A Case Study on Self-configuring Systems in IoT Based on a Model-Driven Prototyping Approach

Fabian Kneer(✉) and Erik Kamsties

Dortmund University of Applied Sciences and Arts,
Emil-Figge-Str. 42, 44227 Dortmund, Germany
{fabian.kneer,erik.kamsties}@fh-dortmund.de
http://www.fh-dortmund.de/

Abstract. [**Context and motivation**] In the last years, the development of the Internet of Things (IoT) with self-configuring systems (SCS) became more important. Consequently, many different solutions have been developed. [**Question/problem**] We observed a lack of common benchmarks, in particular for the IoT domain to evaluate and compare solutions. There are very few accessible cases (examples) for SCSs published at all. [**Principal ideas/results**] We propose a case from the IoT domain, smart cities in particular, which comprises of hardware and software components. Starting point is a smart street lighting system with communication between the lamps and passing cars. [**Contribution**] First, in this paper we present our initial results of running a case study with a model-based prototyping framework on the smart street light. The framework includes a software simulation of the street lamp and the events from the passing cars. Second, an engineer can use the case as a benchmark to compare several approaches in order to make a more informed decision which approach to choose.

Keywords: Self-configuring systems · Case studies · Experimentation

1 Introduction

The Internet of Things (IoT) is a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies [5]. A *thing* is an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks. A *device* is a piece of equipment with the mandatory capabilities of communication and the optional capabilities of sensing, actuation, data capture, data storage, and data processing. One key concern of systems operating in the IoT is to dynamically adapt to changing environments, due uncertainties during requirements-, design-, and run-time.

A considerable number of concepts for self-configuring systems (SCS) has been developed. From a practitioner's perspective, *open source* implementations

G. Dregvaite and R. Damasevicius (Eds.): ICIST 2016, CCIS 639, pp. 732–741, 2016.
DOI: 10.1007/978-3-319-46254-7_59

of the MAPE feedback loop (IBM [1]) for prototyping purposes are missing. This is a show-stopper in practice, as a practitioner would need to work through research papers in order to build such a prototype. A researcher who is interested in the comparison, extension and/or application of existing solutions to a new domain is in a similar situation.

We suggested a *prototyping and evaluation framework* for self-adaptive systems in [3]. The goal of the framework is to ease the prototyping (and possibly development) of self-configuring systems. For this purpose, the framework offers implementations of selected approaches to SCS based on the MAPE loop (e.g., based on feature models as suggested by Pascual et al. [6]). Another goal of the framework is to ease the evaluation of self-configuring systems, to allow for instance benchmarks between different approaches. For this purpose, we developed a case study drawn from the *smart city* domain. The framework is able to collect data on a subset of the metrics at runtime about overall quality, effort, and cost.

In this paper, a academic case study for *Smart Street Lighting* will be presented as an example of a self-configuring system developed using the abovementioned prototyping framework. The case is based on the real world and is transfered to a model world. In the area of IoT a simple representation of the real world is needed as a base for a simulation to verify and test the requirements and decide between the different used approaches for the implementation of a self-configuring system.

The following section presents an overview of the prototyping framework. Section 3 reports on the construction of the case study and the simulation. Section 4 presents the results and the future work.

2 Overview of the Prototyping Framework

The framework bases on experiences we made with the implementation of a goal-oriented (i*-based) approach to self-adaptive systems [2]. We developed a generalized architecture for prototyping SAS in the spirit of a software product-line (SPL), by an analysis of *commonalities* and *differences* of approaches suggested for SAS. We implemented a feature-orientated approach (Pascual et al. [6]) to validate the initial architecture of the prototyping framework. The details of the implementation, and first experiences are described in the remainder of this section.

Implementation. The framework contains components to implement the different activities of the MAPE loop as well as further functions that are needed to build and run an adaptive system.

Figure 1 shows the concept of the prototyping framework. Fundamentally, we separate *development time* and *runtime* artifacts. The development time artifacts represent additional aspects to be captured in the requirements phase. Runtime artifacts are components required to build the SAS. To easy prototyping, some runtime artifacts are generated from the development time artifacts.

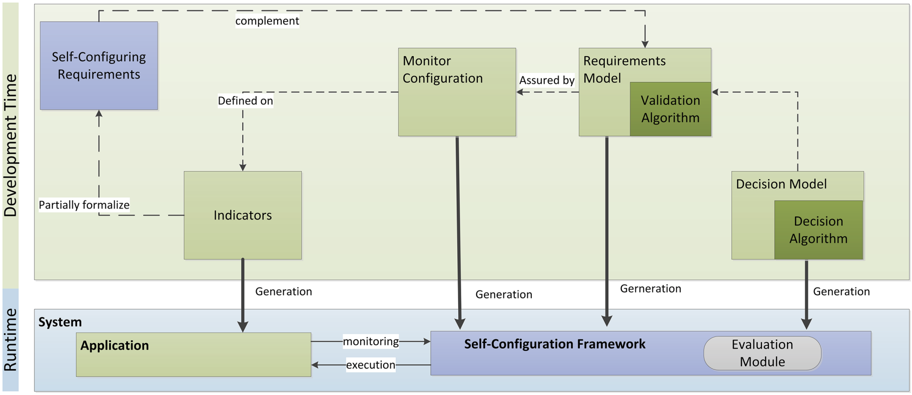


Fig. 1. Prototyping and evaluation framework

The prototyping framework provides a set of components from which can be chosen to develop a self-adaptive system. At the time of writing it contains a feature-oriented approach with utility functions to compute a configuration with a genetic algorithm (see Pascual et al. [6]).

Generator. In order to ease the development of a prototype, a self-adaptive system is partially *generated*. That is, during development time (see upper part of Fig. 1), only the essential information for building the SAS is specified by the user. When using the feature-oriented approach this is: *indicators* (variables of the systems with a type), a *feature model* (different features of the systems and variation points), *ECA rules* (indicator change event, boolean condition, change action on a element, e.g. feature), *utility function* (table with utility values of the different variations of the feature model).

Probes for an application and the whole self-configuring framework (with integrated MAPE loop), see bottom part Fig. 1 can now be *generated* out of a specification. The indicators are used to generate probes for an application. These probes are used by the self-configuring framework to monitor the indicators and check the related ECA rules. The ECA description is used to generate the rules of the ECA rule engine. An application that includes the generated probes can use the self-configuring systems to adapt the behavior to the changing context.

If the specification changes because new elements like indicators, ECA rules, features or utility elements are added or elements are refined for example an ECA rule change, only the generation process is re-triggered and the new prototype can be used for the system which contains the probes. Further details about the generator concept are given in [4].

3 Case Study

Domain. The *Smart City* domain is selected as an instantiation of the Internet of Things. The *Smart Street Lighting* is selected as a subdomain to start with. It is accessible to many readers, it is complex, and comprises many different facets.

Street lights become an important part of smart cities. The lights are extended with new functions beyond the usual function of providing light to a place: the lights are equipped with increasing computing power and communication capabilities like wireless connection, digital street signs, and sensors to measure their environment.

An example for a smart light is provided by the company *Illuminating concepts*[1]. They have designed a flexible wireless solution that is called *Intellistreets*, which includes a energy efficient lighting, audio, digital signage, and more. Figure 2 shows the design of the solution. The light can communicate with other systems and also with humans to help finding a way or send an emergency calls. Also the audio and digital signage can be used for entertainment or announcements.

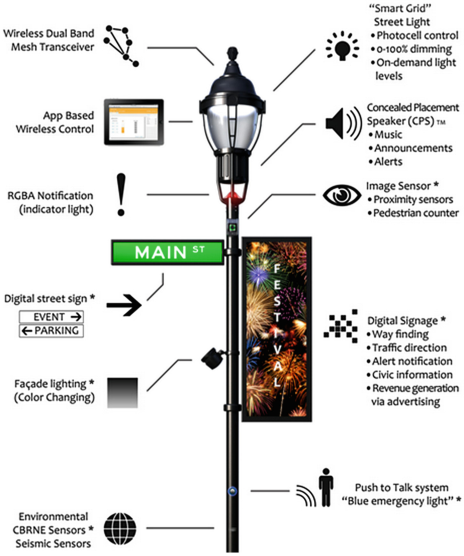
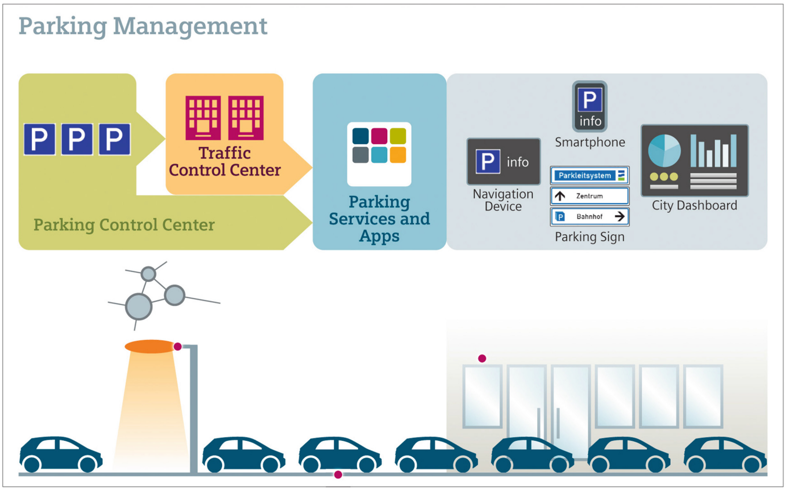

Wireless Dual Band Mesh Transceiver
App Based Wireless Control
RGBA Notification (indicator light)
MAIN ST
Digital street sign *
EVENT →
← PARKING
Façade lighting * (Color Changing)
Environmental CBRNE Sensors * Seismic Sensors
"Smart Grid" Street Light
• Photocell control
• 0-100% dimming
• On-demand light levels
Concealed Placement Speaker (CPS) TM
• Music
• Announcements
• Alerts
Image Sensor *
• Proximity sensors
• Pedestrian counter
FESTIVAL
Digital Signage *
• Way finding
• Traffic direction
• Alert notification
• Civic information
• Revenue generation via advertising
Push to Talk system "Blue emergency light" *

Fig. 2. Intellistreets solution developed by *Illumination concepts* (See Footnote 1).

Further, more companies like *Siemens*[2] are producing parking management systems. The lights have sensors like distance and movement to detect parking

[1] www.illuminatingconcepts.com.

[2] www.siemens.com.



Source: Siemens

Fig. 3. Smart parking systems developed by *Siemens* (See Footnote 3).

vehicles under a light. This information is used to inform driver who are searching for a parking slot. Figure 3 shows the concept of the management system. The information about a free or used parking slot is sent to the traffic control center and also to parking services and apps. The car driver can be informed by navigation systems, smart phones, or over the previously shown digital signage of a smart street light.

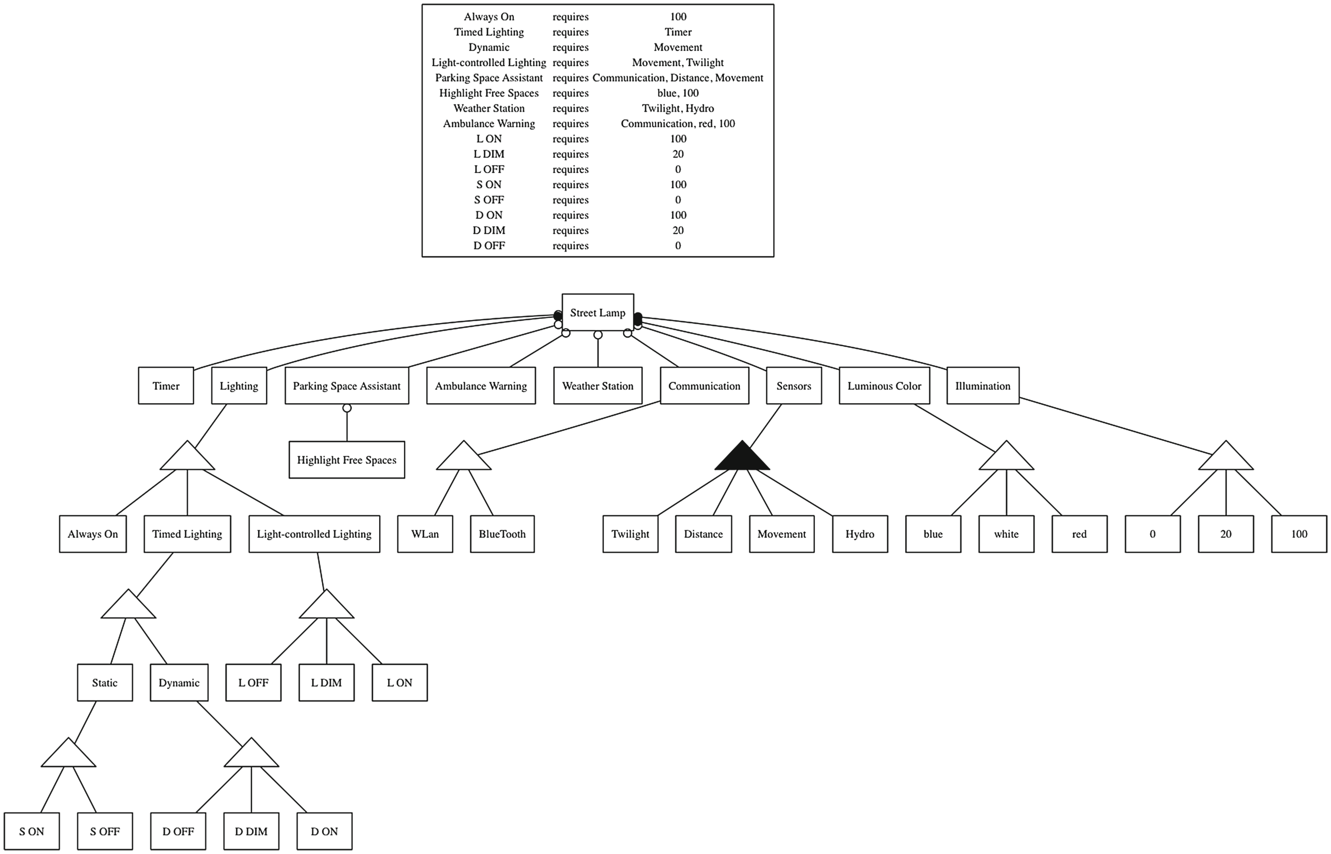
Specification of the Smart Street Light case. The development artifacts of the prototype are developed in a tree structure. The context is *Smart City*, the system is *Public Lighting*, and the subsystem *Smart Street Light*. For the *Smart Street Light* a *Prototype Configuration* is needed. This configuration contains the development artifacts that are chosen for the prototype (e.g. feature model, utility table, and event-condition-action rules (ECAs)). The artifacts are described in a *Software Requirements document*.

The development artifacts are used by a *Generator* to produce source code of a prototype.

In the remainder of the section, the development artifacts that describe the self-configuration of the street lamp are shown. The artifacts are used to generate parts of the self-configuration framework and the probes for the application.

Feature Model. Figure 4 shows the feature model for the street light are shown. The feature model is modeled using the prototyping framework. The different realization strategies are developed as variation points in the model.

The lamp can adapt its luminous color and illumination. The possible values for these parameter are represented as alternative group (Xor-Group) in the feature model. The possible colors are white, blue, and red. The illumination is shown in percent and can variate between 0 %, 20 % and 100 %.



Always On	requires	100
Timed Lighting	requires	Timer
Dynamic	requires	Movement
Light-controlled Lighting	requires	Movement, Twilight
Parking Space Assistant	requires	Communication, Distance, Movement
Highlight Free Spaces	requires	blue, 100
Weather Station	requires	Twilight, Hydro
Ambulance Warning	requires	Communication, red, 100
L ON	requires	100
L DIM	requires	20
L OFF	requires	0
S ON	requires	100
S OFF	requires	0
D ON	requires	100
D DIM	requires	20
D OFF	requires	0

Fig. 4. Feature Model of a Public Street Light. (Color figure online)

The *abstract Sensors* of a street light are *Twilight* to measure light, *Distance* to measure if an object is under the light, *Movement* to react to moves near the light and *Hydro* for weather information.

The lamp can choose between the following three different options to light the street:

- *Always On*: is a error state that results in a street light with maximal illumination. (See constraints: *Always On requires 100*)
- *Timed Lighting*: the light is on during a given time interval. This feature needs a timer to react and activate the light (see constraints). This feature can variate between *Static* and *Dynamic* mode. *Static* leads to a 100 % illumination during the interval and *dynamic* reacts to movements by switching between 20 % and 100 % illumination.
- *Light-controlled Lighting*: this feature requires a *Movement* and a *Twilight* sensor. If the twilight sensor indicates a need for light, the light is put on. Like the dynamic mode of the *Timed Lighting* feature, the light-controlled feature reacts to the movement sensor by switching between 20 % and 100 % illumination.

The next feature is *Parking Space Assistant*. This feature sends information about the free parking spaces under the street light to connected systems like navigation system and also *Highlight Free Spaces* with a blue luminous color.

The last feature is *Ambulance Warning*. If the lamp has an established connection, it gets information about ambulances that pass the street light. The lamp reacts and try to warn its environment by switching to a red luminous color.

Indicators. The case study has status indicator for sensors of the systems, e.g. movement sensor. A boolean value shows if the sensor works as expected or deliver wrong values. In addition to the status indicators, the following indicators are defined:

- `ambulance` shows if an ambulance will pass the street light.
- `detectTwilight` shows a change of the daylight.
- `cars` shows the number of parking cars under the street light.
- `searchingCar` shows if a driver near the street light search for a free parking space.
- `detectMovement` shows if a person or car moves near the street light.

The next three indicators represent the time values of the street light. These are the current system time (`time`), a parameter when the street light should be turned *on* (`turnOnTime`), and a parameter when the street light should be turned *off* (`turnOffTime`).

Simulation. Both the application and the framework are generated as a console application and they need a GUI. Figure 5 shows the GUI for the application. On the left side of the screenshot, the software simulation of the previously presented street light is shown. On the right side, the configurable values are shown.

The first values represent the time indicators (*currentTime*, *turnOffTime*, and *turnOnTime*). The next values represent the status of the sensors. A defect sensor is colored *red*. In this example screen-shot, all sensors are working correctly, which results in green colored sensors. The two buttons with the image of vehicles, can be used to start a moving vehicle (ambulance in the example screen shot). The last configurable values are *parking cars*. Next to the parking symbol are the buttons + and −. Up to three cars can be added to the parking spaces under the street light.

The progress bar and the *Go* button are used to represent and start a scenario. A scenario represents a day of the street light with random events, which are produced during this interval.

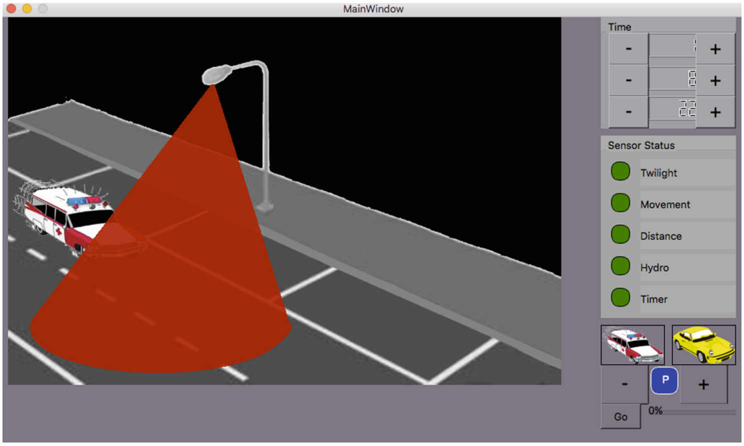


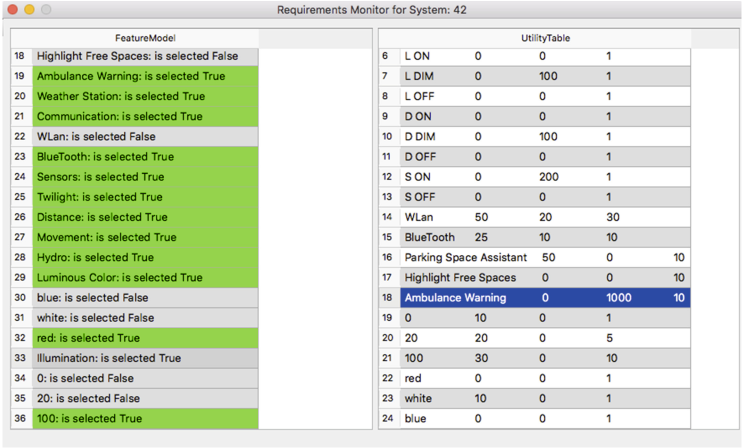
Fig. 5. GUI of the simulated Street Light. (Color figure online)

Figure 6 shows a screenshot of the prototypical GUI for the self-configuring street light. The left table represents the feature model. The *selected* features in the current configuration are colored green and the *deselected* are colored gray. The right table is the utility table. The rows represent a utility element with the utility values and the resource costs.

In the given example, the street light notices a passing ambulance and switches to a configuration with a red light color and maximum illumination.

4 Results and Conclusion

This paper presented an academic case study in which a self-configuring system - a smart street light - was developed using a prototyping framework (developed by the authors in previous work). The case was used to give an impression of the required effort, we provide programming effort data. The development has taken about 3 man-week's and 1500 LoC:



Requirements Monitor for System: 42

	FeatureModel
18	Highlight Free Spaces: is selected False
19	Ambulance Warning: is selected True
20	Weather Station: is selected True
21	Communication: is selected True
22	WLan: is selected False
23	BlueTooth: is selected True
24	Sensors: is selected True
25	Twilight: is selected True
26	Distance: is selected True
27	Movement: is selected True
28	Hydro: is selected True
29	Luminous Color: is selected True
30	blue: is selected False
31	white: is selected False
32	red: is selected True
33	Illumination: is selected True
34	0: is selected False
35	20: is selected False
36	100: is selected True

	UtilityTable			
6	L ON	0	0	1
7	L DIM	0	100	1
8	L OFF	0	0	1
9	D ON	0	0	1
10	D DIM	0	100	1
11	D OFF	0	0	1
12	S ON	0	200	1
13	S OFF	0	0	1
14	WLan	50	20	30
15	BlueTooth	25	10	10
16	Parking Space Assistant	50	0	10
17	Highlight Free Spaces	0	0	10
18	Ambulance Warning	0	1000	10
19	0	10	0	1
20	20	20	0	5
21	100	30	0	10
22	red	0	0	1
23	white	10	0	1
24	blue	0	0	1

Fig. 6. GUI of the Feature-based Prototype. (Color figure online)

- ECA's are used for monitoring the system (*400 LoC 3–4 Days*)
- Feature model is used as requirements model and representation of alternative realization strategies (*600 LoC 3–5 days*)
- Utility Table and - function are used to compute a configuration (*200 LoC 3 Days*)
- A genetic algorithm is used to choose the "best" configuration (*100 LoC 2 Days*)
- Configuration example (*150 LoC 1/2 day*)
- Simple Specification (1 textual requirement and feature model with 19 features) (*200 LoC 1/2 day*)

The case itself can be used by a requirements engineer to validate approaches from the area of self-configuration that are integrated in the framework. The case is taken from a subdomain of the smart city domain as an instantiation of the Internet of Things. The case is scalable and can be extended by additional systems or features of the street light, for example, more lights or a public utility system that is connected to the street lights. These extensions could lead to more complex case studies.

The framework for prototyping and evaluation can be used in a number of ways on the given case study:

- to *understand* a particular SCS approach,
- to *optimize* the application of an approach, or
- to *compare* approaches in a particular target environment.

At the time of writing we have developed a software simulation and a prototype of a hardware street light. To enlarge the case, a RC-car is under development, which enriches for instance the parking scenario. Also part of our future work is to build a small street model with up to five street lights, which can be connected to a virtual simulation of a smart city.

It is planed to make the framework and the case study open source and publicly available using GitHub, when it has passed a first validation.

References

1. An architectural blueprint for autonomic computing. Technical report, IBM (2005)
2. Kamsties, E., Kneer, F., Voelter, M., Igel, B., Kolb, B.: Feedback-aware requirements documents for smart devices. In: Salinesi, C., Weerd, I. (eds.) REFSQ 2014. LNCS, vol. 8396, pp. 119–134. Springer, Heidelberg (2014). doi:10.1007/978-3-319-05843-6_10
3. Kneer, F., Kamsties, E.: A framework for prototyping, evaluating self-adaptive systems-a research preview. In: Bjarnason, E. et al. (ed.) REFSQ Workshops, CEUR Workshop Proceedings,vol.1564. CEUR-WS.org (2016)
4. Kneer, F., Kamsties, E.: Model-based generation of a requirements monitor. In: Joint Proceedings of REFSQ- Workshops, Research Method Track, and Poster Track co-located with the 21st International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2015), Essen, Germany, pp. 156–170, 23 March 2015
5. Overview of the Internet of things. In: IUT-T Y.2060 (2012)
6. Pascual, G.G., Pinto, M., Fuentes, L.: Run-time adaptation of mobile applications using genetic algorithms. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, San Francisco, CA, USA, pp. 73–82, 20–21 May 2013