Adaptive Reshaping of Web of Things and mobile software for the Fog Computing Era

Department of Electrical and Information Engineering

ELECTRICAL AND INFORMATION ENGINEERING
Ph.D. Program

SSD: ING-INF/05–INFORMATION PROCESSING SYSTEMS

Final Dissertation

Adaptive Reshaping of Web of Things and mobile software for the Fog Computing Era

---

by

NOCERA FRANCESCO

Supervisors:

Prof. DI NOIA Tommaso

Prof. MONGIELLO Marina

Coordinator of Ph.D. Program:
Prof. GRIECO Alfredo

Course n° 31, 01/11/2015 - 31/10/2018

Department of Electrical and Information Engineering

ELECTRICAL AND INFORMATION ENGINEERING
Ph.D. Program

SSD: ING-INF/05–INFORMATION PROCESSING SYSTEMS

Final Dissertation

Adaptive Reshaping of Web of Things and mobile software for the Fog Computing Era

---

by

NOCERA FRANCESCO

Referees:

Prof. MIKKONEN Tommi

Prof. PELLICCIONE Patrizio

Supervisors:

Prof. DI NOIA Tommaso

Prof. MONGIELLO Marina

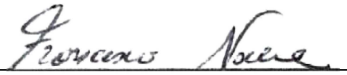
Coordinator of Ph.D. Program:
Prof. GRIECO Alfredo

Course n° 31, 01/11/2015 - 31/10/2018

Declaration of Authorship

I, Francesco NOCERA, declare that this thesis titled, "Adaptive Reshaping of Web of Things and mobile software for the Fog Computing Era" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- This thesis has been composed by myself and the presented work is my own under the guidance of my supervisors Tommaso Di Noia and Marina Mongiello.
- The following Chapters are based on work done by myself with Researchers and Professors from others University: Chapter 2 is co-authored with Niko Mäkitalo (University of Helsinki, Finland) and Stefano Bistarelli (University of Perugia, Italy). The work was accepted as a review paper for IET Special issue SAWoT [3] organized as a result of EnWoT workshop [5]; Chapter 3 is partially co-authored with Umberto Straccia (ISTI-CNR, Italy); Chapter 6 is co-authored with Francesco Maria Donini (Tuscia University, Italy); Chapter 7 is co-authored with Patricia Lago and Ivano Malavolta from Vrije Universiteit Amsterdam (Amsterdam, The Netherlands) where an internship has been carried out.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: 

Date: December 20, 2018

"Twenty years from now you will be more disappointed by the things that you didn't do than by the ones you did do. So throw off the bowlines. Sail away from the safe harbor. Catch the trade winds in your sails. Explore. Dream. Discover."

Mark Twain

"Le cose si fanno quando è il momento. Il mio è adesso."

Laura Pausini

POLYTECHNIC UNIVERSITY OF BARI

Abstract

Department of Electrical and Information Engineering
Electrical and Information Engineering Ph.D. Program

Doctor of Philosophy

Adaptive Reshaping of Web of Things and mobile software for the Fog Computing Era

by Francesco NOCERA

The software has become an essential part of every aspect of the society and our daily life. This is indicated by many recent trends in our society – healthcare, smart homes, smart cities, (autonomous) robots, autonomous connected vehicles and so on – all contain an ever-increasing amount of software. These trends also indicate that the single device computing era is coming to an end. However, the development and software architectures have not changed much during past ten years. At the moment, Cloud-based services are the de facto way how the software is constructed, and then provided for the end users in the form of Web and native apps. Moreover, from a software architecture point of view, Internet of Things (IoT) poses many interesting challenges due to its unpredictable yet adaptive requirements.
In this thesis, we investigate challenges related to Web of Things (WoT) and mobile software in the Fog Computing era in order to improve their adaptive behaviour.

Contents

List of Figures

List of Tables

List of Abbreviations

AR	**A**rchitectural **R**equirements
CEP	**C**omplex **E**vent **P**rocessing
FR	**F**unctional **R**equirement
IoT	**I**nternet of **T**hings
NFR	**N**on-**F**unctional **R**equirement
WoT	**W**eb of **T**hings

Chapter 1

Introduction

The software has become an essential part of every aspect of the society and our daily life. This is indicated by many recent trends in our society – smart homes, smart traffic, smart cities, (autonomous) robots, autonomous connected vehicles and so on – all contain an ever-increasing amount of software. These trends also indicate that the single device computing era is coming to an end. However, the development and software architectures have not changed much during past ten years. At the moment, Cloud-based services are the de facto way how the software is constructed, and then provided for the end users in the form of Web and native apps. This approach, however, cannot accommodate all the needs that come with the multi-device era where programmable objects are everywhere and require efficient and real-time coordination and distributed computations. This has lead to a situation where computation and coordination require new types of software architectures and programming models.

Internet of Things (IoT) has been one key research topics in computer science for some time. IoT has no single definition, and hence it is often referred as an approach for connecting all the physical things to the Internet, or, as an extension to mobile computing. Thus, IoT can be seen to have multiple research subfields, like the Internet of Industrial Things, the Internet of Vehicles, or the Internet of People. Web of Things (WoT) is one of these subfields and a general term used for describing all the approaches to connecting physical things to the World Wide Web (Stirbu, 2008; Guinard, 2009; Guinard et al., 2011a; Tran et al., 2017). In the coming years, people use more and more various types of Web-enabled client devices, and data is stored simultaneously on numerous devices and Cloud-based services. Hence it is the devices together with people and services which form the modern computing environment. The expectations towards interoperability will dramatically raise, which will imply significant changes for software architecture as well since the development is evolving from traditional client-server architectures to decentralized architectures. Thus, Cloud Computing is complemented with two new computing paradigms: *Edge Computing* (computation solely on the device-end) and *Fog Computing* (computation everywhere on the network level).

Fog Computing is an emerging paradigm that was presented by Cisco in 2012. It promises to be an evolution of Cloud-based systems and is primarily targeted for the Internet of Things (Bonomi et al., 2012). While Edge Computing is solely about computations on the network edge devices, the goal

of Fog Computing is to enable exploiting computation and data resources across Cloud services, edge devices, as well as intelligent network nodes. If today the *Cloud* is the most used abstraction and environment to handle remote applications, the *Fog* then offers the advantage of better supporting new computer applications in our connected world. For example, autonomous driving cars, remote monitoring systems for patients, drones for home delivery, the adaptive lighting of streets and homes can all benefit from Fog Computing. All this by leveraging the pervasive computing infrastructure that consists of ad hoc processors, smart routers, and personal devices such as smartphones for computations. This approach allows reducing bandwidth consumption in IoT environments, exploiting a distributed structure that is quite similar to that used in P2P (Peer-to-Peer) communications. In some cases, the Fog can also be seen as a parallel network to the public since it can allow access to resources and computing power without passing through a public Internet connection (Dastjerdi and Buyya, 2016).

Open standards and Web technologies provide tools and a platform for implementing applications in more vendor-neutral ways – in contrast to native apps that can run only on one platform. However, even though the communication is built-in to the Web, the interactions still happen in the same way as with native apps. Also, the Web browser essentially is an app itself and only offers a sandbox for interacting with other entities. Thus, pure Web technology based software partly suffers from the same, and even more limitations than native software do suffer. Despite these limitations, however, Web technologies can still have advantages over native apps (Taivalsaari et al., 2011), and be used for enabling co-operation and interactions between the devices and users. Moreover, Web technologies can teach a lot about standardization for enabling vendor-neutral interactions for the required architectures.

Furthermore, the interest in mobile applications has widely increased in the last years. Execution of these apps often depends on gestures, sensors and location data and allows adaptive behaviors. A variety of techniques and issues related to modeling, implementation and execution of such applications and to the well-known self-adaptive systems is summarized in (R. de Lemos et al., 2013). Adaptation gained increasing attention to classify issues of self-management (Huebscher and McCann, 2008), and of architectural decision making at design and runtime (McKinley et al., 2004). Several reference model have been also proposed (Kramer and Magee, 2007). Anyway runtime adaptation of software components is still a challenging problem (Oreizy, Medvidovic, and Taylor, 1998; Kramer and Magee, 2007).

Usually, adaptation actions are employed at an architectural level to add components, apps, services in a composite model. At behavioral level they enable dynamic changes in an app's behavior, deployment, and execution.

For this reason, the objective of this thesis is to study the future research avenues for architecting WoT software and mobile apps in order to improve adaptation.

1.1 Structure of the Thesis

In this paragraph we outline the structure of the subsequent chapters composing the two Parts of this Thesis: the Web Of Things (WoT) software dimensional view (Part I) and the Mobile software dimensional view (Part II).

Part I

Chapter 2 give a review of the current technological space for architecting Web technology-based IoT software in the coming era of fog computing. They focus on fundamental research challenges and discuss the emerging issues.

In Chapter 3 was proposed as main goal the development of a Decision Support System for supporting designers and software architect in the process of modeling a middleware-induced software system's architecture. To achieve the main goal of this research we propose a theoretical framework based on *Fuzzy Description Logics* (FDLs) for modeling knowledge about NFRs, FRs, ARs, middlewares, design patterns, and define a reasoning algorithm to manipulate the modeled knowledge.

Chapter 4 presents a reflective model whose aim is to inject adaptation into existing middleware. The reflective extension allows a software system to dynamically change its logic without internal changes to the code. In our approach, the awareness of the surrounding context is encoded by means of a rule-based system which drives the dynamic behavior of the middleware.

Part II

Chapter 5 present an approach to complex adaptive mobile applications modeling and implementation, able to dynamically change according to changed behavioral properties, state and/or text variables and user's preference. To this aim, we design a metamodel made up of an Action Repository to store triples composed by logical propositions to define criteria for selecting actions to be executed.

Chapter 6 discuss how user navigation patterns can be used for developing navigation-aware techniques for personalized prefetching of network requests of Android apps. The proposed idea opens for a new family of prefetching opportunities since it focusses at a higher level of abstraction with respect to state of-the-art approaches for network requests prefetching. The proposed idea allows the development of approaches which adapt their prefetching behaviour according to the unique navigation patterns each user exhibits while interacting with a mobile app.

1.2 Published Material

Editor of international journals and of proceedings of international conferences

[1]Marina Mongiello, Eugenio Di Sciascio, Francesco Nocera, and Pietro De Palma. Guest Editorial SPECIAL SECTION: Industry 4.0: the DIGITAl Transformation in the Engineering Findings (DIGITATE), IET journal of Engineering. IET, 2018. To appear.

[2]Antonio Bucchiarone, Marina Mongiello, Francesco Nocera, Michael Sheng. "Preface of the 1st International Workshop on Ensemble-based Software Engineering (EnSEmble 2018)", ESEC/FSE 2018- Proceedings of the 2018 12th Joint Meeting on Foundations of Software Engineering, ACM Digital Library.

[3]Marina Mongiello, Francesco Nocera, Tommaso Di Noia, and Eugenio Di Sciascio. "Guest Editorial: Software Architecture for the Web of Things (SAWoT)", IET Software, 2018, 12, (5), p. 379-380.

[4]Marina Mongiello, Francesco Nocera, Niko Makitalo, and Diego Perez-Palacin. "Preface of the 2nd International Workshop on Engineering the Web of Things (EnWoT 2018)", Càceres, Spain, June 5, 2018. In ICWE International Workshops 2018, LNCS Workshop PostProceedings. Springer, 2018.

[5]Marina Mongiello, Tommaso Di Noia, Eugenio Di Sciascio and Francesco Nocera. "Preface of the 1st International Workshop on Engineering the Web of Things (EnWoT 2017)", Rome, Italy, june 4, 2017. In ICWE International Workshops 2017, ICWE International Workshops 2017, LNCS 10544. Springer, 2017.

International Journals

[6]Niko Mäkitalo, Francesco Nocera, Marina Mongiello, and Stefano Bistarelli. "Architecting the Web of Things for the fog computing era." IET Software (2018).

[7]Marina Mongiello, Francesco Nocera, Angelo Parchitelli, Luigi Patrono, Piercosimo Rametta, Luca Riccardi, and Ilaria Sergi. "A Smart IoT-Aware System For Crisis Scenario Management." Journal of Communications Software and Systems 14, no. 1 (2018): 91-98.

[8]Tommaso Di Noia, Marina Mongiello, Francesco Nocera, and Umberto Straccia. "A fuzzy ontology-based approach for tool-supported decision making in architectural design." Knowledge and Information Systems (2017): 1-30.

[9]Tommaso Di Noia, Eugenio Di Sciascio, Francesco Maria Donini, Marina

Mongiello, and Francesco Nocera. "Formal model for user-centred adaptive mobile devices." IET Software 11, no. 4 (2017): 156-164.

Book chapters and International series

[10]Ivano Malavolta, Francesco Nocera, Patricia Lago, Marina Mongiello. "Navigation-aware and Personalized Prefetching of Network Requests in Android Apps", In Proceedings of the International Conference on Software Engineering (ICSE-NIER 2019).

[11]Francesco Nocera. "Reshaping Distributed Agile and Adaptive Development Environment: Student Research Abstract." In Proceedings of the 2018 12th Joint Meeting on Foundations of Software Engineering, ACM, 2018.

[12]Claudio De Meo, Nicola Siena, Luca Riccardi, Francesco Nocera, Angelo Parchitelli, Marina Mongiello, Eugenio Di Sciascio, Niko Makitalo. "LiquiDADE: a Liquid-based Distributed Agile and Adaptive Development Environment (DADE) Multi-Device tool." In Proceedings of the 2018 12th Joint Meeting on Foundations of Software Engineering, ACM, 2018.

[13]Francesco Nocera, Marina Mongiello, Eugenio Di Sciascio, and Tommaso Di Noia. "MoSAIC: a middleware-induced software archIteCture design decision support system." In Proceedings of the 12th European Conference on Software Architecture (ECSA), p. 5. ACM, 2018.

[14]Marina Mongiello, Francesco Nocera, Angelo Parchitelli, Luca Riccardi, Leonardo Avena, Luigi Patrono, Ilaria Sergi, and Piercosimo Rametta. "A Microservices-based IoT Monitoring System to Improve the Safety in Public Building." In 2018 3rd International Conference on Smart and Sustainable Technologies (SpliTech), pp. 1-6. IEEE, 2018.

[15]Francesco Nocera. "A Liquid Software-driven Semantic Complex Event Processing-based platform for health monitoring", 33rd ACM/SIGAPP Symposium on Applied Computing (SAC SRC 2018), April 2018, Pau (France).

[16]Vito Bellini, Tommaso Di Noia, Marina Mongiello, Francesco Nocera, Angelo Parchitelli, and Eugenio Di Sciascio. "Reflective Internet of Things Middleware -Enabled a Predictive Real-Time Waste Monitoring System." In International Conference on Web Engineering, pp. 375-383. Springer, Cham, 2018.

[17]Stefano Bistarelli, Tommaso Di Noia, Marina Mongiello, and Francesco Nocera. "PrOnto: an Ontology Driven Business Process Mining Tool." Procedia Computer Science 112 (2017): 306-315.

[18]Angelo Parchitelli, Francesco Nocera, Giorgio Iacobellis, Marina Mongiello, Tommaso Di Noia, Eugenio Di Sciascio. "A pre-process clustering methods

for the Waste Collection Problem", IEEE International Conference on Service Operations, Logistics, and Informatics (SOLI 2017), September 2017, Bari (Italy).

[19]Marina Mongiello, Luigi Patrono, Tommaso Di Noia, Francesco Nocera, Angelo Parchitelli, Ilaria Sergi, Piercosimo Rametta. "A Complex Event Processing (CEP)-based aid system for fire and danger management", The 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI 2017), June 2017, Vieste (Italy).

[20]Francesco Nocera, Angelo Parchitelli. "An adaptive Model for Semantic Complex Event Processing- driven Social Internet of Things Network", The 17th International Conference on Web Engineering (ICWE 2017), June 2017, Rome (Italy).

[21]Francesco Nocera, Tommaso Di Noia, Marina Mongiello, and Eugenio Di Sciascio. "Semantic IoT Middleware-enabled Mobile Complex Event Processing for Integrated Pest Management." In CLOSER, pp. 610-617. 2017.

[22]Tommaso Di Noia, Marina Mongiello, Francesco Nocera, Eugenio Di Sciascio. "Ontology-based reflective Iot middleware-enabled agriculture Decision Support System", 9th International Semantic Web Applications and tools for life sciences Conference (SWAT4LS 2016), December 2016, Amsterdam (The Netherlands).

[23]Marina Mongiello, Tommaso Di Noia, Francesco Nocera, Eugenio Di Sciascio. "Case-based reasoning and Knowledge-graph based metamodel for runtime adaptive architectural modeling", 31st ACM/SIGAPP Symposium on Applied Computing (SAC 2016), April 2016, Pisa (Italy).

[24]Francesco Nocera. "Student Research Abstract: Fuzzy ontology-driven web based framework for supporting architectural design", 31st ACM/SIGAPP Symposium on Applied Computing (SAC 2016), April 2016, Pisa (Italy).

[25]Marina Mongiello, Tommaso Di Noia, Francesco Nocera, Eugenio Di Sciascio, Angelo Parchitelli. "Context-aware design of reflective middleware in the Internet of Everything", 1st International Workshop on Formal to Practical Software Verification and Composition (VeryComp 2016) - Co-located with Software Technologies: Applications and Foundations (STAF 2016), July 2016, Vienna (Austria).

National Conferences

[26]Francesco Nocera, Tommaso Di Noia, Marina Mongiello, Angelo Parchitelli, Eugenio Di Sciascio, Stefano Bistarelli. "An adaptive Complex Event Processing-driven SIoT network formal metamodel", 3rd Italian Conference on ICT for Smart Cities & Communities (I-CiTies 2017), September 2017, Bari

(Italy).

[27]Tommaso Di Noia, Marina Mongiello, Francesco Nocera, Eugenio Di Sciascio. "Persistence on different databases via reflective IoT Middleware ", 24th Italian Symposium on Advanced Database Systems (SEBD 2016), June 2016, Ugento (Italy).

Pending Review publication

The content of Chapter 4 is partially (extended version of paper [25]) pending review in:

[28]Francesco Nocera, Marina Mongiello, Tommaso Di Noia, Eugenio Di Sciascio. "Context-aware Middleware for the Internet Of Things based on fuzzy rules and reflective model", Submitted to IEEE Internet of Things Journal.

Part I

Web of Things (WoT) Software dimensional view

Chapter 2

Architecting the Web of Things in the Fog Computing era

Fog Computing paradigm is emerging after a decade's dominance of Cloud-based system design and architecture. Now, instead of centralizing the computation and coordination to remote services, these are deployed and distributed to all over physical surroundings and network nodes, including cloud services, smart gateways, and network edge devices. At the moment, the majority of the Internet of Things (IoT) systems and software has is build on top of open Web-based technologies. We assume that with the ever-growing number and heterogeneity of connected devices, it becomes ever-more crucial to have open standards that support interoperability and enable interactions. In this chapter, we review the current technological space for architecting Web technology based IoT software in the coming era of Fog Computing. We focus on fundamental research challenges and discuss the emerging issues.

The chapter is structured as follows. In Paragraph 2.1 we describe the key enabling technologies for building Web of Things (WoT) software at the moment and in the near future. In Paragraph 2.2 we outline some motivation and research questions for the WoT research. Paragraph 2.3 introduces some research themes that are focused on WoT system and software research. In Paragraph 2.4 we discuss how the current technologies and approaches can respond to the research questions. We also discuss about the possible threats to the validity of our research. Finally, Paragraph 2.5 draws some conclusions.

2.1 The Design Space: Towards Fog Computing

WoT system architecture at the moment typically builds on a Cloud-based, centralized approach where the physical objects are connected to an Internet service, sending data there and possibly getting some actuation instructions back. (We have illustrated this kind of abstract-level example of traditional WoT architecture in Figure 2.1). It has been discussed, however, that the network is becoming the bottleneck in Cloud Computing (Shi and Dustdar, 2016), and relying on such solutions may not be fast enough for the increasing number of mission-critical applications that employ the physical objects (Dastjerdi and Buyya, 2016).

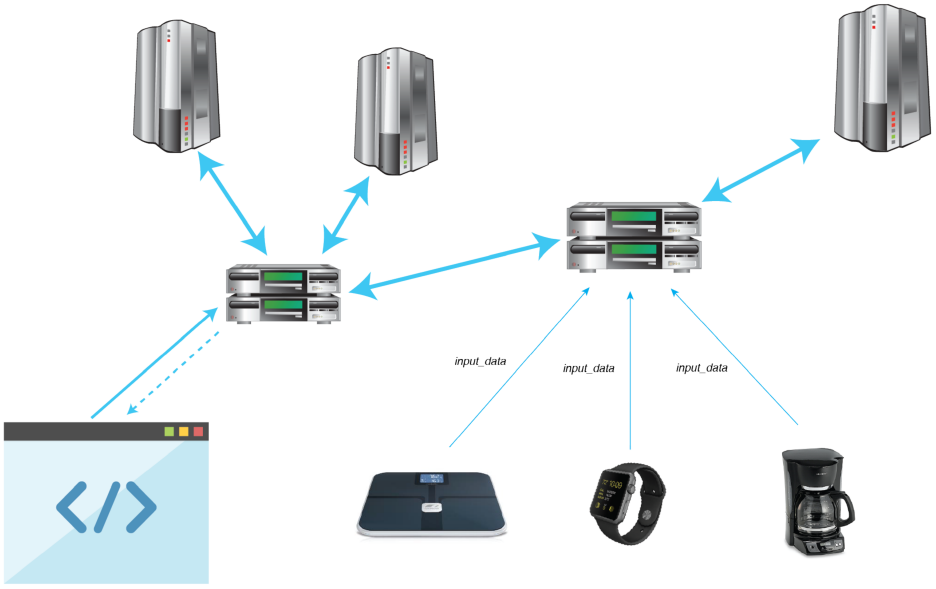


FIGURE 2.1: Generic Cloud-based Web of Things system architecture.

Due to the fast development of ICT technology smaller and more powerful chips have become cheap and can now be embedded to everywhere. In addition to the everyday physical objects to become programmable, this also drives the network to become more and more programmable with the smarter routers and eventually with 5G technologies. The following describes how some key technologies foster the transfer from moving from centralized Cloud-based WoT solutions towards Web of Things that operate everywhere on the network. In Figure 2.3 we present an illustration of an abstract WoT architecture for the Fog where the computation and coordination are distributed and can take place dynamically on various nodes.

2.1.1 Dynamic and Decentralized Computation and Coordination Infrastructures

While the traditional WoT applications appears to run in the browser, the actual computation and coordination have typically taken place on a centralized service. The top-right corner in Figure 2.2 represents these traditional approaches. One key idea of Fog Computing is to exploit the benefits the modern, dynamic computing environment offers. With non-Web based approaches, this goal becomes challenging, however. Fortunately, Web technologies foster for the leveraging of the Edge, Fog, and Cloud, since these technologies typically work everywhere, and hence support harnessing the heterogeneous computational resources. In other words, it is possible (or it is becoming possible) to partition and modularize the software since the Web-based (typically JavaScript) components can be moved and executed on the servers, edge devices as well as on many intelligent computation nodes in between the Edge and Cloud. This further fosters the emergence of the Fog Computing, as depicted in bottom-right in Figure 2.2.

In practice, however, the distinction between Fog and Edge is not always clear since the mobile devices today offer excellent support for personal area networks, namely with Bluetooth. With these, and other network gateways becoming increasingly smart and programmable, this allows bringing the intelligence to the network level. The processing of the data can be taken care by a smart gateway or by a mobile device has many benefits over the Cloud-based computation and coordination. For instance, coordination at the network edges help to reduce the communication lag and allows device coordination in situations when there is no Internet connection, or the quality of the connection is terrible. The more local coordination can also support functional safety since if one device fails to perform some operation, other devices are there to replace it.

Container Technology

One of the challenges while moving from monolith architecture towards more distributed and decentralized microservice architecture, is the management and deployment of the software constructs. This challenge is typically responded with *container technology*, with its (almost de facto) implementation Docker. Its primary objective is to make the microservices easily portable and configurable, and enable them running in isolation from the host computer.

Docker technology is not tied to any other specific programming language or implementation technology, which from the WoT perspective becomes very useful. In contrast, in a monolith architecture, it is often challenging mix other services and libraries implemented with other technologies and programming languages than the remainder of the system. The container-based services communicate with messages, and each service is independent implementation. This allows freely mixing various technologies and improves the interoperability of the different systems.

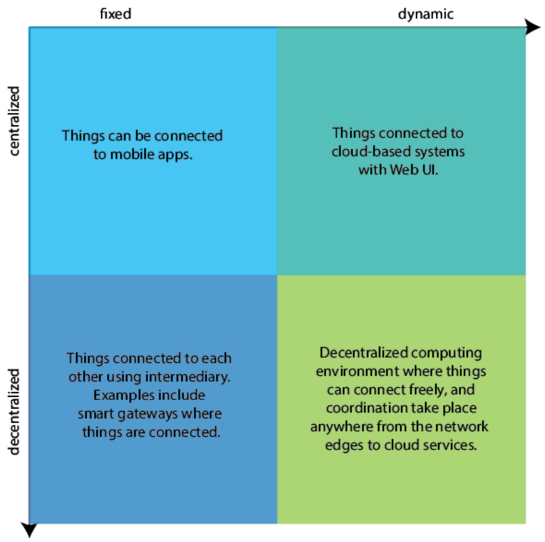


FIGURE 2.2: Web technologies support leveraging the full potential of the Fog.

Serverless Computing

Another alternative for the traditional server-based software is Serverless Computing, which the central idea is to free the software developer from maintaining a server. Instead, the developer implements functions that are deployed and executed on a Cloud service, and the costs of using this service are based on pay-per-execution. Presently the most used and well-known examples of the serverless computing include AWS Lambda, Google Cloud Functions, and Azure Functions, but others are continually emerging. As the number of Serverless Computing service providers increases, it may become challenging to decide which provider to use. Fortunately, the open source Serverless Framework [1] helps in the task by providing a homogenized interface to take benefit of these services.

From WoT development perspective, the serverless computing approaches give a solution for decentralizing the computation easily. Think for example if a physical object has the insufficient computing power, it could then invoke a method remotely. Similarly, some computations can be offloaded from the browser and performed remotely.

From Cloud Services to the Edge Devices

From WoT perspective, the development is often fostered by the emerging APIs of modern browsers since these enable more sophisticated architectures. The browser APIs may, for example, allow connecting the devices directly to the browser. Such connectivity further supports tasks required for coordinating the physical objects and performing computations close to the data sources (Taivalsaari and Mikkonen, 2017).

From WoT point of view, it is also fortunate that now approaches allowing dynamic computations have emerged. For instance, Apple provides JavaScriptCore framework for iOS, macOS, tvOS [2] or Android LiquidCore framework [3] allow the computation and coordination to be performed by some of the edge devices (Mäkitalo, Aaltonen, and Mikkonen, 2016).

Moreover, although most of the Serverless Computing approaches (discussed above) are yet targeted to run in the Cloud, some evidence exists that the Cloud providers have realized the potential of the edge devices, and now offer their solutions for Fog Computing: Amazon's Greengrass software [4] allows running AWS Lambda functions on the users' devices.It appears that the computation and coordination are again coming closer to network edges and prevents dispatching loads of useless sensor data to the Cloud which eats up bandwidth and can be costly.

[1] https://serverless.com
[2] https://developer.apple.com/documentation/javascriptcore
[3] https://github.com/LiquidPlayer/LiquidCore
[4] https://aws.amazon.com/greengrass/

2.1.2 Connectivity and Communication

Connectivity technologies can roughly be categorized into two groups: technologies that require infrastructure's support for communication, and technologies which don't require a separate intermediary technology. These both have pros and cons, as we discuss next.

Infrastructure-based Communication

By communication requiring intermediary technology for enabling relaying the messages between the communicating entities, we refer to infrastructure-based communication. The concrete infrastructure typically is either wireless or wired local area network (W/LAN), or wide area network (WAN), which is more or less a synonym for the Internet. Currently, the Internet-based communication is yet the most typical way to implement the communication for a WoT system, and the protocol used for the task is often HTTP (Hypertext Transfer Protocol). In many cases, the WoT system architecture follows REST design principles (Fielding, 2000) (although this indeed is not tied to the used communication). Other much-used protocols at the moment are MQTT (Message Queuing Telemetry Transport) and CoAP (Constrained Application Protocol) which both enable lightweight communication for more constrained devices and more direct messaging between the entities. Especially MQTT (in addition to REST) at the moment has gained particular popularity for the Internet of Things systems. However, using MQTT or CoAP require a broker for relaying the messages. Moreover, although HTTP protocol may not be the best fit anymore, in the recent 2.0 version (HTTP/2) offers support for two-way communication (M. Belshe, 2015). The downside yet is that one entity acts as the server and the others as clients, which initialize the connection.

Device-to-Device Communication

In the Fog Computing, there is a growing demand for more direct communication technologies with support for bi-directional communication between the entities. The distinction between Edge and Fog Computing is often not very clear (Bermbach et al., 2017), since the modern mobile devices offer excellent connectivity, and thus act as gateways for many peripheral devices. At the moment, for example, Personal and Body Area Networks are formed mainly with Bluetooth and Bluetooth Low Energy (Smart) technologies between the things and the mobile phone.In general, smart gateway technology use ZigBee or Z-Wave to communicate with the smart home electronics, and these gateways are then typically connected LAN and sometimes to WAN for external access. Another device-to-device connectivity technology is WiFi Direct, which allow using the existing WiFi technology to form connections between the supported devices. In practice, WiFi Direct means that one of the devices say a mobile phone, forms a group and acts as its leader, and other devices then connect to this established group.

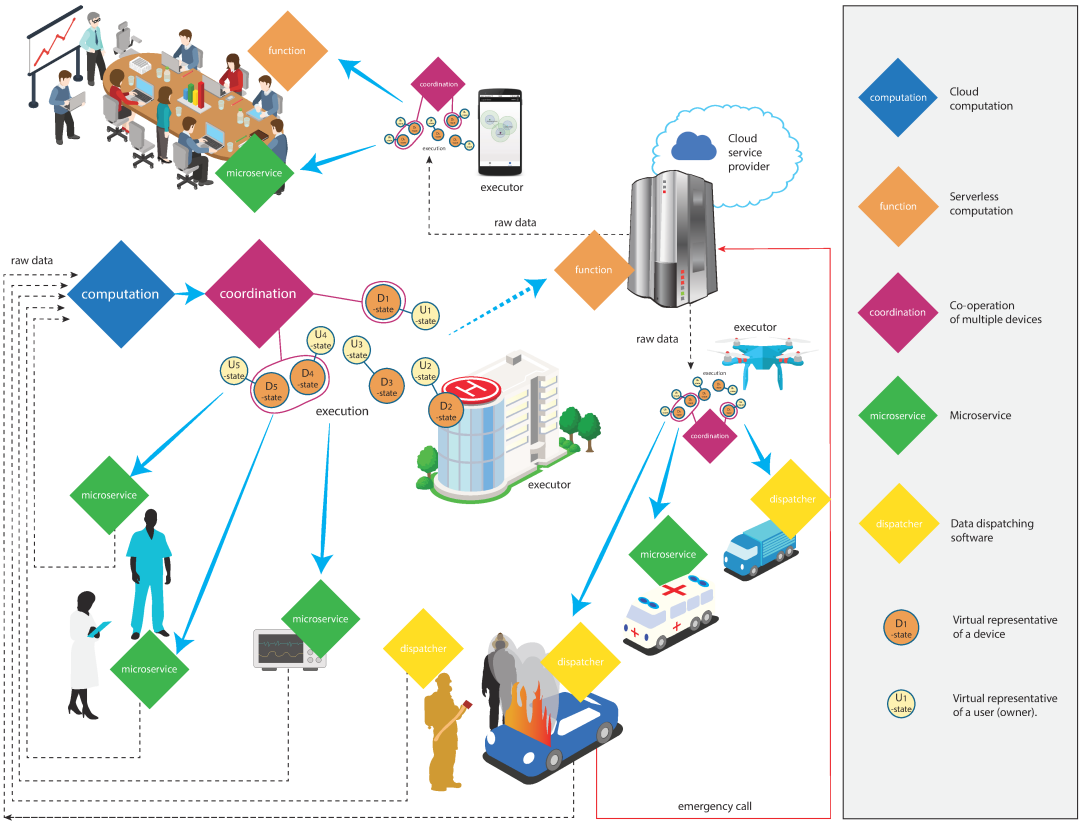


FIGURE 2.3: An illustration of open Web technology based computation and coordination in the Fog.

2.2 Motivation

The objective of this chapter is to study the future research avenues for architecting WoT software, and especially on the Fog Computing era. Our primary research question is:

What are the present and future Web of Things research themes?

We set the scope of our study and answer the primary research question with four supportive and more focused research questions. While these research questions are targeted to WoT research, they are at least partially intertwined with the IoT research. Thus, while discussing them, we also refer the two interchangeably. The primary research question is:

- RQ1: *How to use data and hardware resources for perception and interaction?*
- RQ2: *What are the current building blocks for Web of Things, and who is providing them?*
- RQ3: *Is interoperability with other systems supported, and how can this aspect be improved?*
- RQ4: *What are the current security and privacy issues, and can these threats be covered?*

In the remainder of this chapter, we use these research question for discussing the current state of the WoT research, as well as outlining some future research ideas. To answer the primary research question, we start by outlining some of the current research themes in the next paragraph. Then, we continue towards a comprehensive discussion on the other research questions (RQ1–RQ4).

2.3 Research Themes for Web of Things

WoT has been studied extensively since 2011 (Guinard et al., 2011a; Tran et al., 2017). Because WoT is an umbrella term for multi-device IoT systems that implemented with Web technologies, many research areas are closely related. Here we outline some of the areas that we find to be the most important ones in future research.

2.3.1 Liquid User Experience

Liquid Software refers to the approaches in which applications and data can seamlessly flow from one device to another, allowing the users to roam freely across all the computing devices that they have (Gallidabino et al., 2017). The goal is that users of Liquid Software do not need to worry about data copying, manual synchronization of device settings, application installation, or other burdensome device management tasks. Instead, things should work with minimal effort. From the software development perspective, Liquid Software should dynamically adapt to the almost infinite set of devices that are available to run it.

From the usability of WoT perspective, Liquid Software research studies important paradigm: How the applications can roam from one device to another, following the user from everywhere. The studies aspects include for example state management and synchronization, and various types of user interfaces (Gallidabino et al., 2017). While this paradigm is barely about beginning to work with full-fledged computing devices there is much research to be carried out in the context of constrained devices. (The most advanced is Apple's Continuity [5]).

2.3.2 Complex Event Processing

Complex Event Processing (CEP) refers to real-time analysis and filtering of large amounts of data as explained in (Luckham, 2002). The aim is at detecting meaningful events to be used either directly by the end users, or more commonly, by other systems. So-called event patterns are used for this task, which are conditions to be met for detecting the interesting situations. The data streams leveraged for the processing can, for example, come from logistics, communication networks, social networks, health and wellness services and so on. Many example CEP systems can be found from: (Cugola and

[5] http://www.apple.com/macos/continuity/

Margara, 2012). Many issues yet must be solved to make CEP applicable to commercial systems. These issues include for example fault tolerance and scalability aspects (Dayarathna and Perera, 2018; Randika et al., 2010). Also, query languages should be studied since these could help utilizing the processed events (Zhang, Diao, and Immerman, 2014). Despite of these, many new data management and optimization techniques for processing data has emerged: (Cugola and Margara, 2013; Soto et al., 2016; Mayer, Tariq, and Rothermel, 2017; Starks, Plagemann, and Kristiansen, 2017).

In the context of WoT and Fog Computing, complex events and processing them becomes increasingly important as this has genuinely potential to augment people in their lives. Using Web technologies bring both, opportunities and challenges. The opportunities include that the data can be processed – or pre-processed – closer its origin, say on an intelligent network node for instance. However, such support is limited, which brings us to the challenges. While JavaScript adoption has been vast, there yet is plenty of research to be carried on this area to process the data anywhere in the Fog efficiently: Presently, JavaScript's primary function is to enable data visualizations instead of the actual processing (e.g., with the popular D3.js library).

2.3.3 Microservice Architectural Style

The microservices architecture emphasizes dividing the system into small and lightweight services. Whether considering Microservices as a new architectural style; as an implementation of Service-Oriented Architecture (SOA) (Alshuqayran, Ali, and Evans, 2016; Francesco, Malavolta, and Lago, 2017); or as an evolution of the traditional Service-Oriented Architectural style (Lewis and Fowler., 2015), the approach is one of the most recent avenues towards more flexible installations and executions. The motivation for this transition comes from the fact that continually maintaining a complex monolithic architecture has resulted in difficulties in keeping up in pace with new development approaches such as DevOps, calling for deployment several times a day.

In (Dragoni et al., 2016) microservice architecture is defined as a distributed application where all modules are microservices. The microservices that can be implemented, tested and executed individually, help to manage the development. Some other benefits include increased agility and developer productivity, and also scalability, reliability, and maintainability of the whole system. However, the benefits come with challenges. For instance, discovering services over the network is often challenging and may introduce new weak point to the system. Also, security and privacy management often come more challenging when the architecture is highly decentralized. Other challenges include optimizing communication and performance. So finally, benchmarking and testing the system as a whole may become challenging as well. Despite the many challenges, the system can often benefit from the most advantages (Thönes, 2015; Pahl and Jamshidi, 2016).

From WoT research perspective, this transition at the moment affects to the backend of the system. However, as has been discussed, the physical objects in our surroundings are becoming more and more capable of performing computations. Thus, some of the physical objects may then run some microservices. The benefit is that deployments can be automated and made continuous. From the perspective of Fog Computing, the benefit of microservices is that the data can stay close to its source, and as a result, the lag in the communication reduces. Moreover, the privacy of the users may improve if all the data does not need to be transferred over to a service provided by an Internet company that benefits from the user-produced data.

2.4 Discussion

In this paragraph, we continue the discussion on the remaining four research questions (RQ1–RQ4). In the end of this paragraph, we discuss about threats to the validity of our study.

2.4.1 RQ1: *How to use data and hardware resources for perception and interaction?*

Physical devices (ever-more often constrained and embedded devices) are producing vast amounts of data with their sensing capabilities. Presently, this data is typically dispatched further without processing its raw form. In future, this data can be expected to act as one of the critical enablers for many WoT scenarios, for instance helping people as well as other devices to make more well-informed decisions (Mayer and Karam, 2012). However, the raw data itself does not support making these decisions, and there are mainly four reasons for this (Aggarwal, Ashish, and Sheth, 2013): *First*, the data is distributed to various locations and must be acquired someway. *Second*, the amount of data is very large (much steeper any human could ever interpret). *Third*, the data is often noisy and contains false values. *Fourth*, the data in its raw form (e.g., temperature values) is not very useful, but instead must be processed to more abstract-level information (e.g., weather conditions). Hence to leverage this data we need new tools for collecting, processing, analyzing, describing, and then, finally, for using it in the decision-making processes (Mayer and Karam, 2012).

The hardware resources can be used for interacting and giving "the output" to the physical world. In essence, this means sending some actuation instructions over the network. While now the network is typically the Internet, in Fog Computing context heterogeneous networks play the ever-more important role, giving specific benefits like reduced dependency on high-quality connection, shorter communication lag, and some fault tolerance. Like utilizing the data, also the actuation requires abstractions: For example, commanding each servo motor with an appropriate amount of power would not work. In the context of WoT, the most famous approaches come

from Guinard and Trifa in their many publications and books (Guinard and Trifa, 2016).

In essence, there are two kinds of programming models for the Fog Computing: *sense-process-actuate* models, and *stream processing* models (Dastjerdi and Buyya, 2016). Of these two, the stream processing model has been the typical one and has previously been used in many IoT and Cloud Computing approaches. In the Fog Computing context, the stream processing is still a popular model as it is more straight-forward to implement. The Complex Event Processing approaches often belong to this category, although the processing typically takes places on the edge devices (Cugola and Margara, 2013; Soto et al., 2016; Mayer, Tariq, and Rothermel, 2017; Starks, Plagemann, and Kristiansen, 2017). The sense-process-actuate model, on the other hand, often requires more high-level abstractions when used with heterogeneous devices. Until now there only has been few such models in the context of Fog Computing (Bermbach et al., 2017), and especially in the context WoT and Fog Computing. For this reason, many have simply used APIs for the task, which makes the programming less effective and affects to the maintainability aspect as well.

We believe that higher abstraction models will start to emerge sooner than later, which is indicated by the recent developments in Fog Computing. Mahmud et al. for example, discusses challenges regarding structural, service and security-related issues (Mahmud, Kotagiri, and Buyya, 2018). Also, a taxonomy of Fog Computing classifies and analyses the existing works based on their approaches towards addressing the challenges, proposing some promising research directions to pursue in the future. Also Wen et al. provide an overview of the core issues, challenges and future research directions in Fog-enabled orchestration for IoT services in (Wen et al., 2017). Additionally, they also present experiences of an orchestration scenario as a workflow across all layers of Fog architecture. The reported experiences are based on their own Fog Orchestrator.

The work (Byers, 2017) discusses some of the more significant architectural requirements for the critical IoT networks in the context of exemplary use cases, and how Fog Computing techniques can help fulfill them. A wide variety of potential IoT use cases, serving many critical vertical market switch several selected use cases in each market that have requirements to which Fog techniques are potentially applicable, such as agriculture, health, transportation, smart cities/smart buildings, and so on. The work provides useful guidelines to make more informed decisions in the architecture, partitioning, design, and deployment of Fog Computing in IoT networks.

The work (Rahmani et al., 2018) present a Fog-assisted system architecture capable of coping with many challenges in ubiquitous healthcare systems such as mobility, energy efficiency, scalability, and reliability issues. According to the Rahmani et al., successful implementation of Smart e-Health Gateways can enable massive deployment of ubiquitous health monitoring systems, especially in clinical environments. The work presents a prototype of a Smart e-Health Gateway called UT-GATE and implements an IoT-based

Early Warning Score (EWS) health monitoring to practically show the efficiency and relevance of the system on addressing a medical case study.

An example from some of the authors of this paper for building concrete Fog Computing applications was released lately. (Mäkitalo et al., 2018) presents Action-Oriented Programming model (AcOP) for developing and coordinating interactions between entities operating in the Fog. In addition, in the paper, the pain points of the traditional mobile app and Cloud Computing development in contrast to Fog Computing are highlighted.

2.4.2 RQ2: *What are the current building blocks for Web of Things, and who is providing them?*

At the moment, the world is full of closed and open APIs (Taivalsaari and Mikkonen, 2017), and these are most basic building blocks of the Web of Things development as well. Nearly all the new IoT devices have an API for communicating and accessing it via cloud service. In some case also the gateways offer an API that can be used directly communicating with the IoT devices when operating inside the same local area network (LAN). These APIs typically follow at least some of the REST design principles (Fielding, 2000). Other building blocks are various protocols (e.g., HTTP and WebSocket) that then enable the actual communication, as was described above. Moreover, in addition to the things offering their APIs via Cloud services, some things also embed Web servers in them (Guinard, Trifa, and Wilde, 2010; Fielding and Taylor, 2002) and then provide APIs. However, this approach has been changing after introducing new two-way communication protocols which are typically more light-weight and work even when the topology of the network changes or the device loses its IP address. For the user interface, the current responsive HTML/CSS frameworks (Bootstrap and Foundation) together with the front-end JavaScript frameworks (e.g., React.js, Angular.js, Vue.js) are great building blocks. For other more constrained devices, however, the building block for user interfaces are yet very limited, and for this reason, we need new ways for implementing user interfaces.

From Fog Computing perspective, Docker technology is one of the most critical building blocks. It allows building microservices that then can be deployed and maintained in various locations, often in automated ways. Other technologies that help to leverage the Fog Computing's potential are the current serverless computing frameworks (e.g., AWS Lambda, Google Cloud Functions, etc.) that can be used in a homogenized way with the open source Serverless Framework.

Despite these current building blocks, the development is at the moment is yet forced to focus a lot on the connection and communication-related aspects, rather than the actual application logic. The actual development tools (e.g., Node-RED, Meshblu, etc.) poorly fit the Fog Computing or are such research-oriented that their long-term support cannot be expected. Fortunately, while looking at the numbers of published papers, the IoT and WoT development are timely topics in the research community, and new approaches can be expected to emerge.

At some point, it may even become possible to leverage the existing social structures for sharing building blocks and open APIs, as some researchers have been proposing (Guinard et al., 2011b). Also, posting one's creations to social networks (e.g., Facebook, Linkedin, Twitter, etc.) which share them automatically with trusted or otherwise relevant people sounds promising idea. This would also mean that we would not need to create the social networks for the sharing from scratch. Sharing could also enable advertising one's skills (e.g., a student seeking first job). In social media, however, some things go viral in these networks, meaning they get an enormous amount of popularity, in the good as well as in the bad sense.

Although sharing one's creations or sensor data directly with friends sounds like a great idea, we, however, think that yet we are pretty far from this, as there barely is any programming models or tools (in the context of Fog Computing and WoT). Moreover, the danger also is that this would introduce too many new security and privacy threats, as this is the case already with the things created by the industry as we discuss later.

2.4.3 RQ3: *Is interoperability with other systems supported, and how can this aspect be improved?*

Often standardization is considered to be the key to interoperability. The World Wide Web is one of the most successful technologies in interoperability-wise, and now it is going towards harnessing physical objects to augment us in our daily lives. For this reason, it is vital that the World Wide Web Consortium (W3C) is drafting an abstract WoT architecture (Kajimoto, Kovatsch, and Davuluru, 2017). In the draft, they mention the objective of this work to be that the WoT is intended to enable interoperability across IoT Platforms and application domains. The fundamental idea is to maximize using the existing and emerging tools to be used on for building new IoT scenarios.

As part of the same work, W3C is also drafting some other standards: WoT Thing Description (Kaebisch and Kamiya, 2017), WoT Binding Templates (Koster, 2017b), and WoT Scripting API (Kis et al., 2017). Of these, the WoT Thing Description is the primary building block, which role is to describe an interface of a thing (WoT Interface) so that other things can then interact with other services and things. The role of WoT Binding Template is to enable binding the interface with multiple protocols. A thing may use WoT Scripting API internally, which means that the application logic can be done using JavaScript. The above is supposed to simplify the development significantly and enable moving the developed components fluidly. In Fog Computing this would become very useful and could be used in various use cases.

The work by W3C has a lot in common with Evrythng's Web Thing Model approach. While Evrythng's approach has been developed already since 2014, the W3C member submission was submitted on 2017 (Martins, Mazayev, and Correia, 2017; Trifa, Guinard, and Carrera, 2017). The W3C approach can be considered as one of the leading approaches. However, there are a number of other hypermedia API-level abstractions for constructing

WoT applications and for improving the interoperability that come from consortiums as well as from individual authors. These with include: JSON Hypertext Application Language (JSON-HAL) (Kelly, 2016), Media Types for Hypertext Sensor Markup (HSML) (Koster, 2017a), Constrained RESTful Application Language (CoRAL) (Hartke, 2017) and Web Thing API by Mozilla Francis, 2018.

One of the main purposes of the all above approaches (except JSON HAL) is to offer bindings for different protocols to support programming interactions between the things. However, considering how intuitive it is to use the approach by developer it is also important to offer high-level architecture definitions. From the above approaches, Mozilla's Web Thing API, Evrythng's Web Thing Model, and W3C approach are the only ones that offer such definitions. Mozilla's Web Thing API is on of the most recent approaches and has a lot in common with the Evrythng's Web Thing Model approach. The fact that this approach comes from a company, may however, be a downside since other companies may not be willing to follow this approach. On the other hand, it is also a positive thing that the approach comes from a big company with with a long tradition of open-source publishing their works is good thing since it supports continuity. For example, some of the the approaches coming from small groups or individual authors (JSON-HAL (Kelly, 2016) and HSML (Koster, 2017a) seem to have expired for now. Nevertheless, all these attempts for standardization are important since these highlight the importance of the standardization work for Web of Things. A more in-depth comparison of these APIs has been conducted in (Martins, Mazayev, and Correia, 2017).

At present, the IoT has a strong focus on establishing connectivity between a variety of constrained devices and services. Therefore, the next logical goal is to build on top of this connectivity and begin focusing on the application layer. Thus, in contrast to the IoT approaches, it would be great if the standardization work by W3C serves its purpose as this would enable considering the devices as first-class citizens of the Web. If the developers would have clear abstractions and they could consider that the connection between the things established, this would allow the developers to focus on building the applications.

We believe that at large, the WoT has all the potential to materialize into an open ecosystem of digitally augmented physical objects and new experiences which genuinely can help people in their lives. At this point, we are not there yet, and plenty of research must be conducted on improving the interoperability between the things, as well as between cloud services.

2.4.4 RQ4: *What are the current security and privacy issues, and can these threats be covered?*

WoT involves numerous heterogeneous entities interacting with each other. Given the enormous number of connected devices that are potentially vulnerable, security and privacy protection became extremely necessary (El Jaouhari,

Bouabdallah, and Bonnin, 2017). In fact, poorly secured interconnected (malicious) IoT devices could serve as entry points for cyber attacks towards more critical targets. In this paragraph, we give some considerations on current IoT technology and related security breach and solutions (if any). Source of the following discussion comes from some recent state of the art survey in the area (Mendez, Papapanagiotou, and Yang, 2017; Yang et al., 2017; Alaba et al., 2017; Granjal, Monteiro, and Silva, 2015; Fremantle and Scott, 2017) as well as security issues analysis of the Web of Things (El Jaouhari, Bouabdallah, and Bonnin, 2017), where a more through discussion can be found.

The attacker model for IoT architecture is described in (Atamli and Martin, 2014) where the attacker can be a malicious user, a bad manufacturer, or an external adversary:

- The malicious user is the owner of the IoT device with the potential to perform attacks to learn the secrets of the manufacturer and gain access to restricted functionality. By uncovering the flaws in the system, the malicious user can obtain information, sell secrets to third parties, or even attack similar systems.

- The bad manufacturer is the producer of the device with the ability to exploit the technology to gain information about the users, or other IoT devices. Such a manufacturer can deliberately introduce security holes in its design to be exploited in the future for accessing the user's data and exposing it to third parties. Equally, the production of poorly secured goods results in compromising the users' privacy. Besides, in IoT context, where different objects are connected to each other, a manufacturer can attack other competitors' devices to harm their reputation.

- The external adversary is an outside entity that is not part of the system and has no authorized access to it. An adversary would try to gain information about the user of the system for malicious purposes such as causing financial damage and undermining the user's credibility.

Notice that this is indeed different from classic Dolev-Yaho model (Dolev and Yao, 1983), where the adversary can overhear, intercept, and synthesize any message and is only limited by the constraints of the cryptographic methods used. In other words: "the attacker carries the message" and has a sort of "omnipotence" not easy to implement and verify, and usually considered diminished.

The work (Dragoni, Giaretta, and Mazzara, 2017) reports an in-depth study of possible weakness and their exploitation on IoT devices. In particular, a study of HP conducted some years ago (*Internet of things research study* 2015), analyzed 10 of the most popular IoT devices on the market and revealed a generalized poor security level of the majority of them: most of the devices showed privacy and confidential information leakage; Two third of them used too low authentication requirements, and not strong enough password used; only some of them used encrypted network services, and suffer from XSS weakness.

The most common and easily addressable security issues of Internet of Things devices reported by HP in 2015 include (*Internet of things research study* 2015; El Jaouhari, Bouabdallah, and Bonnin, 2017):

- Privacy concerns: The study reports that 80% of the devices were leaking private information.
- Insufficient authorization: According to the study 80% of the tested devices were not protected by a proper password.
- Lack of transport encryption: According to the study 70% of the tested devices did not encrypt communication.
- Insecure Web interface: According to the study 60% of the tested devices had security concerns in their Web-based user interfaces, and 70% of the systems behind the devices allowed resolving the users' accounts.
- Inadequate software protection: The study reports that 60% of the devices were not securing their software updates with encryption.

In (Zhao and Ge, 2013) the IoT architecture is composed of three layers (perception, network and application). We follow such view and we will analyze security issue in each of the levels.

Perception Layer

The perception layer is strictly connected to the technology used for the communication. Wireless Sensor Network (WSN) is the general term to classify all such (mesh) connected devices, from centimeters to several meters of distances. When dealing with low distance technology and protocol (such as Near Field Communication (NFC) or wireless network of wearable devices (Body Area Network (BAN))) built following 802.15.6 standards, security is inferior. Indeed, NFC suffers from many threats (Denial of Service, and information leakage) (Madlmayr et al., 2008). Moreover, since for backward compatibility reason with RFID it is not encrypted, it suffers from many man-in-the-middle attacks, using antennas or skimmers to intercept the signals.

WSN also refers to protocols connected to the 802.15.4 standard, whose major implementation are ZigBee and 6LowPan (and maybe someway still Bluetooth). Bluetooth is indeed the oldest of the three technologies. Despite it is currently adopted for indoor application with iBeacons, and the particular care of energy consumption in the Low Energy (BLE) version (Zafari and Papapanagiotou, 2015; Zafari et al., 2017), the technology will be probably substituted for security reasons soon by the more advanced implementation of Zigbee and 6LowPan. In fact, despite BLE uses AES-128 CCM for encryption and authentication purposes, it still suffers from many vulnerabilities, and at today many of the countermeasures rely upon user security problem awareness.

ZigBee represents a new protocol for WSN and the main used implementation of the IEEE standard 802.15.4. Differently, from Bluetooth, the ZigBee

protocol came natively with security features and management of both long-term and session keys. The long-term (Master Key) is part of the factory-setting, while all the devices share the session (network) keys on the network. ZigBee uses AES-128 encryption as the default, however, since often some trade-offs between security and power consumption (Boyle and Newe, 2008) and some threats such as traffic sniffing (eavesdropping), packet decoding, and data manipulation/injection could be possible.

6LowPan (that stand for IPv6 Low Power Personal Area Networks) is the newest WSN standard. Its main innovation is the use of an IPv6 address for each sensor in the mesh. The use of IPv6 addressing gives to 6LowPan universality, extensibility and stability (Sheng et al., 2013), and wrt IPv6 small packet size and low bandwidth. Unfortunately, devices that support 6LowPan are still resource consuming and this is the primary challenge the solve in the future for global adoption of 6lowPan. However, from the security point of view, 6LowPan implements Elliptic Curve Cryptography (ECC) encryption algorithm that has smaller-packet sizes w.r.t. RSA.

Middleware and Application Layers

The middleware layer in IoT contains a vast number of proposals, each of them with their pros and cons. Indeed, no real standard is used in this layer because all the vendors propose their solutions. Describing numerous solutions and their security concerns would result in a not complete description, and thus we suggest to the reader to start from the top survey (Razzaque et al., 2016a).

The application layer, on the contrary, is today enough standardized and the majority of the implementations use the Message Queue Telemetry Transport (MQTT) protocol. The MQTT protocol was proposed in (OASIS Message Queuing Telemetry Transport (MQTT) TC, 2014) as a light-weight protocol designed for constrained devices and low-bandwidth. At today MQTT is an OASIS and ISO/IEC JTC1 standard (technology, 2016). The OASIS MQTT TC is producing a standard for the Message Queuing Telemetry Transport Protocol compatible with MQTT V3.1, together with requirements for enhancements, documented usage examples, best practices, and guidance for the use of MQTT topics with a commonly available registry and discovery mechanisms. The standard supports bi-directional messaging to uniformly handle both signals and commands, deterministic message delivery, necessary QoS levels, always/sometimes-connected scenarios, loose coupling, and scalability to support large numbers of devices. Candidates for enhancements include message priority and expiry, message payload typing, request/reply, and subscription expiry. As an IoT connectivity protocol, MQTT is designed to support messaging transport from remote locations/devices involving small code footprints (e.g., 8-bit, 256KB ram controllers), low power, low bandwidth, high-cost connections, high latency, variable availability, and negotiated delivery guarantees.

However, the present implementation of MQTT provides support for only identity, authentication and authorization policies. Identity specifies the client

that is being authorized. Authentication provides the identity of the client and authorization is the management of rights given to the client.
The primary approaches used to support these policies are by using a username/password pair, which is set by the client, for identification or by authentication performed by the MQTT server via client certificate validation through the SSL protocol. The MQTT server identifies itself with its IP address and digital certificate. Its communication uses TCP as transport layer protocol. By itself, the MQTT protocol does not provide encrypted communication. Authorization is also not part of MQTT protocol but can be provided by the servers. MQTT authorization rules control which client can connect to the server and what topics a client can subscribe to or publish.

In addition to investigating and discussing other security and privacy challenges of introducing Fog Computing in IoT environments, (Alrawais et al., 2017) necessitate that the Fog Computing research should be focused on how to overcome the challenges related to authentication in the context of Fog Computing in IoT applications since this is far from trivial in such decentralized environment. Also Zhang et al. discuss about similar security and privacy threats towards IoT applications and discuss the security and privacy requirements in Fog Computing in (Ni et al., 2017). Further, they demonstrate potential challenges to secure Fog Computing and review the state-of-the-art solutions used to address security and privacy issues in Fog Computing for IoT applications and define several open research issues. In a recent paper, a possible solution in (Mäkitalo et al., 2018) by forming trusted coalitions of devices (Ometov et al., 2016) is presented. The approach is connectivity agnostic, but requires a library support for the leveraged device which may limit its usage in WoT environment.

2.4.5 Threats to Validity

We discuss threats to the validity of this work in the different steps of our study.

Threats to the Identification of Primary Studies

In this chapter, we assume that the emerging Fog Computing will eventually enclose Cloud Computing. Hence, we reviewed some of the most recent Fog Computing articles and mapped the key enabling technologies that were mentioned to foster the paradigm shift. The microservice architectural style and communication-related aspects were then present in most articles, and thus formed the core of our study and were used as search terms for finding papers. The importance of data and its analysis was also a significant theme among Fog Computing and Web of Things papers, and for this reason, complex event processing was used as a search term. The IoT and WoT are expected to increase privacy and security threats dramatically, and thus we decided to focus primarily on this aspect while selecting the studies. While this protocol helped us to focus on selecting the major themes for our study, we naturally were not able to include all the minor themes.

Threats to Selection and Data Extraction Consistency

We formulated the research questions RQ1–RQ4 which helped us to select the relevant papers for our study. Naturally, new studies are being published all the time, especially related to security and privacy issues, Fog Computing, and microservices since these topics are very relevant at the moment. We also intentionally excluded thesis as well as some older papers from our study.

Threats to Data Synthesis and Results

We tried to mitigate the threat with a standard protocol of several steps, by piloting and by externally evaluating our process by professors that were not participating in making our study.

2.5 Conclusions

In this chapter, we focused on describing the current technical design space for WoT, focusing on the emerging Fog Computing paradigm. We went through the critical enabling technologies that have gained particular popularity among the developers, and which form the basis of building WoT systems and software. Afterwards, we defined research questions of WoT research and outlined motivation for future research in this area. We described some research areas trying to solve our outlined research challenges. Then, we continued the discussion on the retrospective.

The outcome of this work is that WoT research should take a new path, moving from centralized RESTful and Cloud service based approaches towards more decentralized approaches, and aim at leveraging the full potential of the dynamic and modern computing environment, reaching from the edge devices and network nodes to the Clouds. As the modern computing environment is heterogeneous, the Web-based technologies give a good base and support the fluidity required by the Fog Computing. Like all technologies, also Web technologies have their flaws which mainly are related to their limited support in some hardware platforms and some limitations in accessing the hardware resources. Nevertheless, due to the openness and support for heterogeneity, Web technologies in the context of the Internet of Things have earned their place, and thus will continue growing their popularity among the IoT development.

Chapter 3

A Fuzzy Ontology-based Tool for Decision Making in Architectural Design

3.1 Introduction and motivation

In software development the main goals to accomplish are customer and quality requirements satisfaction, correct execution of the software systems, and cost effective adaptation to future changes (Avgeriou et al., 2011). In order to reach these challenging goals, tools and techniques may help to manage and retrieve the knowledge necessary for decision making processes. For this reason, recent research trends are focused to strengthen the reasoning and decision-making process to reach these goals (Avgeriou et al., 2011).

The development of a software system is in fact determined partly by its functionality i.e., what the system does - and partly by requirements about quality attributes or about development (Chung et al., 2012). Such requirements are known as Non-Functional Requirements (NFRs). During architectural design the selection of NFRs is a relevant task, since can be used as selection criteria for choosing the proper design solution among several ones. On the other hand, exploiting design decisions during development emerges as a Software Architecture (Bass, Clements, and Kazman, 2005; Shaw and Garlan, 1996; Garlan and Shaw, 1994; Di Noia, Mongiello, and Di Sciascio, 2014). The software architecture is the result of the work of an architect or of a designers team (Bass, Clements, and Kazman, 2005). The architectural design of a software system is made up of a set of selected quality attributes; in this sense, the architecture can be modeled by taking into account the needed quality attributes. To reach this goal, an architect can use primitive design techniques. These primitives are known as tactics. Generally a tactic can be considered as a modeling solution and is related to satisfying a given quality attribute. An architectural strategy is made up of several tactics. The architect takes design decision based on a strategy, hence selects a set of tactics, and hence a set of patterns.

The design process requires a choice of the best combination of tactics to achieve the system's desired goals (Bass, Clements, and Kazman, 2005). For this reason during the design of Middleware-induced Software Architecture a challenging tasks are the selection of *(i)* best patterns and *(ii)* best existing

Middleware able to satisfy desired requirements (Egyed and Grunbacher, 2004; Mairiza, Zowghi, and Nurmuliani, 2009; Razzaque et al., 2016b). A main difficult derives from the relationship between requirements and patterns that can be complementary, can be composed, sometimes cooperate to solve a larger problem or are exclusive in a modeling task (Buschmann, Henney, and Schmidt, 2007a). Existing classifications or quality models have been defined to categorize QAs and requirements, anyway a systematic classification of NFRs towards architectural design and a description of their use during system modeling are missing (N.Harrison, 2007; Avgeriou and Zdun, 2005). Moreover, only little work has been done in using a knowledge-based approach to support such activities (Li, Liang, and Avgeriou, 2013).

The main objective of this research is the *implementation of a semi-automated tool aimed at supporting decision makers to derive knowledge able to solve architectural problems.*

More precisely we develop a Decision Support System for supporting designers and software architect – having greater or lesser experience – in the architectural design of Middleware-induced Software Architecture. Specifically, we will show how the following task can be addressed: *given a desired set of FRs, NFRs and ARs perform the tasks of (i) retrieving the smallest subset of patterns that best match them and (ii) recommending existing Middleware can support all the necessary requirements*.

To achieve this goal, we will define a theoretical framework to model and reason on the knowledge about requirements and reusable schema for design in order to obtain a model of the system satisfying given requirements, quality attributes optimized with reusable and composable design schema.

Toward this goal, the following research questions will be addressed:

Q1. What are the main categories of requirements a software may be asked to fulfill? Q2. What are the main reusable solving schema for given classes (or families) of design problems? Q3. How do categories of requirements, reusable schema and classes of problems are related? Q4. How to facilitate the decision making process for middleware-induced software design modeling by using relations among these categories of elements? Q5. How to use this knowledge to support modeling during software design?

To achieve the main objective of this research and answer the research questions, the following research process has been taken: 1. Propose a theoretical framework for modeling knowledge about NFRs,FRs, ARs, middlewares, design patterns, and define a reasoning algorithm to manipulate the modeled knowledge; 2. Build a prototype system to implement the theoretical framework; 3. Validate the prototype with use cases; 4. Compare solutions provided by the decision support system with human proposed solutions to software design problems.

The framework we propose to represent and reason about requirements is based on *Fuzzy Description Logics* (FDLs) (we refer the reader to (Straccia, 2013) for a description about FDLs), which are the theoretical counterpart of *Fuzzy OWL 2* (Bobillo and Straccia, 2011), the main formalism to specify fuzzy ontologies. We recall that FDLs are logical languages specifically tailored to model structured information about vague concepts that naturally

occur in NFRs. For instance, in our context, FDLs allow one to model that "*portability and adaptability are directly proportionate*", "*stability and adaptability are inversely proportionate*" (ontological knowledge) or that "*the Adapter pattern has high portability*" (factual knowledge). New knowledge about a pattern is obtained by combining existing knowledge, for example ontological or factual through a reasoning task. For instance it can be inferred that "*the Adapter pattern has high adaptability and low stability*". Another type of expression allowed in our framework is "*high adaptability* implies a *medium maintainability*". Let us note that in the previous statements, we can use fuzzy sets (Zadeh, 1965)to characterize concepts like *high*, *medium* and *low*. We would like to stress the point that a formal encoding of the knowledge about patterns, requirements, family of patterns and so on is particularly useful to automatize the task of selecting a set of patterns that encounters user's needs. Specifically, we will show how the following task can be addressed: *given a desired set of FRs, NFRs and ARs perform the tasks of (i) retrieving the smallest subset of patterns that best match them and (ii) recommending existing middleware can support all the necessary requirements*. Eventually, in order to make the approach feasible, we collected the knowledge about 109 design patterns, 28 pattern families, 37 NFRs, 61 existing Middleware in literature grouped based on their 7 design approaches and its 14 NFRs, 5 FRs, 8 ARs together with their mutual relations and represented them as a Fuzzy OWL 2 ontology and show its application in a use case scenario.[1]

In summary, this chapter provides the following contributions:

- we provide a method to model and reason with mutual relations among requirements in architectural software patterns by relying on fuzzy ontologies;
- the definition of a novel formal reasoning task being able to retrieve a set of patterns/middlewares that maximally match a user query expressed in terms of desired requirements;
- the construction of a Fuzzy OWL 2 ontology;
- a use case scenario application to validate the method;
- an implementation of the theoretical framework in a decision support system.

No other method similar to the proposed one was found. This chapter extend MD thesis where the approach by using Fuzzy DL is proposed and the knowledge limited to NFRs and design pattern is collected. In the following, we proceed as follows. We start by recalling basic definitions about ontologies and FDLs. Then, we proceed with main properties of design patterns and quality models used to model NFRs. In Paragraph 3.4, we state the addressed problem and the proposed approach. In Paragraph 3.5 we describe a

[1]The ontology is available online at http://sisinflab.poliba.it/semanticweb/ontologies/mosaic/.

case study. In Paragraph 3.7 the implemented decision support system is described. In Paragraph 3.8 we validate the approach and in Paragraph 3.9 we address related work about existing (ontological) approaches for Knowledge representation in software engineering, design patterns and NFRs modelling. Conclusions close the chapter.

3.2 Fuzzy Description Logics

Ontologies play a key role in the success of the *Semantic Web* (Berners-Lee, Hendler, and Lassila, 2001) since they are the recognized knowledge representation formalism for specifying domain knowledge and for sharing and reusing this knowledge. *Description Logics* (DLs) (Baader et al., 2003) are at the core of the Semantic Web ontology description language OWL 2 (Cuenca-Grau et al., 2008). In fact, OWL 2 is based on a specific DL named $\mathcal{SROIQ}(\mathbf{D})$ (Horrocks, Kutz, and Sattler, 2006). In recent times, it has been noted that classical ontologies and its languages are not appropriate to deal with vagueness and imprecise knowledge, which is fundamental to several real world domains (Sánchez and Tettamanzi, 2006). To handle this problem, the use of fuzzy logics with ontology offers a solution. Fuzzy ontologies and its description logics for the semantic web can handle probabilistic or possibilistic uncertainties and vagueness. Research on fuzzy ontologies began in the early 2000's with the focus on simplistic models used for improve an Information Retrieval System (Calegari and Sanchez, 2007). *Fuzzy DLs* (FDLs) are an extension of DLs to deal with *fuzzy* concepts (see (Straccia, 2013; Bobillo et al., 2015) for an overview). In these logics, satisfactions of axioms is based on a degree of truth that generally is a value in $[0, 1]$.

3.2.1 Recap of Fuzzy Description Logics Basics

The fuzzy DL we are considering here is called $\mathcal{ALCB}(\mathbf{D})$, whose expressiveness is enough to illustrate our approach. Our framework extends to more expressive fuzzy DLs easily. We recall that $\mathcal{ALCB}(\mathbf{D})$ is a $\mathcal{ALC}$, in which the letter $\mathcal{B}$ represents the *individual value restrictions* that are restricted kind of nominals, and the letter $\mathbf{D}$ indicates the *fuzzy concrete domains* (Straccia, 2005).

For making the chapter self-contained, we recap here some basic definitions about fuzzy sets and the fuzzy DL $\mathcal{ALCB}(\mathbf{D})$.

Fuzzy Sets

Let X be a countable crisp set, i.e. a set in which an element is a either member of the set or not. A *fuzzy set* (Zadeh, 1965) A over X is characterized by a function $A\colon X \to [0, 1]$, called *fuzzy membership* function. Typical operations on fuzzy sets are defined as: Intersection $(A \cap B)(x) = \min(A(x), B(x))$, Union $(A \cup B)(x) = \max(A(x), B(x))$ and Complement $\bar{A}(x) = 1 - A(x)$. Figure 3.1 illustrates some frequently used membership functions for specifying fuzzy sets.

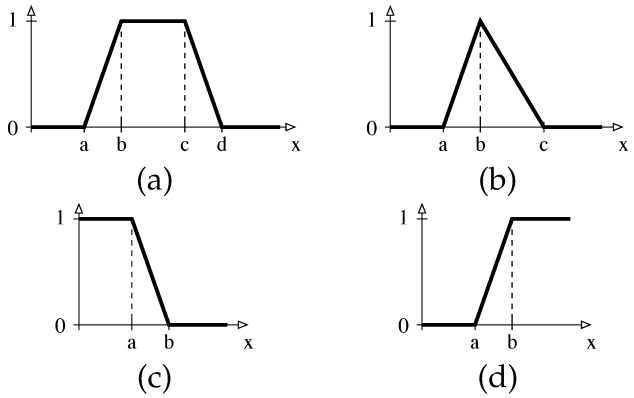


FIGURE 3.1: (a) Trapezoidal function $trz(a,b,c,d)$, (b) triangular function $tri(a,b,c)$, (c) left shoulder function $ls(a,b)$, and (d) right shoulder function $rs(a,b)$.

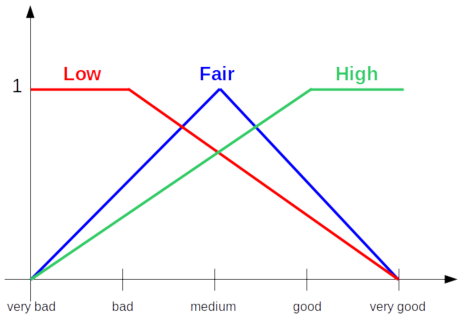


FIGURE 3.2: The fuzzy sets we use to deal with Non-Functional Requirements.

The Fuzzy DL $\mathcal{ALCB}(\mathbf{D})$

At first, let us recap the notion of *fuzzy concrete domain* (Straccia, 2005) that is a tuple $\mathbf{D} = \langle \Delta^{\mathbf{D}}, \cdot^{\mathbf{D}} \rangle$ where $\Delta^{\mathbf{D}}$ is the data type domain and $\cdot^{\mathbf{D}}$ is a mapping that assigns an element of $\Delta^{\mathbf{D}}$ to each data value, and a 1-ary fuzzy relation over $\Delta^{\mathbf{D}}$ to every 1-ary data type predicate $\mathbf{d}$. Hence the mapping $\cdot^{\mathbf{D}}$ relates each data type predicate to a function from the domain $\Delta^{\mathbf{D}}$ to $[0,1]$. The data type predicates $\mathbf{d}$ we are considering here are the membership functions shown in Figure 3.1: trapezoidal $trz(a,b,c,d)$, triangular $tri(a,b,c)$, left-shoulder $ls(a,b)$ and right-shoulder $rs(a,b)$:

$$\begin{aligned} \mathbf{d} \;\rightarrow\; & ls(x,y) \mid rs(x,y) \mid tri(x,y,z) \mid trz(x,y,z,t) \\ & \mid \geq_v \mid \leq_v \mid =_v \mid \in_V , \end{aligned}$$

and $v \in \Delta^{\mathbf{D}}$ and $V \subseteq \Delta^{\mathbf{D}}$. To what concerns our framework, we will consider the following fuzzy concrete domain as illustrated in Figure 3.2. The values/ratings in

$$\{\texttt{verybad}, \texttt{bad}, \texttt{medium}, \texttt{good}, \texttt{verygood}\}$$

are macros for some predefined numerical values.

Now, let us consider the following notations: the set of atomic concepts or *concept names* is denoted as **A**, the set of *role names* denoted as **R**, the set of *individual names* denoted as **I**. We assume that a role can be a *data type property* or an *object property*. Using DL constructors, *concepts* are built from concept names A, object properties R and data type properties S. The syntactic rule used for concept construction is the following:

$$\begin{aligned} C \;\rightarrow\; & \top \mid \bot \mid A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid C_1 \rightarrow C_2 \\ & \mid \exists R.C \mid \forall R.C \mid \exists S.\mathbf{d} \mid \forall S.\mathbf{d} \mid \exists R.\{a\} \; . \end{aligned}$$

Concepts of the form $\{a\}$, with $a \in \mathbf{I}$, are called *nominals*. In $\mathcal{ALCB}(\mathbf{D})$, they can only appear in concepts of the form $\exists R.\{a\}$.

A *Fuzzy Knowledge Base* (or *fuzzy Ontology*) $\mathcal{K} = \langle \mathcal{A}, \mathcal{T} \rangle$ contains a fuzzy ABox $\mathcal{A}$ with axioms about individuals and a fuzzy TBox $\mathcal{T}$ with axioms about concepts.

A *fuzzy ABox* contains a finite set of *fuzzy assertions* of the following types:
(i) *Concept assertions* of the form $\langle a{:}C, \alpha \rangle$, with $\alpha \in (0, 1]$ and stating that individual a is an instance of concept C with degree greater than or equal to α;
(ii) *Role assertions* of the form $\langle (a, b){:}R, \alpha \rangle$, $\alpha \in (0, 1]$, meaning that the pair of individuals (a, b) is an instance of role R with degree greater than or equal to α. A *fuzzy TBox* consists of a finite set of *fuzzy General Concept Inclusions (fuzzy GCIs)*, which are expressions of the form $\langle C_1 \sqsubseteq C_2, \alpha \rangle$, $\alpha \in (0, 1]$, meaning that the degree of concept C_1 being subsumed by C_2 is greater than or equal to α.

For the rest of the chapter we also make the following assumptions:
(i) in fuzzy assertions and GCIs, we may omit the degree α and in that case the value 1 is assumed;
(ii) we may write $\exists R$ in place of $\exists R.\top$;
(iii) we use the GCI $C \equiv D$ as a macro of both expressions $C \sqsubseteq D$ and $D \sqsubseteq C$, dictating that C and D are 'equivalent' concept expressions.

Concerning the semantics, let us fix a fuzzy logic. Unlike classical DLs, in fuzzy DLs, an interpretation $\mathcal{I}$ maps a concept C into a function $C^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow [0, 1]$ and, thus, an individual belongs to the extension of C to some degree in $[0, 1]$, i.e. $C^{\mathcal{I}}$ is a fuzzy set.

Specifically, a *fuzzy interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a nonempty (crisp) set $\Delta^{\mathcal{I}}$ (the *domain*) and of a *fuzzy interpretation function* $\cdot^{\mathcal{I}}$ that assigns:
i to each atomic concept A a function $A^{\mathcal{I}} \colon \Delta^{\mathcal{I}} \rightarrow [0, 1]$;
ii to each object property R a function $R^{\mathcal{I}} \colon \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow [0, 1]$;
iii to each data type property S a function $S^{\mathcal{I}} \colon \Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}} \rightarrow [0, 1]$;
iv to each individual a an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (Unique Name Assumption);
v to each data value v an element $v^{\mathcal{I}} \in \Delta^{\mathbf{D}}$. Now, a fuzzy interpretation

function is extended to concepts as specified below (where $x \in \Delta^{\mathcal{I}}$):

$$
\begin{aligned}
&\bot^{\mathcal{I}}(x) = 0\,,\ \top^{\mathcal{I}}(x) = 1\,,\\
&(C \sqcap D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \otimes D^{\mathcal{I}}(x)\,,\\
&(C \sqcup D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \oplus D^{\mathcal{I}}(x)\,,\\
&(\neg C)^{\mathcal{I}}(x) = \ominus C^{\mathcal{I}}(x)\,,\\
&(C \rightarrow D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \Rightarrow D^{\mathcal{I}}(x)\,,\\
&(\forall R.C)^{\mathcal{I}}(x) = \inf_{y \in \Delta^{\mathcal{I}}}\{R^{\mathcal{I}}(x,y) \Rightarrow C^{\mathcal{I}}(y)\}\,,\\
&(\exists R.C)^{\mathcal{I}}(x) = \sup_{y \in \Delta^{\mathcal{I}}}\{R^{\mathcal{I}}(x,y) \otimes C^{\mathcal{I}}(y)\}\,,\\
&(\exists R.\{a\})^{\mathcal{I}}(x) = R^{\mathcal{I}}(x, a^{\mathcal{I}})\,,\\
&(\forall S.\mathbf{d})^{\mathcal{I}}(x) = \inf_{y \in \Delta^{\mathbf{D}}}\{S^{\mathcal{I}}(x,y) \Rightarrow \mathbf{d}^{\mathbf{D}}(y)\}\,,\\
&(\exists S.\mathbf{d})^{\mathcal{I}}(x) = \sup_{y \in \Delta^{\mathbf{D}}}\{S^{\mathcal{I}}(x,y) \otimes \mathbf{d}^{\mathbf{D}}(y)\}\,.
\end{aligned}
$$

The *satisfiability of axioms* is then defined by the following conditions:
(i) $\mathcal{I}$ satisfies an axiom $\langle a{:}C, \alpha\rangle$ if $C^{\mathcal{I}}(a^{\mathcal{I}}) \geq \alpha$;
(ii) $\mathcal{I}$ satisfies an axiom $\langle (a,b){:}R, \alpha\rangle$ if $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq \alpha$;
(iii) $\mathcal{I}$ satisfies an axiom $\langle C \sqsubseteq D, \alpha\rangle$ if $(C \sqsubseteq D)^{\mathcal{I}} \geq \alpha$ where[2] $(C \sqsubseteq D)^{\mathcal{I}} = \inf_{x \in \Delta^{\mathcal{I}}}\{C^{\mathcal{I}}(x) \Rightarrow D^{\mathcal{I}}(x)\}$. $\mathcal{I}$ is a *model* of $\mathcal{K} = \langle \mathcal{A}, \mathcal{T}\rangle$ iff $\mathcal{I}$ satisfies each axiom in $\mathcal{K}$.

The most common reasoning tasks on fuzzy DLs, which are usually inter-definable (Straccia, 2013), are the following ones ($\mathcal{K}$ is a fuzzy KB):
(i) *Consistency* (or KB satisfiability). Check if $\mathcal{K}$ has a *model*;
(ii) *Fuzzy concept satisfiability*. A fuzzy concept C is α-satisfiable w.r.t. $\mathcal{K}$ iff there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C(x)^{\mathcal{I}} \geq \alpha$ for some element $x \in \Delta^{\mathcal{I}}$;
(iii) *Entailment*. A fuzzy axiom τ is a *logical consequence* of $\mathcal{K}$ (or $\mathcal{K}$ entails τ), denoted $\mathcal{K} \models \tau$, iff every model of $\mathcal{K}$ is a model of τ;
(iv) *Fuzzy concept subsumption*. C_2 α-subsumes C_1 w.r.t. $\mathcal{K}$ iff every model $\mathcal{I}$ of $\mathcal{K}$ satisfies $\forall x \in \Delta^{\mathcal{I}}, C_1^{\mathcal{I}}(x) \Rightarrow C_2^{\mathcal{I}}(x) \geq \alpha$;
(v) *Best Entailment Degree* (BED). The BED of an axiom $\phi \in \{a{:}C, (a,b){:}R, C \sqsubseteq D\}$ w.r.t. $\mathcal{K}$ is defined as $\mathtt{bed}(\mathcal{K}, \phi) = \sup\{\alpha \mid \mathcal{K} \models \langle \phi, \alpha\rangle\}$;
(vi) *Best Satisfiability Degree* (BSD). The BSD of a concept C w.r.t. $\mathcal{K}$ is defined as $\mathtt{bsd}(\mathcal{K}, C) = \sup_{\mathcal{I} \models \mathcal{K}} \sup_{x \in \Delta^{\mathcal{I}}} C^{\mathcal{I}}(x)$;
(vii) *Answer Set*. The *answer set* of a concept C w.r.t. $\mathcal{K}$ is the set $\mathtt{ans}(\mathcal{K}, C) = \{\langle a, \alpha\rangle \mid \alpha = \mathtt{bed}(\mathcal{K}, a{:}C)\}$. Each element in $\mathtt{ans}(\mathcal{K}, C)$ is called an *answer* to C w.r.t. $\mathcal{K}$;
(viii) *Top-k Answer Set*. The *top-k answer set* of C w.r.t. $\mathcal{K}$, denoted $\mathtt{ans}_k(\mathcal{K}, C)$, is the set of top-$k$ ranked answers of to C w.r.t. $\mathcal{K}$. Formally, $\mathtt{ans}_k(\mathcal{K}, C) \subseteq \mathtt{ans}(\mathcal{K}, C)$ is maximal un set inclusion, $|\mathtt{ans}_k(\mathcal{K}, C)| \leq k$, there is no $\langle a, 0\rangle \in \mathtt{ans}_k(\mathcal{K}, C)$, and there is no $\langle b, \beta\rangle \in \mathtt{ans}(\mathcal{K}, C) \setminus \mathtt{ans}_k(\mathcal{K}, C)$ with $\beta > \alpha$ for some $\langle a, \alpha\rangle \in \mathtt{ans}_k(\mathcal{K}, C)$. Each element in $\mathtt{ans}_k(\mathcal{K}, C)$ is called a *top-k answer* to C w.r.t. $\mathcal{K}$.

Eventually, fuzzy DL reasoners enable to manage fuzzy ontologies in practice (see (Bobillo and Straccia, 2016) for an overview). *fuzzyDL* is an expressive fuzzy ontology reasoner (the supported language is more expressive that the one presented here) that supports the three main semantics of

[2] However, note that under standard logic $\sqsubseteq$ is interpreted as $\Rightarrow_z$ and not as $\Rightarrow_{kd}$.

fuzzy logics: Zadeh, Łukasiewicz, and classical DLs, for ensuring compatibility with crisp ontologies.

3.3 Requirements, Design and Architectural patterns

Design and Architectural Pattern. Reusable solutions of design models are available mainly realized using patterns. These approaches are important vehicles for constructing high-quality software architectures since provide already tested solutions (Gamma et al., 1994; Buschmann et al., 1996b).

Design patterns were proposed during the last decades are reusable solutions to modeling recurrent problems. They are mainly based on the expert's experience that use solution proposed for similar problems (Gamma et al., 1994; Buschmann et al., 1996b).

Therefore, the use of patterns or tactics (N.Harrison, 2007; Bass, Klein, and Bachmann, 2002; Gross and Yu, 2001) for architectural modeling constitutes an effective solution for addressing design decisions (Harrison and Avgeriou, 2010b). They also support the construction and documentation of software architectures. In summary, patterns provide a set of predefined design schemes for software systems organization, and provide an abstract formalization of the design solution (Buschmann et al., 1996b; Henninger and Corrêa, 2007; Taylor, Medvidovic, and Dashofy, 2009).

According to (Buschmann, Henney, and Schmidt, 2007a), design patterns can be classified into families that identify a problem area addressing a specific topic. The pattern language proposed in (Buschmann, Henney, and Schmidt, 2007a) includes 114 patterns, which are grouped into thirteen problem areas addressing a specific technical topic.

The main purpose of families and problem areas is to make the language and its patterns more tangible and comprehensible.
In Table 3.1 we briefly describe and summarize some relevant families of patterns as described in (Buschmann, Henney, and Schmidt, 2007a) to which we also add the family of Cloud patterns (Fehling et al., 2014).

Non-Functional Requirements. A software system design is obtained as the results of both functional requirements implementation i.e., what the system does - and requirements about development specifications or quality attributes. Such requirements concern development features, operational costs, but also quality attributes (Chung et al., 2012). Non-Functional Requirements (NFRs) are crucial in software design since help designers in selecting the supposed best design solution among several alternatives. For this reason, best practices in taking into account these requirements is the basis for ensuring successful development. Also taking properly into account requirements can prevent errors which may have negative impact on management and costs of the whole software project (FP Jr, 1987; Davis, 1993; Chung et al., 2012). Non-Functional Requirements lend to a qualitative assessment more than an objective definition, for this reason identifying a particular NFR is a challenging task. Besides, identifying NFRs is a architectural task, so the

TABLE 3.1: Some of the most relevant families of patterns.

Family	Description
Distribution Infrastructure	Patterns concerning middleware's issues, i.e. distribution infrastructure software that help to simplify applications in distributed systems.
Interface Partitioning	Patterns that specify usable and meaningful component interfaces. Interfaces inform clients about the component's responsibilities and usage protocols.
Object Interaction	Patterns for make object interact in standalone programs.
Application Control	These patterns separate interface from application's main functionality. Transforming user input for an application into concrete service requests, executing these requests, and transforming results back into an output that is meaningful for users.
Event Demultiplexing and Dispatching	It concerns patterns that describe different approaches for managing events in distributed and networked systems.
Synchronization	Patterns addressing the problem of synchronous access to shared components, objects and resources.
Concurrency	Patterns that deal with concurrency control.
Adaptation and Extension	Patterns for making applications and components adaptive to specific environments.
Modal Behavior	Patterns that support the implementation of state-machines.
Resource Management	These patterns for managing resource's lifecycle and availability to clients.
Database Access	Patterns belonging to this family manage the access to relational databases and the mapping with other data models.
Cloud	Patterns belonging to this family describe different types of clouds, the services they provide, how to build applications with them and the interconnections between patterns.

availability of reusable solutions that ensure satisfaction of a given NFR or of a set of NFR gives a valid support to performing architectural modeling. Description of knowledge about NFRs is proposed in NFR catalogues (López, Cysneiros, and Astudillo, 2008; Cysneiros, 2007) those enabling reuse and catalogation and a useful way of addressing the need for help on NFR elicitation. In literature, a plethora of definitions can be found of NFRs. A series of such definitions is summarized in (Glinz, 2007):

a) "Describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate."

b) "The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability."

c) "Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet."

d) "...global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. ... There is not a formal definition or a complete list of nonfunctional requirements."

e) "The behavioral properties that the specified functions must have, such as performance, usability."

f) "A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property."

g) "A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior."

Middleware in Internet of Things (IoT) and its requirements. Middleware is necessary to ease the development of the diverse applications and services in IoT. Many solutions have been proposed and implemented, especially in the last couple of years. These solutions are highly diverse in their design approaches, level of programming abstractions, and implementation domains (e.g., WSNs, RFID, M2M, and SCADA) (Razzaque et al., 2016a). The proposals are diverse and involve various middleware design approaches and support different requirements. Based on the analysis in recent surveys (Razzaque et al., 2016a; Zhou, 2012), the existing middleware solution are grouped based on their design approaches, as below:

- Event-based (e.g. Hermes, EMMA, RUNES, PRISMA, SensorBus, Mires and so on),
- Service-oriented (e.g. Hydra, SOCRADES, ubiSOAP, MOSDEN and so on),
- Virtual Machine-based (e.g. VM, DVM, TinyVM, SwissQM, and so on),
- Agent-based (e.g. Ubiware, Impala, MASPOT, and so on),
- Tuple-spaces (e.g. LIME, TS-Mid, TeenyLIME, TinyLIME and so on),
- Database-oriented (e.g. SINA, COUGAR, IrisNet and so on),
- Application-specific (e.g. Milan, AutoSec, Adaptive Middleware and so on)

Some middleware use a combination of different design approaches. In the survey (Razzaque et al., 2016a), all existing middleware in literature are described briefly. A middleware provides a software layer between applications, the operating system and the network communications layers, which facilitates and coordinates some aspect of cooperative processing. From the computing perspective, a middleware provides a layer between application software and system software. Based on previously described characteristics of the IoT's infrastructure and the applications that depend on it, a set

of requirements for a middleware to support the IoT is outlined. These requirements are grouped into the following two sets: *(i) Middleware Service Requirements.* Middleware Service Requirements for the IoT can be categorized as both Functional and Non-Functional. FR capture the services or functions (e.g., resource discovery, resource management, data management, event management, code management) a middleware provides and NFRs (e.g., scalability, real-time o Timeliness, reliability, availability, security & privacy, ease-of deployment and popularity) capture QoS support or performance issue; *(ii) Architectural Requirements.* The Architectural Requirements support application developers: programming, abstraction, interoperable, service-based, adaptive, context-aware, autonomous and distributed. For a detailed description of this requirements we refer to (Razzaque et al., 2016a) and its related references.

3.4 Problem statement and approach

A typical software design problem can be stated as follows:

> *"Given a set of requirements define the software design that best matches them*".

Software design and software architecture design model, requirements and specifications are composed of FRs, NFRs and ARs.

With respect to the general problem stated before, we can re formulate its statement by restricting our attention to the relation among patterns, their families and all categories of requirements. Specifically, with reference to the design problem at the beginning of this paragraph, we address the following problem:

> *""Given a Middleware-induced Software Architecture design to model, a set of FRs, NFRs, ARs and the problem areas the software refers to, which are the design patterns that best fit them? Which are existing middleware that can support all the necessary requirements?"*

The task poses some difficulties that we are going to highlight:

- some requirements cannot be satisfied together some others may be disjoint;
- patterns satisfying the required NFRs may not be contained in families;
- patterns can cooperate, are composable, are complementary or exclusive;
- not all the patterns providing a given NFR or belonging to a family may be known by the designer;
- IoT Middleware solutions are grouped based on their design approaches;
- not all existing middleware support all FRs, NFRs and ARs.

3.4.1 Representing and reasoning about NFRs via fuzzy DLs

We propose the use of fuzzy DLs and fuzzy reasoning services (i) to define a formal model of the relations among problem areas, design patterns, middleware design aproaches and supported NFRs, FRs and ARs; and (ii) to provide the designer a reasoning mechanism as a support for selecting a set of patterns and existing middlewares that ensure the satisfaction of a set of desired requirements. More specifically:

- we use fuzzy DL statements to model the domain of architectural modeling at a high level and to model relations among NFRs, FRs and ARs;
- we propose a fuzzy DL reasoning service to deduce new knowledge about mutual relation between requirements and to answer the retrieval problem.

In the following, detail about addressing the two above mentioned steps are provided.

The (upper) ontology (TBox) is used to formalize the knowledge about the NFRs, FRs, ARs, middleware, about patterns and families that pattern belong to.
The upper ontology is composed by six main classes `SoftwareDesignPattern`, `Families`, `Middleware`, `FunctionalRequirement`, `ArchitecturalRequirement` and `NonFunctionalRequirement`. Its formal definition can be encoded in (classical) DLs as:

$$\exists \texttt{isInFamily} \sqsubseteq \texttt{SoftwareDesignPattern} \tag{3.1}$$
$$\exists \texttt{nFR} \sqsubseteq \texttt{SoftwareDesignPattern} \tag{3.2}$$
$$\exists \texttt{FR} \sqsubseteq \texttt{Middleware} \tag{3.3}$$
$$\exists \texttt{AR} \sqsubseteq \texttt{Middleware} \tag{3.4}$$
$$\top \sqsubseteq \forall \texttt{isInFamily.Families} \tag{3.5}$$
$$\top \sqsubseteq \forall \texttt{NFR.NonFunctionalRequirement} \tag{3.6}$$
$$\top \sqsubseteq \forall \texttt{FR.FunctionalRequirement} \tag{3.7}$$
$$\top \sqsubseteq \forall \texttt{AR.ArchitecturalRequirement} \tag{3.8}$$

In the first four statements are defined the *Domain* restrictions to check the first four statements while *range* restrictions are defined in the last four statement.
To better explain the meaning of restriction, let us consider that `isInFamily` is a role that the concept `SoftwareDesignPattern` to the concept `Families` by connecting their instances.
Instead, the role `NFR` relates the class `SoftwareDesignPattern` to the class `NonFunctionalRequirement` through their instances, the role `FR` relates the class `Middleware` to the class `FunctionalRequirement` and the role `AR` relates the class `Middleware` to the class `ArchitecturalRequirement`.

In our ontology, we make statements about the description of a Middleware, FR, AR, NFR, design pattern, related the pattern to the family it belongs to and make statements about the NFRs it satisfies. Such statements make up

the ABox of the knowledge base we defined. To better explain the knowledge modeling in the ontology let us consider the following statements using the classical DL syntax:

$$\texttt{proxy:SoftwareDesignPattern} \tag{3.9}$$
$$\texttt{resourceManagement:Families} \tag{3.10}$$
$$\texttt{reliability:NonFunctionalRequirement} \tag{3.11}$$
$$\texttt{loadBalancing:NonFunctionalRequirement} \tag{3.12}$$
$$\texttt{reusability:NonFunctionalRequirement} \tag{3.13}$$
$$\texttt{eventManagement:FunctionalRequirement} \tag{3.14}$$
$$\texttt{contextAware:ArchitecturalRequirement} \tag{3.15}$$
$$(\texttt{proxy}, \texttt{resourceManagement}):\texttt{isInFamily} \tag{3.16}$$
$$(\texttt{Hermes}, \texttt{eventBased}):\texttt{isInFamily} \tag{3.17}$$

In the example, the pattern `proxy` is defined as an instance of the class `SoftwareDesignPattern`; the family `resourceManagement` of the class `Families`, the NFRs `reliability`, `loadBalancing`, `reusability` as individuals of the class `NonFunctionalRequirement`, the FR `eventManagement` as individuals of the class `FunctionalRequirement` and `contextaware` as istance of the class `ArchitecturalRequirement`; definition are provided respectively in the statements 3.9-3.15. Moreover, statement 3.16 asserts that `proxy` belongs to the `resourceManagement` family, statement 3.17 asserts that `Hermes` belongs to `eventBased` middleware design approach family.
Additionally, the pattern *proxy* has a level of NFRs *Load Balancing* and *Reliability*. To specify such properties we define new properties, that are related to NFRs. The fuzzy sets used by our framework are the ones represented in Figure 3.2. Such datatype properties are formally represented using **R** that stands for rating. The rating models a ordered set $\{\texttt{verybad}, \texttt{bad}, \texttt{medium}, \texttt{good}, \texttt{verygood}\}$. We refer to the following axioms:

$$\exists \texttt{nFR}.\{\texttt{reliability}\} \equiv \exists \texttt{reliabilityRate}. \in_{\mathbf{R}} \tag{3.18}$$
$$\exists \texttt{nFR}.\{\texttt{loadBalancing}\} \equiv \exists \texttt{loadBalancingRate}. \in_{\mathbf{R}} \tag{3.19}$$
$$\exists \texttt{nFR}.\{\texttt{reusability}\} \equiv \exists \texttt{reusabilityRate}. \in_{\mathbf{R}} \tag{3.20}$$
$$\exists \texttt{nFR}.\{\texttt{adaptability}\} \equiv \exists \texttt{adaptabilityRate}. \in_{\mathbf{R}} \tag{3.21}$$
$$\exists \texttt{nFR}.\{\texttt{maintainability}\} \equiv \exists \texttt{maintainabilityRate}. \in_{\mathbf{R}} \tag{3.22}$$

These axioms states that a pattern that has associated a NFR will have a degree.So we can state for example:

$$\texttt{proxy}:\exists \texttt{loadBalancingRate}. =_{\texttt{good}} \tag{3.23}$$
$$\texttt{proxy}:\exists \texttt{reliabilityRate}. =_{\texttt{good}} \tag{3.24}$$

Besides the modeling of the ABox relations, we use FDLs also to explicitly model relations among NFRs. Consider the NFRs *reliability*, *load balancing*, *reusability* as previously defined (statement (3.11)-(3.13)). An example of mutual relation between NFRs is the one between *load balancing* and *reliability*. Indeed, they are directly proportionate, i.e., if the *loadBalancing* increases (decreases) the same happens for the *reliability*. Such a relation can be written in

fuzzy DLs as

$$\exists \texttt{loadBalancingRate.High} \sqsubseteq \exists \texttt{reliabilityRate.High} \tag{3.25}$$

$$\exists \texttt{loadBalancingRate.Fair} \sqsubseteq \exists \texttt{reliabilityRate.Fair} \tag{3.26}$$

$$\exists \texttt{loadBalancingRate.Low} \sqsubseteq \exists \texttt{reliabilityRate.Low} \tag{3.27}$$

We also know that a system cannot be *reliable* and *reusable* at the same time. That is, the two NFRs are inversely proportionate. Hence, if a pattern guarantees *reliability* it cannot guarantee also *reusability*. We may encode such disjoint relations with the following statement:

$$\exists \texttt{reusabilityRate.High} \sqsubseteq \exists \texttt{reliabilityRate.Low} \tag{3.28}$$

In the same manner, we may also represent statements like

$$\exists \texttt{adaptabilityRate.High} \sqsubseteq \exists \texttt{maintainabilityRate.Fair} \tag{3.29}$$

stating that *"a system with a high adaptability has a fair maintainability"*, in fact if a system has an adaptable behavior to different requirements or to changes in the context, it is poorly maintainable.

Note that from statements (3.25)-(3.28), it can be inferred that the pattern *proxy* cannot guarantee a high degree of *reusability*, since it has a high degree of *load balancing*. Of course, many other kind of implicit relations can be automatically inferred that support the search for better results during the design phase.

3.4.2 Proposed reasoning task

We conclude this paragraph by proposing a novel reasoning task called *Covering Answer Set*, which will result fundamental for our modelling method to solve the defined problem statement.

Let $C_1, \ldots, C_n$ be concepts and let @ be an *aggregation operator* (Torra and Narukawa, 2007). We recall the *Aggregation Operators* (AOs). They are mathematical functions that combine real values. Specifically, an AO has a dimension n and is a mapping $@ : [0,1]^n \to [0,1]$ such that[3] $@(\vec{0}) = 0$, $@(\vec{1}) = 1$. Besides, the aggregation operator @ is monotone in its arguments. Typical examples of Aggregation Operators are *maximum* ($@^{MAX}$), *weighted maximum* $@_W^{\text{wmax}}$, *minimum* ($@^{min}$), *weighted minimum* $@_W^{\text{wmin}}$, *median*, *arithmetic mean* ($@^{AM}$), *weighted sum* $@_W^{\text{ws}}$, strict weighted sum ($@_W^{\text{wsz}}$).

Now, the *covering answer set* of $C_1, \ldots, C_n$ w.r.t. @ and $\mathcal{K}$,

$$\texttt{ans}(@, \mathcal{K}, C_1, \ldots, C_n) \ ,$$

is defined as follows:

1. determine $\texttt{ans}(\mathcal{K}, C_1), \ldots, \texttt{ans}(\mathcal{K}, C_n)$;

[3] With $\vec{0}$ and $\vec{1}$ we identify a vector whose elements are all 0 or 1 respectively.

2. consider the set of tuples

$$A^{@}_{C_1,\ldots,C_n,\mathcal{K}} = \{\langle\{a_1,\ldots,a_n\},\beta\rangle \mid \langle a_1,\alpha_1\rangle \in \texttt{ans}(\mathcal{K},C_1),\ldots, \langle a_n,\alpha_n\rangle \in \texttt{ans}(\mathcal{K},C_n), \beta = @(\alpha_1,\ldots,\alpha_n)\}\ .$$

That is, a tuple in $A^{@}_{C_1,\ldots,C_n,\mathcal{K}}$ is built by picking up an element from each answer set and then by aggregating the individual scores.

3. Eventually, $\texttt{ans}(@,\mathcal{K},C_1,\ldots,C_n)$ is obtained from $A^{@}_{C_1,\ldots,C_n,\mathcal{K}}$ by removing from $A^{@}_{C_1,\ldots,C_n,\mathcal{K}}$ all non-maximal scores and non-minimal subsets, i.e.,

$$\texttt{ans}(@,\mathcal{K},C_1,\ldots,C_n) = \{\langle S,\beta\rangle \in A^{@}_{C_1,\ldots,C_n,\mathcal{K}} \mid \beta > 0, \nexists\langle S',\beta'\rangle \in A^{@}_{C_1,\ldots,C_n,\mathcal{K}} \text{ s.t. } \beta' > \beta, S \subseteq S' \text{ and } \nexists\langle S',\beta'\rangle \in A^{@}_{C_1,\ldots,C_n,\mathcal{K}} \text{ s.t. } S' \subset S\}\ .$$

Each element in $\texttt{ans}(@,\mathcal{K},C_1,\ldots,C_n)$ is a *covering answer* to $C_1,\ldots,C_n$ w.r.t. @ and $\mathcal{K}$. Eventually, the *top-k covering answer set* of $C_1,\ldots,C_n$ w.r.t. @ and $\mathcal{K}$, denoted $\texttt{ans}_k(@,\mathcal{K},C_1,\ldots,C_n)$, is the set top-$k$ ranked covering answers to $C_1,\ldots,C_n$ w.r.t. @ and $\mathcal{K}$.
We say that each element in $\texttt{ans}_k(@,\mathcal{K},C_1,\ldots,C_n)$ is a *top-k covering answer* to $C_1,\ldots,C_n$ w.r.t. @ and $\mathcal{K}$.

3.5 Use Case Scenario n. 01: Cloud-Social- Adaptable System

In this paragraph we describe a Cloud-Social-Adaptable System use case to illustrate how to apply the proposed modeling and solve the first task (retrieve best design patterns to implement).

We consider a social domain in which user's data are stored on cloud platforms and distributed on different clusters or data centers; data are managed by interacting distributed applications. A similar context would require a software architecture based on an extensible model consisting of loosely coupled components. Let us think of web applications for mobile devices, client applications for web-based systems and let us assume that there is a main component – a sort of manager – that performs coordination activities, and other two key components: a manager to gather and manage multimedia data and a location-based application that records all movements made by the user.

On the basis of variations in the user's information needs or changes in the external environment, the system is able to dynamically and extensively change applications to be loaded.

Suppose a user is traveling on a weekend or holiday, the idea is that the system automatically launches an app that organizes the archived multimedia material (photo, movies etc.) relating to the travel destination by creating albums, photo collections with captions, stories etc.

In addition to user's localization, other conditions dependent on the context set in the application may allow the system to dynamically load different applications. The dynamically loaded applications may compromise the system properties, therefore runtime mechanisms to monitor and guarantee the preservation of the properties of interest are needed.

This solution provides flexibility in the architecture as well as access to a public service which is made possible by exploiting the resources available and preserving costs.

Furthermore, virtual machines are loosely coupled, so a possible failure in one of them does not impair the operation of the other guaranteeing an acceptable level of fault tolerance. The virtual machines are located on the middleware and are launched directly from it only if requested by the consumer.

In case of failure the remaining virtual machines would not be affected and the content is made available by the presence of a network.

In order to model the scenarios just described, we have to search for patterns that belong to the family *Cloud* as regards the management of features related to the cloud; but it also requires the solution of problems relating to the communication of process and the middleware so belong to the family *Distribution Infrastructure*. Furthermore the model also requires the use of patterns able to make the system adaptable to changes in the context. So another family to consider is that related to pattern ensuring adaptability.

So, given the application context of the system to be modeled, it is necessary to ensure *adaptability*. Also, since the software must operate in the cloud the model will have to contemplate *elasticity* requirements. Another fundamental requirement to ensure *fault tolerance*, and hence reliability. A low level of *coupling* is also needed since the system implements features of a cloud environment the coupling should be low. Formally, the previously described analysis can be posed as a query as follows:

- retrieve pattern belonging to the following families: *Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*;
- and satisfying the following NFRs: a low degree of *coupling*, a high level of *adaptability*, a high *fault tolerance*, a high level of *elasticity*.

To show how requirements and families necessary to describe the proposed scenario can be modelled using fuzzy DLs, we are going to define the related knowledge base $\mathcal{K}$. The Abox $\mathcal{A}$ contains the following statements. We initially introduce families, patterns and their relations:

applicationControl:FamiliesadaptationAndExtension:Families distributionInfrastructure:Families
cloud:Families

observer:SoftwareDesignPattern
broker:SoftwareDesignPatternreflection:SoftwareDesignPattern
hypervisor:SoftwareDesignPattern
strictConsistency:SoftwareDesignPattern

(hypervisor, cloud):isInFamily
(broker, distributionInfrastructure):isInFamily .(strictConsistency, cloud):isInFamily
(observer, applicationControl):isInFamily
(reflection, adaptationAndExtension):isInFamily

We then define the NFRs and we state how the patterns previously introduced satisfy them:

$$\begin{array}{c}
\texttt{adaptability:NonFunctionalRequirement} \\
\texttt{dependability:NonFunctionalRequirement} \\
\texttt{reliability:NonFunctionalRequirement} \\
\texttt{elasticity:NonFunctionalRequirement} \\
\texttt{faultTolerance:NonFunctionalRequirement} \\
\texttt{loadBalancing:NonFunctionalRequirement} \\
\texttt{flexibility:NonFunctionalRequirement} \\
\texttt{coupling:NonFunctionalRequirement} \\
\texttt{robustness:NonFunctionalRequirement} \\
\texttt{reflection:}\exists\texttt{adaptabilityRate.} =_{\texttt{verygood}}
\end{array}$$

$$\begin{array}{c}
\texttt{strictConsistency:}\exists\texttt{dependabilityRate.} =_{\texttt{verygood}} \\
\texttt{strictConsistency:}\exists\texttt{reliabilityRate.} =_{\texttt{good}}
\end{array}$$

$$\begin{array}{c}
\texttt{hypervisor:}\exists\texttt{elasticityRate.} =_{\texttt{verygood}} \\
\texttt{hypervisor:}\exists\texttt{faultToleranceRate.} =_{\texttt{good}} \\
\texttt{hypervisor:}\exists\texttt{loadBalancingRate.} =_{\texttt{good}}
\end{array}$$

$$\texttt{observer:}\exists\texttt{flexibilityRate.} =_{\texttt{good}} \quad \texttt{observer:}\exists\texttt{adaptabilityRate.} =_{\texttt{medium}}$$

$$\begin{array}{ll}
\texttt{broker:}\exists\texttt{loadBalancingRate.} =_{\texttt{verygood}} & \texttt{broker:}\exists\texttt{robustnessRate.} =_{\texttt{medium}} \\
\texttt{broker:}\exists\texttt{faultToleranceRate.} =_{\texttt{bad}} & \texttt{broker:}\exists\texttt{reliabilityRate.} =_{\texttt{good}} .
\end{array}$$

The TBox contains axioms defining inverse and direct relations between pairs of NFRs. Namely:

$$\exists\texttt{flexibilityRate.High} \sqsubseteq \exists\texttt{couplingRate.Low}$$

$$\begin{array}{c}
\exists\texttt{elasticityRate.High} \sqsubseteq \exists\texttt{adaptabilityRate.High} \\
\exists\texttt{elasticityRate.Fair} \sqsubseteq \exists\texttt{adaptabilityRate.Fair} \\
\exists\texttt{elasticityRate.Low} \sqsubseteq \exists\texttt{adaptabilityRate.Low}
\end{array}$$

$$\begin{array}{c}
\exists\texttt{robustnessRate.High} \sqsubseteq \exists\texttt{faultToleranceRate.High} \\
\exists\texttt{robustnessRate.Medium} \sqsubseteq \exists\texttt{faultToleranceRate.Medium} \\
\exists\texttt{robustnessRate.Low} \sqsubseteq \exists\texttt{faultToleranceRate.Low}
\end{array}$$

$$\begin{array}{c}
\exists\texttt{faultToleranceRate.High} \sqsubseteq \exists\texttt{reliabilityRate.High} \\
\exists\texttt{faultToleranceRate.Medium} \sqsubseteq \exists\texttt{reliabilityRate.Medium} \\
\exists\texttt{faultToleranceRate.Low} \sqsubseteq \exists\texttt{reliabilityRate.Low}
\end{array}$$

$$\begin{array}{c}
\exists\texttt{reliabilityRate.High} \sqsubseteq \exists\texttt{dependabilityRate.High} \\
\exists\texttt{reliabilityRate.Medium} \sqsubseteq \exists\texttt{dependabilityRate.Medium} \\
\exists\texttt{reliabilityRate.Low} \sqsubseteq \exists\texttt{dependabilityRate.Low}
\end{array}$$

$$\begin{array}{c}
\exists\texttt{loadBalancingRate.High} \sqsubseteq \exists\texttt{elasticityRate.High} \\
\exists\texttt{loadBalancingRate.Medium} \sqsubseteq \exists\texttt{elasticityRate.Medium} \\
\exists\texttt{loadBalancingRate.Low} \sqsubseteq \exists\texttt{elasticityRate.Low} .
\end{array}$$

Retrieval of the set of patterns that best satisfies the needed requirements, will be obtained using the novel *covering answer set* reasoning task we have introduced in Paragraph 3.4.2. That is, we define a covering answer set query of the form

$$Q = \texttt{ans}_3(@^{AM}, \mathcal{K}, C_1, C_2, C_3, C_4) , \tag{3.30}$$

which we are going now to build incrementally. First of all let us now consider the query:

- families to consider are the following: *Adaptation and Extension*;
- needed NFRs is *adaptability* with a high degree.

The requirements is modeled as:

$$C' = \exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\} \sqcap \exists \texttt{adaptabilityRate}.\texttt{High}.$$

Under standard fuzzy logic, it can be shown that $\texttt{ans}(\mathcal{K}, C')$ contains the following statements:

$\langle \texttt{reflection}:\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\}, 1\rangle$
$\langle \texttt{reflection}:\exists \texttt{adaptability}.\texttt{High}, 1\rangle$
$\langle \texttt{reflection}:C', 1\rangle$

$\langle \texttt{strictConsistency}:\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\}, 0\rangle$
$\langle \texttt{strictConsistency}:\exists \texttt{adaptability}.\texttt{High}, 0\rangle$
$\langle \texttt{strictConsistency}:C', 0\rangle$

$\langle \texttt{hypervisor}:\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\}, 0\rangle$
$\langle \texttt{hypervisor}:\exists \texttt{adaptability}.\texttt{High}, 1\rangle$
$\langle \texttt{hypervisor}:C', 0\rangle$

$\langle \texttt{observer}:\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\}, 0\rangle$
$\langle \texttt{observer}:\exists \texttt{adaptability}.\texttt{High}, 0.66\rangle$
$\langle \texttt{observer}:C', 0\rangle$

$\langle \texttt{broker}:\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\}, 0\rangle$
$\langle \texttt{broker}:\exists \texttt{adaptability}.\texttt{High}, 0.66\rangle$
$\langle \texttt{broker}:C', 0\rangle$.

$\exists \texttt{couplingRate}.\texttt{Low} \sqsubseteq \exists \texttt{scalabilityRate}.\texttt{High}$

$\exists \texttt{adapatabilityRate}.\texttt{High} \sqsubseteq \exists \texttt{interoperabilityRate}.\texttt{High}$
$\exists \texttt{adapatabilityRate}.\texttt{Fair} \sqsubseteq \exists \texttt{interoperabilityRate}.\texttt{Faie}$
$\exists \texttt{adapatabilityRate}.\texttt{Low} \sqsubseteq \exists \texttt{interoperabilityRate}.\texttt{Low}$

$\exists \texttt{responseTimeRate}.\texttt{High} \sqsubseteq \exists \texttt{performanceRate}.\texttt{High}$
$\exists \texttt{responseTimeRate}.\texttt{Medium} \sqsubseteq \exists \texttt{performanceRate}.\texttt{Medium}$
$\exists \texttt{responseTimeRate}.\texttt{Low} \sqsubseteq \exists \texttt{performanceRate}.\texttt{Low}$.

C' is satisfied by the list of pattern in:

$$\begin{aligned}\texttt{ans}(\mathcal{K}, C') \quad = \{ \quad & \langle \texttt{reflection}, 1\rangle, \langle \texttt{strictConsistency}, 0\rangle, \\ & \langle \texttt{hypervisor}, 0\rangle, \langle \texttt{observer}, 0\rangle, \langle \texttt{broker}, 0\rangle \quad \} .\end{aligned}$$

Please note that, although there is no explicit statements about the adaptability value of *Hypervisor* patterns, thanks to the interaction between `elasticityRate` and `adaptabilityRate` stated in the TBox, we infer that *Hypervisor* has a high level of adaptability.

We now add all the other families required to solve our task.

Therefore, we modify C' in C_1 as follows:

$$\begin{aligned} C_1 \quad = \quad & (\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\} \\ & \quad \sqcup \exists \texttt{isInFamily}.\{\texttt{cloud}\} \\ & \quad \sqcup \exists \texttt{isInFamily}.\{\texttt{applicationControl}\} \\ & \quad \sqcup \exists \texttt{isInFamily}.\{\texttt{distributionInfrastructure}\}) \\ & \sqcap \exists \texttt{adaptabilityRate}.\texttt{High}. \end{aligned}$$

It can be shown that $\texttt{ans}(\mathcal{K}, C_1)$ under standard fuzzy logic contains the following statements:

$\langle \texttt{reflection}:\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\}, 1\rangle$
$\langle \texttt{reflection}:\exists \texttt{isInFamily}.\{\texttt{cloud}\}, 0\rangle$
$\langle \texttt{reflection}:\exists \texttt{isInFamily}.\{\texttt{applicationControl}\}, 0\rangle$
$\langle \texttt{reflection}:\exists \texttt{isInFamily}.\{\texttt{adaptationAndExtension}\}, 0\rangle$
$\langle \texttt{reflection}:\exists \texttt{adaptability}.\texttt{High}, 1\rangle$
$\langle \texttt{reflection}:C_1, 1\rangle$

$\langle$strictConsistency:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$strictConsistency:$\exists$isInFamily.{cloud}, 1$\rangle$
$\langle$strictConsistency:$\exists$isInFamily.{applicationControl}, 0$\rangle$
$\langle$strictConsistency:$\exists$isInFamily.{distributionInfrastructure}, 0$\rangle$
$\langle$strictConsistency:$\exists$adaptability.High, 0$\rangle$
$\langle$strictConsistency:C_1, 0$\rangle$

$\langle$hypervisor:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$hypervisor:$\exists$isInFamily.{cloud}, 1$\rangle$
$\langle$hypervisor:$\exists$isInFamily.{applicationControl}, 0$\rangle$
$\langle$hypervisor:$\exists$isInFamily.{distributionInfrastructure}, 0$\rangle$
$\langle$hypervisor:$\exists$adaptability.High, 1$\rangle$
$\langle$hypervisor:C_1, 1$\rangle$

$\langle$observer:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$observer:$\exists$isInFamily.{cloud}, 0$\rangle$
$\langle$observer:$\exists$isInFamily.{applicationControl}, 1$\rangle$
$\langle$observer:$\exists$isInFamily.{distributionInfrastructure}, 0$\rangle$
$\langle$observer:$\exists$adaptability.High, 0.66$\rangle$
$\langle$observer:C_1, 0.66$\rangle$

$\langle$broker:$\exists$isInFamily.{adaptationAndExtension}, 0$\rangle$
$\langle$broker:$\exists$isInFamily.{cloud}, 0$\rangle$
$\langle$broker:$\exists$isInFamily.{applicationControl}, 0$\rangle$
$\langle$broker:$\exists$isInFamily.{distributionInfrastructure}, 1$\rangle$
$\langle$broker:$\exists$adaptability.High, 0.66$\rangle$
$\langle$broker:C_1, 0.66$\rangle$.

Based on the previous results, it follows that

$$\begin{aligned}\mathtt{ans}(\mathcal{K}, C_1) \quad = \{ \quad & \langle \mathtt{reflection}, 1\rangle, \langle \mathtt{strictConsistency}, 0\rangle, \\ & \langle \mathtt{hypervisor}, 1\rangle, \langle \mathtt{observer}, 0.66\rangle, \langle \mathtt{broker}, 0.66\rangle \quad \} .\end{aligned}$$

We are now ready to define also C_2, C_3 and C_4 in Equation (3.30).

$$\begin{aligned} C_2 \quad = \quad & (\exists \mathtt{isInFamily}.\{\mathtt{adaptationAndExtension}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{cloud}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{applicationControl}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{distributionInfrastructure}\}) \\ & \sqcap \exists \mathtt{elasticityRate}.\mathtt{High}. \end{aligned}$$

$$\begin{aligned} C_3 \quad = \quad & (\exists \mathtt{isInFamily}.\{\mathtt{adaptationAndExtension}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{cloud}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{applicationControl}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{distributionInfrastructure}\}) \\ & \sqcap \exists \mathtt{faultToleranceRate}.\mathtt{High}. \end{aligned}$$

$$\begin{aligned} C_4 \quad = \quad & (\exists \mathtt{isInFamily}.\{\mathtt{adaptationAndExtension}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{cloud}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{applicationControl}\} \\ & \quad \sqcup \exists \mathtt{isInFamily}.\{\mathtt{distributionInfrastructure}\}) \\ & \sqcap \exists \mathtt{couplingRate}.\mathtt{Low}. \end{aligned}$$

With reference to these concepts, the results for $\mathtt{ans}(\mathcal{K}, C_i)$ with $i = 2 \ldots 4$ are illustrated in Table 3.2.

TABLE 3.2: Answer Sets Use Case I.

$\mathtt{ans}(\mathcal{K}, C_2)$	$\mathtt{ans}(\mathcal{K}, C_3)$	$\mathtt{ans}(\mathcal{K}, C_4)$
$\langle$reflection, 0$\rangle$	$\langle$reflection, 0$\rangle$	$\langle$reflection, 0$\rangle$
$\langle$strictConsistency, 0$\rangle$	$\langle$strictConsistency, 0$\rangle$	$\langle$strictConsistency, 0$\rangle$
$\langle$hypervisor, 1$\rangle$	$\langle$hypervisor, 1$\rangle$	$\langle$hypervisor, 0$\rangle$
$\langle$observer, 0$\rangle$	$\langle$observer, 0$\rangle$	$\langle$observer, 0$\rangle$
$\langle$broker, 1$\rangle$	$\langle$broker, 0.33$\rangle$	$\langle$broker, 1$\rangle$

We can finally compute the result for Equation (3.30):

$$\begin{aligned} Q = \{ \quad & \langle \{\texttt{hypervisor}, \texttt{broker}\}, 1 \rangle, \\ & \langle \{\texttt{reflection}, \texttt{broker}\}, 0.8325 \rangle, \\ & \langle \{\texttt{observer}, \texttt{broker}\}, 0.75 \rangle \quad \}. \end{aligned}$$

The pair of pattern `hypervisor` and `broker` is the best solution; the second rank is the pair `broker` and `reflection`; the third is the pair `broker` and `observer`.

Please note that e.g. $\langle \{\texttt{broker}\}, 0.75 \rangle$ is ruled out from Q as there exists a superset with strictly higher score (e.g., $\{\langle \texttt{hypervisor}, \texttt{broker}\}, 1 \rangle$).

3.6 Use Case Scenario n. 02: IoT, in a Healthcare scenario

In this paragraph we describe a second use-case within the field of the IoT, focusing on a scenario of Healthcare.
The new frontier of medicine is moving towards the improvement of the quality of medical treatment and, at the same time, the reduction of the related costs. In this sense, the Internet of Things paradigm is helping to develop a new generation of telemedicine applications that exploit the data collected by sensors worn by the patients. The data collected from all sensors are then transferred to the gateway, which is responsible for analyzing the raw data and forward the processed information to a Cloud system. This information will be later refined and used by doctors to cure/treat the patient. Moreover, in case critical situations are detected in the parameters, an alert (emergency services) is forwarded to the nearest emergency center.

To model this use case scenario just described, we have to search for an existing IoT middleware solution belong to *application-specific* design approach family and patterns that belong to the *Cloud* family as regards the management of features related to the cloud and the *eventHandling* family for managing events in the distributed and networked architecture and publish/subscribe dissemination. But it also requires the solution of problems relating to the communication of process and the middleware so belong to the family *Distribution Infrastructure* and *applicationControl* family for separate interface from application's main functionality. Furthermore the model also requires the use of patterns able to make the system adaptable to changes in the context. So another family to consider is that related to pattern ensuring adaptability to the family *Adaptation and Extension*.

So, given the application context of the system to be modeled, it is necessary to ensure *adaptability*, *performance*, *security*, *availability*, *scalability* and so on. Another fundamental requirement to ensure *fault tolerance*, and hence reliability. A low level of *coupling* is also needed since the system implements features of a cloud environment the coupling should be low. Formally, the previously described analysis can be posed as a query as follows:

- retrieve patterns belonging to the following families: *Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*, *eventHandling*;
- retrieve middleware belonging to the design approach family: *applicationSpecific*;

and satisfying the following requirements:

- NFRs: a low degree of *coupling*, a high level of *interoperability*, a high level of *interoperability*, a high *fault tolerance*, a high level of *elasticity*;
- FRs: *Resource Discovery*, *Resource Management*, *Data Management*, *Event Management*;
- ARs: *interoperable*, *Context-aware*, *autonomous*, *adaptive*, *service-based*, *ligh-weight*, *distributed*;

To show how requirements and families necessary to describe the proposed scenario can be modelled using fuzzy DLs, we are going to define the related knowledge base $\mathcal{K}$. The Abox $\mathcal{A}$ contains the following statements. In order to solve the first question of the problem statement (best set of design patterns), we initially introduce families, patterns and their relations:

$$\begin{array}{l} \texttt{applicationControl:Families} \\ \texttt{adaptationAndExtension:Families} \\ \texttt{distributionInfrastructure:Families} \\ \texttt{cloud:Families} \\ \texttt{eventHandling:Families} \end{array}$$

$$\begin{array}{l} \texttt{observer:SoftwareDesignPattern} \\ \texttt{broker:SoftwareDesignPattern} \\ \texttt{proxy:SoftwareDesignPattern} \\ \texttt{bridge:SoftwareDesignPattern} \\ \texttt{hypervisor:SoftwareDesignPattern} \\ \texttt{strictConsistency:SoftwareDesignPattern} \\ \texttt{bridge:SoftwareDesignPattern} \\ \texttt{proxy:SoftwareDesignPattern} \\ \texttt{publishSubscribe:SoftwareDesignPattern} \\ \texttt{proactor:SoftwareDesignPattern} \\ \texttt{statelessComponent:SoftwareDesignPattern} \end{array}$$

$$\begin{array}{l} (\texttt{hypervisor}, \texttt{cloud}):\texttt{isInFamily} \\ (\texttt{broker}, \texttt{distributionInfrastructure}):\texttt{isInFamily} \\ (\texttt{strictConsistency}, \texttt{cloud}):\texttt{isInFamily} \\ (\texttt{observer}, \texttt{applicationControl}):\texttt{isInFamily} \\ (\texttt{reflection}, \texttt{adaptationAndExtension}):\texttt{isInFamily} \\ (\texttt{bridge}, \texttt{adaptationAndExtension}):\texttt{isInFamily} \\ (\texttt{proxy}, \texttt{adaptationAndExtension}):\texttt{isInFamily} \\ (\texttt{publishSubscribe}, \texttt{distributionInfrastructure}):\texttt{isInFamily} \\ (\texttt{proactor}, \texttt{eventHandling}):\texttt{isInFamily} \\ (\texttt{statelessComponent}, \texttt{cloud}):\texttt{isInFamily} \end{array}$$

We then define the NFRs and we state how the patterns previously introduced satisfy them:

$$
\begin{aligned}
&\texttt{interoperability:NonFunctionalRequirement}\\
&\texttt{scalability:NonFunctionalRequirement}\\
&\texttt{availability:NonFunctionalRequirement}\\
&\texttt{security:NonFunctionalRequirement}\\
&\texttt{performance:NonFunctionalRequirement}\\
&\texttt{responseTime:NonFunctionalRequirement}\\
&\texttt{resilience:NonFunctionalRequirement}\\
&\texttt{estensibility:NonFunctionalRequirement}\\
&\texttt{adaptability:NonFunctionalRequirement}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{broker:}\exists\texttt{interoperabilityRate.} =_{\texttt{good}}\\
&\texttt{broker:}\exists\texttt{scalabilityRate.} =_{\texttt{good}}\\
&\texttt{broker:}\exists\texttt{faultToleranceRate.} =_{\texttt{bad}}\\
&\texttt{broker:}\exists\texttt{couplingRate.} =_{\texttt{good}}\\
&\texttt{broker:}\exists\texttt{performanceRate.} =_{\texttt{medium}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{proxy:}\exists\texttt{securityRate.} =_{\texttt{good}}\\
&\texttt{proxy:}\exists\texttt{scalabilityRate.} =_{\texttt{bad}}\\
&\texttt{proxy:}\exists\texttt{couplingRate.} =_{\texttt{bad}}\\
&\texttt{proxy:}\exists\texttt{availabilityRate.} =_{\texttt{good}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{publishSubscribe:}\exists\texttt{couplingRate.} =_{\texttt{medium}}\\
&\texttt{publishSubscribe:}\exists\texttt{interoperabilityRate.} =_{\texttt{good}}\\
&\texttt{publishSubscribe:}\exists\texttt{performanceRate.} =_{\texttt{bad}}\\
&\texttt{publishSubscribe:}\exists\texttt{scalabilityRate.} =_{\texttt{good}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{proactor:}\exists\texttt{couplingRate.} =_{\texttt{good}}\\
&\texttt{proactor:}\exists\texttt{performanceRate.} =_{\texttt{verygood}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{reflection:}\exists\texttt{adaptabilityRate.} =_{\texttt{verygood}}\\
&\texttt{reflection:}\exists\texttt{interoperabilityRate.} =_{\texttt{good}}\\
&\texttt{reflection:}\exists\texttt{robustnessRate.} =_{\texttt{medium}}\\
&\texttt{reflection:}\exists\texttt{availabilityRate.} =_{\texttt{medium}}\\
&\texttt{reflection:}\exists\texttt{performanceRate.} =_{\texttt{bad}}
\end{aligned}
$$

$$
\texttt{bridge:}\exists\texttt{adaptabilityRate.} =_{\texttt{good}}
$$

$$
\begin{aligned}
&\texttt{cloudStrictConsistency:}\exists\texttt{availabilityRate.} =_{\texttt{good}}\\
&\texttt{cloudStrictConsistency:}\exists\texttt{scalabilityRate.} =_{\texttt{medium}}\\
&\texttt{cloudStrictConsistency:}\exists\texttt{performanceRate.} =_{\texttt{good}}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{statelessComponent:}\exists\texttt{availabilityRate.} =_{\texttt{verygood}}\\
&\texttt{statelessComponent:}\exists\texttt{robustnessRate.} =_{\texttt{verygood}}\\
&\texttt{statelessComponent:}\exists\texttt{securityRate.} =_{\texttt{medium}}.
\end{aligned}
$$

The TBox contains axioms defining inverse and direct relations between pairs of NFRs. Namely:

$$
\exists\texttt{flexibilityRate.High} \sqsubseteq \exists\texttt{couplingRate.Low}
$$

$$
\begin{aligned}
&\exists\texttt{elasticityRate.High} \sqsubseteq \exists\texttt{adaptabilityRate.High}\\
&\exists\texttt{elasticityRate.Fair} \sqsubseteq \exists\texttt{adaptabilityRate.Fair}\\
&\exists\texttt{elasticityRate.Low} \sqsubseteq \exists\texttt{adaptabilityRate.Low}
\end{aligned}
$$

$$
\begin{aligned}
&\exists\texttt{robustnessRate.High} \sqsubseteq \exists\texttt{faultToleranceRate.High}\\
&\exists\texttt{robustnessRate.Medium} \sqsubseteq \exists\texttt{faultToleranceRate.Medium}\\
&\exists\texttt{robustnessRate.Low} \sqsubseteq \exists\texttt{faultToleranceRate.Low}
\end{aligned}
$$

$$
\begin{aligned}
&\exists\texttt{faultToleranceRate.High} \sqsubseteq \exists\texttt{reliabilityRate.High}\\
&\exists\texttt{faultToleranceRate.Medium} \sqsubseteq \exists\texttt{reliabilityRate.Medium}\\
&\exists\texttt{faultToleranceRate.Low} \sqsubseteq \exists\texttt{reliabilityRate.Low}
\end{aligned}
$$

$$
\begin{aligned}
&\exists\texttt{reliabilityRate.High} \sqsubseteq \exists\texttt{dependabilityRate.High}\\
&\exists\texttt{reliabilityRate.Medium} \sqsubseteq \exists\texttt{dependabilityRate.Medium}\\
&\exists\texttt{reliabilityRate.Low} \sqsubseteq \exists\texttt{dependabilityRate.Low}
\end{aligned}
$$

$$
\begin{aligned}
&\exists\texttt{loadBalancingRate.High} \sqsubseteq \exists\texttt{elasticityRate.High}\\
&\exists\texttt{loadBalancingRate.Medium} \sqsubseteq \exists\texttt{elasticityRate.Medium}\\
&\exists\texttt{loadBalancingRate.Low} \sqsubseteq \exists\texttt{elasticityRate.Low}.
\end{aligned}
$$

To retrieve the best set of design pattern that fulfill the necessary requirements as described at the beginning of this use case, we model the query as:

$$Q = \mathtt{ans}_1(@^{AM}, \mathcal{K}, C_1, C_2, C_3, C_4, C_5, C_6, C_7)\ . \tag{3.31}$$

Table 3.3 summarises the scores of each individual w.r.t. queries $C_1, \ldots, C_7$. From Table 3.3, using the $@^{AM}$ (average) aggregation operator, it can be shown that the result to query Q in Equation (3.31) is:

TABLE 3.3: Answer Sets Use Case scenario.

	broker	proxy	publ.Subs.	proactor	bridge	reflection	CloudStr.Cons.	statelessComp.
C_1	1	0	1	0	1	1	0	0
C_2	1	0.33	1	1	1	0	0.66	0
C_3	0	1	0	0	0	0.66	1	1
C_4	0.33	1	0	0	0	0.66	1	1
C_5	0	1	0	0	0	0	0	0.66
C_6	1	0.33	1	1	1	0	0	0
C_7	0.66	0	0.33	1	0	0.33	1	0

$$\begin{aligned} Q \quad = \{ \quad & \langle \{\texttt{broker}, \texttt{proxy}\}, 0.951 \rangle, \\ & \langle \{\texttt{proxy}, \texttt{publishSubscribe}\}, 0.904 \rangle, \\ & \langle \{\texttt{broker}, \texttt{statelessComponent}\}, 0.904 \rangle \} \\ & \langle \{\texttt{publishSubscribe}, \texttt{statelessComponent}\}, 0.855 \rangle, \\ & \langle \{\texttt{proxy}, \texttt{reflection}\}, 0.714 \rangle \quad \}\ . \end{aligned}$$

Hence, the best solution to adopt is $\{\texttt{broker}, \texttt{proxy}\}$.

Enriching the KB with the related fuzzy DL statement in order to retrieve the middleware can support all the necessary requirements, by adopting the same approach, the first best solution to adopt is $\{\texttt{adaptiveMiddlware}\}$.

3.7 Implementation

In this paragraph we present the implemented ontology-driven web application called MoSAIC. The overall architecture is depicted in Figure 3.3. MoSAIC [4] was developed using Eclipse as a Web development platform integrating Apache Tomcat [5] as application server. The business layer is consisting of a modules that implements the functionalities of the tool. First of all enable *(i)* search of Middleware, design patterns, NFRs, FRs, ARs and Pattern Families query the ontology showing their *annotations*; and *(ii)* solve the queries of retrieving the smallest subset of patterns that best match the input requirements and recommending existing Middleware can support all the necessary requirements.

Ontology Administration provides for three main functionalities, that are, the insertion of a new individual (NFR, FR, AR, Middleware or Design pattern), the insertion

[4] MoSAIC's website at: http://sisinflab.poliba.it/tools/softarch/mosaic.

[5] Apache Tomcat software: https://tomcat.apache.org/

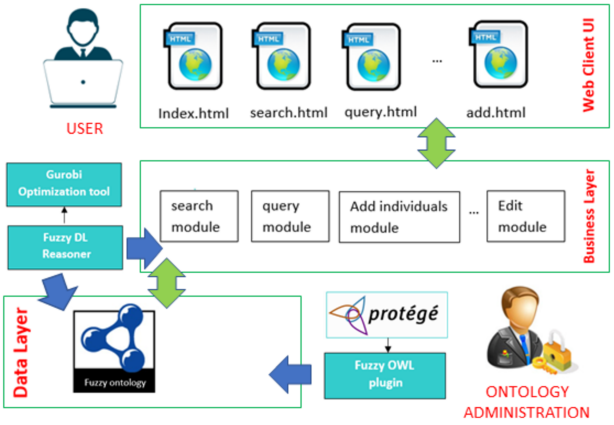


FIGURE 3.3: MoSAIC Architecture, and linked components.

of a Fuzzy DL statements (relations intercurring between individuals) and the editing of ontology individuals. Before executing any change on the ontology, i.e. any of the administrator functionalities, a consistency check is performed.
The Linked components are the following:

- **Fuzzy DL Reasoner** is a java-based reasoner allowing to work with vague information, previously described;
- **Gurobi**[6] is a library for mathematical programming. It is focused on linear programming solver (LP solver), quadratic programming solver (QP solver), quadratically constrained programming solver (QCP solver), mixed-integer linear programming solver (MILP solver), mixed-integer quadratic programming solver (MIQP solver), and mixed-integer quadratically constrained programming solver (MIQCP solver) and it is used within the Fuzzy DL reasoner for MILP calculations;
- **Protégé**[7] is an open source ontology editor that supports the Web Ontology Language (OWL);
- **FuzzyOWL plugin**[8] is a plugin for Protégé that allows users to edit, save Fuzzy OWL 2 ontologies, and submit queries to the underlying inference engine FuzzyDL.

The MoSAIC core component is the Fuzzy ontology. We constructed a Fuzzy OWL 2 ontology (Bobillo and Straccia, 2011) according to the Linked Data principles[9] by reusing URIs already available in the Web. We referred to DBpedia[10] as it is a "de facto" standard in the representation of entities in the Web. In Figure 3.4 is depicted the MoSAIC Fuzzy ontology classes hierarchy (The green nodes are reused classes defined for the ontology developed for MD thesis).
We collected the knowledge about 109 design patterns, 28 pattern families, 37 NFRs, 61 existing Middleware in literature grouped based on their 7 design approaches

[6]The Gurobi Optimizer library is available at www.gurobi.com.
[7]It is available for free download at http://protege.stanford.edu/.
[8]Fuzzy OWL plugin: https://protegewiki.stanford.edu/wiki/FuzzyOWL2
[9]http://www.w3.org/DesignIssues/LinkedData.html
[10]http://dbpedia.org

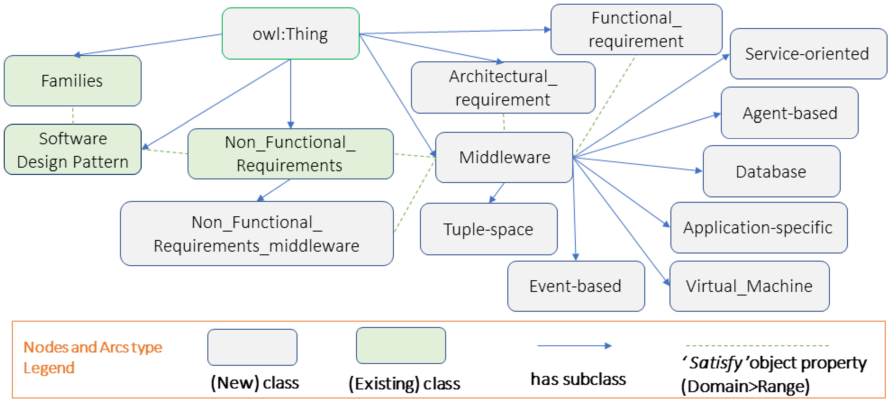


FIGURE 3.4: MoSAIC Fuzzy Ontology classes hierarchy.

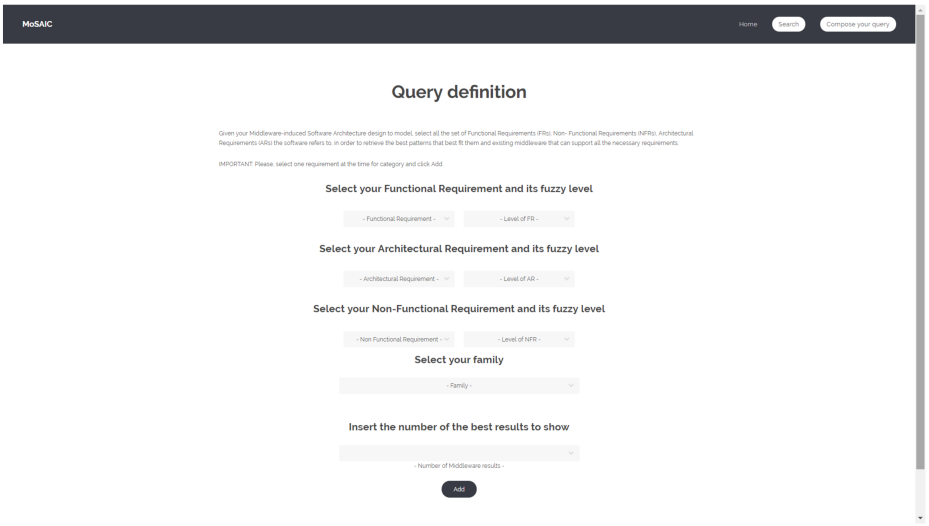


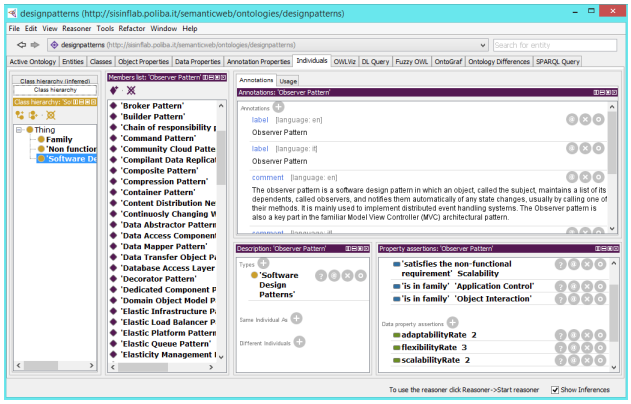
FIGURE 3.5: Screenshot of tool query definition.

and its 14 NFRs, 5 FRs, 8 ARs together with their mutual relations. Other important metrics associated to our ontology, whose consistency has been checked and validated with the *fuzzyDL* Reasoner are the following: *Axiom* (4.682), *Logical axiom count* (3.474), *Functional Data Property axioms* (69), *Data Property Range axioms* (69), *Class Assertion axioms* (280), *Object Property Assertion axioms* (1.560), *Data Property Assertion axioms* (1.434) [11]. According to this metrics, the new knowledge encoded within the MoSAIC Ontology extends the previous ontology of around 308 %.

Screenshot in Figure 3.5 show shows the form for selecting the main elements needed to solve the task that will be translated in the Covering Answer Set query formalism.

Figure 3.6 (a) shows a screenshot of the *Protégé* editor GUI illustrating the main classes of our ontology and part of the individuals in the ABox. The Fuzzy OWL 2 plug-in for Protégé made aleviated the modeling of the fuzzy sets as well as of

[11] MoSAIC ontology is at http://sisinflab.poliba.it/semanticweb/ontologies/mosaic/



(a)

(b)

FIGURE 3.6: (a) Individuals Tab, (b) Creation of a fuzzy datatype *Fair* with the Fuzzy OWL plug-in.

data properties assertions as shown in Figure 3.6 (a) - (b). Axioms as those in Equations (3.9) - (3.16) and Equations (3.25) - (3.29) can be modeled in the *General class axioms* tab. Figure 3.7 reports their equivalent form with the Manchester syntax of OWL 2.[12] During the modeling we adopted the Linked Data principles[13] and tried to reuse URIs already available in the Web.
For instance, we associated the URI `http://dbpedia.org/resource/Non-functional_requirement` to the class `NonFunctionalRequirement`. The same principle has guided the selection of the URIs for design patterns and NFRs. We referred to DBpedia[14] as it is a "de facto" standard in the representation of entities in the Web.[15]

3.8 Discussion

In order to evaluate the degree of usefulness for the tool, we designed a controlled experiment. We sketched the Cloud-Social-Adaptable System example presented in Paragraph 3.5 and we then proposed it to six different teams of students. The teams were assembled so that each one would be composed by three second year

[12] http://www.w3.org/TR/owl2-manchester-syntax/
[13] http://www.w3.org/DesignIssues/LinkedData.html
[14] http://dbpedia.org
[15] See, e.g. the diagram available at http://lod-cloud.net.

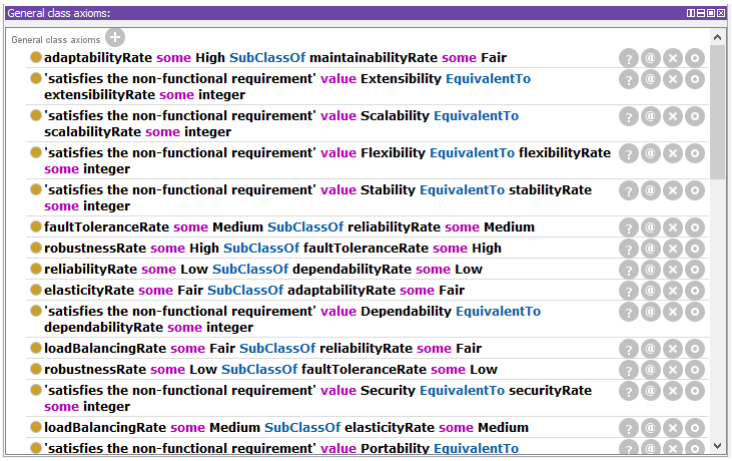


FIGURE 3.7: General class axioms Tab.

graduate students. All students were trained during the MSc course on software design, architectural pattern, NFRs modelling, architectural design.

Three teams (T_1, T_2, T_3) solved the problem supported by the tool while the other three teams (T_4, T_5, T_6) solved the problem using only their own experience. The teams had no idea that other teams were working on the same use case and they had been instructed not to comment the experiment with anyone else. The solution to each design problem was provided as the design of an architecture considering quality aspects and choosing the patterns that best model NFRs suitable for the given context, domain and goals.

The teams T_1, T_2, T_3 were provided with an abstract description in natural language of the use case scenario and each team formulated the query to be submitted to the tool. As shown in Table 3.4, all the three teams composed a query that slightly differs from one another only for the NFR degree (*high*, *medium*, *low*).

The other three teams – T_4, T_5, T_6 –, being provided with the entire query, solved the problems using only their own experience.

TABLE 3.4: Queries formulated by each team and corresponding answers (best solution).

Team	Query		Best Solution
	Families	NFRs	
T_1	*Distribution Infrastructure, Application Control, Adaptation and Extension, Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_2	*Distribution Infrastructure, Application Control, Adaptation and Extension, Cloud Patterns*	medium *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_3	*Distribution Infrastructure, Application Control, Adaptation and Extension, Cloud Patterns*	low *coupling*, medium *adaptability*, medium *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_4	*Distribution Infrastructure, Application Control, Adaptation and Extension, Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Hypervisor, Broker
T_5	*Distribution Infrastructure, Application Control, Adaptation and Extension, Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Reflection, Broker
T_6	*Distribution Infrastructure, Application Control, Adaptation and Extension, Cloud Patterns*	low *coupling*, high *adaptability*, high *fault tolerance*, high *elasticity*	Integration Provider Pattern, Observer

The three solutions provided by the last teams are not expected to be the same, since the answer derives from the experience, from their reasoning, and from creativity. Measuring the goodness of results is not immediate. We compared the solutions provided by the three teams which were supported by the tool with those provided

by the teams not supported by the tool and with the human expert-provided solution. Based on the previously described analysis in Paragraph3.5 the best formulated query is composed as follows:

- Families: *Distribution Infrastructure*, *Application Control*, *Adaptation and Extension*, *Cloud Patterns*;
- NFRs: a low degree of *coupling*, a high level of *adaptability*, a high *fault tolerance*, a high level of *elasticity*.

The set of top-3 ranked answers Q for the provided query is the same as the one reported at the end of Paragraph 3.5.

At the end of the modeling process, each team was supplied with the solution proposed by the others. A qualitative evaluation of the approaches was asked to the teams. Table 3.5 summarizes the results for this qualitative evaluation: the solutions provided with the support of the tool where judged fully correct and complete by the others teams; on the other hand, among the solutions provided by the teams not using the tool, one was considered fully acceptable by all of the other teams while the other two needed respectively minor/ major revisions.

TABLE 3.5: Cross Evaluation among teams.

	full acceptance	minor revision	major revision
T_1, T_2, T_3 Solutions	3	0	0
T_4, T_5, T_6 Solutions	1	1	1

Note that the initial queries for the first three teams were slightly different since each team faced the problem starting from a different point of view. Nevertheless, the answers for these queries was the same, corresponding to the best solution.

Table 3.6 shows the time used by the teams to solve the problem. We measured the efficiency of the process, in term of time elapsed between the starting point of the work and the final solution provided by each team.

TABLE 3.6: Elapsed time to solve the problem.

Team	Elapsed time
T_1	2 days
T_2	1 day and a half
T_3	2 days
T_4	4 days
T_5	5 days
T_6	4 days and a half

Subsequently, in order to measure the matching degree of retrieved patterns with the three ones proposed by the human expert, we measured the similarity between the proposed solutions by the six teams with respect to these three. To this end we

used a combination of Jaccard coefficients (Levandowsky and Winter, 1971). We recap that the Jaccard similarity between two sets A and B is defined as:

$$J(A,B) = \frac{\mid A \cap B \mid}{\mid A \cup B \mid}$$

In detail, we compare the similarity of each solution with the ranking provided by the covering answer set algorithm; w.r.t. the standard Jaccard Similarity coefficient we introduce an exponential decay for each retrieved covering answer set. Formally, being:

- k, the number of covering answer set to retrieve;
- ans^{Tj} the solution set provided by the j-th Team (T_j);
- Q_i the i-th answer set composing the *top-k covering answer set* Q;

the similarity between the solutions set provided by each Team ans^{Tj} and the top-k ranked answers Q is defined as in the following:

$$sim(ans^{Tj}, Q) = \frac{\sum_{i=1}^{k} J(ans^{Tj}, Q_i) \cdot e^{-(i-1)}}{k}, \text{ with } i = 1 \dots k \tag{3.1}$$

Table 3.7 illustrates the similarity values obtained by using Equation (3.1).

In particular, the first column illustrates the similarity value with respect to the best solution Q_1. Note that the similarity analysis is in line with the qualitative analysis carried out in the first step and that the teams using the tool performed better than those without it.

TABLE 3.7: Similarity values with respect to Q_1 (best solution) and Q (the top-3 ranked answers).

Team	$J(ans^{T_j},Q_1)$	$J(ans^{T_j},Q)$
T_4	1.00	0.389
T_5	0.333	0.249
T_6	0.00	0.056

3.9 Related work

Existing approaches for knowledge representation in software engineering are listed in this paragraph. An emphasis is posed on modeling of NFRs, modeling of design pattern, relationships among NFRs and design patterns, approaches for supporting decision making in software architecture design and existing tools.

A systematic mapping study about application of method based on knowledge Representation to solve typical problems of software architecture is proposed in (Li, Liang, and Avgeriou, 2013). The authors outline the most relevant research directions and the weaknesses in the two domains and in their combination. The study witnesses an increased use of knowledge representation based methods to modeling software architectures, especially concerning the model of architectural essentials and relationships among them. In particular, (1) the study calls for more deep

investigation on use of knowledge representation to study impact analysis of software architectures. (2) also approaches of knowledge recovery needs to be further explored; (3) the architectural implementation can benefit from knowledge sharing; (4) automatic and semiautomatic reasoning should be improved; (5) the study shows the need of deep studies about benefit of knowledge approach to software architecture; (6) knowledge based approaches could be implied in several new domains.

Ontological approaches. Ontologies are frequently used in several contexts of software engineering. (Pan et al., 2013) provides a complete analysis for using ontologies in Software Engineering, especially in the development process. In (Gašević, Kaviani, and Milanović, 2009) the use of ontologies in software engineering is surveyed, by considering all the phases of software development Kruchten (Kruchten, 2004a) models architectural design decision in software-intensive systems using an ontology in which each architectural design decision belongs to one of the following categories: existence decisions, behavior decisions, property decisions. The ontology is supported by a tool able to list decision and relations, visualizing design structure and temporal view of design decisions. The works (Henninger and Ashokkumar, 2005; Henninger and Ashokkumar, 2006) propose the modeling of patterns by means of ontologies, but only for a set of patterns, i.e. the usability design patterns. A metamodel for design pattern language in proposed in (Henninger and Ashokkumar, 2006). In (Kampffmeyer and Zschaler, 2007) a *Design Pattern Intent Ontology* (DPIO) to formalize relationships among *Gang of Four's* (GoF) patterns is proposed, the ontology is used to suggest a pattern to solve a given design problem. An Extended version of DPIO ontology to suggest patterns to solve integration problems is in (Harb, Bouhours, and Leblanc, 2009). Formalization of web design pattern is modeled in (Montero, Díaz, and Aedo, 2003) using ontologies. A pattern scanner for the Java language based on a OWL design pattern ontology is proposed in (Dietrich and Elgar, 2005) for recognizing patterns in source code. In (Bakhshandeh et al., 2013) an ontology for modeling the enterprise architecture domain is proposed, while an ontology to model architectural design decisions is described in)(Kruchten, 2004b). More recently, in (Guizzardi et al., 2014) NFRs are defined from an ontological point of view and a language is proposed to model them. Differently from our approach in (Guizzardi et al., 2014) there is no relation with patterns and their families. The authors identify the fuzziness of NFR specifications but their approach is to a more conceptual level. Analogously, the authors of (Rashwan, Ormandjieva, and Witte, 2013) define an annotated ontological vocabulary of NFRs with the main aim of classifying natural language sentences with reference to software specifications. Chi-Lun Liu proposes in (Liu, 2010) a modeling of NFRs that mixes ontologies with rules with the goal of catching inconsistencies in information systems specifications. Unfortunately, neither the overall modeling does not take into account the fuzzy nature of NFRs nor considers relations with architectural patterns. The same issues can be found in (Dobson, Hall, and Kotonya, 2007) where an ontology to model NFRs is proposed.

Design patterns and NFRs modeling. In (Mikkonen, 1998) features of design pattern are modeled using formal methods that capture temporal properties. In (Taibi and Ngo, 2003) a language named *Balanced Pattern Specification Language* (BPSL) is proposed for the specification of behavioral features and structural aspects of patterns (Tichy, 1997). The language derives from Temporal Logics of Action and First Order Logic. In (Harrison and Avgeriou, 2010a) the authors study relationships between design pattern: they consider application of pattern and tactics by using an entity diagram for annotating information about used tactics. Use of NFRs is studied

in (Chung and Prado Leite, 2009) and in (Botella et al., 2001). In (Glinz, 2007; Franch, 1998) several definitions of NFRs are probed. The work (Mylopoulos, Chung, and Nixon, 1992) describe the use of NFRs with process or product oriented approaches. Several automated tools have been proposed for supporting the enterprise architect in architectural modeling, such as (Jansen et al., 2007; Capilla et al., 2006; Tran and Chung, 1999; Diaz-Pace et al., 2008).

Eventually, relationship between patterns and NFRs has been addressed in (Rosa, Cunha, and Justo, 2002) that defines the policy to specify how an attribute affects the quality of non-functional properties. (Gross and Yu, 2001) studies the relationships between NFRs and design patterns. A framework to formalize relations between patterns and tactics is proposed by Harrison et al. in (Harrison, Avgeriou, and Zdun, 2010) to study the impact at implementation level. The impact of a tactic on quality attributes is described in (Harrison, Avgeriou, and Zdun, 2010).

3.10 Conclusion

In this chapter was proposed as main goal the development of a Decision Support System for supporting designers and software architect – having greater or lesser experience – in the process of modeling a software system's architecture. Achievement of the proposed goal was supported by listing some useful research questions. We describe hereby how the five research questions have been addressed throughout the work. For the first two questions, Q1, and Q2 we studied state of the art concerning main categories of requirements – functional and non-functional – by posing great interest in NFRs. Answer to question Q1 can be found in the Paragraph 3.3 and in a more extended study in the related work in Paragraph 3.9, where the use of catalogues of NFRs was found as the more relevant approach. To answer question Q2, we reviewed some state of the art approaches for pattern categorization or classification. The results of this study is summarized in Paragraph 3.3, where pattern, pattern languages, problem area and frameworks available are cited; a more extended study is reported in the related work in Paragraph 3.9. The answer to question Q3 was derived from studying all the approaches to relate NFRs and pattern described in Paragraph 3.3: relationship between NFRs and reusable schema are introduced as the starting point of the work we developed. This study allowed us to build the theoretical framework of our approach that is the answer to question Q4 by defining the fuzzy ontology in which we catalog the NFRs, the Families and the Pattern according to catalogues and categorizations found in the state of the art descriptions. By defining the *Covering Answer Set* algorithm for retrieval of pattern from the fuzzy ontology we solved the problem of facilitating the decision making problem in architectural design.

Question Q5 led us to implement and validate the decision support tool described in this chapter through the use case scenario and preliminary performed experiment on the method.

Main strengths of our approach is the use of *fuzzy ontologies* for modeling Knowledge that relates NFRs with patterns and with the Families they belong to. The fuzzy nature of the adopted language makes it possible to express and reason with concepts like "*the Proxy pattern has a good load balancing level*" or that "*the load balancing level is directly proportional to the level of reliability*". The proposed formalism and the reasoning service was implemented in a tool for supporting the human expert, the architect or the designer to select a set of patterns that matched the desired requirements. To this end, we have also defined a novel reasoning task able to retrieve a

ranked set of patterns that match the user requirements expressed in terms of NFRs. The use of a formal approach inherently guarantees the correctness and consistency of the data entered and the trust of the approach. The reasoner used to build the ontology is able to detect and prevent inconsistent data and incongruences. At last, we have presented a use case and have developed a Fuzzy OWL 2 knowledge base describing 28 pattern families, 37 Non-Functional Requirements, 109 design patterns and their relations. The use case was experienced using the tool by an expert whose solution is provided in the work and by a number of teams whose solution were compared with those provided by the expert supported by the tool. Results proved that solutions proposed by the tool-supported teams were more efficient in term of elapsed time and similarity.

Chapter 4

Context-aware Middleware for the Internet of Things based on fuzzy rules and reflective model

4.1 Introduction and Motivation

One of the challenges in building an Internet of Things (and Services) Middleware is the way software has to be developed and composed on the top of flexible infrastructures and integration architectures.

In this direction, the strongest challenges still regard the way a smart IoT-based architecture is designed in order to make it robust with respect to the contextual changes it continually undergoes. In fact, we all know that the physical world is not a static environment and it changes according to sometimes unpredictable events. Then, informative objects disseminated in the physical world should be able to adapt and change their behavior by following the surrounding context. New software composition paradigms and patterns that deal with heterogeneity, dynamicity, and adaptation need to be introduced thus paving the way to adaptive architectures. In order to be as effective as possible, the intelligent objects in IoT solutions are usually coordinated by a middleware that acts as a facilitator for a smoother and homogeneous communication among the various components.

The design of IoT middlewares has become a popular research area and refers to three main problems in developing distributed applications: the complexity of programming interprocess communication, the need to support services across heterogeneous platforms, and the need to adapt to changing conditions.

Traditional middleware (such as CORBA, DCOM, and Java RMI) addresses the first two problems to some extent through the use of a "black-box" approach, such as encapsulation in object-oriented programming. However, a traditional middleware is limited in its ability to support adaptation. To address all the three problems, the notion of adaptive middleware has been proposed.

In this chapter we propose a reflective model whose aim is to inject context-awareness into an IoT middleware. The reflective extension allows a software system to dynamically change its logic without internal changes to the code. In our approach, the awareness of the surrounding context is encoded by means of a rule-based system which drives the dynamic behavior of the middleware. Rules are interpreted in a fuzzy manner in order to make the middleware adaptive and flexible also to slight changes in sensor's data. A use case scenario of a Smart Home has been experimented to check the rule-based model on a real implementation of an IoT middleware.

This chapter is structured as follows: in the next Paragraph we introduce background technologies concerning IoT. Then we formalize the model to build the reflective architecture and later we show a practical example in a use case scenario. Experiments and results are discussed in Paragraph 4.4.3. After a related work Paragraph, conclusion and future work close the paper.

4.2 Background

Supporting adaptation and evolution is the key to long-lived applications.
In this direction, a key solution is represented by the Reflection architectural pattern (Buschmann et al., 1996c). It objectifies system's details about structure and behavior and enables applications to use or control functionalities of other applications without having a built-in knowledge of their behavior. Reflection, also known as computational reflection, was originally introduced by B.C. Smith to access and manipulate the LISP program as a set of data in execution (Smith, 1982). Starting from this seminal idea, we see how a IoT middleware can surely benefit from the use of a reflective approach. As an example, we have the possibility of designing a completely configurable system which results adaptable to different operating environments.

The main concept in the Reflection pattern is the distinction between the *base-level* and the *meta-level*. A base-level includes the core application logic and its run-time behavior is observed by a meta-level that maintains information about selected system properties to make the software self-aware. Changes to information kept in the meta-level thus affect the subsequent base-level behavior (Buschmann, Henney, and Schmidt, 2007b). The adaptation of the meta level is performed indirectly with the help of a specific interface, the Meta-Object Protocol (MOP) which allows users to specify a change, check its correctness, and automatically integrate the change into the meta-level (Buschmann et al., 1996c). MOP also makes possible the change of connections between the *base-level* and the *meta-objects*. A *meta-object* is an object that creates, manipulates, describes, or implements other objects (including itself). Thus, a proper configuration of the base-level and meta-objects defines the behavior of an application (Maes, 1987). In addition, using a programming language that supports reflection, it is possible to change the structure of the objects themselves at run-time and then make the software system much more flexible.

4.3 A Formal Model to Design a Reflective IoT Middleware

In this paragraph we provide a Formal model for a reflective middleware in a IoT setting. Without loss of generality, we assume that objects sense contextual dimensions and return the corresponding data. From now on, we will use $S = \{s^1, s^2, ..., s^n\}$ to denote the set of objects/sensors in the network and with $D^i = \{d^1, d^2, ..., d^n\}$ the domain of the data returned by the i-th object. As an example, in case we have s^1 being a temperature sensor we may have the corresponding domain $D^1 = \{-25, \ldots, +40\}$. We do not impose that different objects must have different domains. In our network we can have diverse temperature sensors whose domains contain the same values.

The dynamic behavior of the middleware we present, relies on the definition and usage of rules of the form ***given*** *a condition evaluated as true* ***perform*** *an action.*

Definition 1 (Condition – Syntax and Semantics)**.** *Given a set* $\overline{S} = \{s^1, \ldots, s^k\} \subseteq S$ *of objects, an* ***Atomic Condition*** *is defined as the relation* $C_a \subseteq D^1 \times \ldots \times D^k$*.*

Syntax. *Conditions* $C_1, \ldots, C_m$ *are defined recursively as in the following*

$C_1, \ldots, C_m \rightarrow$	C_a	ATOMIC CONDITION
	$\neg C_i$	NEGATION
	$C_i \wedge C_j$	CONJUNCTION
	$C_i \vee C_j$	DISJUNCTION

Semantics. *Given a tuple* $t = \langle d^1, \ldots, d^k \rangle$ *we say that the* ***Atomic Condition*** $C_a \subseteq D^1 \times \ldots \times D^k$ *is evaluated as true in* t *if* $t \in C_a$*. A Condition is evaluated as true if*

ATOMIC CONDITION	C_a *is evaluated as true*
NEGATION	C_i *is not evaluated as true*
CONJUNCTION	*both* C_i *and* C_j *are evaluated as true*
DISJUNCTION	*either* C_i *or* C_j *are evaluated as true*

Given a condition $C \subseteq D^1 \times \ldots \times D^k$ *and a tuple* $t \in D^1 \times \ldots \times D^k$*, we may introduce a function* f *defined as*

$$f(C, t) = \begin{cases} 1 & \text{if } C \text{ is evaluated as true in } t \\ 0 & \text{otherwise} \end{cases}$$

The simplest Atomic Condition involves one sensor only. As an example, suppose sensor s^i detects changes in concentration of pm_{10}[1] in the air. A simple atomic condition can be represented as $pm^i_{10} > 40$. Stating that the condition is evaluated as true if the value of pm_{10} sensed by s^i is greater than 40. Analogously, having the objects s^j and s^k designed to retrieve humidity values h we can define a condition involving the three corresponding domains as in the case of

$$(pm^i_{10} > 40 \wedge h^j < 15) \vee h^j \geq h^k \tag{4.1}$$

In our model, once a condition is evaluated as true, it triggers an action that drives the reflective behavior of the middleware. Adaptation and evolution of the system are then driven by a set of rules.

Definition 2 (Rule)**.** *Given a set of actions* $\mathcal{A} = \{a_1, \ldots, a_n\}$*, a set of rules* $\mathcal{R}$ *is defined as a function associating a Condition* $C \in \mathcal{C} = \{C_1, \ldots, C_m\}$ *to an Action* $a \in \mathcal{A}$*.*

$$\mathcal{R} : \mathcal{C} \rightarrow \mathcal{A}$$

We use the notation $C \Rightarrow a$ *to denote the single rule* $r \in \mathcal{R}$*. Given a context* t*, the action* a *is executed if and only if* C *is evaluated as true in* t*.*

[1] The abbreviation pm_{10} identifies one of several fractions in which it is ranked the particulate: solid and liquid particles dispersed in the air with relatively small dimensions. These particles in the atmosphere are indicated by many common names: dust and soot for those solid, mist and fog for liquids.

For example, we may want to model a rule where in case the condition in Equation (4.1) is evaluated as true in a given context then we send a notification email. In this case, the action is *send email* and the corresponding rule is formulated as

$$(pm_{10}^{i} > 40 \wedge h^{j} < 15) \vee h^{j} \geq h^{k} \Rightarrow send\ email \tag{4.2}$$

In order to be effective, an IoT middleware must also react as quick as possible to contextual changes. Its behavior needs to mimic a real-time one. Unfortunately, performing an action may require a preparation time thus preventing the system to perform an action on time when a specific condition C results to be true with reference to the current context t. On the other side, before we reach t other contextual values are sensed by the objects within the network and, usually, we do not have big leaps while moving from a sensed context to the following one. Consider the simple condition $C = pm_{10}^{i} > 40$ for the sensor s^{i}. Before s^{i} senses the context $t = \langle 40.5 \rangle$ that makes C to be evaluated as true, the sensor also measures other values for pm_{10} which are reasonably close to 40.5. We may expect a series of pm_{10} samples like $(\ldots, 36.3, 37.0, 39.9, 40.0, 40.5)$ for which C can be considered "almost true" to a certain degree. The best way to model degrees of truth for a condition is by means of fuzzy logics Zadeh, 1965. Thanks to the introduction of membership functions like the ones represented in Figure 4.1 we may assign a truth degree to a condition C. As

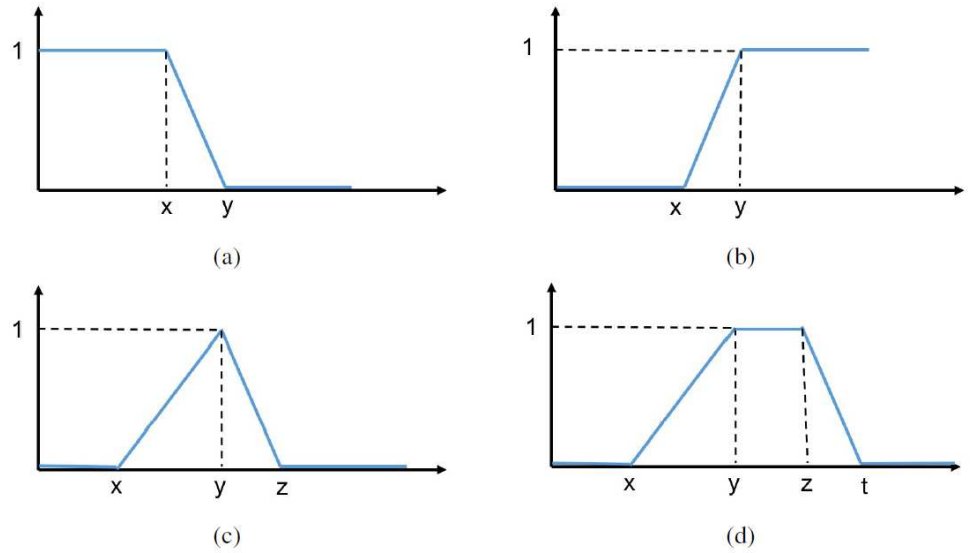


FIGURE 4.1: Fuzzy Membership functions. (a) Left Shoulder function $ls(x, y)$, (b) Right Shoulder function $rs(x, y)$, (c) Triangular function $tri(x, y, z)$, and (d) Trapezoidal function $tra(x, y, z, t)$.

an example, if we assign a *right shoulder* function to the domain pm_{10} with $x = 35.0$ and $y = 40.0$, once s^{i} senses a value equal to 37.0 we may say that C is evaluated as true with a degree of 0.4 while for a value of 39.0 we have a degree of truth for C equal to 0.8. Given the introduction of fuzzy membership functions we may define *Fuzzy Conditions* and *Fuzzy Rules*. As for the former we refer to the standard semantics adopted to evaluate the truth if fuzzy logic formulas involving Boolean operators.

Definition 3 (Fuzzy Condition - Syntax and Semantics). *Given a set $\overline{S} = \{s^1, \ldots, s^k\} \subseteq S$ of objects, an **Atomic Fuzzy Condition** is defined as the relation $C_{af} \subseteq D^1 \times \ldots \times D^k$ and a membership function*

$$mf : D^1 \times \ldots \times D^k \rightarrow [0, \ldots, 1]$$

Syntax. *The syntax of Fuzzy Conditions $C_1, \ldots, C_m$ is the same as for Conditions introduced in Definition 1.*
Semantics. *Given a tuple $t = \langle d^1, \ldots, d^k \rangle$ we say that the **Atomic Fuzzy Condition** $C_a \subseteq D^1 \times \ldots \times D^k$ is evaluated as true with a degree of $mf(t)$. A Fuzzy Condition is evaluated as true with a degree computed as in the following*

ATOMIC FUZZY CONDITION	$mf(t)$
NEGATION	$1 - mf(t)$
CONJUNCTION	$min(mf(C_i), mf(C_j))$
DISJUNCTION	$max(mf(C_i), mf(C_j))$

We use $dt(C, t)$ to denote the degree of truth of the Fuzzy Condition C given the context t.

We may see that the degree of truth for fuzzy condition extends the function f introduced in Definition 1.

Definition 4 (Fuzzy Rule). *Given a set of actions $\mathcal{A} = \{a_1, \ldots, a_n\}$, a set of Fuzzy Rules $\mathcal{FR}$ is defined as a function associating a Fuzzy Condition $C \in \mathcal{C}$ and an activation threshold $v \in [0, \ldots, 1]$ to an Action $a \in \mathcal{A}$.*

$$\mathcal{FR} : \mathcal{C} \times [0, \ldots, 1] \rightarrow \mathcal{A}$$

*We use the notation $C[v] \Rightarrow a$ to denote the single fuzzy rule $fr \in \mathcal{FR}$. Given a context t and a fuzzy rule, we say that the corresponding action is **activated** if $v \leq df(C, t) < 1$, while we say that the action is **executed** if $dt(C, t) = v$.*

A safe value for the activation threshold in a fuzzy rule is 0.5 but it can be set according to the workload needed to prepare the execution of the action a.

For the activation and execution of a set of fuzzy rules, we built an architectural model as shown in Figure 4.2 made up of software components described as follows.

Definition 5 (Message Broker). *Let $MP = \{MP_1, MP_2, ..., MP_k\}$ be the set of message producers and let $MC = \{MC_1, MC_2, .., MC_n\}$ be the set of message consumers. Given the formal messaging protocol of the message producer MP_i and the messaging protocol of the message consumer MC_i, the `Message Broker` MB is a software module able to translate and forward messages through a communication channel.*

A Message Broker (MB) is an architectural pattern for message validation, transformation and routing. Generally, MBs are components of a Message Oriented Middleware (MOM) (Curry, 2004) and provide content and topic-based message routing. As shown in Figure 4.2, the policy inside each Message Broker is based on a First In First Out queuing mechanism. Each queue represents a channel to which the message consumer can subscribe according to the Publish/Subscriber architectural pattern. A Message Broker can be used to manage a workload queue or message queue for multiple consumers. Generally, MBs are used to decouple end-points

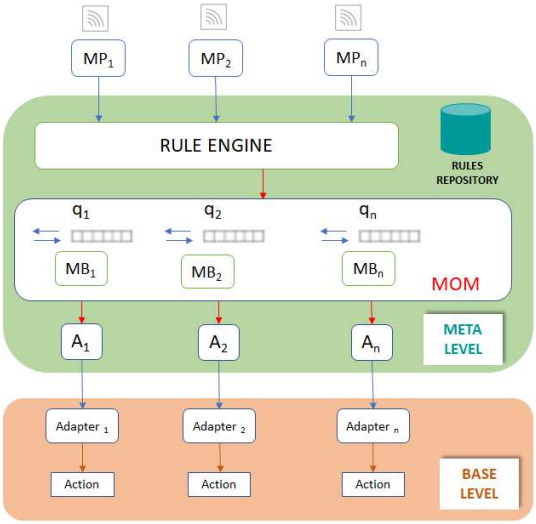


FIGURE 4.2: Architectural schema of the proposed model.

and/or meet specific non-functional requirements. The Publish/Subscribe mechanism is based on one-to-many and many-to-many policies and it allows a single producer to send a message directly to one or more interested consumers (Buschmann et al., 1996a).

Let *MB* be a Message Broker component that models the Publish/Subscribe mechanism for defining the relationships between physical sensors and actions; let us now give the following definition:

Definition 6 (Message and Message Bus). *A Message m is defined as $m = \langle s, t, A \rangle$ where s is the sensor, t the context sensed by s, while a is the action to execute. The Message Bus is the channel where the* MESSAGE BROKER *publishes messages from sensors while consumers subscribe to receive notifications of a relevant message.*

With reference to Figure 4.2, the RULE ENGINE models the reasoning algorithm that fires the fuzzy rules $\mathcal{FR}$. The ADAPTER component works as a driver and translates the received command in a real action. It is possible to have more ADAPTERS, one for each desired application. The abstract architecture models the reflection by means of the meta-level and the base-level as in the following. The meta-level is composed by the MOM with MBs inside, the rule engine connected to the Rules Repository and abstract Actions. The base-level is made up of the Adapters that triggers the real Actions.
The implementation of reflection mechanism is based on the following steps:

- Step 1: A Producer publishes a message m on a given *Message Bus*;
- Step 2: A fuzzy rule fr is executed;
- Step 3: Interested Consumers subscribe to the some *Message Bus*;
- Step 4: The related MESSAGE BROKER passes the message m through the *Message Bus* to the Consumers;
- Step 5: An action can be activated or executed.

4.4 Use Case Scenario

The abstract architecture we have presented can be instantiated in several contexts and scenarios. We now explain the formal model by instantiating it in a use case scenario: a smart domestic environment made up of IoT components, with the purpose of monitoring and automation. The use case has been implemented with a network of sensors to monitor environmental variables of one or more rooms inside an apartment. The devices have been placed along the entire home extension to monitor environmental variables (for example, temperature and air humidity, brightness of the room, CO_2 concentration, etc..) and variables related to the use of services (e.g. electricity consumption of the room, entertainment services, and so on).

4.4.1 Implementation

An implementation of the proposed architecture we used for our experimental evaluation is depicted in Figure 4.3.
Among the various IoT middleware available in the market, we choose DeviceHive mainly for usability reasons: ease of installation, documentation and high integration with a wide range of programming languages and IoT protocols. By using DeviceHive, the set up of a IoT system is made by the following three steps:

- Create: the user creates an instance of the *Cloud by DeviceHive*. There are instances of *Microsoft Azure*, *Juju*, *Docker* and *Cloud Playground*. In our implementation, we used the shell instance *Cloud Playground*.
- Connect: using the specific IoT Toolkit, a dedicated gateway is installed. This is the link between the devices and the cloud of DeviceHive. The gateway is written in Go, and in our case it communicates with a Python script that is responsible for receiving data from the sensors within the network.
- Visualize: sent data can be displayed via a Web page.

Redis as MOM. In our instantiation of the model we adopt Redis[2] as MOM. Although it uses the key-value paradigm, it exposes some characteristics that make it different from other databases in this category: it completely works in RAM, provides support to the storage of the key-value pair, offers four data structures: lists, sets, ordered sets and hash. Using Redis as MB allows the system to translate a message from the messaging protocol of the producer to the recipient's messaging protocol. Through appropriate Publish/Subscribe directives, Redis implements the mechanism of Publish/Subscribe, whereby producers do not send messages directly to recipients; messages are published and sorted into channels, without knowing who is really writing to the channel. Interested parties express their intention to subscribe to notifications of one or more channels of interest. This decoupling between publishers and subscribers guarantee a greater scalability and allows the system to manage a dynamic network topology.

Being a message well know as *a discrete unit of communication intended by the source for consumption by some recipient or group of recipients*, we give the following definition.
The instantiation for the *Message Bus* of our model is obtained by using the Redis

[2]http://redis.io/

Message channel. To implement the Publish/Subscribe mechanism, publishers publish messages on the Redis channel, recipients subscribe to the channel and read information about the available services.

The OBSERVER component in Figure 4.3 observes rules extracted from the Rules Repository. It notifies the MESSAGE BROKER (Redis) about the arrival of new data as well as of the active rule and its status (activation or execution).

We use Node JS as Event-driven component. It receives data about the sensor and the domain value of the sensor through a RESTful interface. The Observer and the Event-driven component instantiate the Rule Engine of our Formal Model.

Follows the formal definition of the rule.

```
 {
  "id": "string",
  "sensors": "array",
  "action":
[
 {
  "action": "array",
  "activation": "string",
  "execution": "string"
 }
],
  "condition": [
[
 {
  "attribute": "array",
  "relation": "string",
  "value": "string"
   }
]
}
```

In the instantiation, it contains a simple call to a generic function with the string identifying the methods and the classes of the base level. According to this mechanism, the logic of the program can be dynamically changed depending on the data got from the sensors and changes are transparent to the internal structure of the software. Within the message JSON received by Redis the class name and the name of the function to call are specified in special strings. The data to be written is contained in a separate object, which will be the argument of the function.

In order to determine which are the Actions available at runtime, a system for loading dynamic components within the same *Meta level* has been implemented. Two files for each *Action Levels* are available: a JSON file containing configuration data and a JavaScript file containing the class itself. Objects instantiated with the data in the JSON file will then be stored in an array, and referenced by the name of the same class, specified within each rule.

4.4.2 Reflective behavior

In this paragraph we analyze the control flow of the implementation in our use case scenario. The system automatically performs actions according to the values received by the sensors through the activation of a formal rule.

A configuration file contains the definition of the rules on the data and the actions to take if the rule is fired. The OBSERVER (of the data flow) notifies the NODE JS COMPONENT that forwards the extracted active rule to the MESSAGE BROKER. The active rule and its status together with the information about sensors and domain value is published on the *Message Bus*. On the message channel information about the arrival of new data is published. The MESSAGE BROKER works as a through for data flow and the active rule that are forwarded to the Reflective part of our component.

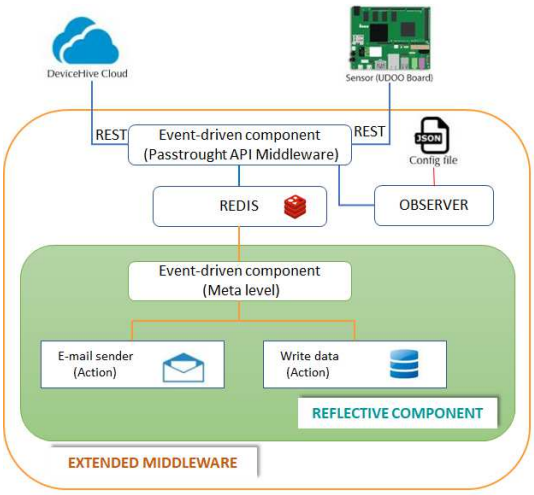


FIGURE 4.3: Implementation of the framework.

We may analyze the exchange of information between the various components involved in which the home environmental monitoring unit sends a message to the middleware with the following form:

```
{
"id":"EJH8rmcDl",
"rule":"send",
"rule_where":"email",
"data":
{
"device":" UDOO-QUAD-T_H_L-1",
"sensor": ["t_sensor",h_sensor"],
"timestamp":"1475353576",
"value":[19,40]
}
}
```

Let us now consider the case in which there is a decrease of domestic temperature due to a sharp decrease in external temperatures: the continuous-monitoring control unit sends the data to the middleware which in turn activates the necessary countermeasures for the resolution of the problem (sending an email or a notification on your mobile device). We can analyze the exchange of information between the various components involved. The monitoring unit sends a message to the middleware:

```
{ "device":" UDOO-QUAD-T_H_L-1",
"value":19,
"sensor":"t_sensor",
"timestamp":"1475353576"
}
```

The message reaches the middleware where it is published on the *Message Bus*:

```
{ "id":"EJH8rmcDl",
"rule":"write",
"rule_where":"mysql","data":
{"device":" UDOO-QUAD-T_H_L-1",
"value":19,
"sensor":" t_sensor ",
"timestamp":"1475353576"}
}
```

The consumers subscribed to the MESSAGE BROKER are then notified about the message. The MESSAGE BROKER forwards the whole message made up of sensor, domain value and the rule (extracted from the config file) to the Rule Engine that

executes it. Reflection is applied thanks to the signing of the consumer component of the Redis channel. This component reads the published messages that contain specific information on the method to be called, the class of which the function is a member and the topics. The *Meta Level* enables the actions of the reflective component. The *Base level* contains, among other specifications, a simple call to the generic function (employed only by identifying strings function and class) whose aim is, in this case, to store in a MySQL database the detected data since it exceeds the threshold more than the value contained in the rules repository. In this case, the rule is as follows:

```
{
"id" : "UDOO-DUAL-T_H_L-1",
"sensors" : ["t_sensor","h_sensor","l_sensor"],
"action" : [
{
"action":['write']
"activation":"database",
"execution":"mysql"
}],
"condition": [
[
{
"attribute":"t_sensor",
"value":"is cold"
}
]
]
}
```

In this case the system checks if the temperature value is included in a cold range of temperature defined by a trapezoidal function according to fuzzy logic. In this example the condition relates to one single sensor, but it can group multiple conditions from different sensors using Boolean operators. In our current implementation, the activations are managed via a plugin system.

4.4.3 Validation of the model and Experiments

We validate our implementation in a controlled environment. Two stations were placed in two different houses, each one with three sensors: humidity, temperature and brightness. To validate the middleware in a distributed environment, the entire core of the middleware including the DBMS was placed on a cloud architecture. Experiments were performed using a Linux virtual machine with Ubuntu 16.04 (single core, 2 GB of RAM, 25GB HD space) hosted on the Microsoft Azure Cloud infrastructure. A web panel allowed an external user to control and monitor of data sent to the middleware. We tested all the software components of the middleware on a continuous flow of data from sensors. The activation of the boards and continuous detection of environmental parameters lasted 10 days. The flow of information went from the cards that produced data placed in the two different houses to the middleware then towards the Cloud of DeviceHive. The core of the entire model is the stream of data that starts from the IoT board, passes through our middleware and reaches the DeviceHive Cloud. We monitored and evaluated the data flow from the sensor units to the middleware, according to rules encoded in the rule manager of the formal model which has been instantiated.

Experimental field

We performed intensive tests on the IoT middleware with a focus on the evaluation of the following NFRs: scalability and quality of service (QoS). From the evaluation of these two NFRs we indirectly derived evaluations about other requirements, i.e. accuracy, reliability, context-awareness and strength of the proposed method.

The tool can be subject to any type of working load increase: as an example let us consider the simultaneous forwarding to the middleware of a large amount of data collected from several board physically located in different environments. Hence, scalability issues need to be taken into account. Scalability refers to the ability of a system to increase or reduce its performance depending on its requirements and availability. Two types of scalability are known in literature: vertical and horizontal ones. Vertical scalability concerns with changes in the hardware with more powerful components (e.g. replacement of a processor with a more powerful one, increase in the amount of RAM, etc.). Horizontal scalability allows the user to insert into the system other components similar to those already instantiated with the aim of distributing the workload [3]. Vertical scaling has been excluded because the most delicate component is the Load Balancer. We considered workload scalability, i.e. the ability of a system to increase its performance depending on the load it is subject to when the system is provided with new resources to scale on, for example entire virtual machines. To this purpose, we consider workload as the quantity of simultaneous requests to the system for exploiting services: e.g. the workload is measured as the quantity Q of objects that simultaneously make a HTTP GET request to display a Web page to a given web-server. The number of allowed requests is computed based on the hardware capacity of the server: useful parameters to determine this parameter is the number of available CPUs, the amount of RAM that can be allocated and the available output bandwidth. Generally, computer systems are designed to support a specified quantity of simultaneous connections; once this load has been overcome, it is necessary to allocate new resources. In our implementation, for the two stations we used we performed experiments to test both quantitatively and qualitatively the performance of the middleware. First of all, we aim to state whether the implemented middleware can scale workload without reducing its performance. Generally, under ideal conditions this means that an additional load requires only additional resources rather than an extensive modification of the entire middleware.

Test of Scalability

To address the NFR of Scalability, the middleware was appropriately modified by introducing new components. The main component we introduced is the *load balancer*. This is modeled as software component that allows to distribute the load between several available nodes thus facilitating not only the addition of new nodes to support the increase of the work load, but also introducing a higher *reliability*. Indeed, the crash of one node or its temporary unavailability does not compromise the data ingestion service but hijacks the workflow to other free nodes of the network.

The implemented load balancer acts at the application layer of the ISO/OSI model, and verifies with a predefined frequency the effectiveness of the target nodes; it also redirects the incoming request to the destination node according to a predetermined routing policy. Among the various routing policies available in the literature we choose the Weighted Round Robin which differs from the classic Round Robin algorithm for the introduction of a system of weights on the connecting edges between the load balancer and the node of interest. It is well- known that in situation where different resources are available between servers connected in a cluster it is recommended a weighted round robin that rebalances the weaknesses of the classical round robin. The Weighted Round Robin policy makes it possible to channel

[3] Performed tests allowed us to evaluate horizontal scalability.

TABLE 4.1: Weights of the Round Robin policy.

hardware features	Weights
1 CPU single Core 1 GByte RAM	0.5
1 CPU Dual Core 1 GByte RAM	1
2 CPU Dual Core 2 GByte RAM	2

requests according to the order of arrival preferring nodes with higher weight. The weight associated to the link is a real number selected based on the performance offered by the hardware performance of the node and by the quality of the link (Table 4.1 shows an example of network weights).

Example of load balancer configuration file with two node and two different weights is:

```
{"Algorithm": "RRW", "servers":
[{"name": "Node1",
"IP", "192.168.1.95",
"port": 3000, "w": 1},
{"name" "Node2", "IP",
"192.168.1.96", "port": 3000, "w": 2}]}
```

The *Algorithm* field in the configuration file allows the choice of the algorithm that will manage the load balancer. The server field contains nodes allocated with their respective weights. The experiments were performed on an architecture made up of virtual machines having Ubuntu Operating System with single core and 2 GB of RAM.

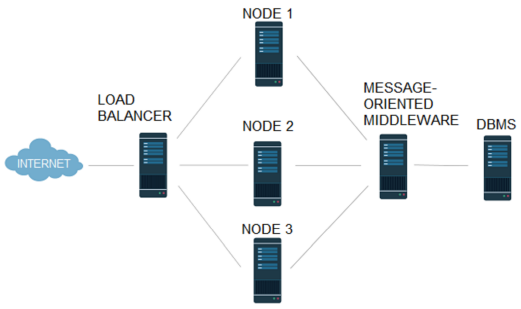


FIGURE 4.4: Network Architecture.

As shown in Figure 4.4, the entire system consists of the following six server nodes on which the components of the whole middleware are distributed.

- Node 1: Load Balancer
- Node 2 – Node 3 – Node 4: Observer and Rule Based
- Node 5: Message Broker REDIS
- Node 6: DBMS

To better test the use of a large amount of stations, we simulated the flow of multiple stations through a client-side script for massive sending of data to the middleware. The script simulates the sending of a given temperature value compatible

TABLE 4.2: Comparison of existing middleware with respect to desired requirements.

Middleware	Scalability	QoS	Cloud API
DeviceHive (https://devicehive.com)	X	X	X
ThingSPeak (http://thingspeak.com)	X	X	X
Zetta (http://www.zettajs.org)	X	X	
Kaa (http://www.kaaproject.org)	X	X	

TABLE 4.3: A table summary the result of scalability test with one, two or three nodes.

♯ of nodes	10 Simultaneous Data	20 Simultaneous Data	40 Simultaneous Data	80 Simultaneous Data	100 Simultaneous Data	500 Simultaneous Data
1 Node	14 seconds	27 seconds	51 seconds	108 seconds	132 seconds	631 seconds
2 Nodes	12 seconds	20 seconds	38 seconds	61 seconds	81 seconds	490 seconds
3 Nodes	11 seconds	17 seconds	29 seconds	42 seconds	60 seconds	385 seconds

with the input fuzzy rule and its activation. To develop a qualitative and quantitative evaluation of the performance of the developed middleware we compared its performance with those another commercial tool: *ThingSpeak*, a platform for the Internet of Things that allows the collection, storage and analysis of data from sensors to the Cloud.
Table 4.2 lists some of the available IoT middleware. Some of the characteristics that have been identified as priorities in the application under consideration have been evaluated: scalability, QoS and availability of Cloud API useful for the pass-through functionality of our middleware. The choice obviously falls in our case on the choice between DeviceHive and ThingSpeak for all the parameters evaluated but above all for the availability of a RESTfull endpoint and of a dashboarding system. The data is sent directly from the board to the Cloud of ThingSpeak using RESTful methods accessible and usable thanks to a token access to the API. The simulation, of the duration of 12 hours, allowed to send data from 3 sensors (temperature, humidity and brightness) at frequency intervals of 30 seconds. The license used (Free) posed restrictions on sending of data: we could not send below 15 seconds interval for sending continuously data with a computational interval of 20 seconds (Table 4.5 shows the comparison result). Anyway all the data has been correctly stored in the predetermined time.

Figure 4.5 shows the trend of the time-scale test (y-axis). On the horizontal axis it is possible to see the increase in the number of data sent and the corresponding complete transit time inside the middleware until the storage within the database. The series 1 (use of a single node) provides the time of ingestion of data without applying scalability. The second and third series show the use respectively of 2 and 3 input nodes and allow to better exploit the load balancer as a mean of coordination in the data injection. Table 4.3 summarizes the results of scalability test. Furthermore, in Figure 4.5 we can observe that by increasing the number of nodes on which we perform load balancing, for a fixed amount of sent data, we improve the throughput of the entire system: if consider for example 500 simultaneous deliveries – the last column in the picture– we see a 30% decrease of the time of ingestion.

This result was expected, in fact by spreading the workload across multiple servers it is possible to get a more performance-related balance of the load of the entire system. It is likely to think that translating the performed experience within a Cloud architecture having a powerful CPU, a more powerful network, more performing drives and a greater amount of RAM it should be possible to improve the obtained results. In fact, the experimental hardware has slowed down the system since the instances of Virtual Machines share the same CPU and the same hard disk.

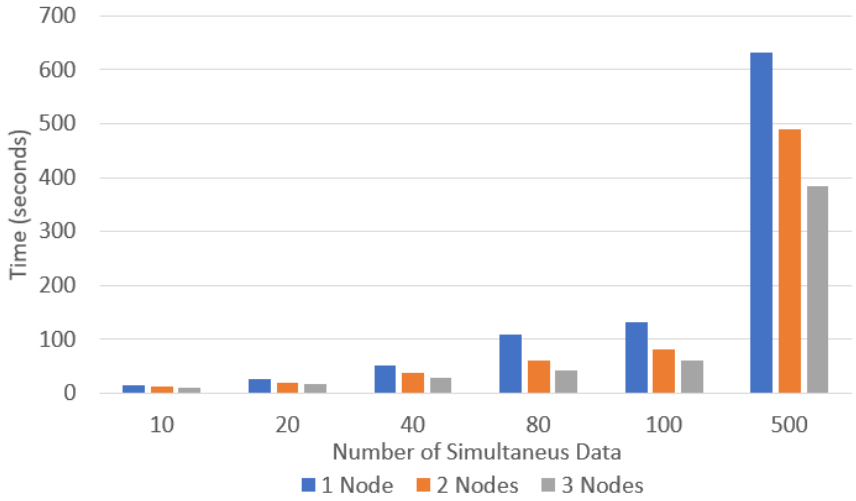


FIGURE 4.5: Results of Scalability test.

From the system's point of view, the bottlenecks are further represented by the Message Broker and by the DBMS. The bottlenecks could be overcame by introducing cluster instances with additional nodes to support the normal stand-alone instances of components. Besides, the use fuzzy-logic based rules has allowed a reduction in the size of the files (10 %) with the consequent reduction of RAM.

To sum up, the test on the scalability has shown that the implemented middleware provides excellent performance related to load distribution on multiple nodes in load situations.

QoS Test

In this paragraph we experimented the implemented middleware from the standpoint of the QoS requirement. First of all, let us dwell on some fundamental aspects relevant for the evaluation of the provided quality of service. We refer to the assessment of QoS at the application layer of the ISO/OSI stack. In fact, the experiment aims to validate properties of input data management and the trigger of actions as a consequence of the application of the rules. The architectural setting for this trial has changed w.r.t. the setting of the scalability evaluation. In fact, in this case we refer to the Cloud architecture. The entire middleware was embedded within a Docker instance on a cloud machine, thus instantiating the entire formal model. We conducted a two-phases evaluation: the first one was performed by varying the time of sending data to the middleware from 1 minute to 5 seconds for a period of 12 hours. The second phase of experiment was performed by introducing competition within the middleware.

In detail the experiment carried out: *(i)* an evaluation of data flow and operations coming from a board on which two sensors for the monitoring of three environmental variables (temperature, humidity and brightness) were installed; *(ii)* an evaluation of the data flow and operations arriving from two boards, each one monitoring three environmental variables (temperature, humidity brightness).

The first evaluation provided positive results on all the tests. Specifically, all the produced data were analyzed and stored within the database. The experimentation

TABLE 4.4: Table summarized the number of data sent, stored and lost in our middleware during the QoS test.

Frequency (seconds)	# Delivered	Lost
60	2160	0
30	4320	0
15	8640	0
5	25920	0
5 (Two Boards)	51840	0

TABLE 4.5: Comparison between our reflective middleware and ThingSpeak middelware.

	Our Middleware	ThingSpeak
Stored data	8640	5800
Delay	About 0	About 5 seconds

proved that the data flow passed from 2160 data records produced in 12 hours to a total of 25920 data records in the 12 hours without loss of information.

The second evaluation aimed to increase the number of input data and create a situation of competition within the middleware: transmission of data from the two stations is independent from one another, this creates situations of competition both in input and in performing reflective actions. This experiment had positive results since during 12 hours of testing the totality of the data was analyzed and stored by the middleware; this raised the number of writings of DBMS to 51840. Table 4.4 summarizes the result of QoS test.

The previous experiment proved that our Middleware is able to support various work loads at different transmission frequency and concurrent ingestion proving to be reliable. Reliability was ensured also by the component and methodologies used in the middleware construction: for example the used MoM proved to be robust and flexible in work load management and without affecting the performance of the virtual machine.

The comparison of the behavior of our middleware with the commercial one proved that the use of reflective middleware allowed storage of all the data without a medium delay time. This last aspect is not relevant in scenarios where data have a low transmission rate (e.g. the air temperature), but it is important in real-time control production scenarios which need a continuous stream of data without delay to provide a decision support.

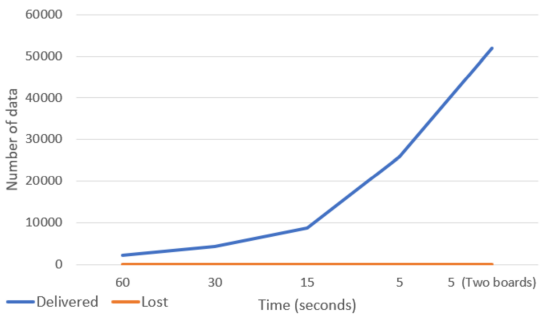


FIGURE 4.6: Results of QoS test.

Finally, result of QoS test as depicted in Figure 4.6 shows that the implemented middleware provides a high QoS in working conditions of processes competition;

besides it allows to receive and storage information with a high transmission rate.

Discussion

We discuss hereby about results of the experiments performed in a smart home environment by the instantiation of our proposed approach to reflective middleware in IoT environment.

The formal model was thought starting from the taxonomy of requirements we found in (Razzaque et al., 2016a), more precisely, also as outcome of the performed state of the art study, we selected some relevant architectural requirements, namely Adaptiveness, Context-awareness and Program abstraction. These emerged as powerful requirements to address the features of an IoT middleware, the interaction with physical objects which are becoming smarter and smarter with the ability to communicate with each other as well as with different information systems, the need of a new generation of objects able to adapt to external inputs coming from the environment they are dipped in.

We addressed the formal model towards satisfaction of program abstraction, formally and practically ensured by the reflection mechanism, the adaptiveness has been implicitly realized through the rules-based approach and the context-awareness guaranteed by the fuzzy rules to select and determine actions to be executed. It emerged that new modeling techniques, patterns, and paradigms for composing and developing software and services are needed to deal with changing context and requirements to give more flexibility and adaptability to the network behavior.

To validate the approach we considered the category of NFRs that an IoT middleware should ensure. Moreover, we selected in the category essential qualities that an IoT middleware should posses. More precisely, we identified Scalability and Reliability ensured also as a result of a desired Qos. The performed experiments proved desirable properties of the model in term of these two parameters. Experiments have been performed in a smart domain environment by setting an experimental field with stations monitoring sensors of temperature, humidity and brightness with variable requests coming from objects.

4.5 Related work

Different approaches are available in the literature of reflective middlewares. Hereby we recap some relevant approaches in chronological order: starting from the earliest to the most modern ones. One of the first approaches is in (Blair et al., 1999) that presents an architecture for reflective middleware based on a multi-model approach. Through a number of working examples, they demonstrate that the approach can support introspection, as well as fine- and coarse- grained adaptation of the resource management framework. In particular, the implemented reflective architecture is supported by a component framework, offering a re-usable set of services for the configuration and re-configuration of meta-spaces. In general, re-configuration is achieved through reification and adaptation of object graph structures.

In (Cazzola et al., 1999) the authors present an aspect of Architectural Reflection called Strategic Reflection, which is an extension of classic reflection to the software architecture level. The basic application of this extension allows the designer a systematic and conceptually clean approach to the development of systems with self-management functionality which also supports such functionality to be added to an existing system without modifying the system itself. The system's strategy is

described by a set of rules.
DART, a Reflective Middleware for Adaptive Applications is presented in (Raverdy, Gong, and Lea, 1998). The project clearly separates each aspect of the application in a distinct entity. Application behavior, properties and needs, environment features, and adaptation policies are designed independently. Interoperability is achieved by using adaptive events and the DART manager. Our framework is compatible with this idea, anyway we use adaptiveness and the more relevant aspect we introduce is fuzziness in modeling rule of reflective behavior. As previously stated, w.r.t. to these early works, our approach is based on a combination of novel reflective technique with adaptive mechanisms.

More recent relevant approaches of reflective middleware are in SOAR (SOA with Reflection)(Huang, Liu, and Mei, 2007). The authors define a two-level meta model which describes the basic characteristics of a SOA system, captures the threats to dependability and identifies the adaptations for preventing or recovering service failures. Based on the meta model, they empirically evaluated the features of dependability mechanisms for service consumers, service brokers, service providers and message passing and then define a set of equations to precisely calculate the factors for making a correct tradeoff. SOAR is implemented in PKUAS, a reflective J2EE application server with support to web services. The work in (Blair, Coulson, and Grace, 2004) surveys and evaluates three generations of reflective middleware research at Lancaster University. In the "depth"dimension, they believe that there is great potential in developing future systems that are "vertically integrated"and can be seamlessly inspected and adapted as a unified "pool"of component-based functionality. Once again, our reflective approach is compatible with this idea, anyway the work is limited to three distinct generations of middleware developed at Lancaster, while we introduce an innovative approach in our model definition.

In (Ikram et al., 2013) the authors present a chemical reaction-inspired computational model, using the concepts of graphs and reflection, which attempts to address the complexities associated with the visualization, modelling, interaction, analysis and abstraction of information in the IoT. At the same time, the work also highlights the fact that there are not many higher-level languages and middleware tools for chemical computing. The authors simulated and observed that the linear timelines can be inefficient and time consuming in complex scenarios, suggesting that an alternative can be to explore peer-to-peer architectures to realise the collection, generation and propagation of Social Smart Spaces (SSss).
W.r.t. these approaches, we start from the related perspective, define a high level and abstract meta model than hence can be tailored on different platform, mainly IoT-based, easily extensible with different IoT middleware. So our approach is more recent as far as the supported technologies is concerned; it also is flexible and can be easily extended being supported by an abstract model driven by a fuzzy rule engine.

A more recent stream of research is focusing on leveraging reflective and adaptive middleware for the IoT domain. A survey of reflective middleware for Iot is in (Perera et al., 2014). The survey addresses a broad range of techniques, methods, models, functionalities, systems, applications, and middleware solutions related to context awareness and IoT. The paper analyzes, compares and consolidates past research work by adopting the following objectives: *(i)* learn how context-aware computing techniques helped to develop solutions in the past, *(ii)* how we will be able to apply those techniques to solve problems in the future in different environments such as the IoT, and *(iii)* highlight open challenges and discuss future research directions. With respect to the idea of adaptive and reflective middleware for IoT we

introduce a fuzzy rule based abstract model to describe the sensor's data and their management depending on these rules. In (Vasconcelos, Vasconcelos, and Endler, 2014) the authors present a middleware that supports distributed dynamic software adaptation, in transactional and non- transactional fashion, among devices. They are focused in providing scalable management of coordinated and distributed dynamic adaptation, and facilitating the development of adaptation plans. The proposed idea is an adaptive framework, but is focused on mobile embedded systems. Our approach is compatible with this idea, anyway, we overcome the mobile perspective by proposing an approach that can be tailored on different platform; in fact it is focused on an IoT domain and can be easily extended to comply with different middleware. The awareness of the surrounding context is encoded by means of a rule-based system which drives the dynamic behavior of the middleware. Rules in our framework are encoded by means of a fuzzy interpretation that enables more flexibility and adaptability of the middleware and improves its context-awareness.

4.6 Conclusion and Future Work

In the recent years, there has been a huge effort to provide an immediate access to information about the physical world through Internet technologies. IoT vision aims to integrating the virtual world of information to the real world of things. The role of a IoT middleware is twofold. On the one hand, it aims to providing the connectivity between the virtual world and the physical one. On the other hand, it provides an interface between heterogeneous physical devices and applications. In this chapter a reflective extension of an IoT middleware which makes possible the design of a software resulting completely configurable and adaptable to different operating environments is presented. The proposed framework enables to automatically perform actions according to the sensor values received by triggering a set of fuzzy rules able to better describe environmental data w.r.t. to actions to be performed.

We validated the proposed model on a real IoT middleware in a smart home scenario performing intensive tests to evaluate non-functional requirements more specifically scalability and Quality of Service (QoS).

Improvements on the software side can be performed by expanding the range of external cloud services for making data flow from sensors to middleware in order to enhance the scalability of the system to other services. The scalability and the QoS tests has shown that the implemented middleware provides excellent performance.

We are currently working to extend the proposed framework with the addition of new Plug and Play actions; and to extend the framework for enabling it to the integration of several middleware. The idea is to build an "adaptive wrapper"of middleware able to adapt the reflective middleware to different Clouds.

Part II

Mobile Software dimensional view

Chapter 5

A formal model for user-centered adaptive mobile devices

5.1 Introduction and motivation

This study was carried out starting from key research concepts within the self-adaptive systems and focused on some Research Questions(RQs) about the same class of applications as far as modeling and implementation are concerned. First of all we identified as key concepts the following factors emerging as the more responsible of changes at run-time: *1.* changes in state/context/environment; *2.* changes in requirements; *3.* user's habits and needs. The key concepts were analyzed by highlighting their role in runtime design and implementation. Table 5.1 presents the RQs that will be addressed and the Research Process(RP) that has been taken.

In this chapter we propose an approach to comprise relevant aspects of adaptation: knowledge about environment, context and user's habits and information extracted from external sources and sensors. A formal metamodel that uses an Action Repository with stored triples of actions and related pre and postcondition about state and context is proposed to model properties of a device. The metamodel is instantiated in real scenarios, by contextualizing the main elements, thus obtaining an adaptive mobile software.

The rest of the paper is organized as follows. Paragraph 2 introduces the proposed approach. Paragraph 3 instantiates the approach in two real scenarios: i.e., a proximity environment application, and an applications to dynamically modify the home screen of a smartphone. Paragraph 4 validates the approach on empirical setting experiments. Paragraph 5 discusses advantages of the proposed framework according to RQs. Paragraph 6 compares our approach with existing state of the art works. The last paragraph concludes the chapter and outlines future work.

5.2 Approach

5.2.1 Action Repository

We propose a formal approach to build runtime mobile applications based on a metamodel to support runtime modeling of adaptive applications. Modeling depends on behavioral/contextual changes and observable properties of the user's habits and profiles. The metamodel is made up of a control level where data extracted from sensors are transmitted to effectors. Sensors and Effectors are interpreted in the sense of the definition of the MAPE-K model (Huebscher and McCann,

TABLE 5.1: Research Questions (RQ) and Research Process (RP).

ID	**RQ**
RQ1	What are the main challenges in design and implementation and use of self-adaptive systems?
RQ2	How could changes in the state, in the context and in the environment involve the changeability of applications and which constraints on this factors can be significant?
RQ3	How could the user's behavior involve the changeability of applications?
RQ4	How could a formal framework improve the runtime approach to such application with respect to existing approaches (even formal or theoretical)?
ID	**RP**
RP1	Propose a theoretical framework for modeling key concepts about changes in environment requirements and user's habit and needs and defining a reasoning mechanism to manipulate the modeled knowledge.
RP2	Explore use case scenarios to implement applications from the theoretical framework.
RP3	Validate the approach through use cases and experiments with real users.
RP4	Compare solutions provided by the metamodel supported implementation with human proposed solutions to given domain use cases to assessing the usability of the proposed adaptation and prove how the proposed framework is relevant.

2008). Actions and tasks to be performed are derived from high-level properties, preconditions about the state and context.

Knowledge about the adaptive software and its environment is modelled using requirements such as: *if the user is near a restaurant, it's lunchtime, usually she eats Japanese food, then she will be informed about a Japanese restaurant nearby*.

The metamodel allows one to express runtime adaptation at behavioral level: apps, services or components may be runtime loaded, deployed and executed.

Definition 7 (States)**.** *Given a set of* State Variables $SVars = \{v_1, \ldots, v_n\}$*, and a set of corresponding domain values* $\{V_1, \ldots, V_n\}$*, a* State *is an assignment* $s : v_i \mapsto d \in V_i$ *that for each* $i = 1, \ldots, n$ *maps each state variable* v_i *to its current value* $s(v_i) \in V_i$*.*

An Action *can change a state* s *to a state* s'*, by changing the value of any number of state variables.*

For example, a state variable *gps* of a mobile device may record in a boolean value whether the GPS is ON or OFF, another may record whether or not the icon of an app appears in the Home screen, another may contain the present brightness of the screen, etc. Actions model the adaptations of the device, e.g.turn OFF the GPS and reduce the brightness of the screen (see below). State variables not affected by an action keep their values.

Contexts are defined in the same way as states, but with the crucial difference that context variables cannot be changed by actions; the device environment changes them exogenously.

Definition 8 (Contexts)**.** *Given a set of* Context Variables $CVars = \{w_1, \ldots, w_m\}$*, and a set of corresponding domain values* $\{W_1, \ldots, W_m\}$*, a* Context c *is an assignment* $c : w_i \mapsto d \in W_i$ *that for each* $i = 1, \ldots, m$ *maps each context variable* w_i *to its current value* $c(w_i) \in W_i$*.*

No action can change a context variable.

Intuitively, contexts model every condition of the device adaptation can have no effect on, e.g.instance, signal strength, battery level, geographical location, nearest street address, date/time, etc.

Actions are defined by their pre-conditions (conditions of application, that are verified before the action can start) and post-conditions (which change the values of one or more state variables).

Definition 9 (Pre- and post-conditions)**.** *We call* $\mathcal{P}$ *the language containing all possible preconditions. For every action* a*:*

1. *the* pre-condition *of* a *is a formula* $P_a \in \mathcal{P}$ *which is a Boolean combination of*
   - *comparisons of state variables* $(v_i \text{op } d)$*, where* $v_i \in SVars$*,* $d \in D_i$*, and* op $\in \{=, \neq, <, >, \geq, \leq\}$ *is a comparison operator;*
   - *comparisons of context variables* $(c_i \text{op } d)$*, where* $c_i \in CVars$*,* $d \in W_i$*, and* op *is as above.*
2. *the* post-condition *of* a *is a formula* Q_a *which is a* conjunction *of assignments* $(v_i = d)$*, where* $v_i \in SVars$*,* $d \in D_i$*.*

We require that $Q_a \models \neg P_a$*, so that once an action has been performed, its effects prevent the immediate re-application of the action.*

An action a *can be performed when the state* s *of the device in the context* c *makes its pre-condition true. After the action is performed, we denote the resulting state as* $s' =$

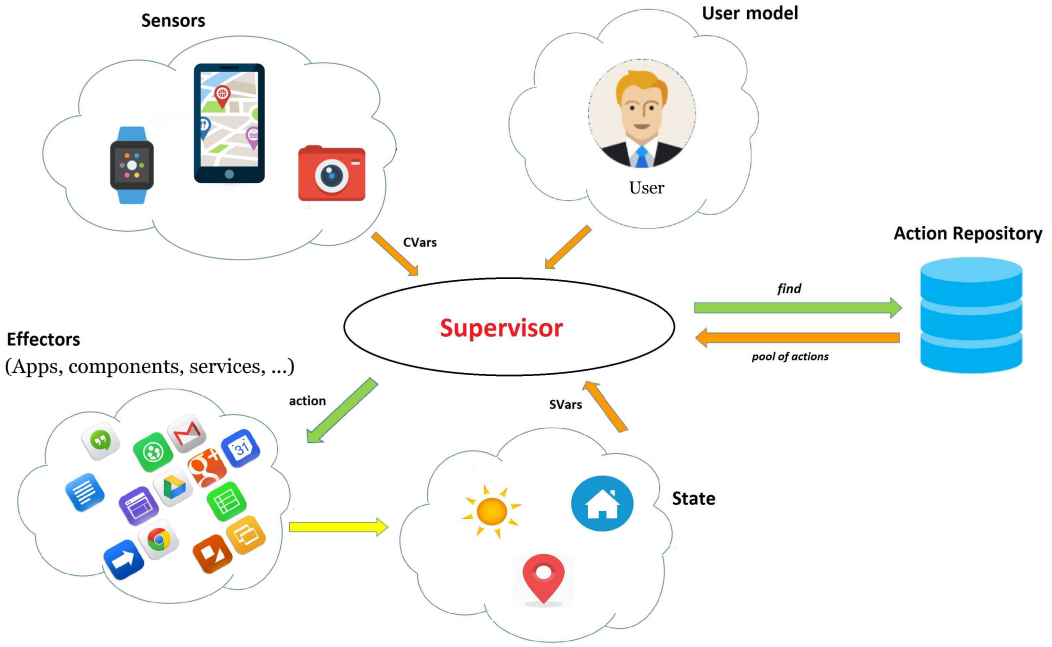


FIGURE 5.1: The proposed Metamodel.

s/Q_a, *meaning that variables not in* Q_a *keep their value, while every variable in* Q_a *changes accordingly.*

For example, a pre-condition could be $((gps = true) \vee (brightness \geq 0.7)) \wedge (batteryLevel \leq 1v)$—denoting some high-consumption state in the context of a low battery level—while a post-condition could be $(gps = false) \wedge (brightness = 0.2)$—which changes the state to a low-consumption one. Intuitively, pre-conditions model a boolean check on sensors, while post-conditions model the activation of effectors. Clearly, while sensors can be checked for a wide range of possible values at once with operators like $<, \neq, \geq$, effectors are modeled as deterministic actions with the only operator =, setting a definite value for a state variable.

We stress that pre-conditions are not triggers, that is, the satisfaction of P_a does not force a to be necessarily performed. In fact, when several such pre-conditions are true at the same moment, the adaptation will choose the most preferable action according to the user model, which will be discussed later on.

Definition 10 (Action Repository (AR))**.** *An* Action Repository (AR) *is a set of triples of the form* $\langle a, P_a, Q_a \rangle$ *where* a *is a (unique) name of an action,* P_a *is its pre-condition and* Q_a *is its post-condition.*

The Action Repository is the core knowledge of our knowledge-based runtime adaptive system, which comprises other parts as listed below.

Definition 11 (Adaptive Architectural MetaModel)**.** *An Adaptive Architectural MetaModel is a tuple* $AAMM =< S, AR, find, U, E >$ *where* S *and* E *are respectively the sensors and effectors,* AR *is an Action Repository,* $find$ *is an algorithm that finds a set of actions that might be applied in a given state and context, and* U *is the user's model.*

Figure 5.1 shows the metamodel of the proposed approach.

5.2.2 Personalized Action Selection

In most cases, a perfect match between the actual state and context and the ones required in the pre-condition is not to be expected. Given a state-context $\tilde{P}$ we need to evaluate if it is "similar enough" to the one specified in the precondition P_a of an action a. Given an Action Repository, we want to execute the action whose preconditions are more similar to $\tilde{P}$. Moreover, it would be advisable that the selection procedure behaves in a personalized way. That is, given $\tilde{P}$, the selection of the action to be executed may change depending on the user. Hence, when evaluating the precondition P_a of an action a, most comparisons in P_a are evaluated as *fuzzy conditions* in Fuzzy Logic (Zadeh, 1965). Indeed, due to their inner nature, it may result hard to evaluate a comparison of state variables in a Boolean setting. In such a setting, the comparison $(time = 12:30)$ would be infrequent considered to have truth value 1 only in the case in which the device time is exactly 12:30, even if *around* 12:30, say 12:25, the comparison still yields a degree of truth which is nearly 1—say, 0.8. Fuzzy logics may surely help in modeling such graded values of truth. They are based on the notion of *fuzzy sets* which are defined, simply put, as functions $f\colon D \to \{0,1\}$ assigning a grade (value) of truth $f(d) \in \{0,1\}$ to a certain value $d \in D$. With reference to the previous example, D is the domain of time and d is a possible hour. In Chapter 2 have been recalled the most common and used membership functions 3.1. The choice of the right function to be associated to a fuzzy set depends on D. Going back to our example, in some scenarios, it may result quite natural to select a *triangular function* with $y = 12:30$.

Given a precondition $P_a \in \mathcal{P}$ represented as a Boolean combination of state/context variables comparisons we now define how to evaluate its truth value.

Definition 12 (Interpretation)**.** *An **interpretation** $\mathcal{I}$ for $\mathcal{P}$ is a function $\cdot^{\mathcal{I}}$ that maps each comparison of state variables $(v_i \text{op } d)$ occurring in P_a to a truth value $(v_i \text{op } d)^{\mathcal{I}} = f(d)$ and, analogously, each comparison of context variables $(c_i \text{op } d)$ to a truth value $(c_i \text{op } d)^{\mathcal{I}} = f(d)$ with f being a fuzzy membership function. Given $P_a, P'_a \in \mathcal{P}$ we recursively define the interpretation of a formula as:*

- $(\neg P_a)^{\mathcal{I}} = 1 - P_a^{\mathcal{I}}$
- $(P_a \wedge P'_a)^{\mathcal{I}} = min(P_a^{\mathcal{I}}, P_a'^{\mathcal{I}})$
- $(P_a \vee P'_a)^{\mathcal{I}} = max(P_a^{\mathcal{I}}, P_a'^{\mathcal{I}})$

With reference to the above definition, given a set of preconditions $\hat{\mathcal{P}} \subseteq \mathcal{P}$ we can compute a total order among its elements by means of the interpretation functions. Indeed, given $P_a, P_b \in \hat{\mathcal{P}}$ we can always evaluate whether $P_a^{\mathcal{I}} \geq P_b^{\mathcal{I}}$ or $P_b^{\mathcal{I}} \geq P_a^{\mathcal{I}}$. Actually, an order among preconditions can be easily reverted to a ranking among the corresponding actions. In other words, if $P_a^{\mathcal{I}} \geq P_b^{\mathcal{I}}$ we assume a is more likely to be executed than b.

Definition 13 (Executable Action)**.** *Let $AR = \{\langle a, P_a, Q_a\rangle, \langle b, P_b, Q_b\rangle, \ldots\}$ be an Action Repository, and $t \in (0,1)$ be a threshold value. We say a is an **executable action** iff both there is no action b such that $P_b^{\mathcal{I}} > P_a^{\mathcal{I}}$ and $P_a^{\mathcal{I}} \geq t$.*

Since we deal with a total order ,we may have more than one executable action $a, a', a'', \ldots$. Indeed, it may occur that $P_a^{\mathcal{I}} = P_{a'}^{\mathcal{I}} = P_{a''}^{\mathcal{I}} = \ldots$. We see that as we do not have any order among $a, a', a'', \ldots$ we may execute any of them randomly. The

reason why we introduce the threshold t is to avoid situations where the executed action has a low truth value (which corresponds to a high untruth value). Given a state-context $\tilde{P}$, in case there is no executable action, the system does nothing until the next change in $\tilde{P}$.

At the end of Paragraph 5.2.1 we argued that, given a state-context $\tilde{P}$, the computation of the executable action should adapt to the user. With respect to the model presented in this paragraph, we can encode user preferences within the fuzzy membership functions. In fact, looking at Figure 3.1, we see they are defined in terms of a set of parameters x, y, z, t. By changing these values, we modify the shape of the functions. Let us go back to our example $(time = 12:30)$ and suppose we define the *fuzzy set* associated to time by means of a triangular function with $y = 12:30$. We may distinguish between an "always on time" user and a "more relaxed" one by setting, for instance, in the former case $x = 12:25$ and $z = 12:35$ while in the latter case $x = 12:00$ and $z = 13:00$. Hence, depending on the user, the truth value associated to P_a may change and then the possible selection of a as executable action. It is noteworthy that x, y, z, t can be either be set manually or be automatically learned by collecting information about the user's behavior.

The history of the user's behavior is stored in a repository through the values of the context and state variables describing the actions generally performed by the user and her preferences. A triangular function elicits the variables values to describe the user's behavior: for example, the history of the places she usually visits (the state variable is position), or the times he usually visits that places. The supervisor will choose among the pool of actions identified in the Action Repository the action that verifies the constraint with the threshold t with respect to the triangular function of variable in the precondition. In case of multiple properties in the precondition expressed with a fuzzy variables the minimum or maximum operator as specified in the corresponding fuzzy interpretation.

To make the formulas fuzzy, we can express the preconditions using intervals, that is, the precondition is not true for only one value of the formula, but for the values in these ranges. For example, the choice of points to be displayed on a map will not be shown only for an exact value of the radius of the area, but depending on user habits, this value can be included in a interval. For example, y is the current position, while the interval $[x, z]$ defines the length of the circumference of the diameter to be displayed (centered at y). In this case the triple is: PRECONDITION: $(gps = true)$ and $(radius > x)$ and $(radius < zx)$ and $(location = y)$ and $(now > 24:20)$ and $(Time < 12:40)$ and (number of high interest points) ACTION: Displays restaurants in the area of interest POSTCONDITION: at least one point is displayed.

5.3 Instantiation of the model

5.3.1 Proximity environment

The metamodel proposed in Paragraph 2 was instantiated in the domain of proximity environments. Nowadays, proximity is being considered as an added value of most applications, especially in social environments.

This phenomenon is widely observed spreading in the social sphere, thanks to the enormous spread of smartphones with GPS. Using the GPS connection, users of social networks can inform in every moment in which attraction, or local shop they are. When the user is in a certain place she launches the application from her smartphone, after the connection with the GPS has been made, she looks at the list

TABLE 5.2: Instantiation of the elements in the tuple *AAMM* for Scenario 1.

Element	Instantiation
Sensors	GPS device.
Action Repository	Codes of features to add / delete / edit points of interest displayed on the map tuples in the action repository contains propositionas as shown in the subsequent examples using state variables and context variables that are: *(i)* State variables: points of interest shown on the map, radius area on the map; *(ii)* Context variables: location, time.
Effectors	Implementation of the operations on the map (simple display or recommendation).
User Model	User habits (time lunch, sleep), the places already visited, how many times they were visited, preferences hotel / restaurant costs, type of medium supply, etc.
Algorithm to find selected points of interest to display	The first time the radius has a default value. From a history of visited sites, it is possible to trace how the user generally moves away and determine the radius of the area to display. The category is chosen based on the time and time zone.

of places in the neighborhood and the so-called check-in to tell users on her list of contacts where she exactly is.

In the field of proximity marketing, we developed AProM (Adaptable Proximity Marketing tool) a mobile app to enable spreading advertisements to end users according to their needs. The application starts and shows the map. Depending on the context (location, time) points can be displayed in different categories of interest. For example, restaurants at lunch time, hotels in the evening, etc. are displayed. The time ranges are decided depending on user's habits (the time usually she has lunch, goes to sleep, etc.). Besides, the area taken into account on the map and the points shown inside have a radius that varies according to the places already visited by the user.

5.3.2 Adaptive Architectural MetaModel instantiation

The instantiated metamodel exploits the *goals*, the objective that the user expects. Table 5.2 summarizes how the elements in the tuple *AAMM* are instantiated.

Effectors are instantiated as actions extracted from the Action Repository based on the *"find"* selection algorithm; they are mainly implementation of the operations on the map (simple display or recommendation).

The repository is conceptually separated, and can virtually be always on the device. Table 5.3 presents an example of the instantiated tuples of actions, precondition and postcondition of the Action Repository.

State and context variables are sensors in the external environments, context variables available on the user mobile device, events extracted from the sensors.

The architectural model derived from the instantiation of the metamodel defined in Paragraph 5.3.1 is shown in Figure 5.2.

AProM was developed on the Android platform and requires, as minimum supported version, Android Ice Cream Sandwich 4.0.3 (API Level 15), GPS Sensor, Internet Connection. The device on which the application is deployed is an LG Nexus

TABLE 5.3: Example of (Fuzzy) tuple for Scenario 1.

PRECONDITION: $(gps = true)$ and $(radius > 0)$ and $(time = 12:00)$ and (low number of points of interest) ACTION: Displays restaurants in the area of interest and suggest new points POSTCONDITION: at least one point is displayed
PRECONDITION: $(gps = true)$ and $(radius > 0)$ and $(time = 21:00)$ and (high number of points of interest) ACTION: Displays hotels in the interest POSTCONDITION: at least one point is displayed
PRECONDITION: $(gps = true)$ and (frequent changes of position) and $(radius > 0)$ and $(time = 21:00)$ ACTION: suggest new hotels POSTCONDITION: (at least one point is displayed) and (position change)
Example of PRECONDITION with fuzzy variables:
PRECONDITION: $(gps = true)$ and $(gps.precision < x_1)$ and $(radius > x_2)$ and $(radius < z_2 - x_2)$ and $(location = y_2)$ and $(time > 12:20)$ and $(time < 12:20)$ and (low number of points of interest) ACTION:Displays restaurants in the area of interest and suggest new points POSTCONDITION: at least one point is displayed
PRECONDITION: $(gps = true)$ and $(gps.precision < x_1)$ and $(radius > x_2)$ and $(radius < z_2 - x_2)$ and $(location = y_2)$ and $(time > 21:20)$ and $(time < 21:40)$ and (low number of points of interest) ACTION: Displays hotels in the interest POSTCONDITION: at least one point is displayed

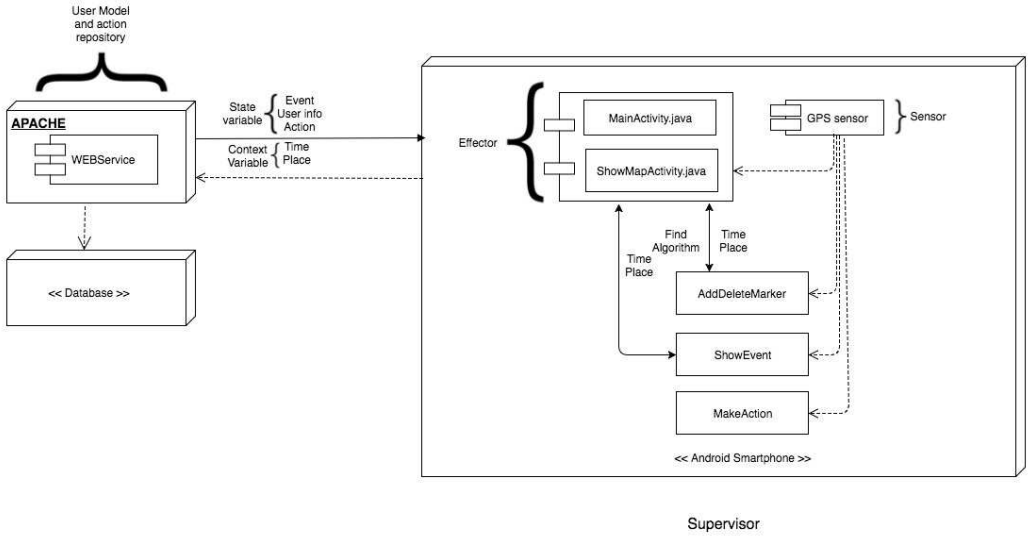


FIGURE 5.2: Architectural model obtained as an instantiation of the metamodel.

5 Android updated to version 5.1, which meets all the required specifications. The development environment is Android Studio, an open source tool for developing Android applications. We chose the Android platform to be able to develop the app without cost and to take advantage of some features of the operating system. AProM presents the typical structure of an application for Android; it is written in Java and XML, respectively, for dynamic and static parts.

Figure 5.3 shows a screenshot of AProM: the area where the user is localized is the circle in which points of interest are shown belonging to the category of food; the right side of the figure shows the list of categories that can be accessed by user for manual search or for inserting advertisements.

5.3.3 A 'smart smartphone'

The second instantiation refers to the implementation of a mobile application that manages dynamic characteristics of the homescreen of a smartphone, depending on the user's habits, by the position detected by GPS in which the user is currently located using the device, by the external context and by user's current behavior. The app, SmartSmartphone, monitors the user's context and shows a list of recommended applications depending on it.

5.3.4 Adaptive Architectural MetaModel instantiation

The instantiated metamodel exploits the *goals*, the objective that the user expects. Table 5.4 summarizes how the elements in the tuple *AAMM* are instantiated.

The repository is conceptually separated, and can virtually be always on the device. Table 5.5 presents example of the instantiated tuples of actions, precondition and postcondition of the Action Repository.

State and context variables are the apps installed on the device and time/position respectively.

TABLE 5.4: Instantiation of the elements in the tuple *AAMM* for Scenario 2.

Element	**Instantiation**
Sensors	GPS device.
Action Repository	Codes of functions to add / remove to display in your smartphone's home (conceptually separated, can virtually always be on the device).
Effectors	Apps displayed in the home of the device.
User Model	Usually visited sites, most used applications, time of use.
Algorithm to find selected points of interest to display	Depending on the place where the user is located, the time and most used installed apps, applications to display on the home device are selected.

TABLE 5.5: Example of (Fuzzy) tuple for Scenario 2.

PRECONDITION: $(gps = true)$ and $(position = home)$ and $(time = 12:00)$ and (most popular home $app = facebook, youtube, netflix, weather$) and (most used apps at 12 $a.m. = Netflix$) ACTION: Netflix displays in home POSTCONDITION: At least one app is displayed
PRECONDITION: $(gps = true)$ and $(position = home)$ and $(time = 7:30)$ and (most popular home $app = facebook, youtube, netflix, weather$) and (most used app at 7.30 $a.m. = weather, facebook$) ACTION: Displays weather and facebook at home POSTCONDITION: At least one app is displayed
PRECONDITION: $(gps = true)$ and $(position = office)$ and $(time = 18:30)$ and (most used app in the $office = spotify, chrome, email, Trenitalia$) and (most used app at 18.30 $= Trenitalia, email, Facebook$) ACTION: Displays Trenitalia and emails in home POSTCONDITION: At least one app is displayed

Example of PRECONDITION with fuzzy variables:
PRECONDITION: $(gps = true)$ and $(gps.precision < x_1)$ and $(radius > x_2)$ and $(radius < z_2 - x_2)$ and $(location = y_2)$ and $(time > 12:20)$ and $(time < 12:20)$ and (most used app at home= facebook, youtube, netflix, meteo) and (most used app at 12.00 $= Netflix$) ACTION: Displays Netflix in home POSTCONDITION: at least one app is displayed
PRECONDITION: $(gps = true)$ and $(gps.precision < x_1)$ and $(radius > x_2)$ and $(radius < z_2 - x_2)$ and $(time > 07:10)$ and $(time < 07:50)$ and (most used app at home= facebook, youtube, netflix, meteo) and (most used app at 12.00 $= Netflix$) ACTION: Displays weather and facebook at home POSTCONDITION: at least one app is displayed

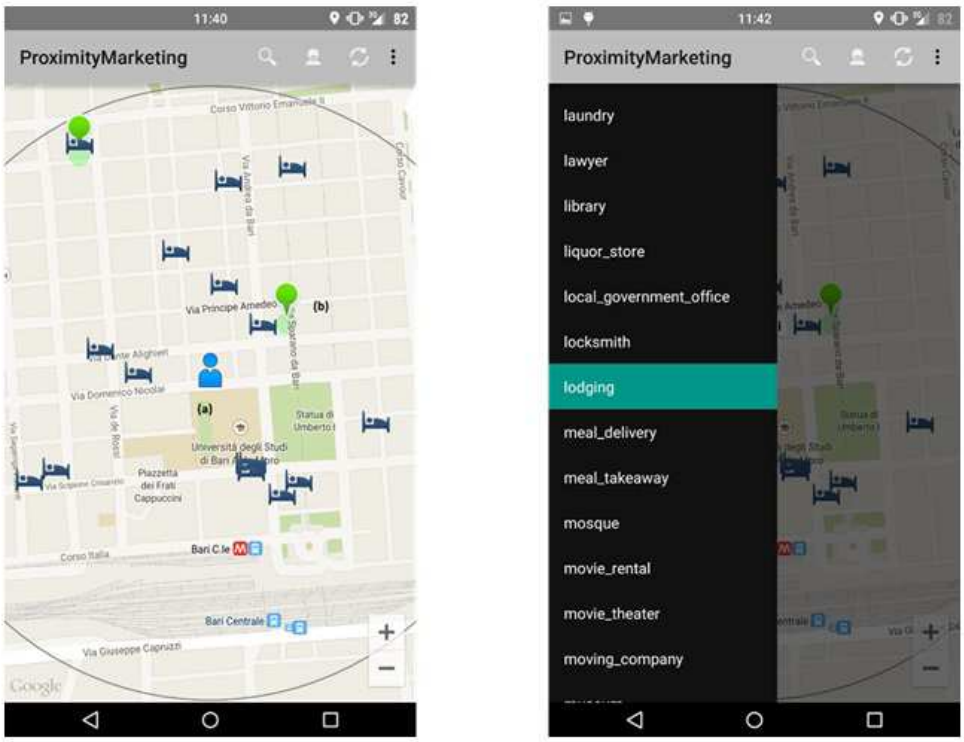


FIGURE 5.3: Screenshot of AProM.

The architectural model derived from the instantiation of the metamodel defined in Paragraph 5.3.1 is shown in Figure 5.4.

To develop this application we chose Android as guest operating system. We chose a native approach to model the architecture, which allows to exploit the system, but also the entire ecosystem associated with it. In fact, thanks to the choice of the Android operating system, it was possible to make use of Google Play Services, a Google proprietary system that runs in the background on devices and that provides developers a set of Application Programming Interface (APIs) that use on the one hand Google services and other hardware on the smartphone in a way transparent, to the developer.

The programming language used is Java. The Integrated Development Environment (IDE) used is Android Studio (version 2.0). This tool is released directly from Google and allows excellent integration with the Android ecosystem.

Figure 5.5 shows three examples of the smartphone home screens related to three different contexts: *home*, *travel* and *work*.

In the *home* context, apps providing functionalities related to the home living are directly available to the user, such as Skygo, Facebook; in the travel contexts apps such as Tripadvisor, Google Maps are more frequently used by the user and hence are available; in the *work* context the user needs apps such as Dropbox, email, Skype and Chrome.

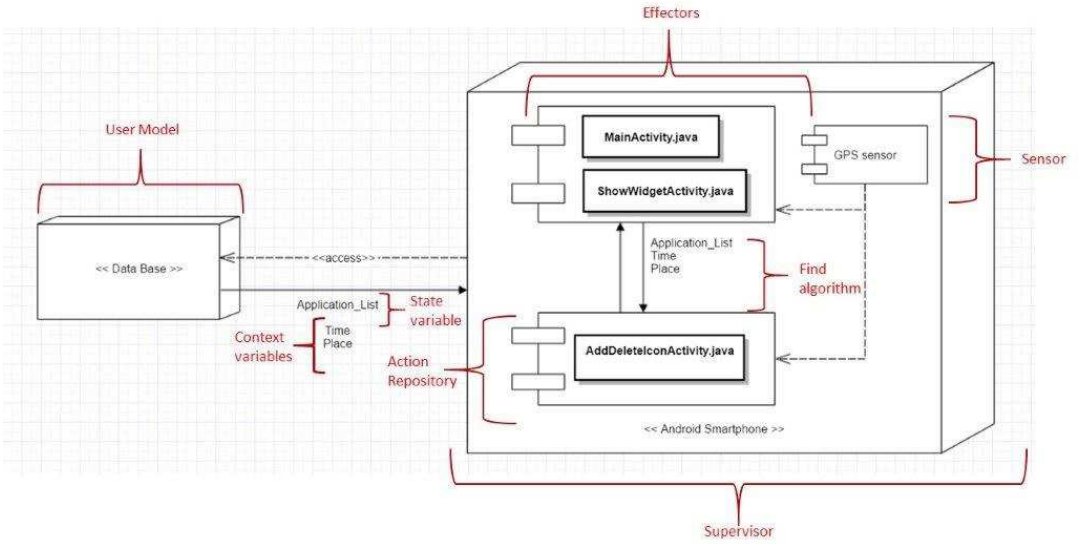


FIGURE 5.4: Architectural model obtained as an instantiation of the metamodel.

5.4 Experiments and validation

In this paragraph we describe a controlled experiment designed to perform a validation of the proposed approach. We aim to prove how the contribution of the proposed framework is relevant.

Two teams of students were assembled, each one composed of three second year graduate students. All students were trained on software design, architectural modeling, mobile and adaptive applications implementation.

The teams had to model and develop two adaptive mobile applications according to scenarios described in Paragraph 3 using only their own experience. Afterward the same teams had to solve the same problem supported by the metamodel, by mapping the main elements of the metamodel to the main elements of an architecture.

The solution to each problem was provided as an adaptive mobile application. A set of NFRs was formerly identified based on general purpose goals of an adaptive software. From the end user perspective we fixed usability, functionality, correctness; from the developer/designer perspective we considered maintainability, correctness, data access, adaptability. Anyway, the two solutions are not expected to be the same, because of details that derive from creativity, experience and from reasoning approaches. Hence, the measure of the goodness of the method was very complex to quantify.

We performed two different experiments: in the first one we compared the solutions provided by the two teams supported by the metamodel with these obtained using only the human experience. In the second experiment, real users evaluated their degree of satisfaction about adaptive functionalities of the applications obtained with/without support of the metamodel. Besides, we aimed to evaluate the impact of user's behavior on the system operation modeling.

FIGURE 5.5: Smartphone home pages.

We measured the advantage of using the metamodel according to designer/developers opinion in terms of: (a) level of difficulty in development for the developers; (b) assessment of requirements satisfaction – in percentage; (c) efficiency: time elapsed between the starting point and the solution.

Hence, a questionnaire was administered to each member of each team to compute the degree of satisfaction of developers for the metamodel supported development procedure that stands at a level of 96%. Table 5.6 summarizes results for the first experiment. The test was conducted as an usability test by considering developers/designers as final users.

In the second experiment a set of end-users was provided with the two applications developed without support of the metamodel, for a fixed period of time. Afterward the same experimenters were provided with the same application obtained by applying the metamodel during its development. At the end of the time period, a qualitative evaluation of the applications was required to the experimenters. Six students were chosen for testing the application. To conduct this experiment we used a slightly modified kind of think-aloud usability test (Lewis, 1982), and administered two questionnaires. The first questionnaire was about the degree of satisfaction expressed through a rating scale ranging from 1, strongly unsatisfied to 10 strongly satisfied.

The degree of satisfaction for metmodel-based applications stands at a level of 97%.

The second questionnaire was made on a System Usability Scale(SUS)-like schema. The questions were not statements about features related to expected non-functional requirements, and the answers had to be expressed through a rating scale from 1, strongly disagree to 5, strongly agree with the proposed statements.

Table 5.7 summarizes results.

5.5 Discussion

We describe hereby how the four research questions defined in the first paragraph of this chapter have been addressed throughout the work.

TABLE 5.6: Designers evaluation of the metamodel.

	elapsed time	% difficulty level	% requirements met with expected rate
metamodelsupported	2 weeks	73%	95%
	1 week and a half	97%	
experience-based	2weeks and a half	87%	89%
	2 weeks and a half	80%	87%

TABLE 5.7: Usability test with real users.

	with metamodel	without metamodel
appropriateness of functionalities w.r.t. the domain	90%	80%
compliance with the user's habits and preferences	86%	49%
correctness of the adaptive actions w.r.t. state and context	96%	50%
speed of response	60%	61%

To take into account the first two key concepts — changes in state/context/environment and changes in requirements — we posed the first two RQs in Table 5.1. To achieve a complete answer to these questions, we studied state of the art concerning main challenges in modeling and implementing self adaptive systems and selected two main categories of related work: *"Formal methods in modeling adaptive architecture", "Composition and interaction between applications in adaptive software."*
By considering the key concept 3 — user's habits and needs — we posed RQ3 in Table 5.1. To answer this question we reviewed the state of the art for user-centered approaches in modeling mobile and adaptive applications. The answer to RQ3 was derived from studying the previously cited related work and devising a formal method to relate the first two key concepts to the third one — state/context/environment changes, requirement changes to user's habits and needs. This issue was very challenging since none of the approach in the state of the art related the key concepts in a single model. This study allowed us to build the theoretical framework of our approach that is the answer to RQ4. We answered this RQ by defining a metamodel, a tuple that comprised all the elements and the relative relations. Each element of the metamodel can be instantiated in real application scenarios thus realizing an adaptive architecture on the fly.

The approach intrinsically satisfies some relevant general purpose requirements of an adaptive application. One of the granted requirement is correctness that is wired in the pre/post condition pairs whose definition is submitted to the constraint that once an action is performed, its effects prevent the immediate re-application of the action. Also, an action can be performed when the state of the device in the context satisfies its precondition. A second requirement is functionality that is elicited by the third element of the tuple, that is the action to be executed when the state and context requirements formalized in the precondition/postcondition are satisfied. The centralized control and management of the framework ensures maintainability, in fact the resulting application when instantiating the metamodel will have an application manager to coordinate the remaining elements of the tuple for the event based flow of action. As external qualities we identified a high level of usability due to the strong weight the metamodel gives to the user profiling.

5.6 Related work

Modeling and analysis of self-adaptive systems has gained increasing attention in the last years. Several approaches intend to face the main challenges of modeling such systems, hence there is a wealth of literature about studying, modeling and implementing this category of systems.

This study was carried out starting from key research concepts within the self-adaptive systems and focused on some research questions about the same class of applications as far as modeling and be concerned. We identified three main categories of state of the art approaches. With respect to these categories we studied shortages or possible ideas for improvement.

Formal methods in modeling adaptive architecture. Formal approaches remain the more powerful ones in order to ensure correctness and guarantee of quality properties. A complete study is proposed in (Weyns et al., 2012), while a survey of architectural modeling in self-management is proposed in (Kramer and Magee, 2007). Formal methods have been used in (Arcaini, Riccobene, and Scandurra, 2015) to model MAPE-K loops through a conceptual and methodological framework based on Abstract State Machines. Scandurra et al. (Riccobene and Scandurra, 2015) define a lightweight formal framework to express adaptive behavior of service components able to monitor and react to environmental changes (context-awareness) and to internal changes (self-awareness). Anyway these approaches are defined at abstract and conceptual level.

On the other hand, approaches that provide an architectural dimension are in (Pelliccione et al., 2008). The authors propose a Software Architecture based approach to perform code synthesis as an assembly of actual components which respect the architectural component behavior. The architectural framework Rainbow (Garlan et al., 2004) uses external mechanisms and a model to carry out adaptation actions at explicit customization points. The Genie approach ()Bencomo and Blair, 2009) manages structural variability of adaptive systems at architectural level and do not provide a way to guarantee desired properties of the systems after each adaptation execution. With respect to the previously described methods, we proposed a framework that is applicable at different levels of abstractions, both conceptual and architectural. A similar advantage is found in (Bucchiarone et al., 2015). The approach uses attributed graph grammars and define consistency and operational properties that are maintained despite adaptation. With respect to this graph based adaptation logic, we introduce a semantic elicitation of knowledge about environment, user's habit and behaviors.

Differently from all the previously described methods, in our framework we comprise both architectural and conceptual issues together with user's behavior and a semantic approach.

Modeling of user's behavior and requirements. User-centric approach for adaptive applications was proposed in Autili, Inverardi, and Tivoli, 2014 to model adaptation for Future Internet Application. Mobile user-aware root planner is proposed in (Chung and Schmandt, 2009). The method collects everyday locations elicited by the user's usual travel patterns.

A data-oriented, context-aware architecture is proposed in (Bolchini et al., 2011). The context is validated at run time. W.r.t. this model we introduce state variables to model changes in the system or in the device.

Besides modeling of user's properties in these approaches is related to a specific

context or environment or behavior, while we model user's behavior that is general purpose. The meta information is split into various categories of context-aware information modeled in the state and context variables.

Context-awareness modeling techniques are summarized in (Bettini et al., 2010). We point out that none of the existing techniques mixes up context, user's behavior and requirements.

A more recent stream of research is focusing on leveraging the new sources of information becoming available through ubiquity of systems, (Serbedzija and Fairclough, 2009). Our approach is compatible with this idea, anyway, we overcome a user's perspective, by considering a formal approach based on pre and postconditions.

To specify requirements, great interest is focused on goal modeling (Cheng et al., 2009).

Among the goal modeling languages, KAOS supports an LTL-based formalism and goals and requirements can be specified by pre- and post-conditions (Fickas and Feather, 1995).

Instead Non-functional requirements may be represented as soft goals or probabilistic patterns (Grunske, 2008). Fuzzy expression of requirements is adopted in (Frigeri, Pasquale, and Spoletini, 2014). Modeling and requirement-based adaptation has been proposed in (Pimentel et al., 2013). Compared with these approaches our method takes into account variation in requirements and variables that may influence the adaptation logic of application.

In (Zhang et al., 2011) the authors proposed an approach to monitoring non-functional requirements.

Adaptation is based only on requirements. Instead we use fuzzy properties to model user's behavior in conjunction with requirements specifications.

Compositional approaches in adaptive applications. Models of service choreography and composition is proposed in (Autili, Benedetto, and Inverardi, 2009). The deployed application is tailored w.r.t. the context at binding time. Composition of self-adaptive systems for dependability is proposed by Cubo et al. (Cubo et al., 2014). Compositional adaptation based on technological dimension is in (Philip, Eric, and Betty, 2004) without a semantic modeling and reasoning. Semantic approaches are proposed in Mongiello et al., 2015 to model self-adaptive architectural model in IoT and (Mongiello, Pelliccione, and Siancalepore, 2015) for runtime verification. We focused our model on the tuples of an action repository where preconditions and postconditions are linked to the actions to be performed, so we guarantee the adaptability also including the tuples to perform the action. If more actions are selected, the user's behavior discriminates on the actions to be performed.

Summarizing, we introduced a formal framework that is innovative with respect to existing approaches since it merges all the relevant changes in state or context, changes in requirements, user's behavior and user's habit and preferences, formalized in a fuzzy logical perspective.

Moreover it wires the correctness requirement in the structure of the metamodel by defining the tern precondition, postcondition, action.

5.7 Conclusion

In the Future Internet era, the way software will be produced and used will depend on the new challenges deriving from the huge number of software apps, component and services that can be composed to build new applications. To face the problem of

dynamic architectural modeling and of runtime composition of distributed complex applications, we proposed a semantic approach to define a metamodel for taking into account relevant aspects adaptation: context, user's habits and profiles, information detached from external sources and sensors. The use of a knowledge-based approach allows modeling and reasoning on complex adaptive software architecture according to changed behavioral properties or context variables. A research questions-based analysis of the state of the art demonstrates the improvements of the metamodel w.r.t. existing approaches. A controlled experiment was conducted with designers and developers. The developed applications was tested by real users. Experimental results proved a high level of satisfaction for designers and developers that used the metamodel, and a high level of satisfaction for end users that used the application in real environments.

Chapter 6

A Navigation-aware Approach for Network Requests Prefetching of Android Apps

6.1 Introduction

Mobile apps dominate our world today. For example, as end users, we are spending more than 2 hours *a day* on mobile apps (Ben Martin, 2018). Android is accounting for more than 85.9% of global smartphone sales worldwide in the first quarter of 2018 (*Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2018* 2018) and more than 3.3 million Android apps are available in the Google Play, the official Android app store (*Number of available applications in the Google Play Store from December 2009 to June 2018* 2018), with more than one thousand apps being published *everyday* (Adam Lella, Andrew Lipsman, 2017). Android apps are not only being published in large numbers, they are also being consumed by users in large numbers. The Google Play store exceeded more than 20 billion downloads in the second quarter of 2018, with an increase of more than 20% w.r.t. 2017 (Sydow, 2018). For surviving in such a highly competitive market, it is fundamental for app developers to deliver apps with extremely high quality in terms of, e.g.performance, energy consumption, user experience.

The fact that mobile apps rely on wireless connectivity (e.g.3G, WiFi) is posing an additional challenge for developers, who must take into account the fact that the network underlying their apps may perform unpredictably in terms of latency or bandwidth (Joorabchi, Mesbah, and Kruchten, 2013). For developers, failing to properly consider network transfers may negatively affect the user experience, and in some cases hinder an effective usage of the app itself (Ravindranath et al., 2012). In turn, this can impact the app user ratings and reviews, which, unless properly addressed, can negatively impact the app's success (Palomba et al., 2018).

In this context, prefetching network requests has been advocated as a highly effective way of reducing network latency experienced by the user since it allows network responses to be generated *immediately* from a local cache (Zhao et al., 2018b; Zhao et al., 2018a). However, despite their promising results, existing approaches for prefetching network requests of mobile apps (e.g.(Zhao et al., 2018b)) can still be improved in many ways: (i) they *neglect the recurrent interaction patterns of each individual user* (e.g.how she navigates within the app), by either prefetching resources that will not be used, or limiting their prefetching algorithms to a set of "safe" situations (hence limiting their potential), (ii) they *do not change as users' interaction patterns change*, thus leading to the same problems mentioned before, and (iii) they

rely on *approximated static analysis techniques* for identifying when and which network resources can be prefetched, thus potentially leading to unhandled control or data flow paths (e.g.in case of reflection or implicit intents) or to the identification of paths that are infeasible at run-time (i.e., false positives) (Li et al., 2017; Yang et al., 2015; Yang et al., 2018).

With the aim of addressing the above mentioned limitations, in this chapter we present a *navigation-aware approach for personalized prefetching of network requests of Android apps*. The approach is fully automated (with the possibility of custom behaviour provided by the developer, if needed), transparent w.r.t. the back-end of the app (i.e., it is independent from the data types provided by the back-end and it does not require any modifications in the business logic of the back-end), and adapts its prefetching behaviour according to the navigation patterns of the user interacting with the app. In order to elicit the design goals of the proposed solution, we firstly carry out an explorative study in order to characterize the state of the practice and developers' needs about network prefetching in Android apps. Given the exploratory nature of the study, we use a mixed-method empirical research design, combining a quantitative analysis of 8,431 real-world open-source Android apps and an online-questionnaire filled in by 56 developers involved in real Android projects. The approach revolves around the concept of *Extended Navigation Graph* (ENG), where nodes represent the Android activities in the app, edges represent navigations of the user among activities, and each edge is annotated with information about the intent used for the navigation and the probability of being performed by the user. The intuition is to build and keep updated the ENG of the app at run-time and to prefetch network requests according to the paths that will be most likely travelled by the user according to the current status of the ENG. As such, our prefetching mechanism is *personalized* since every app installation has its own ENG with different transitions and weights according to the user's unique navigation patterns. Given the source code of an Android app, the approach acts in two phases: (i) at development time X automatically extracts all activities of the app and instruments it in order to continuously probe user navigation events (and related intents) and to inject the business logic for performing network prefetching, and (ii) at run-time the approach builds and keeps the ENG up to date according to the user navigation within the app, prefetches network requests according to the current status of the ENG, and intercepts the app's network requests for serving prefetched resources, instead of using the network. We opted for building the ENG at run-time in order to ensure that its edges represent valid navigation transitions within the app, as opposed to building it via static analysis techniques, which may lead to incomplete ENGs due to the well-known challenges such as the management of the implicit control/data flow among Android components, user-generated events, reflection, and multi-threading (Li et al., 2017). We evaluated the approach by conducting two independent studies aimed at empirically assessing the accuracy in identifying reachable activities within the ENG and the latency reduction achieved by its prefetching mechanism.
The main **contributions** of this chapter are:

1. a quantitative and qualitative *characterization of the state of the practice* on prefetching of network requests of Android apps (explorative study);
2. the definition of a navigation-aware *approach* for personalized pretetching of network requests of Android apps;
3. an *implementation* of the approach as an Android Studio plugin;

4. the results of an *empirical evaluation* of the approach;

The **target audience** of this paper includes both Android developers and researchers. Developers can directly use the implemented plugin for prefetching the network requests of their Android apps, thus reducing the user-perceived latency and loading time of their products. We support researchers by proposing the first quantitative and qualitative characterization of the state of the practice of network prefetching in Android apps. Also, to the best of our knowledge, we are the first to propose an approach for prefetching network requests of Android apps that (i) works at a higher level of abstraction (i.e., the navigation of the user within the app), (ii) adapts to each individual user navigation patterns, and (iii) does not inherit the limitations of current static analysis techniques.
Paper structure. Paragraph 6.2 provides background information. Paragraph 6.3 discusses the design, conduction, and results of our explorative study. Paragraph 6.4 presents the approach, whereas Paragraph 6.5 and 6.6 detail its mechanisms at development and run-time, respectively. Paragraph 6.7 describes the implementation of X and Paragraph 6.8 reports on the experiments we conducted for evaluating it. Paragraph 2.4 discusses their implications. Paragraph 3.9 presents related works and Paragraph 2.5 closes the paper.

6.2 Background

6.2.1 User Navigation in Android Apps

Android is a Linux-based open-source operating system developed by Google. Currently, it is one of the most popular and widely used mobile platforms(**androidshareandroidshare1**). Mobile apps running on the Android platform are mostly developed using the Java programming language and are built via the Android Studio development environment[1]. In some cases, developers use the so-called Android Native Development Kit[2] (NDK) for implementing parts of their apps using native code, mostly C and C++; NDK is often used when developers need to reuse libraries written in C or C++ (e.g., the OpenCV vision library[3]) or for processor-intensive tasks. An Android app is always built into a so-called Android PacKage (APK) which contains the compiled code, its used libraries and resources, and a mandatory XML-based manifest file providing essential metadata about the app (e.g.its main components, required permissions, supported Android APIs).

Android apps are composed of four types of **components**: *Activities*, *Services*, *Broadcast Receivers*, and *Content Providers*. According to the Android programming model (*Android Application Fundamentals* 2018), Android **activities** are the main building blocks of the app and represent single screens of the user interface. Activities are in charge of (i) reacting to user events (e.g.a touch on the screen), (ii) executing some specific functionality (possibly with the help of other app's components like services or content providers), and (iii) updating the user interface of the app for providing information to the user. An Android app comprises multiple activities to provide a cohesive user experience. Listing 6.1 shows excerpts of 3 activities in the context of an example social networking app. `HomeActivity` (lines 2-17) represents the screen

[1] `https://developer.android.com/studio`
[2] `https://developer.android.com/ndk`
[3] `https://opencv.org/platforms/android/`

where the user can see her contacts' updates, `ProfileActivity` (lines 19-42) represents the profile screen of a user, and `FriendsActivity` (lines 44-60) shows all the contacts of a user.

```
1  /** HomeActivity.java **/
2  public class HomeActivity extends Activity {
3    // ...
4    protected void onCreate(Bundle savedInstanceState) {
5      // ...
6      ListView homeListView = (ListView) findViewById(R.id.homelistview);
7      homeListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
8        public void onItemClick(AdapterView<?> av, View v, int pos, long id) {
9          if (this.isUser(v)) {
10           Intent i = new Intent(HomeActivity.this, ProfileActivity.class);
11           i.putExtra("Item", v.getItemId()); // extra field with URL fragment
12           this.startActivity(i); // navigate to Timeline activity
13         }
14       }
15     });
16   }
17 }
18 /** ProfileActivity.java **/
19 public class ProfileActivity extends Activity {
20   private User user;
21   private Button friendsButton;
22   private Button photosButton;
23   // ...
24   protected void onCreate(Bundle savedInstanceState) {
25     friendsButton = findViewById(R.id.button1);
26     photosButton = findViewById(R.id.button2);
27     // ...
28     View.OnClickListener buttonsManager = new View.OnClickListener() {
29       public void onClick(View view) {
30         switch(view.getId()) {
31           case R.id.button1:
32             Intent i = new Intent(MainActivity.this, FriendsListActivity.class);
33             i.putExtra("uId",user.getId()); // set again extra field
34             this.startActivity(i); // navigate to FriendsList activity
35             break;
36           case R.id.button2:
37             // ...
38         }
39       }
40     };
41   }
42 }
43 /** FriendsListActivity.java **/
44 public class FriendsActivity extends Activity {
45   protected void onCreate(Bundle savedInstanceState) {
46     // ...
47     getFriends();
48   }
49   private void getFriends() {
50     Intent i = this.getIntent();
51     String userId = i.getStringExtra("userId"); // get extra field by key
52     OkHttpClient client = OkHttpProvider.getInstance().getOkHttpClient();
53     Request request = new Request.Builder().url("https://api.sn.com/" + userId + "/friends").build(); // build
           target URL
54     client.newCall(request).enqueue(new Callback() { // issue network request
55       public void onResponse(Call call, final Response response) throws IOException {
56         this.addToView(response.body().string()); // usage of remote data
57       }
58     });
59   }
60 }
```

LISTING 6.1: Example of inter-activity prefetchable network request

When a user navigates within an app, the source activity starts a new instance of the target activity by creating an **intent** and passing it to the `startActivity()` method *Android Intents and Intent Filters* 2018. In this context, an Android intent is composed of two main parts *Android Intents and Intent Filters* 2018: (i) the *activity* to start and (ii) the *extras* representing the additional information required to perform the action, defined as a set of key-value pairs. For example, when a user clicks on an item in `HomeActivity` (lines 6-8), if the clicked item represents another user of the platform (line 9), then an explicit intent is created pointing to the target `ProfileActivity` activity (line 10), an extra field is created with `Item` as key and the unique identifier of the clicked user as value (line 11); finally, the target activity is launched in line 12. Similarly, when the user is in the `Profile` activity, she may click on `friendsbutton` (line 29) in order to further navigate to `FriendsActivity` (lines 31-25) and see the list of all friends of the currently visualized user. Also in this case the intent contains another extra field called `uId` containing the unique identifier of the user being visualized.

6.2.2 Network Requests in Android Apps

Being it for loading images, requesting data from a server, or getting resources from the web, most Android apps heavily rely on network requests to work as expected (Liu et al., 2017). The majority of network-connected Android apps request and receive data from their remote backends in a RESTful fashion via the HTTP protocol (*Android connectivity* 2018; Ma et al., 2018). Android developers can choose among a plethora of HTTP client libraries for their apps, such as *OkHttp*[4], *Volley*[5], *Retrofit*[6]. Android HTTP clients provide features such as connection pooling, concurrent requests, socket sharing, etc. In this study we focus on *OkHttp* since it is the HTTP client providing the official implementation of the *HttpUrlConnection* interface from Android 4.4 and it is at the basis of the most widely used networking libraries for Android, such as Volley, Picasso, and Retrofit (Vovk, 2018). As an example, in Listing 6.1 we show the network request for getting the list of all friends of a user of the platform. Specifically, the unique identifier of the user is firstly retrieved from the intent received by the calling activity (lines 50-51), then an instance of HTTP client is retrieved (line 52) and a new network request is created (line 53). In this case, the network request is targeting a dynamically-built URL, which will be called in line 54. The `onResponse` callback is called by the OkHttp library as soon a response is produced by the backend; finally, `addToView` method parses the body of the response for populating the screen with the list of all friends of the visualized user (line 56).

6.2.3 The Prefetching Opportunity

If we consider activities as nodes and explicit intent launches as transitions among nodes, we can obtain the navigation graph of an Android app. As shown in Figure (6.1), the navigation graph of our example is composed of 3 activities and 2 navigation transitions.

If we enrich this graph with (i) weight of the arches, (ii) the URLs requested by every activity, and (iii) how intent extras can map to dynamic fragments of the

[4] http://square.github.io/okhttp
[5] https://developer.android.com/training/volley
[6] https://square.github.io/retrofit/

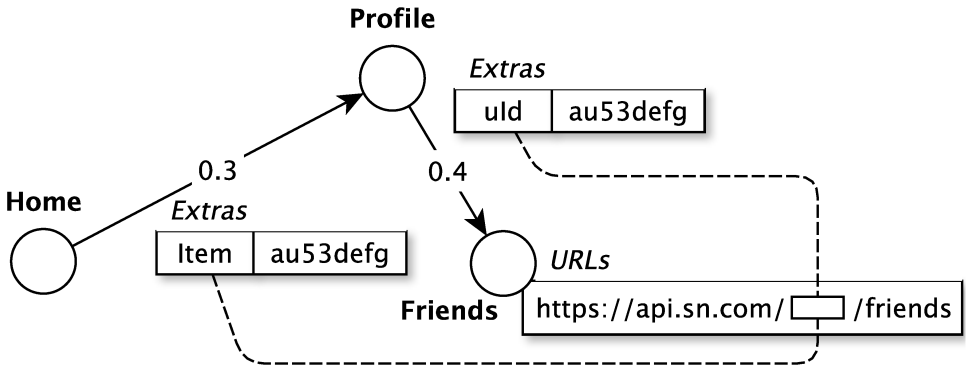


FIGURE 6.1: Example of ENG for Listing 6.1

requested URLs, then we obtain the Extended Navigation Graph (ENG). In our specific example, the ENG contains: (i) weight of the arches, (ii) the extra fields of the intents launched by `HomeActivity` and `ProfileActivity` with keys *Item* and *uId*, respectively, and (iii) how they map to the fragment of the dynamic URL requested in `FriendsActivity`.

The approach exploits the ENG at run-time in order to (i) resolve as soon as possible the URLs to prefetch and (ii) prefetch only the network requests belonging to the activities that will be most *likely* visited by the user according to her own navigation patterns exhibited during previous usage sessions of the app. This reasoning leads to the navigation-aware and personalized prefetching of network requests for Android apps. In our example, as soon as the user is navigating from `HomeActivity` to `ProfileActivity`, the approach maps the `Item` intent extra to the URL fragment needed for completing the URL used in `FriendsActivity` (with a certain degree of confidence). If the prefetching algorithm judges the navigation of the user to `FriendsActivity` as *likely*, then it is able to prefetch its network request two activities in advance w.r.t. the user navigation flow (i.e., when the user is leaving the `HomeActivity`).

6.3 Study Design

We want to assess the state of practice on prefetching approaches for mobile Android apps. Due to the recent emergence of this research topic, in order to achieve the research goal, this study applies an exploratory approach to investigate and characterize the features and the challenges faced by developers when developing (if they do) prefetching approaches for their mobile apps. Fig. 6.2 presents the process we followed for our exploratory study. Our exploratory study consists of an online-questionnaire survey and a quantitative analysis of 8,431 real-world open source Android apps.

In this paragraph we describe the research questions addressed in our study, then we present the overall exploratory study we performed, and finally we highlight the results.

6.3.1 Research Objectives

The main **goal** of this study is to characterize the features and the challenges faced by practitioners when developing prefetching approaches for their mobile apps. We

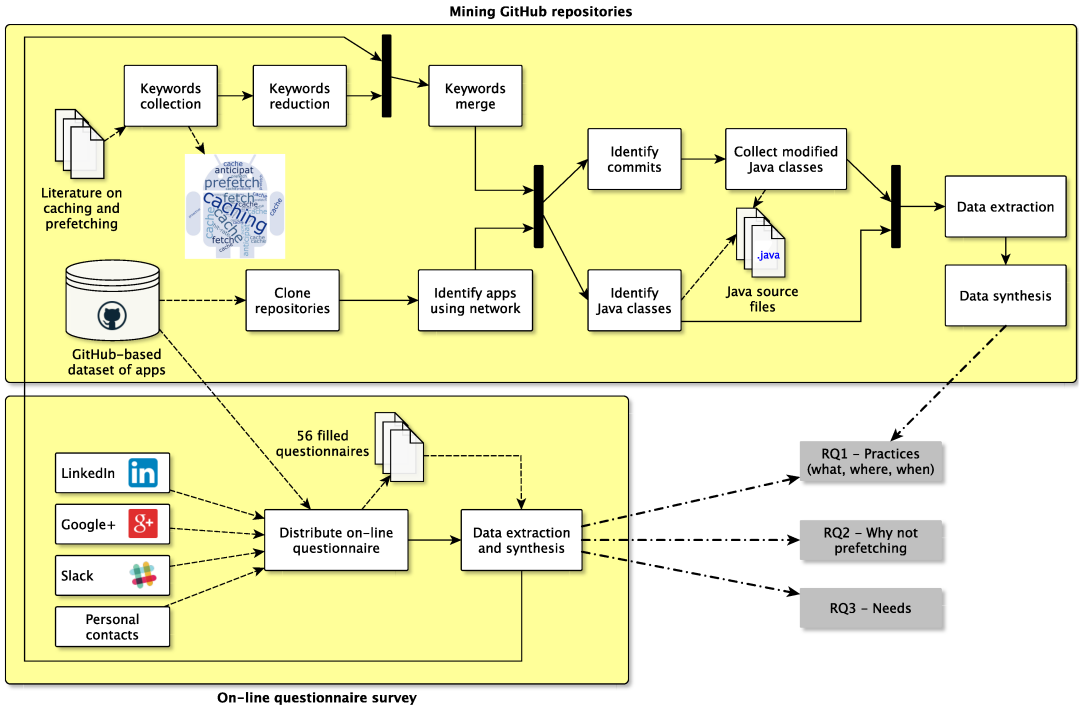


FIGURE 6.2: Overview of Exploratory study process.

refined our goal in the following three Research Questions (RQs).
RQ1 – *How do developers perform data prefetching in mobile apps?* By answering this RQ we aim at characterizing the approaches performed by developers in terms of 'what', 'where' and 'when' to prefetch.
RQ2 – *Why developers do not apply data prefetching approaches in mobile apps?* By answering this RQ we aim at identifying the challenges, risks, reasons of developers about the non-application of prefetching approaches.
RQ3 – *What are the data prefetching needs of developers?* By answering this RQ we aim at identifying the needs of developers about their ideal prefetching approach.

6.3.2 Identification of Representative keywords of prefetching-related commit messages

Approaches implementing prefetching may use other terms than 'prefetch' only. Therefore, in order to design a questionnaire that would cover the terminology in the field, we first reviewed the related works (see paragraph 3.9) and identified representative keywords of prefetching-related commit messages on open-source apps, both mobile-specific and not. As a result, we could identify the following representative keywords: *fetch*, *cach*, *retriev*, *anticipat*, *proactive*, *predict* and *hit-rat*.

6.3.3 Online Questionnaire Survey

Identification of the Target Population

The target population for this study entails Android developers. To address them, we identified the following professional Android developers Communities where most discussions occur: Android Developer Group (LikedIn), Google Android (LikedIn),

Android Development Community (Google+), Android United (Slack). In addition, we included the Android Developers from our personal contacts and a Google Play mailing list from a public and available dataset Scoccia et al., 2018. The resulting population was invited to answer the online questionnaire survey by email or by a post in the social community.

Design of the Questionnaire

As already anticipated, we designed the questionnaire in order to address our three RQs. The questionnaire[7] is composed of 28 questions: 16 are closed-ended questions, whereas the rest are open-ended questions allowing respondents to freely discuss personal experience. In 3 optional questions, the participant was asked to add personal details and comments. The questionnaire is structured in six sections *(a-f)* and follows the flow shown in Figure 6.3. Beside the introduction of the study,

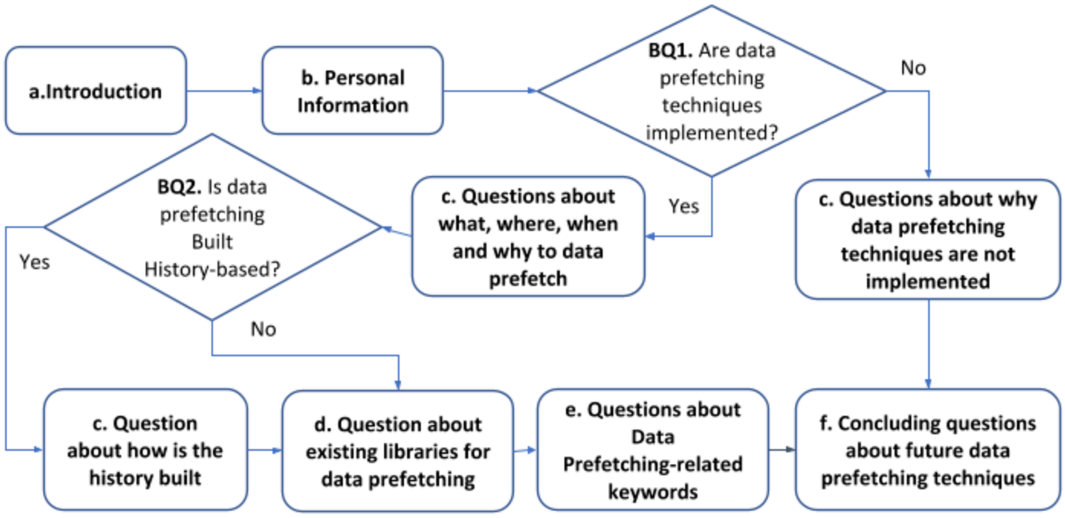


FIGURE 6.3: Questionnaire flow.

other administrative information and a preliminary Prefetching definition *(a)*, the core sections of the questionnaire are: *(b)* demographic questions about the working environment, professional background of the participant and previous consideration of prefetching approach implementation, *(c)* questions about 'what', 'where', 'when' and 'why' to data prefetch, consideration of previous interactions of the user(s), *(d)* a question about existing libraries, *(e)* questions on keywords representative of prefetching-related commit messages in GitHub, and *(f)* questions about the participants needs in order to identify their ideal prefetching approach. The questionnaire was created as an online form.

Data Analysis and Results

The questionnaire has been completed by 56 participants that have been involved in Android development and are from different organizations and countries. All the answers are collected in a single spreadsheet (with a column for each question and

[7] A transcript of the questionnaire is available here: Link to be added after double blind review process.

a row for each participant). For each question (i.e., for each column of the spreadsheet) we applied basic descriptive statistics for better understanding the data about the occurrences of each given response and we extracted key findings. Then, we performed a series of brainstorming meetings to discuss the results presented below.

Among the 56 participants, 36 have an experience under five years, whereas the experience of the remaining participants is between 5 and 10 years.
When asked about how many Android apps they have developed, the majority of participants (about 52%) declare to have developed between two and five apps. Other participants declare that they have developed between six and ten apps (about 23%) and only one app (about 14%); the remaining 11% have developed more than 10 mobile apps. The majority of participants (about 59%) develop Android apps as a hobby while the rest work in a company. In reference to their current job, 71% of the participants declares to be developers, 18% are (PhD) Students, about 7% are Managers and the remaining 4% indicates a higher-level professional profile.

Starting with the first branch question (BQ1) shown in Figure 6.3 of our questionnaire – based on the provided prefetching definition shown in the frame below – about 61% of the participants say they have never implemented a prefetching approach in any of their Android apps. The remaining 39% implemented it at least once.

> ***Prefetching*** *refers to the approach in which the app requests and downloads data (e.g., raw data, images) from the back-end before the user actually needs it. Prefetching is different from caching since the former does not require a previous access to the needed data for saving requests.*

Furthermore, among the participants who explicitly stated that they have never implemented a prefetching approach, about 79% declared that they have never even considered to implement a prefetching approach in their apps. The main reasons brought by respondents for not considering prefetching are equally distributed between: *(i)* no need perceived in implementing a prefetching approach, and *(ii)* lack of knowledge about these approaches. These reasons show lack of awareness of the problem and of possible existing solutions. The reasons brought by respondents who instead just considered the prefetching approach without implementing it (about 21%) are equally distributed between, *(i)* no need perceived, and *(ii)* hard task for approaches not officially published, no presence of libraries, and high risk if not well-enforced.

The following analyzes the answers from participants who have explicitly implemented a prefetching approach, in terms of 'what', 'where', 'when' and 'why' to data prefetch. It is important to note that in these questions *(section c)* we ask the respondents to focus on the last Android app where they have implemented a prefetching approach.

Regarding the Google Play category of the Android app considered by the participants, the categories reported most frequently are: Social, Music & Audio, Food & Drink, Sport, and Finance. However, the apps considered by the participants cover each category at least ones, hence making our survey representative of all Google Play categories.

Regarding the prefetched **type of resources**, the answers fall into the following categories: *images* (about 47%), *raw data* (about 38%), and audio/video streams (about 15 %). This result highlight that images are the most prefetched resource

thanks to the existence of libraries (e.g.Glide, Picasso) that support developers to implement functions for image pre-load.

We then ask participants to rate, for each type of resource, the frequency with which the prefetched resources are modified during the lifetime of the considered app. Table 6.1 shows the occurrences of the provided answers by participants. By the majority of the participants it is perceived that raw data resources change between *sometimes* and *always*; images change between *sometimes* and *very often*. Finally, audio/video stream resources that are the least-frequently prefetched resources tend not to change.

TABLE 6.1: How often do prefetched resources change during the lifetime of your Android app?

Resources category	**Rates occurrences**						
	Always	*Very often*	*Sometimes*	*Rarely*	*Never*	*Don't Know*	*Not prefetched*
Raw data	5	2	6	2	1	0	6
Images	3	5	6	4	2	0	1
Audio/video stream	2	2	2	1	3	2	9

By investigating **where** the prefetched resources are stored, the answers are almost equally distributed between *In-memory* (about 68%), *Local database* (about 64 %), and *Files in the file system* (about 50%). We suppose that the diversity of the answers derives from the absence of official documentation and guidelines.

The participants were also asked **why** they perform prefetching. As shown in Table 6.2, respondents identified as the most relevant reasons: Decrease user-perceived latency (20/22) and improve app performance (12/22). Moreover, when asked **when**

TABLE 6.2: Reasons about **why** to perform prefetching.

Answers	*Occurrences*
Decrease user-perceived latency	20
Improve app performance	12
Reduce network requests	8
Optimize energy consumption	5
Other	3

they performed prefetching – as shown in Table 6.3 – participants indicated that they typically perform prefetching when the app is in foreground AND being used (about 36%) and when the app is launched the first time after installation (about 27%). For what concerns how participants **consider different users** when they perform prefetching, about 77% declare that *users are all the same* – one data prefetching pattern for all users. Only for 3 participants *users are grouped into meaningful groups* – one customized data prefetching pattern for each group of users. Moreover, when we asked in detail how these participants distinguish the specific users, basically they group the users by type of accounts.

In the remaining of this section *(c)* of the questionnaire, we ask participants if they consider previous interactions of the user; in other words if they implement a **history-based approach**. Only 5 participants (about 23%) states that they consider previous interactions of the user with the app. Instead, the informations considered

TABLE 6.3: Reasons about **when** participants perform prefetching.

Answers	*Occurrences*
When the app is in foreground AND being used	12
When the app is launched the first time after installation	9
When the app is in background	7
As soon as the app comes to foreground	5

by participants in order to build the history are: previous actions performed by the users (2/5), previous navigations among screens of the app (2/5) and Previously requested URLs (1/5).

Existing libraries for data prefetching

In this section of the questionnaire *(d)* we ask participants to suggest, based on their experience, potential **libraries** that allow developers to implement a prefetching approach. Among the respondents to this question (22), 5 participants stated that they do not know existing libraries, while the remaining participants (17) indicates existing libraries (i.e. OKHttp, Glide, Picasso, Retrofit, Dbflow, SyncAdapter, Smack, rxjava, room and ExoPlayer) that do not allow to directly implement prefetching approaches. In fact, no library that implements prefetching – in any way– has been found. The developers, who know the formal existence of these approaches, use the very capable networking libraries before reporting, in order to customize the provided functions to implement personal possible solutions of prefetching approach.

Representative data prefetching-related Keyworks

In this section of the questionnaire *(e)* we ask participants two questions in order to identify and evaluate representative keywords of prefetching-related commit messages in Android apps. The answers to these questions will be important in order the complete the overall exploratory study with the commit message analysis in GitHub (Phase 5). Firstly, we ask participants to provide some keywords they think may be representative of prefetching-related commit messages in GitHub. Participants have reported different keywords, similar to each other. The latter indicated with the most occurrences are *preload*, *prefetch* and *predict*. After we ask participants to rate the Keywords identified in the second phase of our Exploratory study. Based on the results reported in Table 6.4 and in the previous question we can affirm and confirm that the most representative keywords of prefetching-related commit messages to be used in the empirical study are all the keywords reported in Table 6.4 (excluding *hit-rat**) plus the additional keyword *preload* indicated by participants.

Participants needs about data prefetching approaches

Finally, in this last section of the questionnaire *(f)*, we asked all participants involved four questions in order to get more details about their ideal prefetching approach for Android apps. Firstly, we asked participant to rate the control they would like to have on the prefetching approach. As shown in Table 6.5, the majority of respondents (about 41 %) declare that it is **"very important" to have control** on the

TABLE 6.4: Rate occurrences for each keyword.

Keywords	Rates occurrences		
	Excellent	*Somewhat*	*Poor*
fetch*	15	5	2
cach*	10	9	3
retriev*	7	10	5
anticipat*	7	7	8
proactive	6	7	9
predict*	6	6	10
hit-rat*	3	4	15

prefetching approach. When asking to motivate their answers, the main reasons regarding the possibility to customize their prefetching approach in reference to resources type, usage of the app, storage size, optimize the use of the bandwidth and probability to get unused data. Moreover, we have asked to indicate if they like an

TABLE 6.5: How important is to have control on the prefetching approach in your Android apps?

Answers	*Occurrences*
Absolutely essential	4
Of average importance	17
Very important	23
Of little importance	4
Not important at all	2
I don't know	6

"optimistic" or a more *"conservative"* prefetching approach according to the following definitions:

- *Conservative prefetching*- potentially high precision (high number of hits) but low benefits (less requests/time saved when having a hit);
- *Optimistic prefetching* - potentially poor precision (low number of hits) but high benefits (high number of requests/time saved when having a hit)

About 55% of participants (31/56) state to have **tradeoff between optimistic and conservative prefetching** (good precision but potentially less requests saved), and about 23% and 21% declare that they prefer a conservative (13/56) and optimistic prefetching (12/56), respectively. Finally we have asked participants to suggest the top-three features about their ideal prefetching approach. Most of the answers fall into the following features: easy to develop, configure, extend, deploy and debug.

6.3.4 Empirical Investigation on Android apps in GitHub

In this paragraph we present the last phase of our exploratory study. This phase aims at addressing RQ1 from the quantitative and qualitative prefetching-related commits in Android Apps dimensional view. The study *context* consists of a dataset of 8,431 real-world open-source Android apps (about 240 GB) obtained as a result of *AndroidTimeMachine* project [8]. It combines source and commit history information available on GitHub with the metadata from Google Play store. First of all

[8]https://androidtimemachine.github.io/

we reduced the dataset about 80 GB, deleting text, audio and video files. In order to identify performed prefetching approaches, a string analysis was performed. In particular we identify all the occurrences of *HTTP* keyword with at least one of the prefetching-related keywords identified in the previous phase: *cach*, *fetch*, *predict*, *retriev*, *anticipat*, *hit-rate* and *proactive*. As results we obtain in two different files the frequency values of the keywords and the flag that certifies the presence of the HTTP keyword related to the individual files(Out1 file), and the information regarding each project(Out2 file). Both the two functions have been implemented with the use of a Thread Pool in order to reduce the execution time of the script.The Out1 and Out2 files contain respectively 3833141 and 7942 occurrences. In order to consider only the projects and files in whose source code at least one of the seven keywords is present, two further scripts have been developed. The *ReaderOutputProject* script, whose output is a CSV file named *ProjectSelected* created from the Out2 file. The script, starting from the complete list of projects, considers only projects containing at least one JAVA file containing in turn one of the seven keywords identified previously. From this selection, the number of projects has decreased from 7942 projects to a number equal to 3738. For the list of individual JAVA files, the *ReaderOutputFile* script was used instead. This script creates, starting from the Out1 file, a file containing the list of all files in which at least one of the keywords is present. The selection led to a reduction of files from 383141 to 38170, whose list was saved in the *FileSelected* file in CSV format. At the same time, the research was carried out considering only the commits, given the possibility of displaying all the information as description and files affected by the same commit. As previously mentioned, we will go to select the commit only, in whose description there is one of the keywords. The number of projects involved in at least one of these commits is 1173, while the number of JAVA files involved is 8159. The names of the files created by the script for saving the lists are *ProjectListWithFlagCommit* and *FileListWithFlagCommit*. Having these two parallel searches available, an intersection was made between the two data sets, in order to obtain files or projects that are involved in at least one "special" commit and a "special" type source. Given the possibility that some special defined commits may involve files not in JAVA format, from this point on, we will consider only the list of JAVA files, leaving out the list of projects. Next, simply group the files by projects to get the final number. To perform the operations described above, the *AddFlagCommit* and *WriterFileFinal* scripts were used. Following the intersection of the two datasets, the JAVA files obtained were 3675 grouped then successively in 673 projects. We manually analyzed the source codes of this projects. Only 9 project implement a prefetching approach without exhibiting any specific patter. The results in terms of "what", "where" and "when" to prefetch are the following: *What-* raw data, images; *Where-* app component that trigger prefetching (in order of occurrences): Activity, service, application, fragment and Broadcast receiver; *When-* at the launch of the app. Also this last phase confirm *OkHttp* as the most used library.

6.3.5 Reflection

The overall exploratory study presented in this paper has been centered around the three research questions introduced at the beginning of this paragraph. The findings are summarized in the following. First of all, the results of this exploratory study reveal that developing prefetching approaches in mobile apps is a very emerging area.

RQ1 asks *"How do developers perform data prefetching in mobile apps?"*. The majority of apps run over HTTP and the massive use of popular HTTP libraries, including OKHttp, URLConnection, Volley and Retrofit. In terms of "what", "where" and "when" the results have been discussed.
RQ2 asks *"Why developers do not apply data prefetching approaches in mobile apps?"* The results of our study reveal that, so far, the majority of Android developers do not implement prefetching approaches for lack of knowledge about these approaches, therefore the developer is not aware of having these problems and the existence of possible solutions. Instead, it is an hard task to implement – not officially published, no presence of libraries, and high risk if not well-enforced.
RQ3 asks *"What are the data prefetching needs of developers?"* Concluding, for all participants is very important to have prefetching approach schema to apply when developing apps. Indeed, for the majority of respondents declare that it is very important to have control on the prefetching approach in order to customize the approach in reference to resources type, usage of the app, storage size, optimize the use of the bandwidth and probability to get unused data.

6.4 The Approach

We designed a two-phased approach, where the first phase is performed only once at development time (see Figure 6.4) and the second phase is carried out at run-time throughout the execution of the app (see Figure 6.5).

6.4.1 At development time

As shown in Figure 6.4, the only input required by the plugin is **the original source code of the app** being developed. The approach does not impose any specific development style to the developer, provided that the app makes HTTP requests by passing through *OkHttp*.

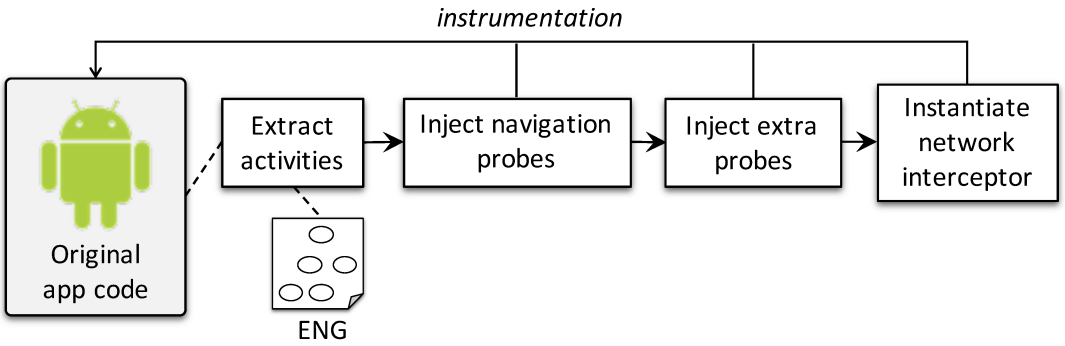


FIGURE 6.4: Overview of the approach at development time.

The first step of the plugin is the **extraction of all activities** of the app in order to build the initial version of the ENG. This step is realized by (i) parsing the `AndroidManifest.xml` file of the app, (ii) producing a node in the ENG for each detected activity and (iii) labelling each node with the full path of the Java class implementing the activity. Since it is mandatory to declare all activities of an Android

app in its manifest file[9], the produced ENG is complete with respect to the coverage of all activities of the app being analysed.

Then, the plugin **injects a navigation probe** in the Java class corresponding to each extracted activity. The navigation probe notifies the plugin when a navigation occurs from/to every activity of the app at run-time. This step is realized by injecting notification statements (i.e., probes) in the body of the `onStart` and `onStop` methods of each previously extracted activity. Since those methods are called by the Android OS every time it needs to make the current activity visible or not visible to the user[10], the plugin is able to correctly and timely detect any transition within the ENG of the app.

The next step consists in the **injection of extra probes**, which will be in charge of notifying the plugin every time an intent's extra is set by the app. This step is realized by injecting notification statements immediately after a call to either the `putExtra` or `putExtras` methods in each activity (they are the only methods in the `Intent` class where its extra fields can be changed).

Finally, the plugin **instantiates a network interceptor** in order to log all HTTP requests issued by the app and serve prefetched resources. At run-time the activities performed by the network interceptor are totally transparent to the developer, who can implement the business logic of the app.

Summarizing, at the end of the steps described above, the plugin has (i) extracted the ENG of the app containing all activities of the app, (ii) instrumented the app with probes for notifying about navigation and intents' extra update events at run-time, and (iii) added a transparent network interceptor for logging all outgoing HTTP requests of the app at run-time and serving prefetched resources. All those steps are performed *automatically* without requiring any intervention by the developer.

6.4.2 At run-time

Figure 6.5 shows the main components of the plugin at run-time. All together they are responsible for (i) keeping the ENG always up-to-date according to the navigations and actions performed by the user, (ii) identifying which network resources can be prefetched, (iii) prefetching network resources, and (iv) making prefetched resources available to the app via a lightweight URL map. In the following the behaviour and main responsibilities of these components of the plugin are presented.

The **Navigation monitor** is a passive component, which is triggered every time a navigation probe in the app raises a navigation event (i.e., every time the user is moving between two screens of the app). The main responsibilities of the navigation monitor are: (i) to keep track of the current activity within the ENG, (ii) to add a transition in the ENG if a raised navigation event is involving a target activity that has never been visited before, (iii) to update the weight on the transitions of the ENG for every received navigation event, and (iv) to trigger the Prefetcher component at every navigation event.

The **Extras monitor** is also a passive component and it is triggered whenever a previously-injected extra probe raises an event related to the setting of one of the intent's extra fields. The main responsibilities of the extra monitor are: (i) to update the ENG with the newly set extra field, and (ii) to trigger the Prefetcher component for every received event.

[9] https://developer.android.com/guide/topics/manifest/activity-element

[10] https://developer.android.com/guide/components/activities/activity-lifecycle

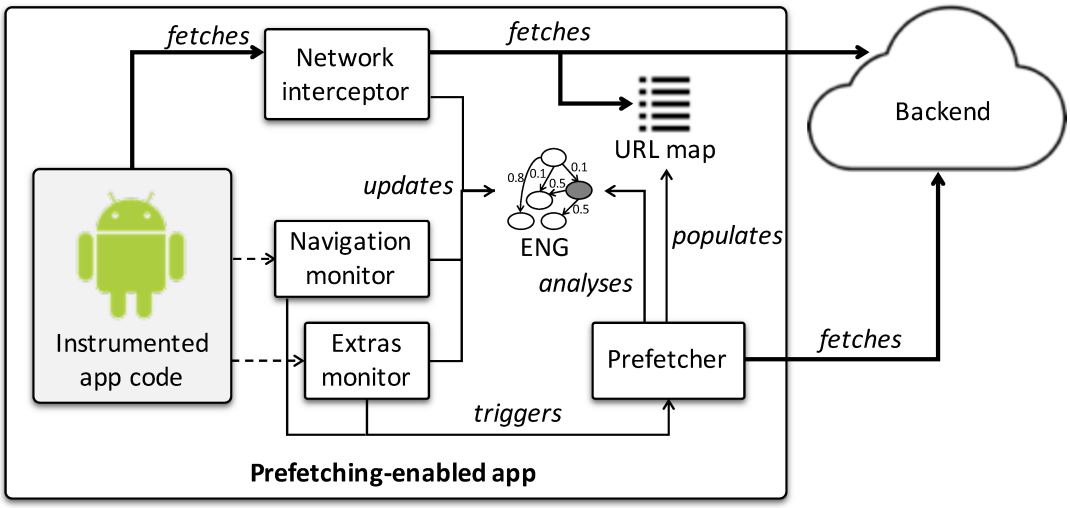


FIGURE 6.5: Overview of the approach at run-time.

The **Prefetcher** lies at the core of the plugin . It is triggered by the two monitor components every time a user navigates within the app or an intent's extra is set; then, the Prefetcher analyses the current ENG and, based on the currently-prefetchable network resources, it fetches them from the backend of the app and stores them into the URL map. It is important to note that we designed the plugin with *separation of concerns and maintainability* in mind: if a different prefetching algorithm will be needed in the future or the developer needs a custom prefetching algorithm (e.g.by exploiting machine learning techniques for better predicting the user's navigation events), the developer can implement such a change simply by updating the Prefetcher component, without impacting the other components of the plugin.

The **URL map** is a lightweight hashmap storing a set of key-value pairs. For each entry of the map, the key and value parts contain the URL and corresponding payload of the prefetched resource, respectively. The URL map is cleared every time the app goes in background or is killed in order to have a relatively low number of entries in the map and to keep only fresh data in the prefetched resources.

The **Network interceptor** has three main responsibilities: (i) to log all outgoing HTTP requests of the app at run-time[11], (ii) to update the ENG according to the logged HTTP requests, and (iii) to serve prefetched resources as soon as one of the outgoing requests made by the app matches an entry in the URL map. Logging HTTP requests is necessary in order to update the list of the performed network requests during the current activity in the ENG; this information will be used by the prefetching algorithm for identifying new mappings between some intent's extra field and dynamic URL fragments requested by the app. Technically, the plugin exploits the interceptors mechanism of OkHttp, which allows us to add our network interceptor as last element of a potentially not-empty chain of already existing interceptors. This makes our prefetching mechanism totally *transparent to the developer* in terms of effort, and *unintrusive* with respect to the business logic of the app w.r.t. network requests.

[11]As done in **ICSE_2018**, in this study we focus exclusively on GET HTTP requests in order to not incur in unwanted add/update/delete operations in the backend.

6.5 Building the Extended Navigation Graph

As described in the previous paragraph we design an *Extended Navigation Graph* to abstract the activity transitions. Here we give the formal definitions of an activity transition $t(\gamma)$ and an Extended Navigation Graph ENG in order to describe the automated app instrumentation.

Definition 14 (Activity Transition)**.** *An activity transition $t(\gamma)$ is triggered by an intent, where γ is the list of 2-tuple $\gamma < K, V >$, where K is the set of keys identifing an intent extra and V is the set of string values associates to each key.*

Definition 15 (Extended Navigation Graph)**.** *An Extended Navigation Graph ENG is a directed Graph with a start vertex v. It is denoted as a 3-tuple, $ENG < A, E, v >$, where a is the set of all activity nodes of an app; E is the set of directed edges, and $e < a_1, a_2 >$ represents an activity transition $t(\gamma)$.*

6.5.1 App Instrumentation

The IntelliJ plugin instruments an app automatically by taking into account the information extracted from the Android Manifest file and from the source code. After the execution of the plugin, the source code will contain the instructions needed by the library to interact with it, will enable the graph building and the intent extras identification at run-time and will allow the library to have access to the Http client.

The automated instrumentation is split in the following three main actions.

Action 1):the plugin extracts all the activity class names from the Android Manifest and uses the associated metadata to identify wether the activity is the Launcher Activity (the one that is shown on the Android app launcher); after this identification the plugin thanks to the defined API *PrefetchingLib::init* and *PrefetchingLib::setCurrentActivity*, adds into the Launcher Activity the lines code needed to initialize the library and to all the activities the code needed to notify to the library that is happening an activity transition $t(\gamma)$.
The *init* method is inserted only in the app's launcher activity and enables the initialization of the library, composed by the initialization of the database, the loading of the ENG and the loading of all the past url candidates for each node of the graph. The setCurrentActivity method allows the library to be notified when an activity transition happen. This method updates the reference to the current activity and adds the edge to the graph if it does not already contains this transition.

Action 2): the plugin looks in each class if there's an Intent that will be enriched with extra information at run-time and adds the code lines needed to notify the library that an Activity after that point will contain a 2-tuple key-value that may be used to fill a candidate URL.

Action 3): the plugin looks in each class where the http client is instantiated in order to notify the library that a http client is available and ready to be used to intercept the network requests.

6.6 Network Requests Prefetching at Run-time

After the app instrumentation phase the library acts at runtime. Also the runtime phase in divided in two subparts: the creation of the navigation graph and the

identification of the URLs to be prefetched. Both parts acts in a parallel way and shares their information to identify the correct parameters needed to fill a URL. At the first launch, the engine has no idea on how the activities are connected to each other; when the user navigates through the activities, the library keeps track of this movements that are notified by the instructions inserted in the app instrumentation phase. During this phase each time a user moves from an activity to another the library records a new transition; this information will be used later to calculate the probability that a user in a future navigation will move from the source activity to the target activity.

6.6.1 Identification of URLs to be prefetched

As for the graph, also the candidate URLs table is empty at the first launch. So the library intercepts every network request as soon as the developers connects the app's http client. When a URL is requested from the app, the graph is traversed backward from the current node to the root node and all the extra fields of the intent generated that edges are evaluated in order to find a substring of the URL that is equal to a value of the extra field. If a match is found, the URL part that matches the value of this extra is marked as parameter and it's value is replaced with the key of this extra intent.

6.6.2 Prefetching spot

When a user is navigating through Activities, as soon as he generates an Intent with some extras, the library checks all the edges starting from the current Activity Node and looks for the nodes that has a probability to be visited higher than a given threshold. Than it expands the selected nodes and checks again if the probability of the next nodes is higher than a threshold and so on. Once it has the complete list of the most probable nodes that will be selected by the user, starts looking for each candidate URL that each node contains if, with the current set of extras, is able to fully resolve the candidate URL. If all the parameters are filled with the current extras 2-tuple key-value, the URL is added to a list that will be used to perform the network request.

Data: Current activity node a_c, Extended Navigation Graph G
Result: Candidates URLs string
$candidates \leftarrow newemptylist\ successors \leftarrow getallsuccessors(a_c, G)$
foreach $succ \in successors$ **do**
  foreach $parameteredURL \in s.getparameteredURLlist()$ **do**
    if $s.keys.containsAll(parameteredURL.getParamKeys())$ **then**
      $candidates.add(parameteredURL.fillParamKeys(s))$;
  end
end
return *candidates*

Algorithm 1: Computing prefetching candidates URLs

Data: Current activity node a_c, Extended Navigation Graph G
Result: List of URL prefetch
$listURLtoprefetch \leftarrow newemptylist\ countlist \leftarrow newemptylist$
$successors \leftarrow getallsuccessors(a_c, G)$
foreach $succ \in successors$ **do**
  $a_e \leftarrow succ.getdestination()\ countlist.add(countedge(a_c, a_e), a_e)$
end
$sizesuccessor \leftarrow getTotalEdges(countList)$
$probabilityList \leftarrow newemptylist$
foreach $count \in countList$ **do**
  $probabilityList.add((count/sizesuccessors), a_e)$
end

Algorithm 2: Computing URL to prefetch

6.7 Implementation

The library has been implemented by extending Android libraries for the runtime execution and IntelliJ plugins for the source code instrumentation phase. The chosen http client is OkHttpClient because it is the default Http client in Android and it is easy to integrate with a lot of existing libraries. To store information has been used the Room Persistence Library [12] which enables the full power of the SQLite Android native driver giving a high level of abstraction.

The source code instrumentation is achieved through an IntelliJ plugin, developed with the IntelliJ Platform Plugin SDK; this SDK enables the interaction with the API of any of the IntelliJ IDEs, in this particular case with Android Studio. This SDK provides off-the-shelf tools to analyze and modify the source code and through these APIs it has been possible to develop a powerful plugin that simplifies the integration of the library from the developer perspective.

6.8 Evaluation

In this paragraph we report the design and the results of first experiment we carried out to evaluate the approach. The *goal* of this study is to assess the accuracy of the plugin in identifying reachable activities within the dynamically-built ENG. We chose to evaluate the reachable activities in the ENG since the ENG is the core of the whole approach and failing to build an accurate ENG at run-time may potentially result in a high number of unused prefetched resources (false positives) or missed prefetching opportunities (false negatives). This study is designed as a multi-test within object study Wohlin et al., 2012, because it is conducted on a single *object* (i.e., the current implementation of the library) across a set of *subjects* (i.e., a set of apps to be executed). More specifically, we firstly consider all the 20 apps included in the replication package of the study proposing Gator Yang et al., 2018, the state-of-the-art approach for building models of Android apps similar to our ENGs called window transition graphs. The initial set of 20 apps is reduced to 6 apps since we had to filter out (i) the apps with less than 3 activities as their navigation graph is trivial and (ii) the apps whose source code in GitHub is not compilable into an executable APK, due to e.g.missing resources, missing Gradle build files, etc. In order to

[12] https://developer.android.com/topic/libraries/architecture/room

extend the current set of 6 apps, we consider the dataset of 8,431 real apps proposed in Geiger et al., 2018 and we randomly select a pool of 50 potential subjects. Then, we manually compile each of the 50 apps and we include all the apps for which the compilation step is successful[13]. Table 6.6 presents the subjects of this experiment, together with their category, number of activities declared in their Manifest file, and source.

TABLE 6.6: Subject apps for study 1

ID	Name	Category	Activities	Source
A1	BarCodeScanner	Tools	9	Yang et al., 2018
A2	Beem	Comm.	14	Yang et al., 2018
A3	Hillffair	Entert.	26	Geiger et al., 2018
A4	Mileage	Finance	50	Yang et al., 2018
A5	OpenManager	Product.	6	Yang et al., 2018
A6	OpenSudoku	Games	10	Yang et al., 2018
A7	Phonograph	Music	14	Geiger et al., 2018
A8	Privacy-Friendly-Weather	Weather	12	Geiger et al., 2018
A9	Radiomenenepro	Music	3	Geiger et al., 2018
A10	TippyTipper	Finance	5	Yang et al., 2018

The apps are quite heterogeneous in terms of store category and number of activities (min = 3, max = 50, mean = 14.9, SD = 13.9). For each app, we (i) extract the ENG built by the plugin while one of the researchers systematically executes all possible interactions provided by the app, (ii) run Gator for producing its window transition graph, and (iii) build its ground-truth navigation graph by manually inspecting its source code. The latter activity has been performed by two researchers and implied 1 week of full-time effort. Finally, we collect and compare the number of reachable activities in the two navigation graphs (i.e., the manually-constructed ones and those produced by the plugin).

Results – As shown in Figure 6.6, for 8 apps the number of reachable activities identified by the plugin is the same as the ones reachable in both the ground truth models and the graphs produced by Gator. In two cases (A2 and A4), the plugin is missing two potentially-reachable activities w.r.t. the ground truth and Gator. We manually inspected the source code of the involved activities in order to understand why the plugin missed the transitions to those 4 Android activities. In all 4 cases, the activities incorrectly marked as unreachable represent corner cases for our experiment, e.g.accessing a management dashboard reserved to administrators, receiving real-time messages from other users of the app, etc. Overall, those 4 activities are mainly due to specific situations that could not be straightforwardly reproduced in our experiment; nevertheless, they will be considered by the plugin as soon as the user will actually navigate to those "less common" activities at run-time.

Overall, the obtained results are promising and indicate that detecting navigation traces among activities at run-time leads to the accurate identification of reachable activities. Interestingly, by looking at Table 6.6, we can observe that the number of activities declared in the Android manifest is often higher than those executed at

[13]This step has been performed fully manually in order to raise the probability of having compilable Android apps by manually adjusting their project configuration (where possible). This step involved two researchers and took about 1 person-month

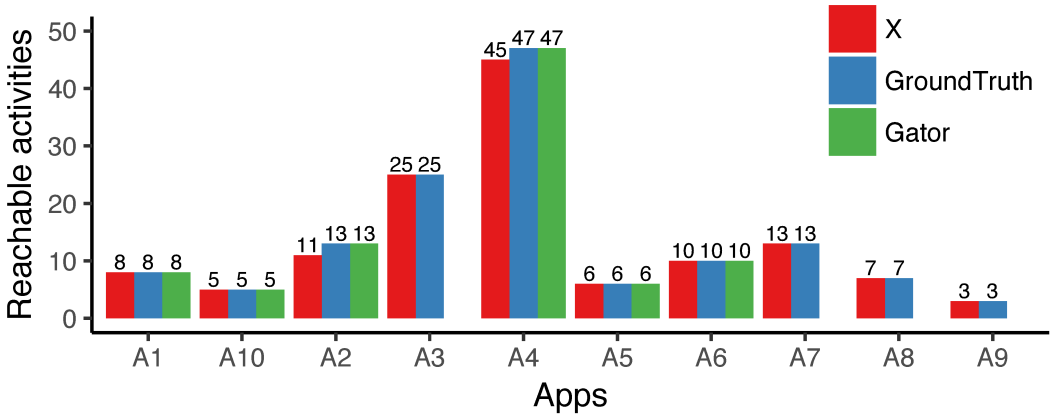


FIGURE 6.6: Reachable activities identified by the plugin, the ground truth, and Gator

run-time. This phenomenon is not related to the plugin and it is due to the fact that developers do not remove unused activities from the manifest files of their apps; those unused activities can be considered as legacy code produced after the release of a new version of the app. Legacy activities do not impact the accuracy of the approach since in the ENGs produced by the plugin they are simple islands composed of one node, which are never considered by the prefetching-related algorithms.

6.9 Related work

Prefetching has been explored and applied successfully in distributed systems previously. To the best of our knowledge Bouquet (Li et al., 2016) and PALOMA (Zhao et al., 2018b) are the first approaches proposed in the literature to prefetch network requests of Android Apps. Despite their promising results these approaches can still be improved.

Bouquet applies program analysis techniques to bundle HTTP requests in order to reduce energy consumption in mobile apps. The approach detects Sequential HTTP Requests Sessions (SHRS), in which the generation of the first request implies that the following requests will also be made, and then bundles the requests together to save energy. This can be considered a form of prefetching. This work, however, does not address inter-callback analysis and the SHRS are always in the same callback. Therefore, the prefetching only happens a few statements ahead (within milliseconds most of the time) and has no tangible effect on app execution time.

PALOMA is the first technique to apply program analysis to address *what* and *when* to prefetch certain HTTP requests in mobile apps in order to reduce user-perceived latency. This technique was only evaluated on a small number of apps and its performance depends on the flaws of web caching schemes employed in the original apps. With respect to PALOMA, our approach acts at development time and at run-time in order to address *when* and *what* network resources can be prefetched considering users' navigation within the app, how users interaction patterns change potentially avoiding to unhandled control/data-flow paths. POLOMA does not consider user navigation through the app nor the history of past requests, but adds method calls to the local proxy in the source code where an HTTP request is spotted.

6.10 Conclusion and future work

In this chapter we presented a new technique for navigation-aware and personalized prefetching of network requests in Android apps. The proposed technique works at a higher level of abstraction with respect to state-of-the-art approaches (e.g., callback-based prefetching) and focusses on the so-called navigation graph of the app. Focusing on the navigation graph opens for a new family of prefetching opportunities. Firstly, the navigation graph can act as internal model for run-time prefetching algorithms, which now can look ahead several steps into the future network resources which will be requested by the app. Secondly, it allows us and other researchers to develop prefetching algorithms which take into account the unique and user-specific navigation patterns exhibited by each user, potentially reaching better results in terms of hit rate w.r.t. the one-size-fits-all prefetching approaches existing today.

In the long term we are planning to design and develop different variations of the prefetching algorithms, each of them following different strategies. For example, we will formulate prefetching as a variation of state reachability analysis with probabilities and exploit model checking at run-time for its resolution; we will map the ENG to different variations of Markov chains with probabilities; we will formulate prefetching as a graph-based combinatorial optimization problem and solve it analytically, etc. We will design and conduct larges-scale experiments on the accuracy of each of the above mentioned variations involving real apps from the Google Play store and (possibly) reusing already existing benchmarks.

Chapter 7

Conclusion

In this thesis, we investigate challenges in design, implementation and use of self-adaptive systems from the Web of Things (WoT) to mobile software perspective. Starting by a state of the art study of the current technological space for architecting Web technology-based IoT software, the main contributions of this thesis are:

- providing a complete, comprehensive and replicable picture of the state of the art on the current technical design space for WoT, focusing on the emerging Fog Computing paradigm;
- designing a modeling framework and implementing a Decision Support System for supporting designers and software architect in the process of modeling a middleware-induced software system's architecture;
- proposing a reflective model whose aim is to inject adaptation into existing middleware allowing a software system to dynamically change its logic without internal changes to the code;
- presenting an approach to complex adaptive mobile applications modeling and implementation, able to dynamically change according to changed behavioral properties, state and/or text variables and user's preference;
- presenting a new technique for navigation-aware and personalized prefetching of network requests in Android apps . The proposed idea allows the development of approaches which adapt their prefetching behaviour according to the unique navigation patterns each user exhibits while interacting with a mobile app.

Bibliography

Adam Lella, Andrew Lipsman (2017). *The 2017 U.S. Mobile App Report*. comsCore white paper.

Aggarwal, Charu C, Naveen Ashish, and Amit Sheth (2013). "The internet of things: A survey from the data-centric perspective". In: *Managing and mining sensor data*. Springer, pp. 383–428.

Alaba, Fadele Ayotunde et al. (2017). "Internet of Things security: A survey". In: *J. Network and Computer Applications* 88, pp. 10–28. DOI: 10.1016/j.jnca.2017.04.002. URL: https://doi.org/10.1016/j.jnca.2017.04.002.

Alrawais, Arwa et al. (2017). "Fog computing for the internet of things: Security and privacy issues". In: *IEEE Internet Computing* 21.2, pp. 34–42.

Alshuqayran, N., N. Ali, and R. Evans (2016). "A Systematic Mapping Study in Microservice Architecture". In: *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44–51. DOI: 10.1109/SOCA.2016.15.

Android Application Fundamentals (2018). URL: https://developer.android.com/guide/components/fundamentals.

Android connectivity (2018). URL: https://developer.android.com/training/basics/network-ops/connecting.

Android Intents and Intent Filters (2018). URL: https://developer.android.com/guide/components/intents-filters.

Arcaini, Paolo, Elvinia Riccobene, and Patrizia Scandurra (2015). "Modeling and Analyzing MAPE-K Feedback Loops for Self-Adaptation". In: *10th IEEE/ACM International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015, Florence, Italy, May 18-19, 2015*, pp. 13–23.

Atamli, Ahmad and Andrew P. Martin (2014). "Threat-Based Security Analysis for the Internet of Things". In: *2014 International Workshop on Secure Internet of Things, SIoT 2014, Wroclaw, Poland, September 10, 2014*. Ed. by Gabriel Ghinita, Razvan Rughinis, and Ahmad-Reza Sadeghi. IEEE Computer Society, pp. 35–43. ISBN: 978-1-4799-7907-3. DOI: 10.1109/SIoT.2014.10. URL: https://doi.org/10.1109/SIoT.2014.10.

Autili, Marco, Paolo Di Benedetto, and Paola Inverardi (2009). "Context-Aware Adaptive Services: The PLASTIC Approach". In: *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pp. 124–139.

Autili, Marco, Paola Inverardi, and Massimo Tivoli (2014). "CHOREOS: Large scale choreographies for the future internet". In: *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse*

Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014, pp. 391–394.

Avgeriou, Paris and Uwe Zdun (2005). "Architectural patterns revisited–a pattern language". In: *Proc. 10th European Conf. Pattern Languages of Programs (EuroPLoP)*, pp. 431–478.

Avgeriou, Paris et al. (2011). *Relating software requirements and architectures*. Springer Science & Business Media.

Baader, Franz et al., eds. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

Bakhshandeh, Marzieh et al. (2013). "A Modular Ontology for the Enterprise Architecture Domain". In: *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2013 17th IEEE International*. IEEE, pp. 5–12.

Bass, L., M. Klein, and F. Bachmann (2002). "Quality attribute design primitives and the attribute driven design method." In: *Revised Papers from 4th Int.Workshop on Software Product-Family Engineering*. Vol. 2290. Springer, pp. 169–186.

Bass, Len, Paul Clements, and Rick Kazman (2005). *Software architecture in practice*. Boston ; Munich [u.a.]: Addison-Wesley. ISBN: 0-321-15495-9.

Ben Martin (2018). *Global Digital Future in Focus - 2018 International Edition*. comsCore white paper.

Bencomo, Nelly and Gordon Blair (2009). "Using architecture models to support the generation and operation of component-based adaptive systems". In: *SEAMS'09*. Springer, pp. 183–200.

Bermbach, David et al. (2017). "A Research Perspective on Fog Computing". In: *Proceedings of the 2nd Workshop on IoT Systems Provisioning & Management for Context-Aware Smart Cities*. Springer.

Berners-Lee, T., J. Hendler, and O. Lassila (2001). "The Semantic Web". In: *The Scientific American* 284.5, pp. 34–43.

Bettini, Claudio et al. (2010). "A survey of context modelling and reasoning techniques". In: *Pervasive and Mobile Computing* 6.2, pp. 161–180.

Blair, Gordon S, Geoff Coulson, and Paul Grace (2004). "Research directions in reflective middleware: the Lancaster experience". In: *Proceedings of the 3rd workshop on Adaptive and reflective middleware*. ACM, pp. 262–267.

Blair, Gordon S et al. (1999). "The design of a resource-aware reflective middleware architecture". In: *Meta-Level Architectures and Reflection*. Springer, pp. 115–134.

Bobillo, Fernando and Umberto Straccia (2011). "Fuzzy Ontology Representation using OWL 2". In: *International Journal of Approximate Reasoning* 52 (7), pp. 1073–1094.

— (2016). "The Fuzzy Ontology Reasoner *fuzzyDL*". In: *Knowledge-Based Systems* 95, pp. 12 –34. DOI: 10.1016/j.knosys.2015.11.017. URL: http://www.sciencedirect.com/science/article/pii/S0950705115004621.

Bobillo, Fernando et al. (2015). "Fuzzy Description Logics in the framework of Mathematical Fuzzy Logic". In: *Handbook of Mathematical Fuzzy Logic, Volume 3*. Ed. by Carles Noguera Petr Cintula Christian Fermüller. Vol. 58. Studies in Logic, Mathematical Logic and Foundations. College Publications. Chap. 16, pp. 1105–1181. ISBN: 978-1848901933.

Bolchini, Cristiana et al. (2011). "Context Modeling and Context Awareness: steps forward in the Context-ADDICT project." In: *IEEE Data Eng. Bull.* 34.2, pp. 47–54.

Bonomi, Flavio et al. (2012). "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, pp. 13–16.

Botella, Pere et al. (2001). "Modeling non-functional requirements". In: *Proceedings of Jornadas de Ingenieria de Requisitos Aplicada JIRA* 2001.

Boyle, David and Thomas Newe (2008). "Securing Wireless Sensor Networks: Security Architectures". In: *JNW* 3.1, pp. 65–77. DOI: 10.4304/jnw.3.1.65-77. URL: https://doi.org/10.4304/jnw.3.1.65-77.

Bucchiarone, Antonio et al. (2015). "Rule-Based Modeling and Static Analysis of Self-adaptive Systems by Graph Transformation". In: *Software, Services, and Systems*, pp. 582–601.

Buschmann, F. et al. (1996a). *Pattern-oriented Software Architecture: A System of Patterns*. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 0-471-95869-7.

Buschmann, Frank, Kevlin Henney, and Douglas C Schmidt (2007a). *Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing*. Wiley.

— (2007b). *Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing*.

Buschmann, Frank et al. (1996b). *Pattern-oriented software architecture: a system of patterns*. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 0-471-95869-7.

— (1996c). *Pattern-oriented software architecture: a system of patterns*. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 0-471-95869-7.

Byers, Charles C (2017). "Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for FOG-enabled IoT networks". In: *IEEE Communications Magazine* 55.8, pp. 14–20.

Calegari, Silvia and Elie Sanchez (2007). "A fuzzy ontology-approach to improve semantic information retrieval". In: *Proceedings of the Third International Conference on Uncertainty Reasoning for the Semantic Web-Volume 327*. CEUR-WS. org, pp. 117–122.

Capilla, Rafael et al. (2006). "A web-based tool for managing architectural design decisions". In: *ACM SIGSOFT software engineering notes* 31.5, p. 4.

Cazzola, Walter et al. (1999). "Rule-based strategic reflection: Observing and modifying behaviour at the architectural level". In: *Automated Software Engineering, 1999. 14th IEEE International Conference on.* IEEE, pp. 263–266.

Cheng, Betty H C et al. (2009). "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty". In: *LNCS*. Vol. 5795. LNCS. Springer, pp. 468–483.

Chung, Jaewoo and Chris Schmandt (2009). "Going my way: a user-aware route planner". In: *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 1899–1902.

Chung, Lawrence and Julio Cesar Sampaio do Prado Leite (2009). "On non-functional requirements in software engineering". In: *Conceptual modeling: Foundations and applications*. Springer, pp. 363–379.

Chung, Lawrence et al. (2012). *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media.

Cubo, Javier et al. (2014). "Adaptive Services for the Future Internet". In: *J. UCS* 20.8, pp. 1046–1048.

Cuenca-Grau, B. et al. (2008). "OWL 2: The next step for OWL". In: *Journal of Web Semantics* 6.4, pp. 309–322.

Cugola, Gianpaolo and Alessandro Margara (2012). "Processing flows of information: From data stream to complex event processing". In: *ACM Computing Surveys (CSUR)* 44.3, p. 15.

— (2013). "Deployment strategies for distributed complex event processing". In: *Computing* 95.2, pp. 129–156.

Curry, Edward (2004). "Message-oriented middleware". In: *Middleware for communications*, pp. 1–28.

Cysneiros, Luiz Marcio (2007). "Evaluating the Effectiveness of Using Catalogues to Elicit Non-Functional Requirements." In: *WER*, pp. 107–115.

Dastjerdi, A. V. and R. Buyya (2016). "Fog Computing: Helping the Internet of Things Realize Its Potential". In: *Computer* 49.8, pp. 112–116. ISSN: 0018-9162. DOI: 10.1109/MC.2016.245.

Davis, Alan M (1993). *Software requirements: objects, functions, and states*. Prentice-Hall, Inc.

Dayarathna, Miyuru and Srinath Perera (Feb. 2018). "Recent Advancements in Event Processing". In: *ACM Comput. Surv.* 51.2, 33:1–33:36. ISSN: 0360-0300. DOI: 10.1145/3170432. URL: http://doi.acm.org/10.1145/3170432.

Di Noia, Tommaso, Marina Mongiello, and Eugenio Di Sciascio (2014). "Ontology-driven pattern selection and matching in software design". In: *European Conference on Software Architecture*. Springer, pp. 82–89. ISBN: 978-331909969-9. DOI: 10.1007/978-3-319-09970-5_8.

Diaz-Pace, Andres et al. (2008). "Integrating quality-attribute reasoning frameworks in the ArchE design assistant". In: *Quality of Software Architectures. Models and Architectures*. Springer, pp. 171–188.

Dietrich, Jens and Chris Elgar (2005). "A formal description of design patterns using OWL". In: *Software Engineering Conference, 2005. Proceedings. 2005 Australian*. IEEE, pp. 243–250.

Dobson, Glen, Stephen Hall, and Gerald Kotonya (2007). "A Domain-Independent Ontology for Non-Functional Requirements". In: *Proceedings of ICEBE 2007, IEEE International Conference on e-Business Engineering and the Workshops SOAIC 2007, SOSE 2007, SOKM 2007, 24-26 October, 2007, Hong Kong, China*, pp. 563–566.

Dolev, Danny and Andrew Chi-Chih Yao (1983). "On the security of public key protocols". In: *IEEE Trans. Information Theory* 29.2, pp. 198–207. DOI: 10.1109/TIT.1983.1056650. URL: https://doi.org/10.1109/TIT.1983.1056650.

Dragoni, Nicola, Alberto Giaretta, and Manuel Mazzara (2017). "The Internet of Hackable Things". In: *CoRR* abs/1707.08380. arXiv: 1707.08380. URL: http://arxiv.org/abs/1707.08380.

Dragoni, Nicola et al. (2016). "Microservices: yesterday, today, and tomorrow". In: *CoRR* abs/1606.04036. arXiv: 1606.04036. URL: http://arxiv.org/abs/1606.04036.

Egyed, Alexander and Paul Grunbacher (2004). "Identifying requirements conflicts and cooperation: How quality attributes and automated traceability can help". In: *Software, IEEE* 21.6, pp. 50–58.

El Jaouhari, Saad, Ahmed Bouabdallah, and Jean-Marie Bonnin (2017). "Security Issues of the Web of Things". In: *Managing the Web of Things*. Elsevier, pp. 389–424.

Fehling, C. et al. (2014). *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Vienna. ISBN: 9783709115671.

Fickas, Stephen and Martin S Feather (1995). "Requirements monitoring in dynamic environments". In: *Proc. of RE'95*. IEEE, pp. 140–147.

Fielding, Roy T and Richard N Taylor (2002). "Principled design of the modern Web architecture". In: *ACM Transactions on Internet Technology (TOIT)* 2.2, pp. 115–150.

Fielding, Roy Thomas (2000). "REST: Architectural Styles and the Design of Network-based Software Architectures". Doctoral dissertation. University of California, Irvine. URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

FP Jr, Brooks (1987). "No Silver Bullet Essence and Accidents of Software Engineering". In: 4. IEEE, pp. 10–19.

Francesco, P. D., I. Malavolta, and P. Lago (2017). "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption". In: *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 21–30. DOI: 10.1109/ICSA.2017.24.

Franch, Xavier (1998). "Systematic formulation of non-functional characteristics of software". In: *Requirements Engineering, 1998. Proceedings. 1998 Third International Conference on*. IEEE, pp. 174–181.

Francis, B. (Mar. 2018). *Web Thing API (Unofficial Draft)*. Tech. rep. World Wide Web Concortium (W3C). URL: https://iot.mozilla.org/wot/.

Fremantle, Paul and Philip Scott (2017). "A survey of secure middleware for the Internet of Things". In: *PeerJ Computer Science* 3, e114. DOI: 10.7717/peerj-cs.114. URL: https://doi.org/10.7717/peerj-cs.114.

Frigeri, Achille, Liliana Pasquale, and Paola Spoletini (Aug. 2014). "Fuzzy Time in Linear Temporal Logic". In: *ACM Trans. Comput. Logic* 15.4, 30:1–30:22.

Gallidabino, ANDREA et al. (2017). "Architecting liquid software". In: *Journal of Web Engineering* 16.5&6, pp. 433–470.

Gamma, Erich et al. (1994). *Design patterns: elements of reusable object-oriented software*. Pearson Education.

Garlan, David and Mary Shaw (1994). "An introduction to software architecture". In: technical report.

Garlan, David et al. (2004). "Rainbow: Architecture-based self-adaptation with reusable infrastructure". In: *Computer* 37.10, pp. 46–54.

Gašević, Dragan, Nima Kaviani, and Milan Milanović (2009). "Ontologies and software engineering". In: *Handbook on Ontologies*. Springer, pp. 593–615.

Geiger, Franz-Xaver et al. (2018). "A Graph-based Dataset of Commit History of Real-World Android apps". In: *Proceedings of the 15th International Conference on Mining Software Repositories, MSR*. New York, NY: ACM, pp. 30–33. URL: http://www.ivanomalavolta.com/files/papers/MSR_2018.pdf.

Glinz, Martin (2007). "On non-functional requirements". In: *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*. IEEE, pp. 21–26.

Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2018 (2018). URL: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

Granjal, Jorge, Edmundo Monteiro, and Jorge Sá Silva (2015). "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues". In: *IEEE Communications Surveys and Tutorials* 17.3, pp. 1294–1312. DOI: 10.1109/COMST.2015.2388550. URL: https://doi.org/10.1109/COMST.2015.2388550.

Gross, Daniel and Eric Yu (2001). "From non-functional requirements to design through patterns". In: *Requirements Engineering* 6.1, pp. 18–36.

Grunske, Lars (2008). "Specification patterns for probabilistic quality properties". In: *Proc. of ICSE'08*. IEEE, pp. 31–40.

Guinard, Dominique (2009). "Towards the web of things: Web mashups for embedded devices". In: *In MEM 2009 in Proceedings of WWW 2009. ACM*.

Guinard, Dominique and Vlad Trifa (2016). *Building the web of things: with examples in node. js and raspberry pi*. Manning Publications Co.

Guinard, Dominique, Vlad Trifa, and Erik Wilde (2010). "A resource oriented architecture for the web of things". In: *Internet of Things (IOT), 2010*. IEEE, pp. 1–8.

Guinard, Dominique et al. (2011a). "From the internet of things to the web of things: Resource-oriented architecture and best practices". In: *Architecting the Internet of things*, pp. 97–129.

— (2011b). "From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices". In: *Architecting the Internet of Things*. Ed. by Dieter Uckelmann, Mark Harrison, and Florian Michahelles. Springer, pp. 97–129. ISBN: 978-3-642-19156-5.

Guizzardi, Renata S. S. et al. (2014). "An Ontological Interpretation of Non-Functional Requirements". In: *Formal Ontology in Information Systems - Proceedings of the Eighth International Conference, FOIS 2014, September, 22-25, 2014, Rio de Janeiro, Brazil*, pp. 344–357.

Harb, Dania, Cédric Bouhours, and Hervé Leblanc (2009). "Using an ontology to suggest software design patterns integration". In: *Models in Software Engineering*. Springer, pp. 318–331.

Harrison, Neil B and Paris Avgeriou (2010a). "How do architecture patterns and tactics interact? A model and annotation". In: *Journal of Systems and Software* 83.10, pp. 1735–1758.

— (2010b). "Implementing reliability: the interaction of requirements, tactics and architecture patterns". In: *Architecting dependable systems VII*. Springer, pp. 97–122.

Harrison, Neil B, Paris Avgeriou, and Uwe Zdun (2010). "On the impact of fault tolerance tactics on architecture patterns". In: *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems*. ACM, pp. 12–21.

Hartke, K. (Oct. 2017). *The Constrained RESTful Application Language (CoRAL)*. Tech. rep. IETF. URL: https://datatracker.ietf.org/doc/draft-kelly-json-hal/.

Henninger, Scott and Padmapriya Ashokkumar (2005). "An Ontology-Based Infrastructure for Usability Design Patterns". In: *Proc. Semantic Web Enabled Software Engineering (SWESE), Galway, Ireland*, pp. 41–55.

— (2006). "An ontology-based metamodel for software patterns". In: *8th Intl. Conf. on Software Engineering and Knowledge Engineering (SEKE2006)*.

Henninger, Scott and Victor Corrêa (2007). "Software pattern communities: Current practices and challenges". In: *Proceedings of the 14th Conference on Pattern Languages of Programs*. ACM, p. 14.

Hopcroft, John E, Rajeev Motwani, and Jeffrey D Ullman (2001). "Introduction to automata theory, languages, and computation". In: *ACM SIGACT News* 32.1, pp. 60–65.

Horrocks, Ian, Oliver Kutz, and Ulrike Sattler (2006). "The Even More Irresistible $\mathcal{SROIQ}$". In: *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06)*. AAAI Press, pp. 57–67. ISBN: 978-1-57735-271-6.

Huang, Gang, Xuanzhe Liu, and Hong Mei (2007). "SOAR: towards dependable Service-Oriented Architecture via reflective middleware". In: *International Journal of Simulation and Process Modelling* 3.1-2, pp. 55–65.

Huebscher, Markus C and Julie A McCann (2008). "A survey of autonomic computing-degrees, models, and applications". In: *ACM Computing Surveys (CSUR)* 40.3, p. 7.

Ikram, Ahsan et al. (2013). "Approaching the Internet of things (IoT): a modelling, analysis and abstraction framework". In: *Concurrency and Computation: Practice and Experience*.

Internet of things research study (2015). http://www8.hp.com/us/en/hp-news/press-release.html?id=1744676. [online, visited 30.11.2017].

Jansen, Anton et al. (2007). "Tool support for architectural decisions". In: *Software Architecture, 2007. WICSA'07. The Working IEEE/IFIP Conference on*, pp. 4–4.

Joorabchi, Mona Erfani, Ali Mesbah, and Philippe Kruchten (2013). "Real challenges in mobile app development". In: *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*. IEEE, pp. 15–24.

Kaebisch, Sebastian and Takuki Kamiya (Sept. 2017). *Web of Things (WoT) Thing Description*. Tech. rep. World Wide Web Concortium (W3C).

Kajimoto, Kazuo, Matthias Kovatsch, and Uday Davuluru (Sept. 2017). *Web of Things (WoT) Architecture*. Tech. rep. World Wide Web Concortium (W3C).

Kampffmeyer, Holger and Steffen Zschaler (2007). "Finding the pattern you need: The design pattern intent ontology". In: *Model Driven Engineering Languages and Systems*. Springer, pp. 211–225.

Kelly, M. (Nov. 2016). *JSON Hypertext Application Language (JSON-HAL)*. Tech. rep. IETF. URL: https://datatracker.ietf.org/doc/draft-kelly-json-hal/.

Kis, Zoltan et al. (Oct. 2017). *Web of Things (WoT) Scripting API*. Tech. rep. World Wide Web Concortium (W3C).

Koster, M. (Sept. 2017a). *Media Types for Hypertext Sensor Markup (HSML)*. Tech. rep. IETF. URL: https://datatracker.ietf.org/doc/draft-koster-t2trg-hsml/.

Koster, Michael (Oct. 2017b). *Web of Things (WoT) Protocol Binding Templates*. Tech. rep. World Wide Web Concortium (W3C).

Kramer, Jeff and Jeff Magee (2007). "Self-managed systems: an architectural challenge". In: *Future of Software Engineering, 2007. FOSE'07*. IEEE, pp. 259–268.

Kruchten, Philippe (2004a). "An ontology of architectural design decisions in software intensive systems". In: *2nd Groningen Workshop on Software Variability*. Groningen, The Netherlands, pp. 54–61.

— (2004b). "An ontology of architectural design decisions in software intensive systems". In: *2nd Groningen Workshop on Software Variability*, pp. 54–61.

Levandowsky, Michael and David Winter (1971). "Distance between sets". In: *Nature* 234.5323, pp. 34–35.

Lewis, Clayton (1982). *Using the" thinking-aloud" method in cognitive interface design*. IBM TJ Watson Research Center.

Lewis, James and Martin Fowler. (2015). *Microservices.* https://martinfowler.com/articles/mi

Li, Ding et al. (2016). "Automated energy optimization of http requests for mobile applications". In: *Proceedings of the 38th international conference on software engineering*. ACM, pp. 249–260.

Li, Li et al. (2017). "Static analysis of android apps: A systematic literature review". In: *Information and Software Technology* 88, pp. 67–95.

Li, Zengyang, Peng Liang, and Paris Avgeriou (2013). "Application of knowledge-based approaches in software architecture: a systematic mapping study". In: *Information and Software technology* 55, pp. 777–794.

Liu, Chi-Lun (2010). "Ontology-Based Conflict Analysis Method in Non-functional Requirements". In: *9th IEEE/ACIS International Conference on Computer and Information Science, IEEE/ACIS ICIS 2010, 18-20 August 2010, Yamagata, Japan*, pp. 491–496.

Liu, Xuanzhe et al. (2017). "Understanding diverse usage patterns from large-scale appstore-service profiles". In: *IEEE Transactions on Software Engineering*.

López, Claudia, Luiz Marcio Cysneiros, and Hernán Astudillo (2008). "NDR ontology: sharing and reusing NFR and design rationale knowledge". In: *Managing Requirements Knowledge, 2008. MARK'08. First International Workshop on*. IEEE, pp. 1–10.

Luckham, David (2002). *The power of events*. Vol. 204. Addison-Wesley Reading.

M. Belshe R. Peon, M. Thomson, ed. (2015). *RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)*. Internet Engineering Task Force (IETF).

Ma, Yun et al. (2018). "A Tale of Two Fashions: An Empirical Study on the Performance of Native Apps and Web Apps on Android". In: *IEEE Transactions on Mobile Computing* 17.5, pp. 990–1003.

Madlmayr, Gerald et al. (2008). "NFC Devices: Security and Privacy". In: *Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, 2008, Technical University of Catalonia, Barcelona , Spain*. IEEE Computer Society, pp. 642–647. ISBN: 978-0-7695-3102-1. DOI: 10.1109/ARES.2008.105. URL: https://doi.org/10.1109/ARES.2008.105.

Maes, Pattie (1987). "Concepts and experiments in computational reflection". In: *ACM Sigplan Notices* 22.12, pp. 147–155.

Mahmud, Redowan, Ramamohanarao Kotagiri, and Rajkumar Buyya (2018). "Fog computing: A taxonomy, survey and future directions". In: *Internet of Everything*. Springer, pp. 103–130.

Mairiza, Dewi, Didar Zowghi, and Nurie Nurmuliani (2009). "Managing conflicts among non-functional requirements". In: *12th Australian Workshop on Requirements Engineering*. University of Technology, Sydney, pp. 11–19.

Mäkitalo, Aaltonen, and Mikkonen (2016). "Coordinating Proactive Social Devices in a Mobile Cloud: Lessons Learned and a Way Forward". In: *Proceedings of the International Conference on Mobile Software Engineering and Systems*. MOBILESoft '16. Austin, Texas: ACM, pp. 179–188. ISBN: 978-1-4503-4178-3. DOI: 10.1145/2897073.2897079. URL: http://doi.acm.org/10.1145/2897073.2897079.

Mäkitalo, Niko et al. (2018). "Safe and Secure Execution at the Network Edge: A Framework for Coordinating Cloud, Fog, and Edge". In: *IEEE Software*.

Martins, Jaime A, Andriy Mazayev, and Noélia Correia (2017). "Hypermedia APIs for the Web of Things". In: *IEEE Access* 5, pp. 20058–20067.

Mayer, Ruben, Muhammad Adnan Tariq, and Kurt Rothermel (2017). "Minimizing Communication Overhead in Window-Based Parallel Complex Event Processing". In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. ACM, pp. 54–65.

Mayer, Simon and David S Karam (2012). "A computational space for the web of things". In: *Proceedings of the Third International Workshop on the Web of Things*. ACM, p. 8.

McKinley, Philip K et al. (2004). "Composing adaptive software". In: *Computer*, pp. 56–64.

Mendez, Diego M., Ioannis Papapanagiotou, and Baijian Yang (2017). "Internet of Things: Survey on Security and Privacy". In: *CoRR* abs/1707.01879. arXiv: 1707.01879. URL: http://arxiv.org/abs/1707.01879.

Mikkonen, Tommi (1998). "Formalizing design patterns". In: *Proceedings of the 20th international conference on Software engineering*. IEEE Computer Society, pp. 115–124.

Mongiello, Marina, Patrizio Pelliccione, and Massimo Siancalepore (2015). "Ac-Contract: run-time verification of context-aware systems". In: *SEAMS '15. ICSE Workshop*, pp. 106–115.

Mongiello, Marina et al. (2015). "Adaptive architectural model for Future internet applications". In: *Proc. of the 5th International Workshop on Adaptive services for future internet*.

Montero, Susana, Paloma Díaz, and Ignacio Aedo (2003). "Formalization of web design patterns using ontologies". In: *Advances in Web Intelligence*. Springer, pp. 179–188.

Moser, Thomas et al. (2009). "Semantic event correlation using ontologies". In: *On the Move to Meaningful Internet Systems: OTM 2009*. Springer, pp. 1087–1094.

Mylopoulos, John, Lawrence Chung, and Brian Nixon (1992). "Representing and using nonfunctional requirements: A process-oriented approach". In: *Software Engineering, IEEE Transactions on* 18.6, pp. 483–497.

N.Harrison, P. Avgeriou (2007). "Pattern-driven architectural partitioning: Balancing functional and non-functional requirements". In: *Second International Conference on Digital Telecommunications 2007. ICDT '07. IEEE*. IEEE, pp. 21–26.

Ni, Jianbing et al. (2017). "Securing fog computing for internet of things applications: Challenges and solutions". In: *IEEE Communications Surveys & Tutorials*.

Number of available applications in the Google Play Store from December 2009 to June 2018 (2018). URL: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

OASIS Message Queuing Telemetry Transport (MQTT) TC (2014). *MQTT Version 3.1.1*. standard. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. Oasis.

Ometov, A. et al. (2016). "Implementing secure network-assisted D2D framework in live 3GPP LTE deployment". In: *2016 IEEE International Conference on Communications Workshops (ICC)*, pp. 749–754. DOI: 10.1109/ICCW.2016.7503877.

Oreizy, Peyman, Nenad Medvidovic, and Richard N Taylor (1998). "Architecture-based runtime software evolution". In: *Proceedings of the 20th international conference on Software engineering*. IEEE Computer Society, pp. 177–186.

Pahl, Claus and Pooyan Jamshidi (2016). "Microservices: A Systematic Mapping Study." In: *CLOSER (1)*, pp. 137–146.

Palomba, Fabio et al. (2018). "Crowdsourcing user reviews to support the evolution of mobile apps". In: *Journal of Systems and Software* 137, pp. 143–162.

Pan, Jeff Z et al. (2013). *Ontology-Driven Software Development*. Springer.

Pelliccione, Patrizio et al. (2008). "An architectural approach to the correct and automatic assembly of evolving component-based systems". In: *Journal of Systems and Software* 81.12, pp. 2237–2251.

Perera, Charith et al. (2014). "Context aware computing for the internet of things: A survey". In: *Communications Surveys & Tutorials, IEEE* 16.1, pp. 414–454.

Philip, K, P Eric, and HC Betty (2004). "Composing adaptive software". In: *IEEE Computer*.

Pimentel, João et al. (2013). "From Requirements to Architectures for Better Adaptive Software Systems". In: *Proceedings of the 6th International* i* *Workshop 2013, Valencia, Spain, June 17-18, 2013*. Ed. by Jaelson Castro et al. Vol. 978. CEUR Workshop Proceedings. CEUR-WS.org, pp. 91–96. URL: http://ceur-ws.org/Vol-978/paper_16.pdf.

R. de Lemos et al. (2013). "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap". In: *Software Engineering for Self-Adaptive Systems II*. Vol. 7475. LNCS. Springer Berlin Heidelberg, pp. 1–32.

Rahmani, Amir M et al. (2018). "Exploiting smart e-health gateways at the edge of healthcare internet-of-things: a fog computing approach". In: *Future Generation Computer Systems* 78, pp. 641–658.

Randika, HC et al. (2010). "Scalable fault tolerant architecture for complex event processing systems". In: *Advances in ICT for Emerging Regions (ICTer), 2010 International Conference on*. IEEE, pp. 86–96.

Rashwan, Abderahman, Olga Ormandjieva, and René Witte (2013). "Ontology-Based Classification of Non-functional Requirements in Software Specifications: A New Corpus and SVM-Based Classifier". In: *37th Annual IEEE Computer Software and Applications Conference, COMPSAC 2013, Kyoto, Japan, July 22-26, 2013*, pp. 381–386.

Raverdy, Pierre-Guillaume, HLV Gong, and Rodger Lea (1998). "DART: a reflective middleware for adaptive applications". In: *OOPSLA'98 Workshop# 13: Reflective programming in C++ and Java*.

Ravindranath, Lenin et al. (2012). "AppInsight: Mobile App Performance Monitoring in the Wild." In: *OSDI*. Vol. 12, pp. 107–120.

Razzaque, Mohammad Abdur et al. (2016a). "Middleware for Internet of Things: A Survey". In: *IEEE Internet of Things Journal* 3.1, pp. 70–95. DOI: 10.1109/JIOT.2015.2498900. URL: https://doi.org/10.1109/JIOT.2015.2498900.

Razzaque, Mohammad Abdur et al. (2016b). "Middleware for internet of things: a survey". In: *IEEE Internet of Things Journal* 3.1, pp. 70–95.

Riccobene, Elvinia and Patrizia Scandurra (2015). "Formal modeling self-adaptive service-oriented applications". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pp. 1704–1710.

Rosa, Nelson S, Paulo RF Cunha, and George RR Justo (2002). "Process NFL: A language for describing non-functional properties". In: *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*. IEEE, pp. 3676–3685.

Rupnik, Rok (2009). "Decision support system to support the solving of classification problems in telecommunications". In: *Informacije Midem - Journal of microelectronic electronic component and materials* 39.3, pp. 168–177.

Sánchez, Daniel and Andrea GB Tettamanzi (2006). "Fuzzy quantification in fuzzy description logics". In: *Capturing intelligence* 1, pp. 135–159.

Schulte, Roy W and Yefim V Natis (2003). "Event-driven architecture complements SOA". In: *Gartner Research Note, July* 8.

Scoccia, Gian Luca et al. (2018). "An Investigation into Android Run-time Permissions from the End Users' Perspective". In: *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. MOBILESoft '18. Gothenburg, Sweden: ACM, pp. 45–55. ISBN: 978-1-4503-5712-8. DOI: 10.1145/3197231.3197236. URL: http://doi.acm.org/10.1145/3197231.3197236.

Serbedzija, Nikola B and Stephen H Fairclough (2009). "Biocybernetic loop: from awareness to evolution". In: *IEEE Congress on Evolutionary Computation*. IEEE, pp. 2063–2069.

Shaw, Mary and David Garlan (1996). *Software architecture: perspectives on an emerging discipline*. Vol. 1. Prentice Hall Englewood Cliffs.

Sheng, Z. et al. (2013). "A survey on the ietf protocol suite for the internet of things: standards, challenges, and opportunities". In: *IEEE Wireless Communications* 20.6, pp. 91–98. ISSN: 1536-1284. DOI: 10.1109/MWC.2013.6704479.

Shi, W. and S. Dustdar (2016). "The Promise of Edge Computing". In: *Computer* 49.5, pp. 78–81. ISSN: 0018-9162. DOI: 10.1109/MC.2016.145.

Smith, Brian Cantwell (1982). "Procedural reflection in programming languages". PhD thesis. Massachusetts Institute of Technology.

Soto, José Angel Carvajal et al. (2016). "CEML: Mixing and Moving Complex Event Processing and Machine Learning to the Edge of the Network for IoT Applications". In: *Proceedings of the 6th International Conference on the Internet of Things*. IoT'16. Stuttgart, Germany: ACM, pp. 103–110. ISBN: 978-1-4503-4814-0. DOI: 10.1145/2991561.2991575. URL: http://doi.acm.org/10.1145/2991561.2991575.

Starks, Fabrice, Thomas Peter Plagemann, and Stein Kristiansen (2017). "DCEPSim: An Open Simulation Framework for Distributed CEP". In: *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*. ACM, pp. 180–190.

Stirbu, V. (2008). "Towards a RESTful Plug and Play Experience in the Web of Things". In: *2008 IEEE International Conference on Semantic Computing*, pp. 512–517. DOI: 10.1109/ICSC.2008.51.

Straccia, Umberto (2005). "Description Logics with Fuzzy Concrete Domains". In: *21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*. Ed. by Fahiem Bachus and Tommi Jaakkola. Edinburgh, Scotland: AUAI Press, pp. 559–567.

— (2013). *Foundations of Fuzzy Logic and Semantic Web Languages*. CRC Studies in Informatics Series. Chapman & Hall.

Sydow, Lexi (2018). *Global App Downloads Grew 15% and Consumer Spend 20% in Q2 2018 Versus a Year Ago*. App Annie white paper. URL: https://www.appannie.com/en/insights/market-data/global-app-downloads-grew-15-and-consumer-spend-20-in-q2-2018-versus-a-year-ago/.

Taibi, Toufik and David Chek Ling Ngo (2003). "Formal Specification of Design Patterns - A Balanced Approach". In: *Journal of Object Technology* 2.4, pp. 127–140.

Taivalsaari, A. and T. Mikkonen (2017). "A Roadmap to the Programmable World: Software Challenges in the IoT Era". In: *IEEE Software* 34.1, pp. 72–80. ISSN: 0740-7459. DOI: 10.1109/MS.2017.26.

Taivalsaari, A. et al. (2011). "The Death of Binary Software: End User Software Moves to the Web". In: *Creating, Connecting and Collaborating through Computing (C5), 2011 Ninth International Conference on*, pp. 17–23. DOI: 10.1109/C5.2011.9.

Taylor, R. N., N. Medvidovic, and E. M. Dashofy (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing. ISBN: 0470167742, 9780470167748.

technology, ISO/IEC JTC 1 Information (2016). *ISO/IEC 20922:2016*. standard. International Organization for Standardization (ISO).

Thönes, J. (2015). "Microservices". In: *IEEE Software* 32.1, pp. 116–116. ISSN: 0740-7459. DOI: 10.1109/MS.2015.11.

Tichy, Walter F (1997). "A catalogue of general-purpose software design patterns". In: *Technology of Object-Oriented Languages and Systems, 1997. TOOLS 23. Proceedings*. IEEE, pp. 330–339.

Torra, Vicenç and Yasuo Narukawa (2007). *Information Fusion and Aggregation Operators*. Cognitive Technologies. Springer Verlag.

Tran, Nguyen Khoi et al. (Aug. 2017). "Searching the Web of Things: State of the Art, Challenges, and Solutions". In: *ACM Computing Surveys* 50.4, 55:1–55:34. ISSN: 0360-0300. DOI: 10.1145/3092695. URL: http://doi.acm.org/10.1145/3092695.

Tran, Quan and Lawrence Chung (1999). "NFR-Assistant: Tool support for achieving quality". In: *Application-Specific Systems and Software Engineering and Technology, 1999. ASSET'99. Proceedings. 1999 IEEE Symposium on*. IEEE, pp. 284–289.

Trifa, Vlad, Dominique Guinard, and David Carrera (Apr. 2017). *Web thing model W3C Member Submission*. Tech. rep. World Wide Web Concortium (W3C). URL: http://model.webofthings.io/.

Vasconcelos, Rafael Oliveira, Igor Vasconcelos, and Markus Endler (2014). "A middleware for managing dynamic software adaptation". In: *Proceedings of the 13th Workshop on Adaptive and Reflective Middleware*. ACM, p. 5.

Vovk, Leanid (2018). *How to choose an Android HTTP Library*. URL: https://appdevelopermagazine.com/5265/2017/6/5/how-to-choose-an-android-http-library.

Wen, Zhenyu et al. (2017). "Fog Orchestration for IoT Services: Issues, Challenges and Directions". In: *IEEE Internet Computing* 21.2, pp. 16–24.

Weyns, Danny et al. (2012). "A survey of formal methods in self-adaptive systems". In: *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. ACM, pp. 67–79.

Wohlin, C. et al. (2012). *Experimentation in Software Engineering*. Computer Science. Springer.

Yang, Shengqian et al. (2015). "Static window transition graphs for android (t)". In: *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, pp. 658–668.

Yang, Shengqian et al. (2018). "Static window transition graphs for Android". In: *Automated Software Engineering*. ISSN: 1573-7535. DOI: 10.1007/s10515-018-0237-6. URL: https://doi.org/10.1007/s10515-018-0237-6.

Yang, Yuchen et al. (2017). "A Survey on Security and Privacy Issues in Internet-of-Things". In: *IEEE Internet of Things Journal* 4.5, pp. 1250–1258. DOI: 10.1109/JIOT.2017.2694844. URL: https://doi.org/10.1109/JIOT.2017.2694844.

Zadeh, L. A. (1965). "Fuzzy Sets". In: *Information and Control* 8.3, pp. 338–353.

Zafari, Faheem and Ioannis Papapanagiotou (2015). "Enhancing iBeacon Based Micro-Location with Particle Filtering". In: *2015 IEEE Global Communications Conference, GLOBECOM 2015, San Diego, CA, USA, December 6-10, 2015*. IEEE, pp. 1–7. ISBN: 978-1-4799-5952-5. DOI: 10.1109/GLOCOM.2014.7417504. URL: https://doi.org/10.1109/GLOCOM.2014.7417504.

Zafari, Faheem et al. (2017). "Enhancing the accuracy of iBeacons for indoor proximity-based services". In: *IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25, 2017*. IEEE, pp. 1–7. ISBN: 978-1-4673-8999-0. DOI: 10.1109/ICC.2017.7996508. URL: https://doi.org/10.1109/ICC.2017.7996508.

Zhang, Haopeng, Yanlei Diao, and Neil Immerman (2014). "On complexity and optimization of expensive queries in complex event processing". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, pp. 217–228.

Zhang, Pengcheng et al. (2011). "Monitoring of probabilistic timed property sequence charts". In: *Software: Practice and Experience* 41.7, pp. 841–866.

Zhao, Kai and Lina Ge (2013). "A Survey on the Internet of Things Security". In: *Ninth International Conference on Computational Intelligence and Security, CIS 2013, Emei Mountain, Sichan Province, China, December 14-15, 2013*. IEEE Computer Society, pp. 663–667. ISBN: 978-1-4799-2548-3. DOI: 10.1109/CIS.2013.145. URL: https://doi.org/10.1109/CIS.2013.145.

Zhao, Yixue et al. (2018a). "Empirically Assessing Opportunities for Prefetching and Caching in Mobile Apps". In:

Zhao, Yixue et al. (2018b). "Leveraging Program Analysis to Reduce User-Perceived Latency in Mobile Applications". In: *International Conference on Software Engineering*.

Zhou, Honbo (2012). "The internet of things in the cloud: A middleware perspective". In: