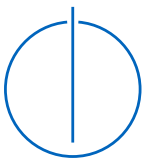
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Ontology Based Practical Rule Engine for Internet of Things

Hakan Uyumaz



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Submission date Hakan Uyumaz

Acknowledgments

Abstract

In the current technology state of the world, internet acts as the central hub for all kinds of communication between different nodes. Regardless of the subject, all types of software use web services that are built with their own well-defined, high-level and abstract architectural design to accomplish this communication with each other. The architecture that will be used to construct the system communication can differ in terms of needs. Plenty of different architectures and protocols exists to build web services. Nevertheless, the choices of protocols would always be the same for the similar work field since constraints that should be satisfied by the web services will be alike. Moreover, the architecture design of these technologies will also be very similar.

The web services in the Internet of Things (IoT) are also defined in a very similar way and mostly based on REST, CoAP or MQTT. The programmers who are building IoT web services are dealing with implementation of the same functionalities with a little difference under this similarity. The differences mainly occur regarding the device choices and how the data obtained from these devices are processed. However, these differences can be parameterized. Therefore, the programmers will only require to configure their system in a high-level perspective. These configurations will consist of the information about the user devices, protocols, how data is stored and how data is processed.

The aim of this thesis is to build a rule engine to easily configure required functionalities of web services in the IoT domain. The rule engine will use rules to process and to analyze any data that is received by the web services. A rule can be applied either on data streams to derive the active state or the data batches to process and to analyze historical data of a device or devices. By achieving this goal, an easy-to-use -with high-level configuration- development environment in the domain of IoT will be introduced. These high-level configurations can either be done in a user-friendly web interface or through an application program interface. Therefore, the web services developer can define functionalities regarding the process and the analysis of the data. Also, the challenging gap between high-level design and actual low-level implementation will be covered with the rules that are configured by the users to match their design.

Contents

1 Introduction

1.1 Background

Internet of Things (IoT) has become an emerging technology with the power of low priced computational units and the cloud technology. There are thousands of different examples in the real world which targets both the end-users and the industry.

The problem with the current state of the art is the requirement of tremendous time investment by the developers. Developers need to spend this time to develop both embedded software and web services to serve their goal. Most of this time is spent on functionalities that are already implemented in different projects with slight differences.

In the IoT practical course offered in the Technical University of Munich, the hardware abstraction layer for single board computers is built. This layer has been extended to a platform with IoT core which covers required web services functionalities to store and access the data of the devices with multi-protocol support. In the following semesters, the development of this platform will be sustained. The goal of this platform is to solve the challenges and problems that developers encounter concerning the need of technical background and time by serving a configurable platform with an easy-to-use user interface. This generic and practical platform can be extended with the easily configurable rule engine to process live data streams or historical data batches of the devices.

In most cases, the applications have similar capability needs to process their data with some little customization. The most basic needs during the data process flow can be listed as;

- Retrieving the data,
- Checking whether a data point in a data stream satisfies a condition,
- Checking whether a data batch in historical data satisfies a condition,
- Forwarding the data to another data process flow,
- Manipulating the data,
- Classifying the data,

- Triggering defined events.

There are requirements and challenges while building a system which can process data. Availability, the need of time, integration of a new process into the system and the vast number of different devices can be considered as some examples these requirements and challenges.

The required functionalities to define data process flows in the area of IoT diverge from each other with only a little customization. Business logic, interoperability of the sensors and actuators, machine to machine (M2M) communication and context-aware services are the divergency items and the primary concern for each IoT system [1]. Therefore, any abstraction to define functionalities shall also allow fundamental freedom not to lose these divergency items.

To point out the challenges in the current state of the art within an example, a company which delivers IoT solutions for the factory automation systems can be taken into account. This company customizes their system to match with the needs of their customers. In the scope of data processing, this customization may be adding the support of the different type of devices, introducing new methods into the data flow, implementation of brand-new feature or integration of features that are already implemented in the past for another customer.

All of these customizations may lead to challenges which will end up with hiring new developers or losing the potential customers. A generic and practical rule engine can be defined to implement and to maintain the functionalities of the necessary system in the IoT domain. The predefined and parameterized building blocks within a functional user interface (UI) and application program interface (API) can sustain a fast and practical environment to ease and erase these challenges.

1.2 Proposed Solution

This master thesis aims to build an ontology-based rule engine to define data flows needed for web services in the area of IoT within a generic and practical development environment. An abstraction of all functionalities that are necessary for any data flow would help to achieve this generic rule engine. The predefined and customizable building blocks, ontologies to store the data and the rules are crucial for a generic and practical platform with the help of the abstraction. On the other hand, the generalization of the system would create an applicability tradeoff. So to maintain practicality, the resulting platform shall be definite and comprehensive enough to support any IoT project and the other areas rather than IoT shall not be a concern. In other words, the building blocks shall not be too broad to support any domain. Additionally, all

customization concerning each building block shall be done in easy-to-use interfaces that are supplied to the users to maintain practicality.

While designing web services that will be used in different use cases in IoT as well as others, some design concerns must be taken into account by the developers. The techniques and the architecture to enable communication between all necessary nodes, the format for the data that will be stored, how and when data will be served can be considered as the primary design concerns [1]. After the design phase of the system has been finished, the design shall be easily implemented to make the prototypes live and to run them.

The resulting platform will provide solutions in different degrees for the practicality challenges and problems in the IoT domain. These challenges and issues can be listed as follows;

1. The requirement of the technical background,
2. The need for the time investment,
3. The maintenance of the system,
4. The reimplementation of already implemented functionality,
5. The deployment of the system,
6. The different technology backgrounds of the developer in the team.

There exist different options with different practicality levels to implement the designed web services in the platform.

The platform that is based on the rule engine provides the solutions to the developers by simplifying the working environment. It solves the problem of the requirement for an enormous amount of time investment and technical background, and maintenance challenge by providing practical tools. The platform shall render a practical and easy to learn web UI for the hobby users with a less technical background. The data and the rules, which are owned by the platform user, will be easily accessible and mutable by the platform user in the UI. The platform users will quickly adapt even if they have little technical background since the system provides building blocks to connect with each other by using the most common user experience (UX) elements such as drag-drop objects, wires, and forms.

The platform shall also store the data of the web applications of the platform users in an easily accessible, organizable and modifiable way without loss of genericity. M. G. Kibria et al. proposed that a semantic ontology which declares every device as objects on the gateways can be used for the energy efficiency [2]. N. Lee and H. Lee

also suggested an IoT service architecture that provides services with an object gateway [3]. The device objectification by using ontologies is a reliable method to store the data of the real world devices in IoT services. Nevertheless, the management of read-write access to the objects, the evaluation of the gathered data and the methods for the service of these data are the topics that need to be covered to build a generic and practical platform that serves any needs of IoT.

To build business logic and define evaluation methods on the ontologies; the rules, which are in if-then form decision-making tools, must be introduced in the platform. The rules function like a chain of blocks, where each does a small fraction of the functionality while interpreting the logic. There are different types of blocks that each has a different function in the rule handling. The limited toolbox of generic rule blocks also preserves the practicality for the users since they do not need to master a vast pool of different block types.

In addition to these simplifications, the platform also supports reusability of any functionality or data hierarchy to avoid the reimplementation challenge. Therefore, the developers such as in the company that is given in the subsection 1.1 will not need to reimplement functionalities that are required by a new customer. Instead, they will reference the data or the features that are already implemented for another customer.

The deployment of the system will be easier for the most basic user with the existing UI. However, there will be more tools for the power users to handle their deployment challenge. The power users such as working in a company are provided with an application program interface (API) to use any functionality of the platform through HTTP requests and Internet of Things Easy Query Language (IoTeQL). By using these, the power users can manipulate their data and rules as well as the UI for their basic operations on their system. With the help of API and IoTeQL, existing systems such as the system of the company can be transferred to the platform quickly with the help of a script. Moreover, when the company needs to provide their services with a little customization for a new customer, they can promptly achieve it with the help of reusable components.

Since the platform is technology independent and focuses on the construction of the logic, different technology background of the team members will not be a concern for the developer teams. They will easily adapt to the development environment of the platform with any background.

In figure 1.1, the high-level architecture of the rule manager and the helper components is shown. The task of the rule manager is to run the rules on the data when required. The management of the data and the rules on the ontology is the responsibility of the data manager. Since they are isolated and loosely coupled with each other, they can autoscale individually behind a load balancer and expansion of the computational unit where it is needed can be achieved. Consequently, only the data

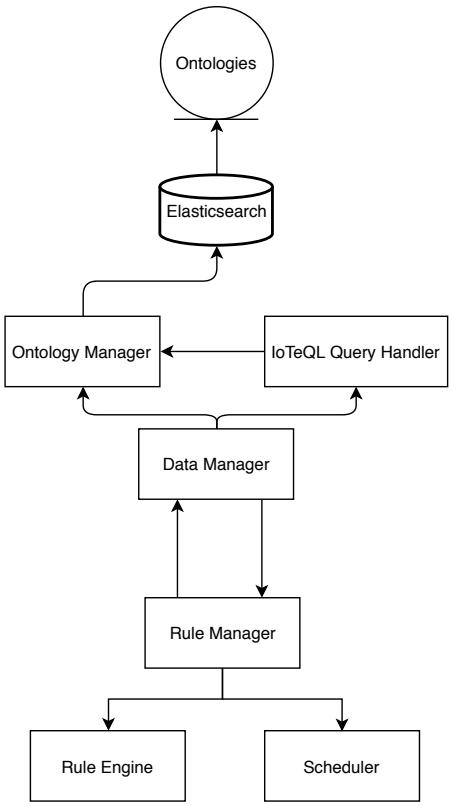


Figure 1.1: The high level architectural diagram of the platform

manager will horizontally scale when the number of queries is increased, or just the rule manager will scale horizontally when the massive amount of scheduled rules is needed to be handled at the same time. A platform user can access all of the subsystems by their APIs, and the connector components of them act like adapters to interact with each other. Therefore, any change in any subsystem will not lead to a change in the overall architecture.

In figure 1.2, the ontology representation of a smart thermometer is shown. The thermometer is represented as a type with the data fields and the data streams. A type can also have another type as one of its fields to describe complex structures. For example, the smart thermostat can be represented as another type and have the thermometer as one of its fields. A single data such as name, location or last data timestamp which describes an object or the state of an object is represented as a data field. The data streams such as hourly or daily temperature contain time-series data in them. A type can also have a field such as broken or time alive which serves data that are calculated on demand with the help of the rules. In this example, the broken

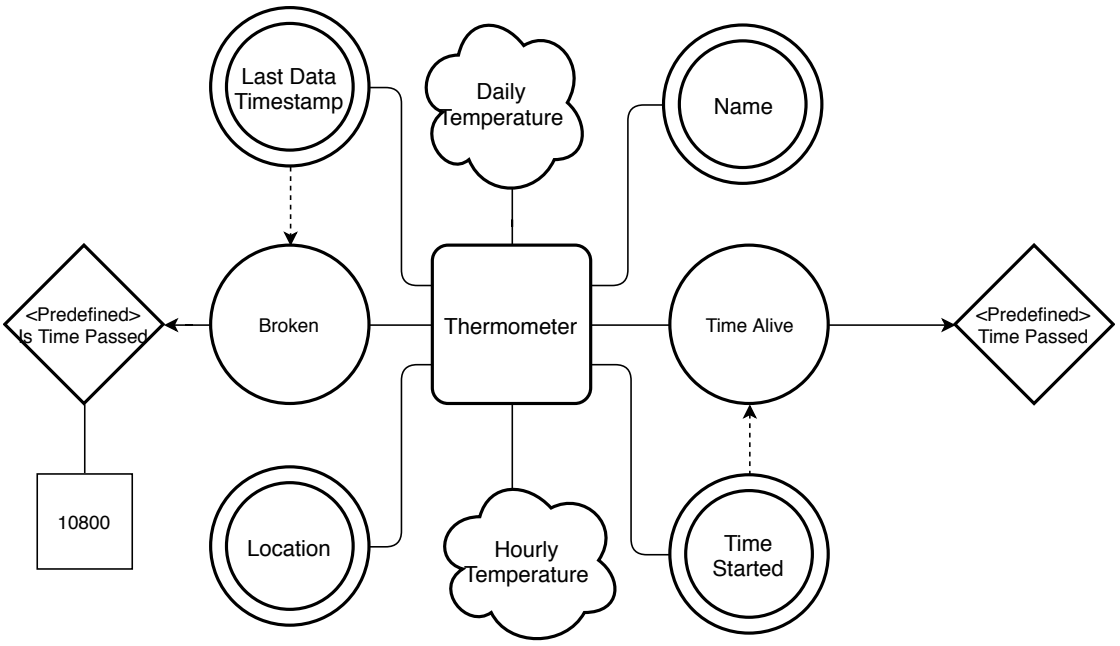


Figure 1.2: The ontology of the smart thermometer

and the time-alive fields serve necessary data to consumers with predefined rule blocks which maps the data on other fields to the data needed. Moreover, the manipulation of the data fields and streams is also possible with the help of the rules. A producer can only produce hourly temperature data, and the rule manager can generate daily data using the average of this hourly data. The implementation of this scenario can be found in the following chapters.

2 Data Management

2.1 Isolation of the Data of the Individual Users

2.2 Ontology Management

2.2.1 Data Hierarchy and Representation in Ontologies

2.2.2 Accessing and Manulapulating the Data in Ontologies

2.3 Rule Management

2.3.1 Rule Representation

2.3.2 Predefined Function Blocks

2.3.3 Rule Engine

2.3.4 Scheduling of the Rules

2.3.5 Support For Edge Computing

2.3.6 Performance Issues and Solutions of Rules

3 Software Tools and Techniques for Rule Engine

3.1 User Interface

3.1.1 The Implementation of the Rules

3.2 Application Programming Interface

3.2.1 Internet of Things Easy Query Language

3.2.2 Performance Results

4 Use Case Implementation

The use case implementation for the smart home thermostat system.

4.1 Ontology Implementation

4.2 Rule Implementation

4.2.1 IoTeQL

```
$create[type]

{
	name: "Thermostat",
	data_fields:
	[
		{
			name: "name",
			type: String,
			required: true
		},
		{
			name: "location",
			type: GeoJSON/point
		},
		{
			name: "time_started",
			type: DateTime,
			updatable: false,
			inital: $time.now
		},
		{
			name: "time_last_data_arrived",
			type: DateTime
		},
		{
			name: "is_broken",
			type: Rule,
			rule: $create[rule]
			{
				name: "is_broken",
				initial: $create[node]
				{
					object: $self,
					data_fields:
					[
						"time_last_data_arrived"
					],
					target: $builtin[map](name: "is_time_passed")
					{
						parameters: [10800],
						target: $create[sink]
						{
							name: "is_broken"
						}
					}
```

```
                                }
                        }
                },
                {
                        name: "time_alive",
                        type: Rule,
                        rule: $create[rule]
                        {
                                name: "time_alive",
                                initial: $create[node]
                                {
                                        object: $self,
                                        data_fields:
                                        [
                                                "time_started"
                                        ],
                                        target: $builtin[map](name: "time_difference")
                                        {
                                                target: $create[sink]
                                                {
                                                        name: "time_alive"
                                                }
                                        }

                                }
                        }
                },
                {
                        name: "temperature_on_last_day_hourly",
                        type: [Float]
                },
                {
                        name: "temperature_on_last_month_daily",
                        type: [Float]
                }

        ]
}

$create[object](name: "Thermostat")

{
        "name": "Foo Thermostat",
        "location":
        {
                "type": "Point",
                "coordinates": [170,11]
        }
}

$create[rule]

{
        name: "save_hourly_temp",
        initial: $create[source]
        {
                name: "save_hourly_temp",
                target: $builtin[query](name: "append")
                {
                        parameters:
                        {
                                "id": $input.device_id,
                                "field": "temperature_on_last_day_hourly"
                        },
                        target: $create[branch]
                        {
                                targets:
                                [
                                        {
                                                condition: true,
                                                target: $create[sink]
                                                {
                                                        name: "save_hourly_temp"
                                                }
```

```
                                        },
                                        {
                                                condition: true,
                                                target: $create[event]
                                                {
                                                        name: "daily_ready",
                                                        condition: ($input.temperature_on_last_day_hourly.size >= 24)
                                                }

                                        }
                                ]
                        }
                }
        }
}

$create[rule]

{
        name: "save_daily_temp",
        initial: $defined[event](name: "daily_ready")
        {
                target: $create[branch]
                {
                        targets:
                        [
                                {
                                        condition: true,
                                        target: $builtin[reduce](average)
                                        {
                                                parameters: {"field": "temperature_on_last_day_hourly"}
                                                target: $builtin[query](append)
                                                {
                                                        parameters:
                                                        {
                                                                "field": "temperature_on_last_day_hourly",
                                                                "value": $input.temperature_on_last_day_hourly
                                                        },
                                                        target: $create[sink]
                                                        {
                                                                name: "daily_temp_saved"
                                                        }
                                                }
                                        }
                                },
                                {
                                        condition: true,
                                        target: $builtin[query](reset)
                                        {
                                                parameters: {"field": "temperature_on_last_day_hourly"}
                                        }

                                }
                        ]
                }
        }
}
```

5 Related Works

5.1 Amazon Web Services Internet of Things Core (AWS IoT Core)

5.2 Azure Iot Hub

6 Literature Overview

7 Impacts and Results

7.1 Need of Time Investment

7.2 Scalability

7.3 Performance Measures

7.4 Reusability and Availability

7.4.1 Dockerization

7.4.2 Open Source

7.4.3 Continous Integration

7.4.4 Unit and Integration Tests

7.5 Future Improvements

List of Figures

List of Tables

Bibliography

[1] G. Suciu, S. Halunga, A. Vulpe, and V. Suciu, "Generic platform for iot and cloud computing interoperability study," in *International Symposium on Signals, Circuits and Systems ISSCS2013*, pp. 1–4, July 2013.

[2] M. G. Kibria, M. A. Jarwar, S. Ali, S. Kumar, and I. Chong, "Web objects based energy efficiency for smart home iot service provisioning," in *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 55–60, July 2017.

[3] N. Lee and H. Lee, "Device objectification for internet of things services," in *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*, pp. 1–2, June 2014.