query, and full name of the class representing the query result.

The interface is shown on Listing 7.

```
public interface Connector {
    void initFromXMLfile(URL url) throws
InitializingException;
    void close();
    void storeCases(Collection<CBRCase>
cases);
    void deleteCases(Collection<CBRCase>
cases);
    Collection<CBRCase> retrieveAllCases();
    Collection<CBRCase>
retrieveSomeCases(CaseBaseFilter filter);
}
```

Listing 7. Interface Connector

Server application is implemented as a JavaEE application for the WildFly application server [23]. It consists of two modules, module for communication with Rule Engine Server, and module for recommending rules. The last one is implemented using jCOLIBRI framework.

Rules recommendation logic is implemented by the method cycle of the StandardCBRApplication interface. The first step of this method is creation of the configuration object followed by setting up global similarity function and local similarity functions. Local similarity functions measure similarities between events and actions appearing in a query with those appearing in rules from the rule base. Thereafter a method calculating similarity for all rules from database is called. Overall similarity between rules is calculated as a mean value of group similarity and action similarity.

5. CONCLUSIONS

This paper presents the software platform RBHAS aimed at automation of IoT devices management.

The platform RBHAS proposed in this paper has capacity to mitigate the problem of IoT devices protocols heterogeneity by offering development of proxy servers that could be programmed to serve communication between specific IoT devices and platform. Besides, the platform facilitates automation of IoT devices control by applying rules that can be set up by user. RBHAS does not require any local central unit, which can be considered both advantage (no need for additional hardware and/or its configuration) and disadvantage (IoT devices must be able to communicate over Internet).

Lack of authentication is the main drawback of the current version of RBHAS and one of the priorities for further development. In addition, RBHAS could be improved by implementing rules contradiction detection, active control of IoT devices (like temporary blocking, etc.), client applications for mobile platforms (Android and iOS), and rules sharing via social networks. Quality and performance of rules recommendation is also worth to be considered as a subject of further work. Better quality and performance could be achieved through defining an ontology for IoT devices, actions and events, introduction of metadata about controlled/controlling entities (homes, users), rules classification against various criteria (entity/object that rules apply to, the goal achieved by rule application like energy savings, etc.), and support for rules evaluation by users. Proxy Servers could publish their functionalities using RDF (Resource Description Framework) format thus enabling querying via SPARQL Protocol and RDF Query Language (SPARQL) that, in turn, could facilitate introduction of semantic technologies and automated reasoning.

Apart from the above mentioned, more substantial shift that could be consideration-worth is the situational calculus instead of rule-based approach, and classical recommender system with filtering instead of CBR.

REFERENCES

[1] "Gartner says the Internet of Things installed base will grow to 26 billion units by 2020", Gartner, December 2013, [Online]. Available: http://www.gartner.com/newsroom/id/2636073.

[2] K. Ashton, "That 'Internet of Things' thing", RFID Journal, 22 July 2009. [Online]. Available: http://www.rfidjournal.com/articles/view?4986.

[3] D. Evans, "The Internet of Things how the next evolution of the internet is c h a n g i n g everything" [Online]. Available: http://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0 411FINAL.pdf

[4] S. Z. Hosain , "How far is the hype surrounding claims of up to 50B IoT and machine-to-machine devices by 2020 away from reality?", RCR WirelessNews,[Online].Available: http://www.rcrwireless.com/20160628/opinion/reality-check-50b-iot- devices-connected-2020-beyond-hype-reality-tag10

[5] H. Güneş, D. Akdaş," Web based, low cost and modular home automation system", Technical Gazette 23, 2(2016), pp. 533-538

[6] https://en.wikipedia.org/wiki/Home_automation

[7] P. Lynggaard, "Artificial intelligence and Internet of Things in a "smart home" context: A Distributed System Architecture. (1 Ed.)", Department of Electronic Systems, Aalborg University, 2014

[8] A. Zeeshan, A. Mujtaba, "Smart house: artificially intelligent home automation system", VDM Verlag Dr. Müller, 2011

[9] S. Kumar, M.A. Qadeer, "Application of AI in home automation", IACSIT International Journal of Engineering and Technology, Vol. 4, No. 6, December 2012, pp. 803-807

[10] M. B. I. Reaz , "Artificial intelligence techniques for advanced smart home implementation", Acta Technica Corviniensis – Bulletin of Engineering, Tome 6, Fascicule 2, 51-57, 2013

[11] Q. Sun, W. Yu, N. Kochurov, Q. Hao, and F. Hu, "A multi-agent-based intelligent sensor and actuator network design for smart house and home automation", Journal of Sensor and Actuator Networks, 2013, 2, pp 557- 588

[12] A. Wood, "The internet of things is revolutionizing our lives, but standards are a must", The Guardian, 31 Mart 2015. [Online] Available: https://www.theguardian.com/media-network/2015/mar/31/the-internet-of-things-is-revolutionising-our-lives-but-standards-are-a- must.

[13] W. van Melle, E. H. Shortliffe, and B. G. Buchanan, "EMYCIN: a knowledge engineer's tool for constructing rule-based expert systems", (in Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Buchanan and Shortliffe (eds.)), The Addison-Wesley Series in Artificial Intelligence, 1984, pp. 302-313

[14] B. G. Buchanan; R. O. Duda, "Principles of rule-based expert systems", Advances in Computers, Vol. 22, 1983., pp. 163-216

[15] J. L. Kolodner, "An introduction to case-based reasoning", Artificial Intelligence Review, March 1992, Volume 6, Issue 1, pp 3–34

[16] B. Díaz-Agudo, P. A. González-Calero, J. A. Recio-García, A. A. Sánchez-Ruiz-Granados, "Building CBR systems with jCOLIBRI", Science of Computer Programming, Volume 69, Issues 1–3, 1 December 2007, pp 68–75

[17] Node.js, Node.js Foundation, [Online] Available: https://nodejs.org.

[18] M. Cantelon, M. Harter, T. Holowaychuk and N. Rajlich, "Node.js in action", Manning Publications, 2014.

[19] J. R. Wilson, "Node.js the right way: practical, server-side JavaScript that scales", Pragmatic Bookshelf, 2013.

[20] A. Lerner, "ng-book the complete book on AngularJS", Fullstack io, 2013.

[21] "SQLite", SQLite Consortium, [Online]. Available: https://sqlite.org/about.html.

[22] "Sequelize", GitHub, Inc. [Online]. Available: https://github.com/sequelize/sequelize.

[23] M. Ćmil, M. Matłoka and F. Marchioni, "Java EE 7 development with WildFly", Packt Publishing, 2014.

[24] M. Little, G. Roth, S. Vinoski, S. Parastatidis, B. Sletten, J. Webber, I. Robinson and S. Tilkov, "InfoQ Explores: REST", InfoQ, 2010.

[25] E. Redmond and J. R. Wilson, "Seven databases in seven weeks a guide to modern databases and the NoSQL movement", The Pragmatic Programmers, 2012.

[26] R. Suter, "MongoDB - An introduction and performance analysis", Master thesis, HSR Hochschule für Technik Rapperswil, 2012.

[27] E. Bernard and S. Grinovero, "Hibernate OGM Reference guide" [Online].Available: https://docs.jboss.org/hibernate/ogm/4.0/reference/en- US/pdf/hibernate_ogm_reference.pdf.

[28] A. Leonard, "Pro Hibernate and MongoDB", Apress, 2013.

[29] W. Van Woensel, A. H. Newres, P. C. Roy, A. M. Ahmad and S. S. R. Abidi, "A comparison of mobile rule engines for reasoning on semantic web based health data", In 2014 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2014) (D. Slezak, H. S. Nguyen, M. Reformat, E. Santos Jr., eds.), IEEE Computer Society Press, 2014.

[30] https://github.com/C2FO/nools