Adaptive Quality of Context Optimization in the Internet of Things: Models and Methods

vorgelegt von
M. Sc.
Elif Eryilmaz-Sigwarth

an der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
-Dr.-Ing.-

genehmigte Dissertation

Promotionsausschuss:
Vorsitzender: Prof. Dr. Odej Kao (TU Berlin)
Gutachter: Prof. Dr. Dr. h.c. Sahin Albayrak (TU Berlin)
Gutachterin: Prof. Dr. Elisabeth André (Universität Augsburg)
Gutachter: Prof. Dr. Arkady Zaslavsky (Deakin University, Australien)
Tag der wissenschaftlichen Aussprache: 9. September 2020

Berlin 2020

Zusammenfassung

Das Internet der Dinge (Internet of Things, IoT) ist eine Grundlage des Internets der Zukunft und wird Milliarden intelligenter Objekte umfassen, die in der Lage sind, einander zu erfassen, zu betätigen und miteinander zu kommunizieren. Die jüngsten Fortschritte im Internet der Dinge (IoT) bieten reichhaltige Möglichkeiten für die Entwicklung flexiblerer, kontextsensitiver Lösungen. Anwendungen können von diesen flexiblen Mechanismen profitieren, indem sie sich an Veränderungen in der Erfassungsumgebung anpassen, wie z.B. das Verschwinden/die Verschlechterung von Sensoren oder die Nichtverfügbarkeit von Diensten. Diese Vision trägt dazu bei, dass IoT für flexiblere kontextsensitive Anwendungen und Dienste auf breiter Basis effektiv zu nutzen.

Das Verständnis von Sensordaten spielt eine wichtige Rolle beim kontextsensitiven Rechnen. Aufgrund der Skalierbarkeitsprobleme ist es jedoch nicht möglich, Daten aus allen verfügbaren Datenquellen über das IoT zu sammeln, um die Qualität kontextbezogener Anwendungen und Dienste zu verbessern. Das Hauptproblem besteht darin, die richtigen Datenquellen zu finden, die die besten Daten für spezifische Aufgaben der Kontexterkennung liefern. Die Auswahl geeigneter Sensoren für die spezifische Kontexterkennungsaufgabe ist eine wichtige Herausforderung, da im IoT eine große Anzahl von Sensoren zur Verfügung steht und sich ihre Qualität voneinander unterscheidet. Die verwandte Verarbeitungsmethode, um Informationen aus den Daten eines Sensors zu erhalten, wirkt sich auch auf die Gesamtqualität der Kontexterkennung aus.

In dieser Doktorarbeit wurden Modelle und Methoden zur Verbesserung der Qualität der Kontexterkennung erforscht, entwickelt und evaluiert. Zu diesem Zweck werden geeignete Modelle vorgeschlagen, um die Qualität zunächst mit der Datenquelle selbst auszudrücken. Dann wurden Methoden zur Auswahl und Anpassung qualitätsoptimaler Datenquellen entwickelt, um mit der Dynamik von Sensorumgebungen umzugehen. Die vorgeschlagenen

Modelle und Methoden wurden anhand einer Fallstudie im Bereich des autonomen Fahrens evaluiert.

Abstract

The Internet of Things (IoT) is a basis of the Internet of the future and will cover billions of intelligent objects being able to sense, actuate, and communicate with each other. Recent advances in the Internet of Things (IoT) provides rich opportunities for the development of more flexible context-aware solutions. Applications can benefit from those flexible mechanisms by adapting to change in the sensing environment such as sensor disappearance/degradation or service unavailability. This vision helps to effectively use IoT widely for more flexible context-aware applications and services.

Understanding sensor data plays an important role in context-aware computing. However, collecting data from all available data sources over IoT is not feasible to improve the quality of context-aware applications and services due to scalability problems. The main issue is to find the right data sources providing the best data for specific context recognition tasks. Selecting appropriate sensors to address the specific context recognition task is an important challenge as a huge number of sensors is available in IoT and their quality differs from each other. Related processing method to get information from a sensor data also affects the overall context recognition quality.

In this doctoral study, models and methods for improving the quality of context recognition have been researched, developed and evaluated. For this purpose, adequate models are proposed to express the quality with the data source itself first. Then, methods for selecting and adapting quality-optimal data sources have been developed to deal with the dynamicity of sensing environments. Proposed models and methods have been evaluated through a case study in the scope of autonomous driving.

Dedicated to my parents Hatice and Mustafa Eryilmaz

who made everything possible for me...

Acknowledgements

It is very difficult to put everybody contributed directly or indirectly during my journey in the scope of this thesis. I would like to start with my advisor Prof. Sahin Albayrak. His constant support enabled me to conduct this research in many inspiring and challenging projects at the DAI-Labor of the Technische Universität Berlin. I would also like to thank the other members of my committee for their advice and critique to improve my work.

In my time at the DAI-Labor, I worked in many different research projects which some of them contributed a lot to this research work. I would like to thank all members of Diginet-PS and NeMO projects and especially Manzoor-Ahmed Khan, Nils Masuch, Jan Keiser, Axel Heßler, Martin Berger, Christian Rakow and Timo Ruck. It was a great pleasure to work with you during the last years.

My colleagues particularly in the competence center NGS at DAI-Labor have supported my work during this journey with valuable suggestions and discussions. I would like to thank Frank Trollmann first for his valuable feedback and support anytime I needed it. I would like to thank all of my NGS colleagues, Andreas Rieger, Johannes Faehndrich, Sebastian Ahrndt, Stephen Prochnow, Izgh Hadachi, Nizar Ben-Sassi, Marcus Voß, Mahmoud Draz, Daniel Freund, Mathias Wilhelm, Paul Zernicke. It has been a great pleasure to have worked with you over the years. I want to tank Fikret Sivrikaya and Nuri Kayaoglu as well for their help and support during my time at DAI-Labor.

I want to thank all of my friends for their constant support and patience during this work, especially to my lovely friends Arda Yildirim, Can Goeruer, Nese Cakmak-Goeruer, and Meric Benderlioglu. Without their continued support, I would not be able to finish this thesis.

And a big thank you goes to my parents Hatice and Mustafa Eryilmaz, my sister Asli Yaylali, my brother-in-law Kerem Yaylali, and my cutest nephew Cinar Yaylali. You are always there for me with full of love and support whenever I need it. I also want to thank my parents-in-law Marianne and Burkhard Sigwarth as well as my brother-in-law Valentin

Sigwarth which I have had the chance to know them for the last couple of years during my journey. Thank you for always welcoming me with family warmth.

Last but certainly not least, I would like to thank my lovely husband Vincent Sigwarth. His constant support enabled me to keep going every time. Thank you for always being there for me and taking care of everything with endless love and energy.

It is almost impossible not to forget to acknowledge someone when finishing a project of this size and duration. I would like to thank and apologize to everyone I forgot to mention.

Table of Contents

List of Figures

List of Tables

List of Publications

This section presents the published work related to dissertation as a main and co-author:

Elif Eryilmaz, Manzoor Ahmed Khan, Frank Trollmann, Sahin Albayrak; *Adaptive Service Selection for Enabling the Mobility of Autonomous Vehicles*; In European Conference on Ambient Intelligence, 203-218, 2019.

Elif Eryilmaz, Frank Trollmann, Sahin Albayrak; *Quality-aware service selection approach for adaptive context recognition in IoT*; In Proceedings of the 9th International Conference on the Internet of Things, 2019. **(Honourable Mention Award)**

Elif Eryilmaz, Frank Trollmann, Sahin Albayrak; *An Architecture for Dynamic Context Recognition in an Autonomous Driving Testing Environment*, In: The 11th IEEE International Conference on Service-Oriented Computing and Applications; 2018. **(Best Paper Award)**

Elif Eryilmaz, Frank Trollmann, Sebastian Ahrndt, and Sahin Albayrak; *Challenges for Adaptable Quality of Context Recognition in Opportunistic Sensing*; In: VDE Kongress 2016; 2016.

Elif Eryilmaz, and Sahin Albayrak; *Quality of Context Optimization in Opportunistic Sensing for the Automatization of Sensor Selection over the Internet of Things*; In: ECAI 2016, Workshop Modelling and Reasoning in Context (MRC), August 2016; 2016.

Elif Eryilmaz, Frank Trollmann, and Sahin Albayrak; *Conceptual Application of the MAPE-K Feedback Loop to Opportunistic Sensing*; In: 10th Workshop: Sensor Data Fusion:

Trends, Solutions, Applications (SDF 2015); 2015.

Nils Masuch, **Elif Eryilmaz**, Tobias Küster, et al.; *Decentralized Service Platform for Interoperable Electro-mobility Services throughout Europe*, In: Towards User-Centric Transport in Europe 2: Enablers of Inclusive, Seamless and Sustainable Mobility. Springer International Publishing, 2020, pp. 184–199.

Andrey Boytsov, Arkady Zaslavsky, **Elif Eryilmaz**, and Sahin Albayrak; *Situation Awareness Meets Ontologies: A Context Spaces Case Study*; In: The Ninth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'15); 2015.

Apart from the publications presented above, this research work was awarded with **3rd Prize** from **"Forum for Excellent Young Scientists: Internet of Things"** by Humboldt Innovation GmbH [1] in 2015.

Other publications of the author outside of this research work can be found in Google Scholar [2].

[1] https://www.humboldt-innovation.de/en/events/m_eventID-206.html
[2] https://scholar.google.de/citations?user=KfZ4OVYAAAAJ

List of Supervised Theses

This section presents the supervised bachelor theses and seminar work related to this dissertation:

Kristina Radeva, *Optimization Methods for QoS-aware Service Composition*; Technische Universitaet Berlin, Seminar, 2018.

Sonia Grozdanova, *Derivation of Quality of Sensors from Deployed Ones*; Technische Universitaet Berlin, Bachelor, 2017.

Stefan Gierke, *Retrieval of sensor data for context information*; Technische Universitaet Berlin, Bachelor, 2017.

Sonia Grozdanova, *A Survey on Quality of Context Awareness for the Internet of Things*; Technische Universitaet Berlin, Seminar, 2015.

1

Introduction

Ubiquitous computing is a concept in Computer Science for many years representing the idea of making computing devices smaller and unobtrusive in order to integrate them in everyday life [1]. With the rise of the Internet of Things (IoT), everyday life has become more connected as IoT enables vast number of intelligent objects being able to sense, act and communicate with each other [2].

Context-awareness plays an important role in ubiquitous computing. The system or application is context-aware if it uses context to provide relevant information to the user depending on the user's task [3]. Dey and Abowd defined context as any information that can be used to describe an entity [3]. Context information can be used by the software application to increase its effectiveness. Although context-awareness is mostly recognised with ubiquitous computing, it also becomes associated with the IoT as the IoT provides transparent access to many sensors, processors and actuators using standardized protocols via its underlying infrastructure [4].

As more and more applications need to be aware of their environment it becomes less and less feasible to install specific sensors for each application. Instead, a more flexible context-management that is able to make use of existing sensors is required. Classical context recognition approaches often assume the deployment of specific sensors for a context recognition goal [5]. The sensors are selected at design time. Enabling the detection of a new recognition goal is impractical as it either (a) requires the deployment of new sensors, which can be expensive, or (b) the adaptation of the detection mechanism to the existing sensors, which

requires expert knowledge. Therefore, research in context-aware computing is moving towards dynamic mechanisms for utilizing available sensors [6].

Opportunistic sensing follows the idea of utilizing available data sources [2] by using signal and information processing techniques to enable the involved sensing infrastructures to automatically discover and select data sources [7]. Throughout this work, the following terminology from opportunistic sensing by Roggen et al.[5] to abstract dynamic context recognition approaches will be used. In this terminology, a **context recognition** task is defined as calculating a recognition chain to fulfil a recognition goal. A **recognition goal** specifies the type of information to be derived via context recognition as well as constraints on that information. A **recognition chain** is calculated by a dynamic context recognition method to fulfil the recognition goal. It consists of sensors providing information as well as processing methods to derive high-level information from the low-level sensor values.

Although IoT can exploit opportunistic sensing approaches, not all sensors are relevant to the detection of a specific situation and some sensors can provide better quality for context recognition. Thus, finding sensors that provide the correct data in the desired quality for a specific context recognition task is important. Quality of Context recognition (QoC) plays a critical role in this respect to improve the desired behaviour of context-aware applications and services. As proposed by Kim and Lee [8], context information can be defined by many quality aspects, such as accuracy, precision, completeness, access security, and up-to-dateness. The importance of these factors is determined by the requirements of the recognition goal which could be affected by the current situation. For instance, when the battery of a sensor is low, a downgrade in accuracy in favour of energy consumption might be acceptable. Additionally, the quality requirements of context recognition tasks can change during run time, e.g., when the application decides to downgrade accuracy in order to maintain functionality. When a large number of the sensors become available over the IoT, selecting suitable sensors fulfilling the changing quality requirements of the recognition goal is a challenging issue. Sensor selection should be flexible enough to adapt to such circumstances on a big scale over the IoT.

The aim of this study is to optimize the quality of context recognition by selecting recognition chains (a combination of sensors and processing functions) autonomously to provide the best quality data to the recognition goal amongst available sensors over the IoT. The evaluation criterion is finding the optimal recognition chain and updating autonomously in case of any change. In the following sections the details about the topic, goals and contributions of this

thesis are provided (Section 1.1) and described how they are related to the structure (Section 1.2).

1.1 Goals and Contributions

The research challenges addressed in this thesis mainly lie in two main domains; context-awareness and self-adaptability. The main motivation of this doctoral study is to investigate the application of dynamic context management by enabling adaptive resource selection to improve the quality of context recognition in smart environments. From this respect, specific contributions are provided in this section.

The main goal of this work is to address the lack of end-to-end solution for calculating adaptively the best fitting recognition chain to the recognition goal by means of quality. The recognition chain should fulfil the recognition goal under certain quality requirements. The recognition chain consists of sensors and processing functions to derive the required information for the recognition goal. In this respect, main contributions of this thesis are summarized below.

Contribution 1: Selecting and adapting the optimal recognition chain requires an adequate model of the recognition goal and the elements in the recognition chain. Expressive modelling of the recognition goal to be able to describe quality requirements for different applications is the first contribution of this work.

Contribution 2: The recognition chain is assembled from sensors and processing functions. Modelling sensors and processing functions with respect to their impact on the recognition goal by means of quality is the next contribution to enable an automated assembly and optimization of the recognition chain.

Contribution 3: Developing an algorithm to determine the optimal recognition chain given these models mentioned in Contribution 1 and 2 is the third contribution of this work. This requires optimization of search space by quality requirements modelled.

Contribution 4: Enabling adaptation of the recognition chain in case of changes is the fourth contribution of this work. This includes the implementation of feedback loops from control theory to monitor the running system, and analyse, plan and execute the adaptation for the recognition chains.

1.2 Structure

This section presents the structure of the thesis. In Chapter 2, problem statement with related research questions are provided. Chapter 3 presents the fundamentals that are required to understand the content of this thesis in Section 3.1 and state-of-the-art addressing the research questions in Section 3.2. Chapter 4 provides the first part of the approach with proposed models for the quality optimized recognition chain selection. In Chapter 5,the second part of the approach is presented which enables adaptive recognition chain selection. In Section 6, a case study is defined for the implementation and evaluation of the approach in the scope of autonomous driving. The implementation details of the approach in a case study are given Chapter 7. After the implementation, the results of the evaluation are provided in Chapter 8. In Chapter 9, additional application areas are presented for further application of the proposed models and methods. Chapter 10 concludes the thesis with the summary of contributions and potential future directions.

2

Problem Statement

Classical context-recognition approaches are domain-specific as they mostly propose the deployment of specific sensors for the required piece of information. For instance, if we want to develop a smart home solution for the context-aware home automation, every time we need to deploy the fixed set of sensors and bound to achieve only one task. This is impractical and a time-consuming task to deploy new sensors in the smart home for every new context situation to detect and it makes the system costly due to configuration effort for further development. Therefore, to create more flexible solutions that could be applied to more systems, it is required to use dynamic context management methods applying self-adaptive techniques to use the available data sources in order to provide the required piece of information. These methods are required to be able to react to the changes in the environment (e.g. sensor appearance/disappearance and change in sensors characteristics) to continue providing the required information for context-aware applications. As IoT envisions a vast amount of sensors being able to connect over the Internet, it provides rich opportunities for context-recognition systems to provide more flexibility which data source to use for a specific recognition goal.

For this purpose, Section 2.1 outlines a motivational scenario in which flexible methods can be a great benefit. Then, Section 2.2 presents the related challenges to enable that. Based on those challenges, the research statement is provided in Section 2.3. The parts of this Chapter are published in the author's publications [9] and [10].

2.1 Motivational Scenario

In this section, a motivational scenario is presented where adaptive optimal recognition chain selection is necessary to increase the quality of context recognition application. As a developer, Tim wants to develop a context-aware application that needs to know the position of the user. There can be various sensors to infer this information in the environment where the user is. For instance, if the user is at home, the cameras deployed in the house can be used to infer this information. When the user is outside, the application can use a smartphone or smartwatch to determine the position information of the user. The techniques and algorithms to infer position differ for each sensor. For instance, getting information by using cameras at home requires image processing techniques whereas using GPS on the smartphone of the user requires sensor data fusion. The quality of the recognition is also not the same due to the different quality of sensors and algorithms. The scenario is depicted in Figure 2.1.

The aim of quality of context optimization is to continue providing the same information to the application (user position) in case of any change in the user environment affecting the data sources. If there are multiple cameras that exist in the house of the user having the same coverage, the most accurate or reliable one should be selected depending on the application preference. Assuming Camera 1 is the most accurate sensor depicted in Figure 2.1, the application should use this data source to infer the required information.

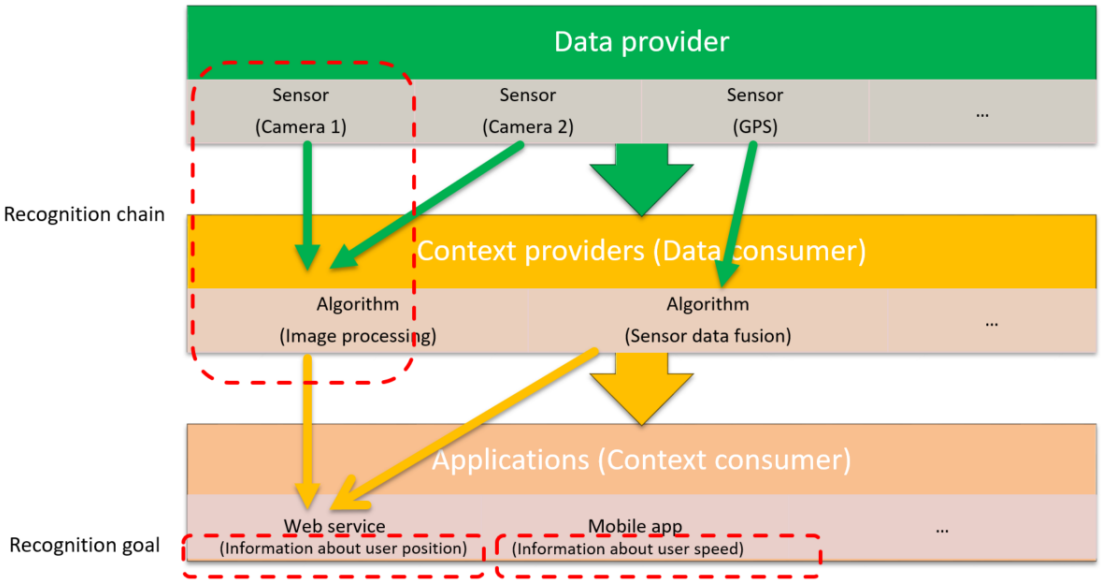


Figure 2.1: Adaptive selection for user position

If the selected sensor degrades or becomes unavailable (e.g. user goes outside), the application should use other data sources (e.g. GPS) to continue getting information about the position of the user. In this scenario, the recognition goal is to get information about the user position. One sample recognition chain is the one using Camera 1 as a data source and applying an image processing algorithm to infer the data.

For selecting the optimal data source for an application, there are different challenges that need to be addressed as presented in the following Section 2.2.

2.2 Challenges

In order to develop adaptive optimal recognition chain selection methods, there are a number of challenges that need to be addressed. The first challenge is the generic representation of data sources together with their processing methods, so-called recognition chains, in a way that they can be substituted with each other to achieve a specific recognition goal. When there is a common level of representation, the second challenge is to specify and model the recognition goals with quality properties. At this point, there are many different aspects regarding the quality (e.g. cost, accuracy, latency, stability) for choosing between different recognition chains to work for the same recognition goal. After modelling the recognition goal, selecting the best fitting recognition chain based on the quality requirements of the recognition goal is the third challenge which requires optimization based on different quality criteria. The fourth and last challenge is updating the selected recognition chain which fulfils the quality requirements when any change occurs in the sensing infrastructure requiring the implementation of self-adaptive methods. Overview of those challenges with respect to the goals of this work presented in Section 1.1 can be seen in Figure 2.2.

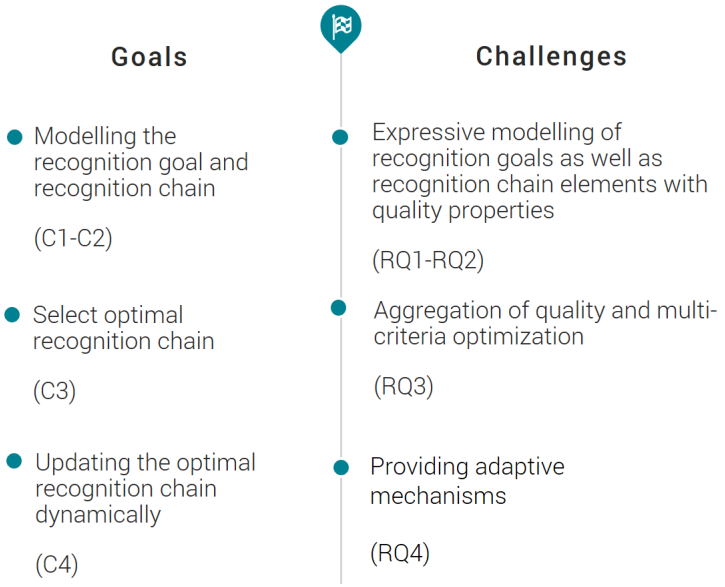


Figure 2.2: Goals and challenges in this research

Based on those identified challenges, the details about the related research questions can be found in Section 2.3.

2.3 Research statement

In this research, the aim is to evaluate the improvement on the quality of context recognition by an autonomous sensor selection that selects a recognition chain by taking into account the quality requirements of the recognition goal and the quality properties of the elements building up the recognition chain. The recognition chain consists of data sources and the processing methods to retrieve high-level information from low-level data sources.

Quality of Context recognition (QoC) plays a critical role to improve the desired behaviour of context-aware applications and services. As proposed in [8], context information can be defined by many QoC aspects, such as accuracy, precision, completeness, access security, and up-to-dateness [8]. However, in context-aware systems, not all context recognition goals require the same QoC. Quality requirements of recognition goals are varied and dependent on the specification of the recognition goals that can have different acceptable values to fulfil the context recognition task. Thus, the first research question of this work is the following:

(RQ-1) *How to define recognition goals and related quality requirements that can be expressive enough to provide context-aware applications and services?*

To select and update the recognition chains based on the recognition goal requirements, the characteristics of each data source as well as processing methods need to be known. In this study, sensors are used as the main data sources in the sensing infrastructure. The quality characteristics include but not limited to the type of data the sensor provided, how frequent the sensor provides this data in which accuracy and reliability level, what the cost of using this sensor is and what the constraints of the sensor to provide data (battery level, environmental constraints). Thus, the second research question is the following:

(RQ-2) *How to define recognition chain elements and related quality properties in a way that they can be substituted to each other?*

In order to fulfil the QoC requirements for a specific recognition goal to improve the overall quality of context recognition by selecting the best recognition chain, there needs to have a selection mechanism based on both quality requirements of recognition goal and quality properties of recognition chains. The most important point here to be addressed is how to decide the percentage of fulfilment required by the recognition goal and what the acceptable variation from this fulfilment is [11]. Therefore, the techniques addressing this problem should provide the results based on these priorities for the fulfilment by using comparison techniques in combination with the elements in the recognition chain and recognition goals dynamically. Thus, the third research question is the following:

(RQ-3) *How to decide that the selection mechanism is fulfilling the recognition goal best for the different quality requirements?*

In autonomous recognition chain selection, when any change occurs in the sensing infrastructure (sensor appearance/disappearance/change in sensor characteristics), the information needs to be detected in a different way to fulfil the requirements of the recognition goal dynamically. In other words, the mechanisms should adapt themselves to react to the sensor infrastructure changes to achieve the recognition goals. This requires autonomous control in a way that the mechanisms should constantly check and optimize its status and automatically adapt themselves [12]. By enabling so, the selected recognition chain is always up-to-date fulfilling the recognition goal. Thus, the fourth research question is as follows:

(RQ-4) *How to provide self-adaptability for context-awareness to react changes in the sensing infrastructure at runtime?*

This work focuses on combining different approaches from sensor and service perspective to create an end-to-end adaptive optimal recognition chain selection framework. For this purpose, we classify the focus of this work under three main groups as follows.

- **Modelling:** This part includes the modelling of elements used in adaptive context recognition task as well as modelling context and its quality.
- **Selection & Optimization:** This part includes the construction of recognition chains among available elements and optimal selection mechanism to select the best fitting one for the context recognition task.
- **Adaptation:** This part focuses on the adaptation of the selection mechanism in case of any changes in the involved elements for the context recognition task.

In Chapter 3, the foundations (Section 3.1) and state-of-the-art work (Section 3.2) based on those three groups are presented from the service and sensor perspectives for modelling and selection & optimization parts of this work. For the adaptation part, reference architectures in self-adaptive software systems and application of them to opportunistic sensing have been in focus.

3

Foundations

3.1 Basic Terms and Concepts

This section describes the main terms and concepts used in this work from various domains. First, as the main problem domain is the Internet of Things and opportunistic sensing, its related concepts are defined in Section 3.1.1 and Section 3.1.2 respectively. Then, in Section 3.1.3, the basic terms for context and situation awareness in relation to IoT are provided. For the proposed models, Section 3.1.4 presents the terms and concepts from the Semantic Web domain. To address the wide-scale of IoT, service and agent-oriented terms are described in Section 3.1.5 and 3.1.6 respectively. Finally, Section 3.1.7 describes the terms used to enable autonomous control in software systems from self-adaptive software systems.

3.1.1 Internet of Things

The IoT envisions creating a smart environment where intelligent objects around us being able to sense, actuate and communicate with each other over the Internet. IoT as a term first mentioned by Ashton and it has been seen as a future of the Internet to make a broader impact in everyday life [13]. Vermesan et al. defines IoT as an enabler to connect people and things at Anytime, Anyplace with Anything and Anyone, using Any path/network and Any service [14].

IoT presents a broad vision to enable its idea. For this purpose, it covers various networks, sensing and information processing approaches to enable the interconnection of any computing

device. IoT provides transparent access to many sensors, processors and actuators using standardized protocols via its underlying infrastructure without considering hardware, operating systems, or locations [15]. Sensor networks are the main building blocks of this connectivity enabling this transparent access.

As a predecessor of IoT, pervasive computing (also called ubiquitous computing) enables computing devices getting so small and ever-present that they eventually surround us everywhere. IoT took this vision to extend the scope by providing machine-to-machine interaction without the need of interacting with users differently from pervasive computing.

Although there is minimal interaction with the user, IoT can still ease the everyday life for humans by providing meaningful information about the current situation in which they are in. People can get benefit from their everyday tasks as well as they can be prevented from dangerous situations. IoT is a part of this mission by inferring the context through sensing and processing capabilities in various different domains such as healthcare, home automation, transportation, automotive, agriculture and environment monitoring, to name a few. The vast amount of sensors that the IoT envisions can bring more flexibility while developing smart applications and services.

3.1.2 Opportunistic Sensing

As one of the main ideas in IoT is sensing. The devices having sensors attached enable to perceive the environment through sensing. The number of sensors deployed around the world is increasing to enable sensing capabilities in different domains. Millions of sensors are being embedded in the physical world that sense, create and communicate data already [16]. It is estimated that by 2020, there will be 50 to 100 billion devices connected to the Internet [17].

The sensors in IoT differ in type and quality. Additionally, as they are physical entities, they are subject to change during time resulting in erroneous results for the applications. Therefore, a way to select suitable sensors dynamically is crucial to react to the changes in the sensing environment. Selecting the appropriate sensors which provide high-quality data plays a critical role in developing smart systems.

One of the approaches addressing this requirement is opportunistic sensing. Opportunistic sensing is a retrieval of information related to human activity and the environment around them by using small computational devices attached to individuals [18]. Different than classical sensor-network approaches requiring to fuse all retrieved data from stationary data sources,

opportunistic sensing is based on signal and information processing techniques that enable the involved sensing infrastructures to automatically discover and select data sources [7]. Kurz et al. used opportunistic sensing for context recognition task as *"the sensors available at any point in time should be best exploited for a recognition task"* and argue to avoid specific sensor deployment for any application which enables the use of sensors discovered opportunistically [2].

Roggen et al. propose an opportunistic activity recognition system called Opportunity Framework [19] as a reference implementation of an opportunistic sensing system for human activity and context recognition [20]. The architecture of this framework is given in Figure 3.1 based on the illustrations in [5].

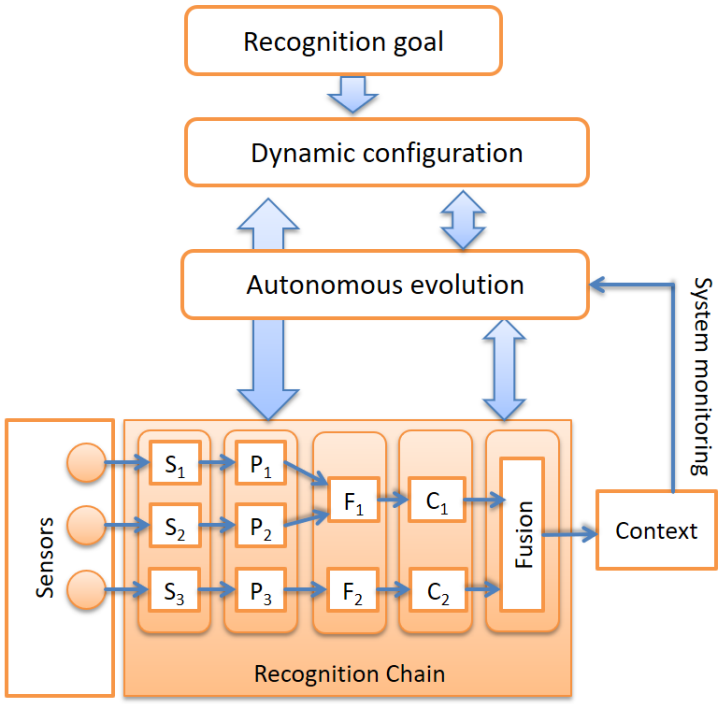


Figure 3.1: Opportunistic activity recognition system architecture in Opportunity Framework (based on [5])

For a given recognition goal the Opportunity Framework configures the recognition chain dynamically based on the available sensors and domain knowledge. The recognition chain is responsible for mapping sensor signals (S) to activity or context classes by applying pre-processing (P), feature extraction (F), classification (C) and decision fusion methods. These data-processing methods and the domain knowledge are also evolving based on the continuous analysis of relations between sensors and activities.

3.1.3 Context

Although a vast number of sensors available in IoT, sensing capability alone does not create additional value for applications and services. Raw sensor data should be processed to make inferences in which the device/system or the user is in. In other words, it is required to have high-level information from low-level data retrieved from sensing devices.

The efficiency of applications in IoT highly depends on how well they represent and reason about the situations happening in the environments both for systems as well as humans. Therefore, context is an essential element in this respect.

According to the well-acknowledged definition, Abowd et al. define context as follows:

> *"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."* [21]

Similarly, Abowd and Dey define that a system is context-aware "if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." [21, 3]

Bazire and Brézillon analyzed 150 definitions of context from different areas and they stated that there is no single definition of context covering all aspects [22, 23]. They conclude their analysis by stating that *"context acts like a set of constraints that influence the behaviour of a system (a user or a computer) embedded in a given task"* [22].

Villegas et al. argue that classical definition of context by Abowd et al. [21] does not completely reflect the operational view of context when it is subject to change in execution as well as it does not cover the other interactions except between users and their tasks [24]. Therefore, the authors proposed the following definition:

> *"Context is any information useful to characterize the state of individual entities and the relationships among them. An entity is any subject which can affect the behavior of the system and/or its interaction with the user. This context information must be modeled in such a way that it can be pre-processed after its acquisition from the environment, classified according to the corresponding domain, handled to be provisioned based on the system's requirements, and maintained to support its dynamic evolution."* [24]

In this study, the definition from Villegas et al. [24] is used as it reflects the dynamic nature of context due to uncertain changes in the environment.

Anagnostopoulos et al. define the situations as *"logically aggregated pieces of context"* in a generic way [25]. Supporting this idea, Ye et al. define a situation as *"external semantic interpretation of sensor data"* where semantic interpretation means *situation assigns meaning to sensor data"* and external depicts *"from the perspective of applications, rather than from sensors"* [26]. Therefore, situation-awareness can be seen as using context to have a higher level of information abstraction. This abstraction proposed by Ye et al. can be seen in Figure 3.2.

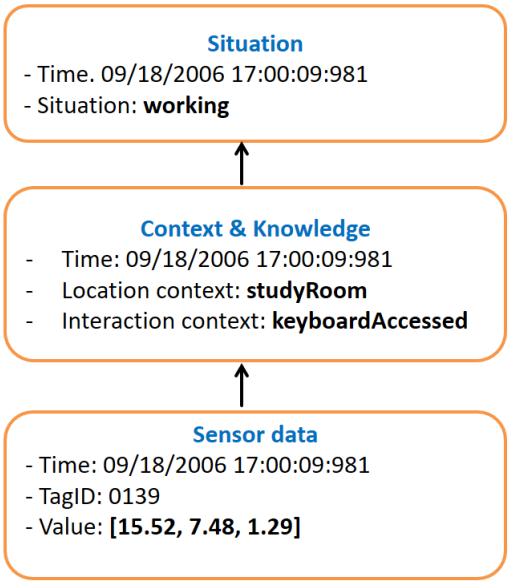


Figure 3.2: Abstraction levels in pervasive computing (based on [26])

As can be seen in Figure 3.2, the sensors get data from the environment and combining with context information (location and interaction), the situation as a high-level abstraction of information can be derived to be used by applications.

Similar to the data flow in software systems, context needs to be managed from where it is produced until where it is used to develop context-aware software. Perera et al. examine different management cycles for data and context [6] and proposed a minimum number of phases for context life-cycle as depicted in Figure 3.3.

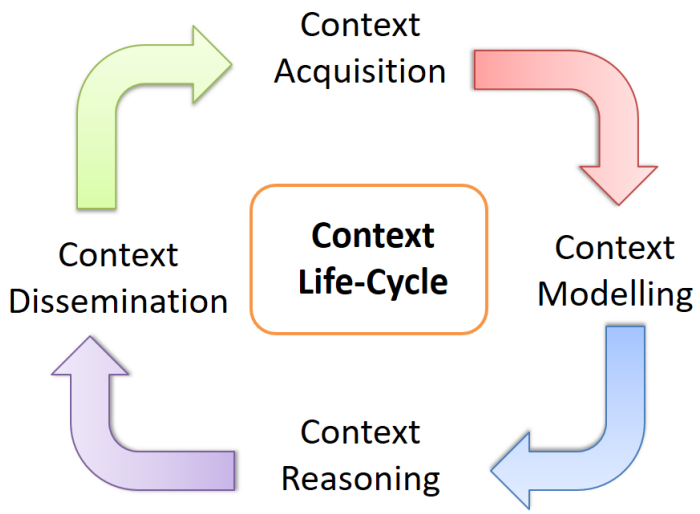


Figure 3.3: Context life-cycle (based on [6])

According to [6], context life-cycle starts with *Context Acquisition* phase where the low-level context data must be acquired from any sources; then, in *Context Modelling* phase, the low-level context has to be structured for further processing; as the next step in *Context Reasoning*, the high-level context must be inferred on the basis of the context model; finally, in *Context Dissemination* phase, high-level context needs to be shared with other applications. In different phases of context life-cycle, there are different methods and approaches proposed by authors.

As the first phase of context life-cycle, the context should be acquired from data sources. In this process, sensors play an important role to acquire the characteristics of the environment. Based on the acquisition process, the context can be retrieved by using three different ways: (1) sense, (2) derive, (3) manually provided [6]. By using (1), the data is sensed through sensors, whereas in (2), additional computation is necessary to retrieve data for context. In (3), the user can provide data about context but it is cumbersome to develop context-aware applications, therefore should be minimized.

With the emerging IoT paradigm, although the increased number of sensors available via the Internet brings many challenges (e.g. scalability, heterogeneity), it also brings opportunities to use those data sources in context acquisition for more flexible context management solutions while developing context-aware applications as presented in Section 3.1.1.

In order to get benefit from a vast amount of sensors to provide flexible context-aware applications, it is important to facilitate interoperability across heterogeneous data sources. Semantic Web (SW) technologies address the similar problem for the integration and management of information from heterogeneous sources, therefore could be also applied

for context-awareness. In the following Section 3.1.4, brief information about SW technologies is provided.

3.1.4 Semantic Web

Semantic Web (SW) is an extension of the World Wide Web (WWW) aiming at unifying the data to be shared and reused through standards [27]. The ultimate goal of the semantic web is making machines understand data available on the Internet without any interference.

To enable this vision, World Wide Web Consortium (W3C)[1] provides standards to enable common data formats and exchange protocols. RDF (Resource Description Framework)[28] and OWL (Web Ontology Language)[29] are the most commonly used technologies to describe the semantics of data on the web. As a fundamental standard, RDF is used to describe linked data on the web with limited expressiveness. OWL extends RDF based on formal logic to provide more expressiveness in order to enable description of ontologies.

Linked Data (LD) is a foundational vision of SW to enable data sharing on the web by enabling links between different data sources. LD uses the following rules and fundamental web technologies to do so [30]:

- Use Uniform Resource Identifiers (URIs) as names for things
- Use HyperText Transfer Protocol (HTTP) URIs so that people can look up those names
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
- Include links to other URIs so that they can discover more things.

LD enables links between data sources by using RDF format. In RDF, data is described as statements by using subject, predicate, object triples [28]. Figure 3.4 shows a sample RDF statement.

[1] World Wide Web Consortium website `https://www.w3.org/`

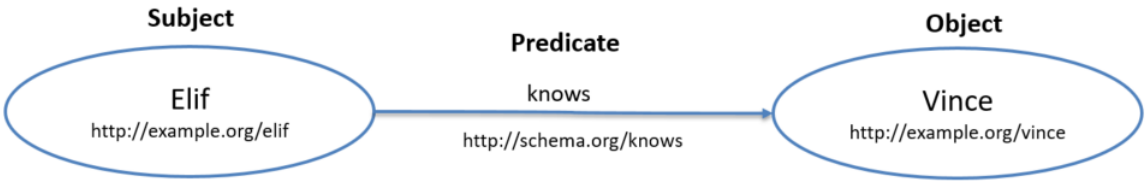


Figure 3.4: RDF statement example

As can be seen in Figure 3.4, the object describes the value of the predicate for the related subject. Every element in a triple needs to be encoded with a URI. URIs can be constructed using vocabularies in the form of RDF Schema or ontologies.

RDF Schema (RDFS)[28] describes RDF semantically by using classes and properties. The main classes and properties from RDFS that are used in this work are presented in Figure 3.5.

Class name	Comment
rdfs:Resource	The class resource, everything.
rdfs:Literal	The class of literal values, e.g. textual strings and integers.
rdfs:Class	The class of classes.
rdfs:Datatype	The class of RDF datatypes.

Property name	Comment	domain	range
rdf:type	The subject is an instance of a class.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class

Figure 3.5: Subset of RDFS classes and properties

Queries over linked data are made using a semantic query language called SPARQL[31] that allows to retrieve and manipulate data stored in RDF format.

RDFS has limited capability regarding expressiveness to represent knowledge. In order to model more complex knowledge, Web Ontology Language (OWL) extends RDF to enable a more complex relationship between classes and properties. In the next section, we will present details about ontologies and OWL in detail.

Ontologies are applied in many different areas to represent knowledge. Gruber defined ontology as *explicit specifications of a conceptualization* which enables knowledge representation, reasoning and sharing independent of a domain [32]. The ontology represents a rich model for knowledge representation with its expressibility and computability.

Ontologies are domain-independent way of storing semantic information [32]. Ye et al. classify ontologies in three main categories as *generic (upper) ontologies*, describing general concepts independent of a domain; *domain ontologies* specifying concepts of a certain domain; *application ontologies* representing specific knowledge about an application [33].

Common components of ontology includes the concepts of individuals, classes, attributes and relations having the following definitions [34]:

- **Individuals:** instances or objects
- **Classes:** concepts, types of objects
- **Attributes:** properties that individuals and classes can have
- **Relations:** description of how classes and individuals are related to each other

In SW, to model the ontologies, Web Ontology Language (OWL) is used to represent complex knowledge. OWL[29] is an ontology representation language which provides expressiveness to define classes, properties, individuals, and data values in ontologies.

In OWL, classes are represented as instances of owl:Class, a subclass of rdfs:Class. Individuals are instances of classes. Properties are defined as object properties and data properties. Object properties are used to relate individuals with individuals, whereas data properties link individuals with data values.

Semantic Web is a one-step forward for dealing with heterogeneity in IoT by providing common vocabularies and semantics with web technologies. Apart from the definition and using the same semantics for IoT data sources, another challenge to tackle is providing computing between those data sources in a reliable and robust way. In Section 3.1.5, the concepts from Service-Oriented Computing aiming to tackle down this problem on a wide scale in IoT are provided.

3.1.5 Service

Service-oriented Computing (SOC) is a computing paradigm for the development of distributed applications in a wide scale. SOC plays an important role in IoT to provide reliable, robust, and

user-centric applications. With the evolving technologies, complex applications and systems need to be developed that deal with different requirements regarding performance and resource limitations.

SOC requires the design of Service-Oriented Architectures (SOAs) to provide solutions for managing the increasing complexity in largely distributed, heterogeneous and dynamic resource environments [35]. SOA is an architectural style of building software by using loosely-coupled components in order to increase the reuse of those components in distributed environments.

SoA principles are commonly applied in IoT due to its unassociated, loosely coupled and self-contained units of functionalities [36]. Main design principles of SOA are [37]:

- Service Reusability by other service consumers
- Service Loose Coupling by working independently
- Service Abstraction by hiding the work logic
- Service Composability by using other services
- Service Statelessness by not maintaining state information
- Service Discoverability by providing enough meta-information to the service consumers to be found
- Service Autonomy by having less interference

SOA uses the publish-find-bind-execute paradigm as shown in Figure 3.6.

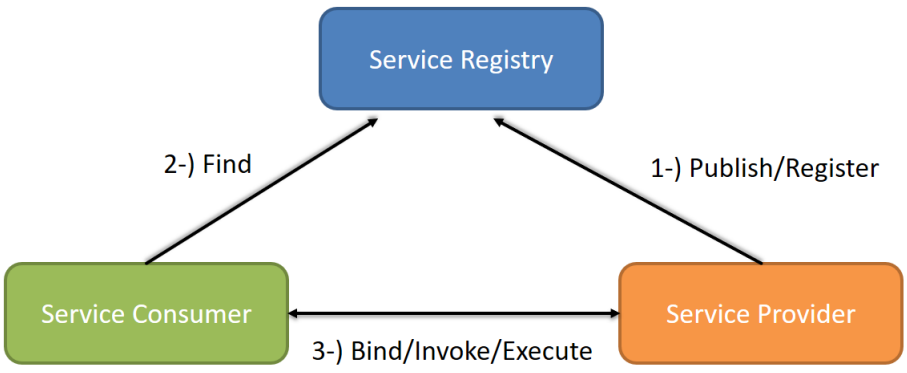


Figure 3.6: Publish-Find-Bind-Execute Paradigm in SOA (adapted from [38])

Based on the Publish-Find-Bind-Execute paradigm, service providers develop services and publish them in a distributed service registry. Service registry stores the service descriptions to be used by service consumers. Service consumers search from the service registry fulfilling their request and when found, the binding takes place to tell service consumers how to call the related service. After that, the service consumer can execute the service.

Web services realize SOA through specific standards to provide services over the web. The World Wide Web Consortium (W3C) defines web services as follows [39]:

> *Web Services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web Services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.* [39]

Web Services use standards to enable those functionalities such as Web Services Description Language (WSDL) [2] for defining service operations and execution properties, Simple Object Access Protocol (SOAP) [3] for exchange of information, Universal Description, Discovery, and Integration (UDDI) [4] as a distributed service registry for the service descriptions.

Besides W3C standards for web services, one of the commonly used software architecture styles to create web services is REpresentational State Transfer (REST) [40]. REST uses HTTP to manipulate data over the web by using predefined stateless operations such as POST to create/update data, GET to read data and DELETE to remove data [40]. REST is widely accepted to create web services as it is lightweight, simple and faster to manipulate data from web resources.

Cardoso described the life-cycle of a web service with five different phases as depicted in Figure 3.7 [41].

[2]Web Services Description Language (WSDL)`https://www.w3.org/TR/wsdl.html`

[3]Simple Object Access Protocol (SOAP) `https://www.w3.org/TR/2000/NOTE-SOAP-20000508/`

[4]Universal Description, Discovery, and Integration (UDDI) `https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`

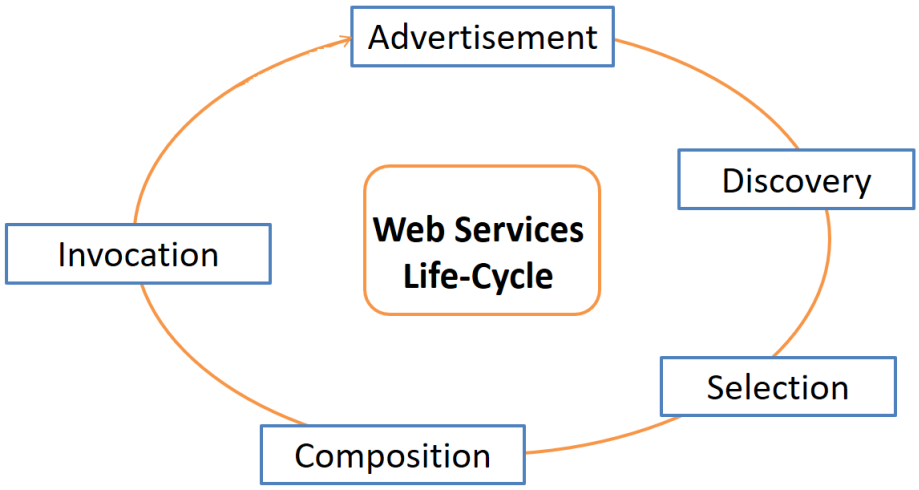


Figure 3.7: Web Services Life-Cycle (adapted from [41])

The web service life-cycle starts with *Advertisement* phase where service is described and published in the distributed service registries; *Discovery* phase finds the list of available services potentially fulfilling the application request (one or more services); *Selection* phase select the most optimal service among available ones based on some non-functional criteria requested by the application; *Composition* phase, combine the found services to fulfil the application request if there is no single service found fulfilling the request; *Invocation* phase executes the found service or services involved in a composition as a process [41].

As mentioned in Section 3.1.4, Semantic Web is an extension of the World Wide Web to semantically describe web resources[27]. Semantic Web Services uses ontologies as an underlying data model from Semantic Web aspect for the automation of phases in the web service life-cycle. The main aim of semantic web services is to overcome the limitations of classical web services by integrating semantic technologies. Figure 3.8 by Dominque and Martin shows the overall relationship between the web and semantic web services.

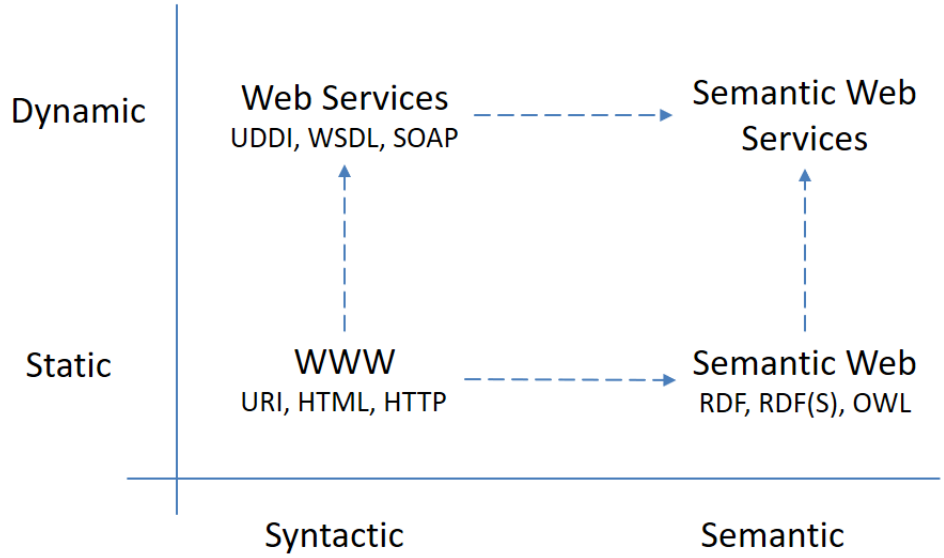


Figure 3.8: Web and Semantic Web Services (adapted from [42])

There have been a vast amount of approaches proposed to describe semantic web services. Web Ontology Language for Services (OWL-S)[43] is a widely used semantic description language for web services. Figure 3.9 shows the top-level components of OWL-S where service is defined.

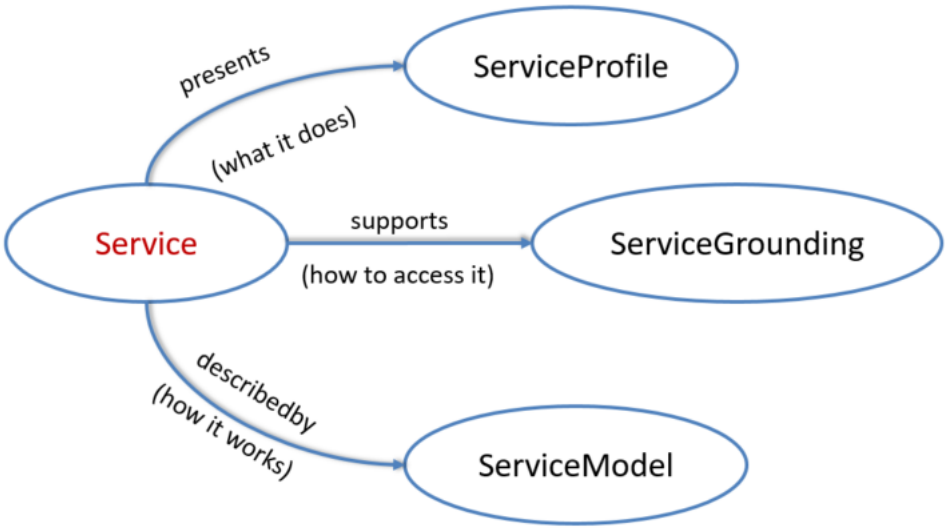


Figure 3.9: Top level of service ontology- OWL-S [43]

ServiceProfile is used to discover a service as it presents what a service does, whereas *ServiceModel* and *ServiceGrounding* are used to make use of a service as they describe how the service works and how to access it respectively[44].

Service profile provides information about the service provider, a functional description (inputs, outputs, preconditions and effects), and non-functional properties (e.g. quality rating, category). The service model describes the operation of a service to enable invocation. The grounding specifies how the service can be invoked technically by the service consumer with the related parameter types required for the operation.

In the *Advertisement* phase of web services life-cycle, *ServiceProfile* is relevant to identify what a service does. In OWL-S, ServiceProfile is represented with the commonly used classes and properties as depicted in Figure 3.10.

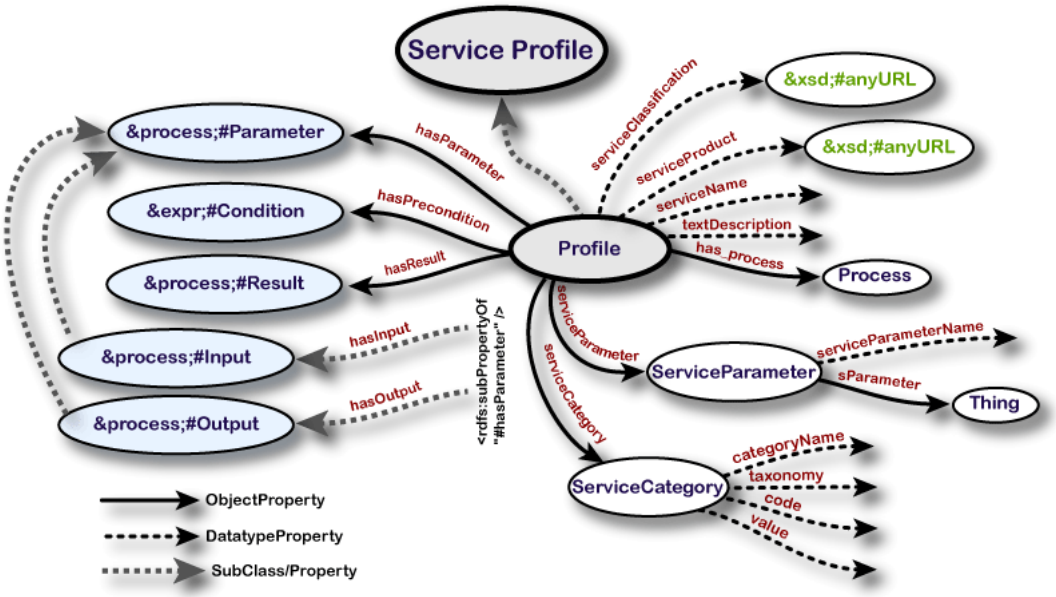


Figure 3.10: ServiceProfile with selected classes and properties [43]

Properties and classes of service profile allow autonomous discovery and matching of fitting services. The mechanisms for semantic matching use a different combination of those properties and classes. The selection of parameters for combination brings up the challenge to increase the search space during the discovery phase. In this work, the ServiceProfile definition of OWL-S is partially used.

3.1.6 Agent

Although different definitions of agent exist in Artificial Intelligence (AI), one of the most well-known definitions of an agent by Russell and Norvig is the following:

"[a]n agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators." [45]

Rational agents can make decisions to solve a problem, act on it with the goal of achieving the best outcome by using past and current perception of its environment. The rationality of an agent depends on performance measure indicating success criterion, knowledge about the environment that the agent is in, the actions that an agent is able to perform and the sequence of actions that an agent took until now.

Russell and Norvig classify the task environments of rational agents in which they are designed to solve problems as follows; fully vs. partially, single vs. multi-agent, deterministic vs. stochastic, episodic vs. sequential, static vs. dynamic, known vs. unknown [45]. Depending on a task environment properties, the representation of the environment perceived by an agent can be done by using different methods such as atomic representation (Hidden Markov Models, Markov decision process), factored representation (constraint satisfaction, propositional logic, Bayesian networks, ML) or structured representation (Relational DBs, first-order logic, first-order probability models) [45].

Agents in AI are grouped in four main categories based on perceived intelligence and their capabilities as reflex agents (simple and model-based), goal-based agents, utility-based agents and learning agents. In the scope of this work, only goal-based and utility-based agents are relevant and further explained.

Goal-based agents take decision based on how far they are currently from their desirable condition (goal) as depicted in Figure 3.11. Their actions aim for reducing the distance from their goal. It uses knowledge to choose from multiple possibilities which require searching and planning.

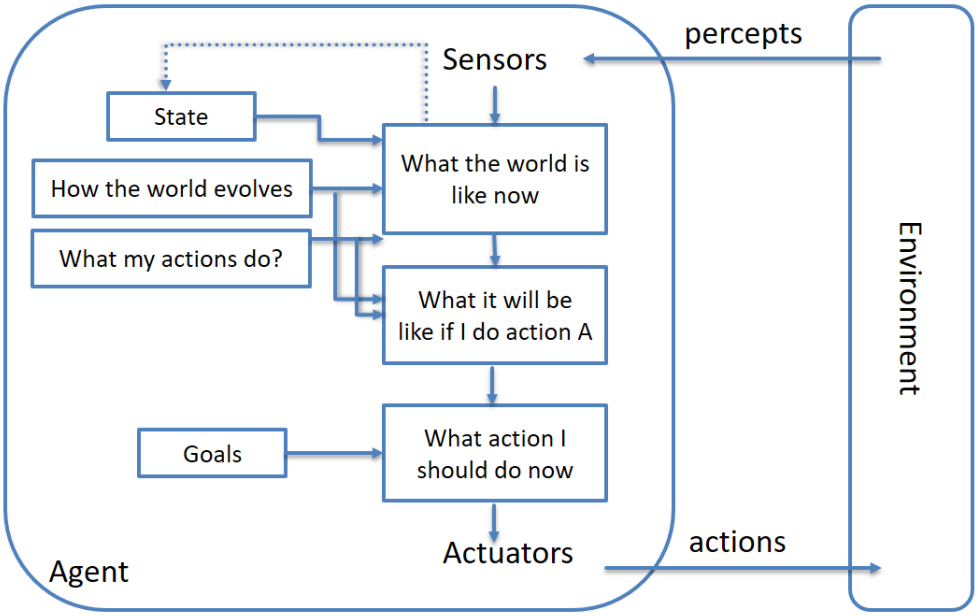


Figure 3.11: Goal-based agent behaviour (adapted from [45])

Utility-based agents take decision based on a preference (utility) on top of a goal as depicted in Figure 3.12. When there are multiple possibilities to achieve a goal, their actions aim for maximizing the expected utility yielding expected performance. This is taken into account by using a utility function in the decision-making process.

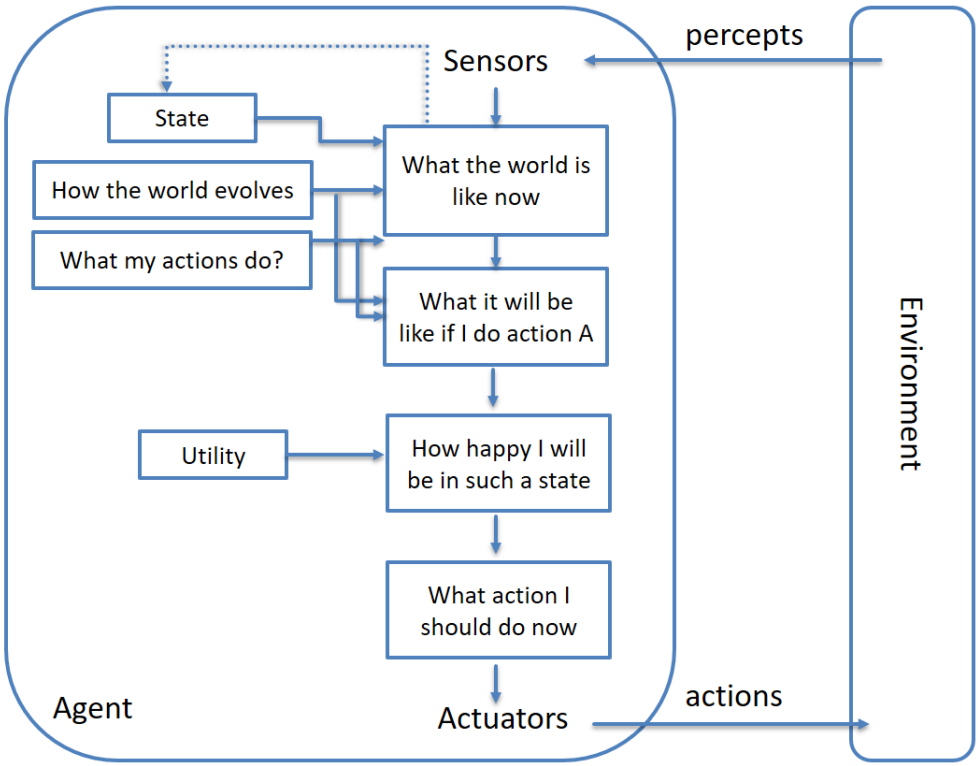


Figure 3.12: Utility-based agent behaviour (adapted from [45])

Problem-solving in AI is considered as a type of goal-based agent. In computer science, problem-solving requires algorithms and heuristics.

To perform a problem-solving, first goal formulation is required based on the current situation of an agent and its performance measure. After that, the problem formulation is necessary based on which actions the agent should take to achieve the formulated goal. Problem formulation has (1) *initial state* a starting step of agent to the goal, (2) *actions* available potential actions to the goal, (3) *transition model* describing each action's results, (4) *goal test* determining whether the current state is the goal and (5) *path cost* is the performance based on the selected actions[46].

The set of all states reachable from the initial state by following any sequence of actions form a state-space of a problem. This can be seen as a directed graph where nodes are the states, links between the nodes are actions, and the path is a sequence of states connected by the sequence of actions[45, 46]. To find a solution to a problem, all possible sequences of actions to reach the goal state from the current state should be searched and the best optimal path should be selected based on the agent's performance measure[45, 47]. There are two main categories of classical search strategies for this purpose as uninformed (blind) search (e.g.

depth-first search, breadth-first search) and informed (heuristic) search (e.g. greedy search, A* search) [45]. The performance measure of each strategy depends on the completeness of the algorithm in the given state-space, optimality of the algorithm to find the optimal solution in the given state-space with the time and space complexity. Besides the classical search strategies, local search algorithms (e.g. hill-climbing search, simulated annealing) can be also used to reach a goal-state without considering performance measure [45].

Knowledge-based agents need information related to their environment. An agent uses information in order to make decisions. There are different techniques that can be used to represent knowledge such as propositional logic, first-order logic, rule-based systems, and semantic networks[26].

In knowledge representation, it is important to choose the sufficient granularity for the problem domain to represent the relationships between different involved concept as well as accessing the stored information efficiently. In order to address this requirement, knowledge engineering focuses on the process of representing a specific domain with the important concepts and the formal representation of the objects and relations in that domain. Ontologies are one of the most commonly used methods to represent knowledge in complex environments.

When the knowledge is represented, the agent needs to understand the stored knowledge via reasoning. Forward and backward chaining are the processes in AI to make certain inferences. Forward chaining is bottom-up and data-driven approach starting with the available data and using inference rules to extract data until the goal is not reached whereas backward chaining is a top-down and goal-driven approach applying the rules on the known facts from the goal[45]. Many problems in AI require mostly more than one action to be taken by an agent in order. Russell and Norvig defined a sequence of actions that will achieve a goal as planning [45].

3.1.7 Feedback loop

To provide adaptive behaviour, the descriptions are made by taking the feedback loop from control theory as a basis.

Feedback loop also called as closed-loop is the basis of control theory to automate the control of systems which are under dynamic change. Figure 3.13 depicts the main components of the feedback loop. The mechanism for control works via comparing the *measured output* of the *target system* to act as defined in *reference input*. While doing so, it is subject to the *control error* which is, by using *control input*, is compensated for target system to achieve the

desired output. Measured output can be affected by *disturbance* produced by control input and *noise* produced by target system. *Transducer* is responsible to convert measured output to be comparable with reference input. Most importantly *controller* is responsible to create a control input to achieve the desired output based on current and past values of control error [48].

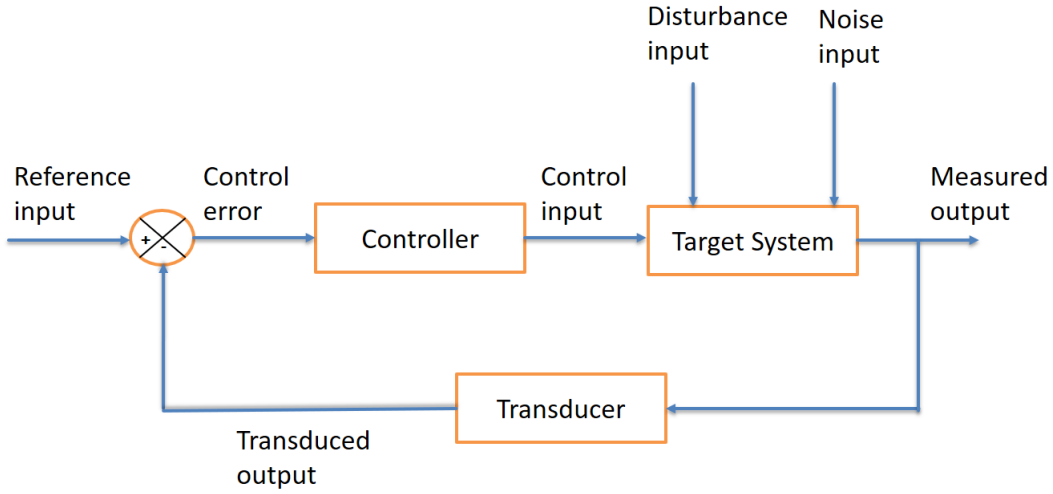


Figure 3.13: Block diagram of a feedback control system (based on [48])

In software engineering, the MAPE-K feedback loop has been proposed [12] to implement the controller functionality of the feedback loop presented in Figure 3.13 for adaptive behaviour of a software system. In the next section, the details of feedback loops in software engineering and proposed architectures for self-adaptability are presented.

The MAPE-K feedback loop has been proposed by IBM as part of their proposal for autonomic computing [49]. The purpose is the implementation of self-adaptation in software systems. The adaptation is done via a dedicated component, called the autonomic manager as presented in Figure 3.14.

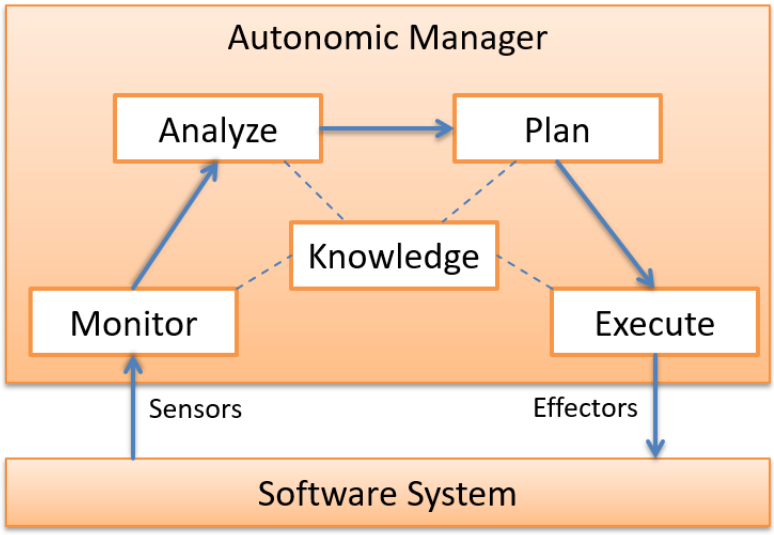


Figure 3.14: The MAPE-K feedback loop (based on [49])

The autonomic manager implements a feedback loop, which **M**onitors information about the software system (and its context of use), **A**nalyses this information, **P**lans changes according to the result of the analysis and **E**xecutes these changes. A central **K**nowledge base is used to pass information between the phases of the feedback loop and to store information between execution cycles. This feedback loop is called MAPE-K feedback loop according to its main components. It is an abstraction of the general processes involved in decisions about adaptations. Each component can be implemented in different ways. The autonomic manager is connected to the software system via sensors for receiving the monitored information and effectors to implement the planned changes. In the scope of SASS, this is an instance of the external approach to adaptation [50] where the autonomic manager corresponds to the adaptation engine.

3.2 State-of-the-Art

This section presents the state-of-the-art from four different main topics related to this research work. First, the approaches and techniques in the area of sensor and observation modelling are discussed in Section 3.2.1 to answer *RQ2* and partially *RQ1* (see Section 2.3). Then in Section 3.2.2, the approaches related to context and quality of context modelling including definition and calculation of quality are presented. This is related to *RQ1* and *RQ2* (see Section 2.3). Section 3.2.3 presents the related work about the selection, ranking and optimization methods both from sensor and service perspective required for *RQ3* (see Section 2.3). Finally, Section

3.2.4 presents the architectures to provide self-adaptability in general as well as application to opportunistic sensing required for *RQ4*.

3.2.1 Sensor and observation modelling

3.2.1.1 Modelling with ontologies

In order to have a generic description model for sensor data, sensor ontologies are commonly used to define sensor description and data modelling for IoT solutions [51, 52]. There have been various OWL-based sensor ontologies developed as presented in the survey of Compton et al. [51]. OntoSensor is one of the sensor ontologies which extends well-known upper-level ontology SUMO (Suggested Upper Merged Ontology) [53] by providing comprehensive set of concepts to represent sensors and their data [54]. CESN is another domain ontology describing sensor types and the relationship between sensors and their measurements with a restriction of measuring one physical property [55]. Another domain ontology called MMI Device Ontology is designed with the purpose of describing oceanographic devices and sensors excluding data observation relations for maritime applications [56]. CSIRO is another domain ontology developed to describe sensors with abstract and concrete properties in relation to domain concepts and observation process [57].

The metadata annotation for sensor characteristics is proposed to capture the characteristics of a sensor accurately [15]. For this purpose, Sensor Web Enablement standards are proposed to annotate sensors and observations by Open Geospatial Consortium (OGC)[5]. One of those standards is SensorML [58] which enables describing the details of sensors with their observation data from data provider point of view. Another standard by OGC is Observations and Measurements (O&M) [59] complementing SensorML by focusing on observation and observable properties from a data consumer point of view.

To address the integration gap of those standards by OGC with Semantic Web and Linked Data, The W3C Incubator Group proposed Semantic Sensor Network (SSN) ontology to describe observations measured by sensors [60]. The SSN ontology focuses on providing a domain independent ontology which is generic enough to adapt to different use-cases at the sensor and observation levels. SSN is developed based on an ontology design pattern called the Stimulus Sensor Observation (SSO) [61] to provide minimal ground.

[5]Open Geospatial Consortium (OGC) `https://www.opengeospatial.org/`

SSO pattern is further developed and extended as Sensor, Observation, Sample, and Actuator (SOSA) pattern which become a baseline of current SSN ontology [60]. The current SSN ontology based on SOSA can be seen in Figure 3.15 from observation perspective by depicting SOSA-related components and restrictions in green and SSN-only components in blue. In this work, SSN/SOSA ontology is partially used to describe sensory data as presented in Section 7.3.

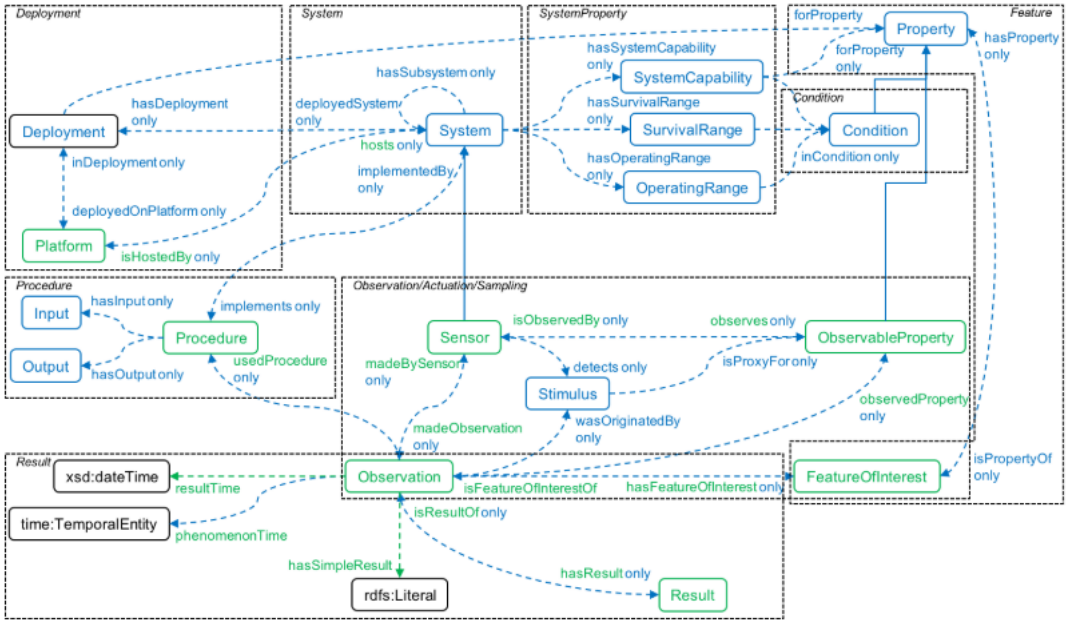


Figure 3.15: Overview of the SSN/SOSA classes and properties (observation perspective) based on [60]

SSN/SOSA ontology does not define result of observation further as quantity values. Quantities, Units, Dimensions and Data Types (QUDT) ontology defines the numeric value and unit of measurement as a quantity [62].

To further close the gap between Semantic Web and IoT, Web of Things (WoT) architecture is introduced to provide an open Internet of Things based on web standards by providing different decoupled layers on top of networked things [63]. The illustration for WoT architecture based on [64] using different web standards can be seen in Figure 3.16.

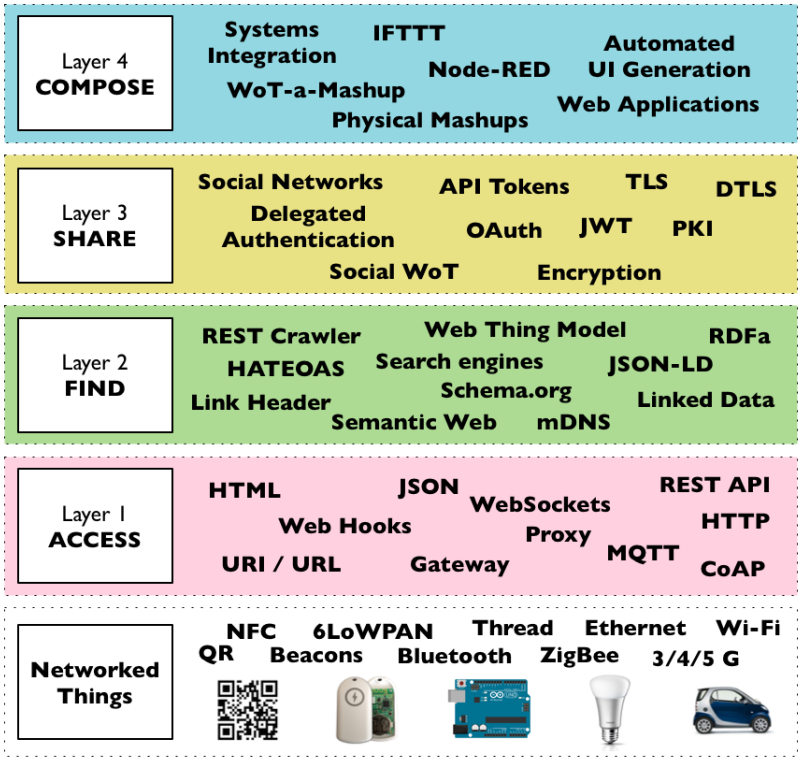


Figure 3.16: Overview of layers in WoT architecture based on [64]

In this work, similar standards like in WoT are used to address the challenges in different levels by combining service-oriented architectures. The next section presents the service perspective for the modelling sensor data.

3.2.1.2 Modelling as a service

Everything-as-a-Service (XaaS) [65] concept has evolved with cloud computing to use available hardware and software resources in an efficient way. Major service models proposed under XaaS, like Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (PaaS), were limited with cloud computing at the early stage. With the evolvement of IoT, new service models are proposed following this idea. One of those models is Sensing-as-a-Service (S2asS) [6, 66] supporting the idea of data exchange between the data owners and data consumers in IoT by connecting sensors to software systems and related functionalities as a service. Several middleware solutions are proposed (such as OpenIoT [11] and GSN [67] to facilitate such a model to address the available sensors in a big scale in IoT. Service-Oriented Architecture (SoA) principles are commonly applied in IoT due to its unassociated, loosely coupled and self-contained units of functionalities [36].Similarly, the Context-as-a-Service (CoaaS) model [68] has evolved as an abstraction layer on top of the data that can be retrieved

from different resources via IoT and aims at addressing the gap between context consumers and producers to provide interoperable data exchange [69].

In the next section, sensor modelling as a service i explained in order to describe sensor data and wider interaction with other entities in IoT.

3.2.1.3 Sensor modelling as a service

Modelling sensor and observation alone is not sufficient to develop an IoT application as sensor data needs other interactions in the environment. De et al. proposed IoT model [70] with key interacting components as depicted in Figure 3.17.

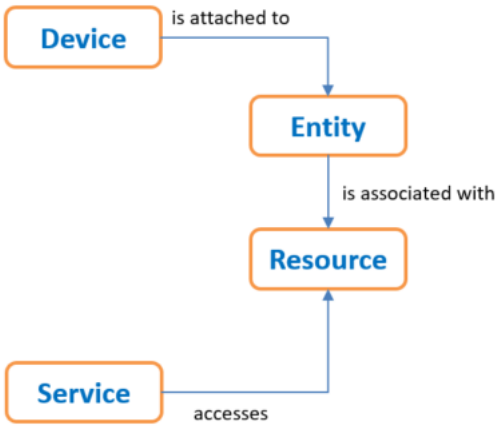


Figure 3.17: IoT Model: key concepts and interactions [70]

In IoT model, *Entity* constitutes "things" (e.g. human, animal, car, store, logistic chain item, electronic appliance, closed or open environment); *Device* is either attaches to an entity or part of the environment of an entity so it can monitor it; *Resource* provides information on the entity or enables controlling of the device and *Service* provides a well-defined and standardized interface, offering all necessary functionalities for interacting with entities and related processes. In summary, the services expose the functionality of a device by accessing its hosted resources [70].

Based on the depicted interaction in Figure 3.17, IoT-A ontology is proposed combining SSN ontology by additionally representing IoT related concepts [70]. IoT-A ontology uses OWL-S upper ontology to map the depicted entries in Figure 3.17. This mapping can be seen in the righ-hand side of Figure 3.18.

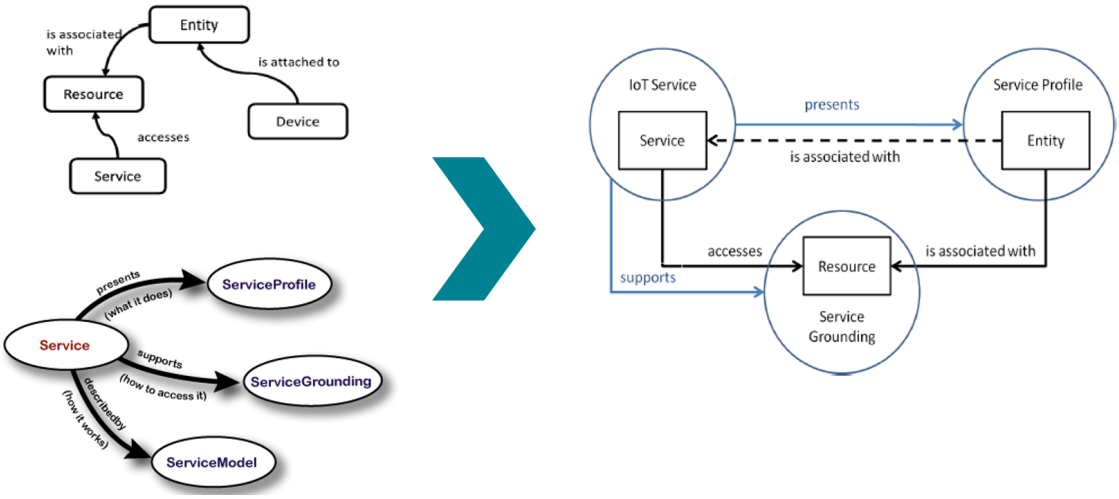


Figure 3.18: IoT model concepts by using OWL-S [70]

This mapping indicates the entity as service profile in OWL-S where it is associated with a service. Resource is represented as service grounding where a service accesses. IoT-A model combined SSN and OWL-S to descibe IoT concepts. The implementation of this model is provided in [71] with detailed concepts, properties and relations.

Although, IoT-A model gives flexibility regarding expressiveness due to using OWL-S as an upper ontology, it is complex for dynamic environments in IoT which requires fast adaptation [72]. To provide lightweight semantics in IoT, Bermudez et al. proposed IoT-Lite ontology as an instantiation of SSN ontology having lightweight semantics [72]. Figure 3.19 shows the core SSN concepts used in IoT-Lite ontology as well as entity, service and resource descriptions and relations.

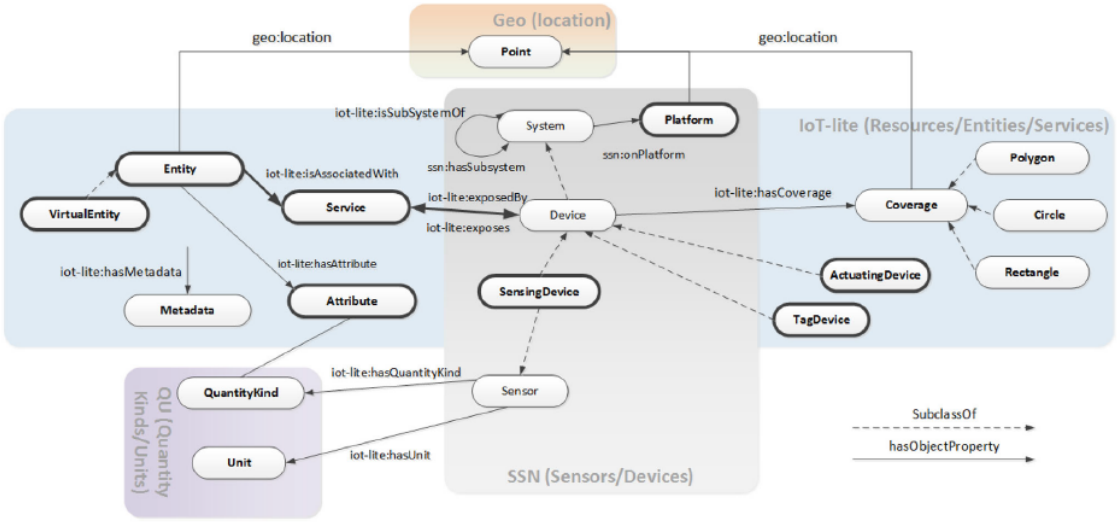


Figure 3.19: Overview of IoT-Lite ontology [72]

IoT-Lite provides lightweight semantics for sensory data in IoT by focusing on the device descirption. However, the ontology does not cover the quality parameters for different entities in ontology like sensor, device and service. This topic is covered in the next section under context and quality of context modelling.

3.2.2 Context and Quality of Context Modelling

In this section, the techniques and approaches related to context and its quality modelling as well as its quality calculation are provided. The aim of this section is to give insights about context consisting of different entities. In this study, context defintion and modelling is related to the recognition chain components in relation to recognition goal (RQ1 and RQ2 in Section 2.3) whereas quality of context combining different quality parameters in the recognition chain is considered for the optimal selection (RQ3).

3.2.2.1 Context Modelling

Sensor data alone does not provide any context information without interpretation. Therefore, context modelling is necessary to interpret sensor data from different sources and aggregate/-correlate them to infer a specific piece of information about context. Bettini et al. stated main requirements for accurate context representation as dealing with heterogeneity and mobility, taking into account relationships and dependencies between context information, timeliness of information, considering imperfection of sensor data, efficient reasoning, usability of modelling formalisms, and efficient context provisioning [73].

Six major context modelling techniques identified by Strang et al. addressing the above requirements as (1) key-value, (2) markup scheme, (3) graph-based, (4) object-based, (5) logic-based, and (6) ontology-based modelling [74]. As each model has advantages and disadvantages affected by also preferred context reasoning methods, using hybrid models for accurate context representation is the most preferable way depending on the requirements of the domain [6].

Context ontologies aim at describing characteristics of context in various domains. Wang et al. developed OWL-based context ontology named CONON (CONtext ONtology) which provides upper ontology to provide generic concepts like location, person or activity as well as extensibility to add domain-specific concepts for context information [75].

SOCAM (Service Oriented Context Aware Middleware) ontology is developed as an extension of CONON for context-aware services in smart environments [76]. It uses CONON's

upper ontology and extends with domain-specific concepts for smart homes to enable the development of services. CoBrA-ONT (ontology for Context Broker Architecture) is another OWL ontology which supports context-aware systems with distributed agents [77]. SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) ontology is defined based on COBRA-ONT to provide a vocabulary for several generic concepts for pervasive applications [78].

Generic representation of context is addressed by theoretical context models. One of those models developed by different researchers is the Context Spaces Theory (CST) which provides a general context representation and model with a theoretical foundation by using a multidimensional space metaphor to represent, reason about, and validate, context [79]. CST allows the combination of specification-based techniques like formal logic and learning-based techniques like the Bayesian classification to determine situation occurrence.

3.2.2.2 Quality of Context

For context-aware applications and services, quality is defined in three categories. Quality of Context (QoC) is defined as a quality of the gathered information (e.g. accuracy, freshness) whereas Quality of Service (QoS) is the quality of the service that provides this information (e.g. availability, response time) and Quality of Device (QoD) is the technical properties of the devices (e.g. sensors) that collect data (e.g. battery life) [80]. These quality parameters are not orthogonal as they influence each other. For example, the distance between data sources (QoD) can influence the speed of the service (QoS) and the accuracy of the information (QoC) [80]. In this section, first the different definitions of QoC are provided, then different techniques for the calculation of those parameters are presented.

Quality Criteria Definition:

Henricksen proposes the most common quality parameters as *accuracy*, *confidence*, *freshness*, *resolution* and *credibility* [81]. While he uses the term accuracy to describe the degree to which the data reflects the reality, Buchholz et al. [80] refer to the same measure as *precision*. In contrast, McKeever et al. [82] and Filho et al. [83] define accuracy as the frequency of correctness from the sensor readings. Instead of confidence, freshness and credibility, Buchholz et al. use *probability of correctness*, *up-to-dateness* and *trust-worthiness* [80].

Although those mentioned studies alone do not reflect a comprehensive list of quality parameters, it can be summarized that there is not a standard definition of parameters and those parameters can be interpreted differently based on the related domain. One of the examples can be given by the work from Villalonga et al. who addresses the quality requirements in activity recognition and wearables domain and extended quality parameters by defining *origin* and *spatial scope* [84]. Perera et al. propose additional quality parameters for device and sensor quality [85]. Commonly used quality criteria with definitions are compiled within this study as presented in Appendix A.

As there is no standard definition of commonly used parameters which are accepted by all researchers, the research on definition of quality parameters moved into a meta definition of related parameters that could cover the commonality of existing parameters.For this purpose, Filho et al. defined QoC ontology where some common parameters are defined and other parameters can be included by extending this ontology [83]. Marie et al. proposed a meta-model for modelling any type of QoC parameter after analyzing different list of quality parameters exist in the literature for context-aware applications [86].

The importance of the quality parameters depends mainly on the domain for a specific application or a situation [83]. For this purpose, if the related importance is not known beforehand, it is essential to define the importance or relative importance between existing quality parameters when developing context-aware applications. For this purpose, Manzoor et al. proposed a *crucial value* parameter, which indicates that the information is of importance for concrete situations [87]. Additionally, Manzoor et al. proposed a model for processing different QoC metrics by differentiating QoC as subjective and objective view depending on the application requirements [88]. This is also similar to Quality of Experience (QoE) definition where the satisfaction of the user by the provided service has both objective and subjective sides. However, The approach from Manzoor et al. is based on using only one entity (context object) to calculate those quality parameters. Different than their work, this work addresses the combination of different context objects (e.g. information providers) using those QoC parameters.

To summarize, the survey of quality parameters in the literature has outlined the heterogeneity in their denomination and definition. Further, credibility is an example of a criterion presented under different names in this area of research. Moreover, timeliness can be found as synonym to both currency and temporal scope, although both parameters differ

in their meaning. Additionally, the following section outlines diverse ways of evaluation of the quality criteria.

Quality Criteria Calculation:

Although there is huge effort describing the quality in context-aware computing, only few research effort present the estimation of those parameters. In the last decade, many different solutions, frameworks, middlewares and approaches were developed to provide those characteristics.

Buchholz et al. [80] proposed the relation between QoS, QoD and QoC as mentioned in Section 3.2.2.2, they did not provide a method to measure those quality parameters and their interrelations. Kim and Le [8] provided statistical formulas to calculate accuracy and completeness. However, those are not tested in real scenarios. Manzoor et al. [87] provide metrics for the context quality parameters they proposed as functions whose interval range is between 0 and 1. Their definitions of context quality parameters were specific to the application scenario. Villalonga et al. presented guidelines to match quality parameters used to assess the quality of context in a generic way and mathematical formulations for the conversion in activity recognition domain [84]. However, those conversion functions do not take into account the requirements from the application side. Filho et al. [83] define QoC indicator (QoCI) to evaluate higher-level quality property based on QoC parameters (QoCP) retrieved from sensors. They also differentiated the same quality parameter for different entities like context source (e.g. sensors) and context provider (e.g. one or more context sources) [83].

Within the scope of opportunistic sensing for activity recognition, Kurz et al. defined two QoS metrics as Degree of Fulfillment (DoF) expresing how well the activity is recognized and Trust Indicator (TI) indicating how trustworthy is the sensor data at specific time [2]. Their approach assumes sensors to be label deriving entities including those QoS metrics. While this can be useful in activity recognition regarding classification, for context recognition in some cases, actual data is more important than rough classification.

Marie et al. [86] proposed meta-model for QoC metric definition as mentioned in Section 3.2.2.2. At their following work [89], they also proposed formulas for the calculation of complex quality metrics from the basic ones (e.g. freshness from current date and date of context production). However, competing objectives for different QoC metrics was not covered at their work.

Sicari et al. [90] argued that the existing work on the calculation of QoC metrics mostly depends on the complete description or meta-data that can be retrieved from sensors. As it is not the case in real deployments, the quality metrics should be dynamically calculated. Therefore, they proposed an architecture for the calculation of some metrics based on using configuration from application/user side and existing quality values of data sources [90]. However, their solution does not take into account multiple conflicting quality metrics.

To summarize, there are a lot of definition and calculation method for QoC metrics in literature. However, resources like power, storage and memory which are used during the calculations should also be taken into consideration. Multi-criteria decision-making approaches should be developed to be able to deal with multiple and conflicting criteria, as well as dynamic update depending on the changes on the application requirements and sensing infrastructure characteristics during runtime.

3.2.3 Selection, ranking & optimization

To address the RQ3 (see Section 2.3) for providing optimal selection among available recognition chains, selection, ranking and optimization techniques from different domains are provided in this section. First, the approaches related to sensor selection and ranking techniques are presented in Section 3.2.3.1. Then, the approaches from service domain are provided for the selection and optimization within the scope of service composition in Section 3.2.3.2.

3.2.3.1 Sensor Selection and Ranking Techniques

Manzoor et al. [91] used selection of best sensor to solve context conflicts by providing the context information and quality threshold values as an input to globally eliminate the sensors not fulfilling those parameters. Filho et al. [83] used similar approach and extended to have more flexibility by having different aggregation methods (default, optimistic, pessimistic, average) for thresholds instead of global ones.

Ostermaier et al. [92] proposed sensor ranking based on calculating likelihood of sensors gathering the requested information and then adjusting the ranking based on the accuracy of past rankings with the assumption of same behaviour for each sensor periodically [93].

In Opportunity Framework, as described in Section 3.2.2.2, Degree of Fulfillment (DoF) and Trust Indicator (TI) are used for the quality. Roggen et al. uses those metrics to determine

best-n sensors for the recognition goal by ranking the available sensors with respect to accuracy [19].

Perera et al. classified the requirements for sensor ranking as point-based requirements which are mandatory to fulfill and proximity-based requirements which are negotiable and significance of each requirement in this category defined by the user [94]. This is similar to hard and soft constraints in service domain for QoS parameters where a hard constraint specifies that the constraint must be satisfied at any cost whereas a soft constraint allows a certain violation of it [95, 96]. Based on point and proximity-based requirements, Perera et al. proposed a Comparative-Priority Based Weighted Index (CPWI) based on a Euclidean distance to represent each sensor in multi-dimensional space where every dimension represents a quality parameter [94] for sensor ranking. To increase the efficiency, the authors also suggested additional Comparative-Priority Based Heuristic Filtering (CPHF) removing the sensors that does not fulfill the user requirements before the ranking and adding lower and upper bounds of each quality parameter by a user [94, 85].

Wang et al. proposed a ranking model by calculating the cost of accessing a sensor in the network and storing this information as metadata [97]. Although this does not cover comprehensive quality criteria for ranking and selection, it can be combined with other methods like CPWI and CPHF proposed by Perera et al. [94].

3.2.3.2 Quality-aware service composition

In this section, the approaches for selection, ranking and optimization from service point of view are explained. In service domain, those approaches are usually used for service composition. Service composition is a way of building a new service by combining existing services. When there are many services for the same task in the composition, it becomes difficult to choose the optimized combination based on some properties. One of the most important properties in the service composition is Quality of Service (QoS) indicating that how well the service serves (reliability, availability, response time, etc.) [98]. The selection of the optimal service composition to maximize the end-to-end QoS requirements of a composition automatically is an NP-hard problem [99]. Optimization problem for QoS-aware service composition can be modeled as a multidimensional, multi-objective, multi-choice knapsack problem (MMMKP) where there is m classes, n items of each class, l dimensions of each item's profit and k dimensions of each item's weight with a goal of selecting one item of each class

to maximize each dimension of profit and not exceed the capacity of each weight dimension [100]. In the service composition classes are the composition tasks, items are available service implementations of the tasks, dimension of profit and weight are the QoS properties and capacities of weights are the composition's QoS constraints [98]. An example composition based on [98] is depicted in Figure 3.20.

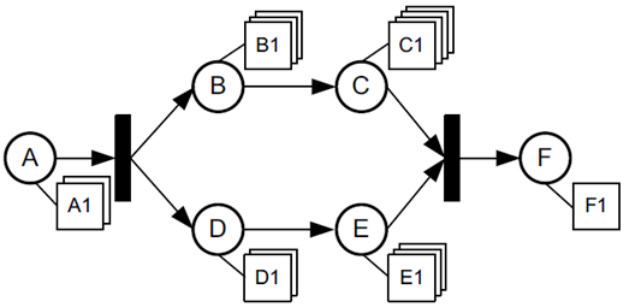


Figure 3.20: Example service composition problem [98]

Figure 3.20 presents the composition problem from A to F with the goal of maximizing three QoS-properties as response time, availability and reliability and two QoS-constraints that a minimal availability of 90 percent and a minimal reliability of 95 percent. The resulting MMMKP is as follows [98]:

- six classes (tasks) with two items in class A (A1, A2), three items in class B (B1, B2, B3), four items in class C (C1, C2, C3, C4), two items in class D (D1, D2), three items in class E (E1, E2, E3) and one item in class F (F1);
- two dimension of weight wA and wR with the capacity of wA=0.90 and wR=0.95
- three dimension of profit pT, pA and pR

Strunk classifies approaches to solve QoS-aware service composition problem in two main categories exact solutions finding the optimal execution plan and (meta-) heuristics solutions selecting a nearly optimal execution plan [98]. There have been vast amount of algorithms proposed for exact solutions including but not limited to the optimization methods such as Linear Integer programming, Dynamic programming, Mixed Integer programming algorithms as well as (meta-) heuristics include Genetic algorithm, particle swarm optimization, simulated annealing and ant colony optimization algorithms [101, 98]. As an overall, each of the solutions uses different strategies with similar principles to maximize or minimize objective functions. (Meta-) heuristics that could provide a sufficiently good solution to an quality-aware service

composition optimization problem, sometimes even more efficient than exact algorithms. However, (meta-) heuristics cannot guarantee a global optimal solution compared to the optimization algorithms and iterative methods.

Cardoso et al. proposed aggregation methods in service composition for specific QoS properties to calculate the overall quality of the composition based on different control flows [102]. Those formulas are depicted in Figure 3.21.

QoS property	*Sequence*	*Concurrency*	*Loop*	*Choice*
Availability	$\prod_{i=1}^{m} A(s_i)$	$\prod_{i=1}^{p} A(s_i)$	$A(s)^k$	$\prod_{i=1}^{n} p_i * A(s_i)$
Reliability	$\prod_{i=1}^{m} R(s_i)$	$\prod_{i=1}^{p} R(s_i)$	$R(s)^k$	$\prod_{i=1}^{n} p_i * R(s_i)$
Time	$\sum_{i=1}^{m} T(s_i)$	$Max(T(s_i)_{i\in\{1\ldots p\}})$	$k * T(s)$	$\sum_{i=1}^{n} p_i * T(s_i)$
Price	$\sum_{i=1}^{m} P(s_i)$	$\sum_{i=1}^{p} P(s_i)$	$k * P(s)$	$\sum_{i=1}^{n} p_i * P(s_i)$
Custom property	$f_S(F(s_i))$, $i \in \{1\ldots m\}$	$f_P(F(s_i))$, $i \in \{1\ldots p\}$	$f_L(F(s_i))$, $i \in \{1\ldots k\}$	$f_C(F(s_i))$, $i \in \{1\ldots n\}$

Figure 3.21: Aggregation formulas for calculating QoS of a service composition [102]

Quality estimation with multiple conflicting criteria is maturely researched in the scope of multi-objective/multi-criteria optimization especially in web services domain for service composition. The area is still challenging as the end-to-end quality of recognition chains is not only influenced by the services but also properties of the digital parts of the devices and sensors. The influence of methods to process device/sensor data is also important to be taken into consideration as they can affect the overall quality of the retrieved information.

3.2.4 Self-adaptability

In this section, the approaches for self-adaptability are provided to react the changes for optimal selection. For this purpose, the research area of Self-Adaptive Software Systems (SASS) is taken as basis. In SAAS, several architectures and models have been proposed to implement self-adaptation. One of the approaches presented in Section 3.1.7 is to define adaptive behaviour for autonomic computing as feedback loops from control theory. This idea provides a conceptualization of the reasoning process to decide whether an adaptation is required or not.

Several authors have argued that SASS usually contain some form of such a feedback loop but it is usually intermingled with the software system itself. Adaptation via a controlling

feedback loop can be done explicit or implicit in the design of self-adaptive software systems. This is categorized by Salehie and Tahvildari as internal and external approaches to engineer SASS [50] as depicted in Figure 3.22.

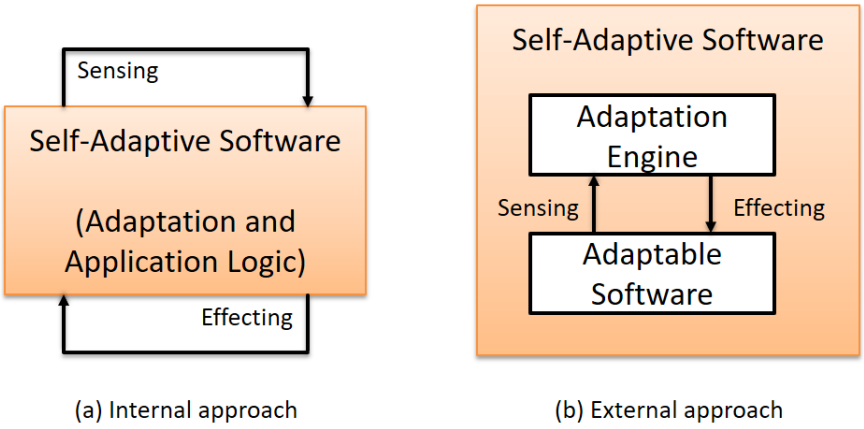


Figure 3.22: Internal and external approach for SASS (based on [50])

As depicted in Figure 3.22, in the *internal* approach (a), the adaptation mechanism and application logic is not separated whereas in the *external* approach (b), adaptation is achieved via separated component called adaptation engine interacting with the adaptable software [50]. MAPE-K is an implementation of adaptation engine in the external approach [49]. Throughout this thesis, adaptation engine is called as adaptation controller.

Making feedback loops explicit from the system design is proposed by several authors [103, 48, 104] as leads to a clear separation of concerns between the adapted system and the adaptation mechanisms and can provide standardized components for the adaptation that can be reused by other self-adaptive software systems. Engineering feedback loops directly enables the comparison of self-adaptive systems, and the structured development and re-use of existing components or ideas.

A SASS may also include multiple feedback loops [49, 105]. In this scope Vromant et al. give an overview of interactions between the components in multiple feedback loops with regards to intra- and inter-loop coordination [106].

Villegas et al. stated that a complex self-adaptive system often involve adaptations in the context infrastructure and in the systems goals along with the adaptation of the software system itself [107]. This is reflected in DYNAMICO model, a reference model for implementing SAS systems that specifies three interacting MAPE-K feedback loops. These three feedback loops are depicted in Figure 3.23.

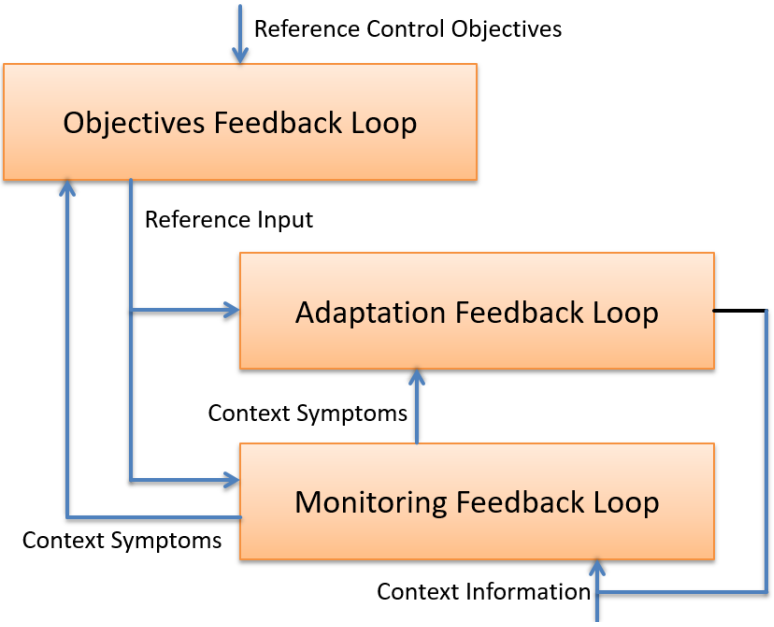


Figure 3.23: Feedback loops and interactions in the DYNAMICO model (based on [107])

The *Adaptation Feedback Loop* is the main MAPE-K feedback loop which controls the adaptation of the software system. The *Objectives Feedback Loop* is responsible for adapting the overall system goals. The adaptation is based on the externally given reference control objectives and the information received as context symptoms. This feedback loop provides the adaptation goals as reference input for the adaptation feedback loop and the detection goals as reference input to the monitoring feedback loop. The *Monitoring Feedback Loop* receives this information sensed from the internal (from the adaptation feedback loop) and external (from other sources) context information and derives the required context symptoms for the adaptation and objectives feedback loops based on its detection goals.

As mentioned in Section 3.1.7, the adaptation via MAPE-K feedback loop is done through its four main phases (MAPE) where the calculations for adaptation take place. Knowledge base (K) provides required information in those phases. However, knowledge base is less structured comparing to the other phases of MAPE-K feedback loop in this respect [108]. To provide an abstraction for the knowledge base, Vogel et al. proposed an extension of MAPE-K by using runtime models [109].

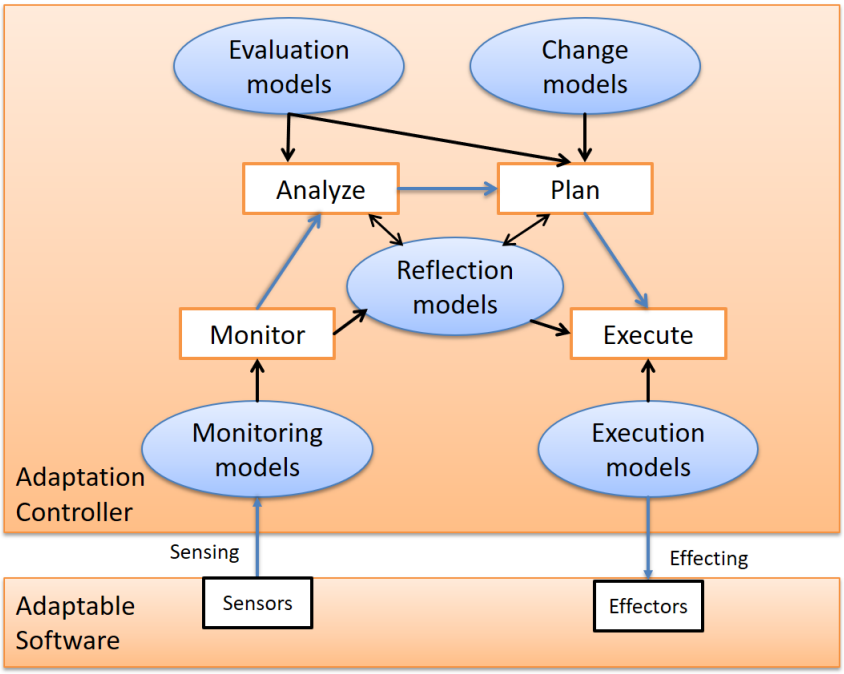


Figure 3.24: Runtime models for MAPE-K (based on [109])

As depicted in Figure 3.24, *Reflection Models* reflect the software and its environment; *Monitoring Models* are responsible to map observations to the reflection models; *Evaluation Models* describes the goal towards to the required adaptations; *Change Models* define the options that are available for planning which can be used to find an appropriate adaptation with the guidance of evaluation models; and *Execution Models* describe to execute the planned adaptation by refining from the model to the adaptable software [109].

To analyse adaptation in opportunistic sensing, Opportunity Framework [19] is used to see how adaptation is implemented in opportunistic sensing. Opportunity Framework as presented in Section 3.1.2 is an adaptable software system requiring flexible mechanisms to react to the changes of the sensor infrastructure (sensor appearance, disappearance, changes in sensor properties). Opportunity deals with the adaptability in dynamic configuration where the recognition chain is configured based on the recognition goal as well as in autonomous evolution where domain knowledge and methods for mapping sensor signals to activities are evolved based on the relations between sensors and activities. The dynamic configuration adapts the composition of the whole recognition chain, on the other hand autonomous evolution adapts individual component in the recognition chain. When any change occurs in the sensor infrastructure, it needs to be evaluated whether a change in the recognition chain, e.g., selecting/substituting one sensor with another or changing the methods in the recognition

chain, is required. This enables the fulfilment of the requirements of the recognition goal despite changes. Opportunity is able to provide this adaptability which requires autonomous control in the sense that the mechanisms constantly check and optimize their status and adapt themselves to the changing conditions. However, this adaptability process is hidden in black-box components in Opportunity Framework which makes it difficult to extend the logic of adaptation for multi-criteria optimization during recognition chain selection. As argued above distinguishing between these phases and making the activities in each phase explicit can help to compare and combine this framework with other approaches.

3.3 Summary

To summarize, the state of the art approaches are presented from sensor and service perspective under the category of (1) sensor and observation data modelling in Section 3.2.1, (2) context and quality modelling in Section 3.2.2, (3) selection, ranking and optimization from both sensor and service perspectives in Section 3.2.3 and finally (4) self-adaptability from the self-adaptive software architecture point of view in Section 3.2.3. The results of the analysis show that there is no end-to-end solution providing adaptive optimal recognition chain selection method although there are promising models and approaches from different domains. In Chapter 4 and Chapter 5, the approach to enable adaptive optimal recognition chain selection.

4

Quality-aware Optimal Selection

In this Chapter, first part of the approach to enable adaptive quality of context optimization in IoT is presented by providing a framework for optimal recognition chain selection. The approach is threefold which are presented in the following subsections. First, in Section 4.1, the models with definitions and formalism used in the approach are presented. Then, in Section 4.2, the methods for quality aggregation for recognition chains, selection mechanism and optimization of selected recognition chain are provided respectively. The parts of this Chapter are published in the author's publications [110] and [111].

4.1 Models

In this section, the models used as a foundation of the approach are described. Section 4.1.1 presents the high-level definitions following with formal descriptions of those in Section 4.1.2.

4.1.1 Definitions

In this section, the definitions for recognition goal, recognition chain, service, and quality are provided. The parts of recognition chain consists of different sensors and processing functions. For this purpose, those are modelled as a service. Quality definition is attached to all those services to create aggregated quality of a recognition chain.

Recognition Goal (RG): Recognition goal is a definition of the required information with related objects. This includes: identifier of the goal, required input and output as a set of objects, and quality requirements specific to this recognition goal. In the recognition goal, the

quality requirements need to be defined to be used later in the optimization function. Quality requirements of recognition goal consist of the identifier for a each single quality property with related maximum and minimum values required to fulfil the goal (if any), the direction to be in favour of (positive when bigger value is better, negative otherwise), and the aggregation method to be used while composing the different parts of a recognition chain (e.g. some quality parameters require to take summation of values in the composition, some require to have the maximum).

Service Model: Sensor data and processing methods are defined as services in the approach. The service model consists of service description identifier, required sets of inputs and outputs for a service to be executed, and the set of quality properties of a service when it is executed. A single quality parameter consists of the parameter id, type and value of it to use later in a quality calculation of an overall recognition chain.

Recognition Chain (RC): Recognition chain consists of recognition chain identifier,the set of services involved with the identifiers and the execution order in the chain (sequential and parallel execution), the quality properties of the recognition chain that results from aggregation of quality properties of involved services by comparing with the recognition goal. This includes a set of different quality parameters including related id and value after the chaining process.

Quality Model: The quality is defined in two main categories to fulfil the requirements of adaptive service selection. The first category is quality requirements (QReq), which are attached to the recognition goal. The other category is the quality properties, which are attached to service and recognition chain. The quality of recognition chain is composed by aggregating the quality properties (QProp) of involved services. The calculation of aggregated value for this purpose is provided in Section 4.2.2.

4.1.2 Formalism

In this section, formal definitions of the mentioned descriptions in Section 4.1.1 are described with related concepts and properties.

4.1.2.1 Recognition Goal

The recognition goal (RG) throughout this research is defined as follows:

$$RG =< RGid, INP, OUT, Set_{Qreq} >$$

where $RGid$ is the recognition goal identifier, INP and OUT are the required sets of inputs and outputs for the recognition goal with different control flows (e.g. sequence, concurrent). It is assumed that data types of input or output parameters are known via an interface description and data type matching.

Set_Qreq is the set of quality requirements of a recognition goal. Set_Qreq represents hard constraints for the recognition goal to be fulfilled in any condition. This represents the QoC to optimize in this research which can be a affected by a combination of QoD and QoS. Set_Qreq is defined to be used for the optimization of selection amongst available recognition chains as presented in Section 4.2.3.

A single $Qreq$ is defined by having the following properties:

$$Qreq =< id, max, min, optDirection, aggMethod >$$

where id is the identifier for a single quality property, max and min are maximum and minimum values for the constraints if any defined in the recognition goal, $optDirection$ is the type for positive or negative direction for the related property to be either maximized or minimized and $aggMethod$ is the aggregation method to be used while chaining the different parts of a recognition chain.

The type of quality parameter defines the related aggregation method to be applied while chaining services to calculate overall quality of a recognition chain. Four main types of aggregation are taken into account in this work as a summation (SUM), maximum (MAX), minimum(MIN) and multiplication (MULT) based on common quality parameter types and aggregation formulas proposed by Cardoso et al. [102].

4.1.2.2 Service Model

For the web services, the functionality can be expressed by the properties of a service that describe their behaviour in terms of its inputs, outputs, and also additional descriptions such as preconditions and effects if applicable [112]. This is commonly annotated as IOPE description of a service for functional properties and used for the composition of different services to accomplish more complex tasks. Apart from the functional properties, Quality of Service (QoS) covering different quality properties of services is also an important factor in service composition to select candidate services for the same functionality.

For describing a service, IOPE is partly used to indicate related inputs and outputs of service and excluded preconditions for simplicity. Quality-related properties of service is defined as a quality factor in the effects of this description. Based on this, the service description(SD) can be formalized as:

$$SD =< Sid, INP, OUT, Set_{Qprop} >$$

where Sid is the service description identifier, INP and OUT are the required sets of inputs and outputs for the service to be executed with different control flows, Set_{Qprop} is the set of quality properties of a service when it is executed. Set_{Qprop} can involve different quality parameters like QoS (availability, response time) as well as and QoD (e.g. accuracy, battery life) [80]. A single quality parameter $Qprop$ includes the parameter id, type and value of it to use later in a quality calculation of an overall recognition chain. The calculation of quality by using $Qprop$ is described in detail in Section 4.2.2.

As can be seen from the definition of recognition goal in Section 4.1.2.1, service description presented in this section is similar to the recognition goal only with a difference of quality parameters instead of quality requirements.

The service model is used to define the recognition chain elements mainly sensor and processing functions. The details about mapping those to a service description are provided in Section 4.1.2.4.

4.1.2.3 Recognition Chain

The recognition chain (RC) is modelled as follows:

$$RC =< RCid, SSer, Qeval >$$

where $RCid$ is the recognition chain identifier, $SSer$ is the set of services involved with the identifiers and the control flow in the chain, $Qeval$ is the relative quality evaluation of the recognition chain based on a recognition goal as presented in Section 4.1.2.1. This includes a set of different quality parameters including related id and value after the chaining process. The reflection model stores the identifiers both for involved services in the $SSer$ as well as recognition chain identifier $RCid$. It stores the mapping of these identifiers to the actual URI of a service/recognition chain to call during execution. The details about how to construct a recognition chain and how to calculate $Qeval$ is provided in Section 4.2.

4.1.2.4 Recognition Chain Elements as Services

This section includes the details of how the data sources and processing functions as services are represented based on the service model described in Section 4.1.2.2 to be used in constructing recognition chains.

Data Source Model: In order to model data sources as services, there is a need to interpret the sensor data to express input and output of the services as well as quality properties.

To model the sensor data as services, a data source ontology is proposed with the involved entities. The model representing the relations between a sensor, related observation and service is depicted in Fig. 4.1 in detail. This model allows us to capture all types of sensors and the properties they measured in a specific area. The location where the sensor is located and the observation where it is measured are separated in this model. This allows differentiation of sensor location and observation location. Each sensor is modelled based on the data it provides, its location, measured data as observation, and other information through a single service. Sensors and related properties (Observation entity in Fig. 4.1), as well as services, have quality parameters that are necessary to express in the model for the quality-aware service selection and composition.

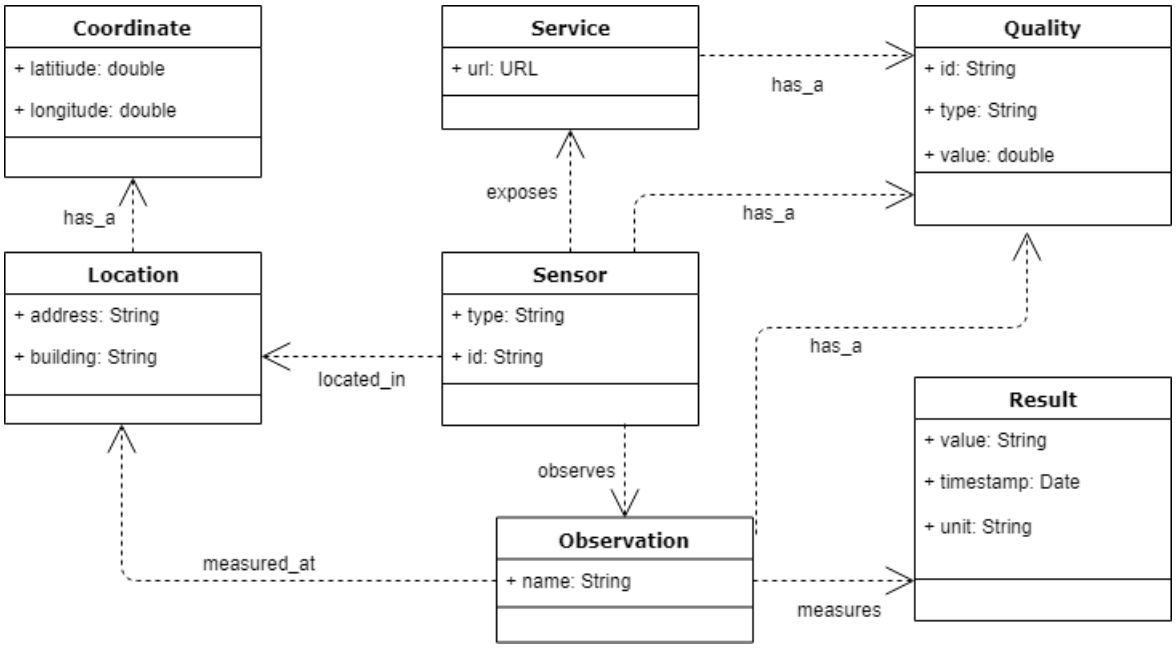


Figure 4.1: Detailed view of entities for the representation of sensor data

Sensors can provide different types of information, based on the observations they can provide. This information comes from the sensor self description or meta-data. The observations

of the different type of sensors are properties and values which the sensor can provide. This information can come from different sources. For many sensor systems in IoT, the sensors provide streaming data via a streaming platform or a database. Quality properties of a sensor include the QoD properties like battery status, accuracy if it is reported by the sensor provider. As the quality is also related to observation quality, some quality properties like up-to-dateness for observations are used as quality properties for observations. For services regarding the quality, response time and availability are used as a quality properties of a service as quality properties.

This model is different than IoA-A and IoT-Lite ontologies depicted in Section 3.2.1.3 as it allows to include quality for different entities including Observation, Service and Sensor.

Data Processing Model: In order to define processing methods as a service, the service definition presented in Section 4.1.2.2 is used. The inputs of processing methods can be either other processing methods or directly the sensors. The outputs of the methods can be directly the information that fulfils the recognition goal or sub-goals.

For the quality definition of a processing method in the service description, two types of processing methods are used in this work as analysis and prediction. This separation is necessary to define the quality for processing methods, as the prediction tasks require much computational power and not suitable for fast responses. This separation is reflected as a property of service description for the processing methods (*Qprop* in the service description).

4.2 Selection and Optimization Process

In this section, a generic algorithm is provided for the quality-optimal selection of recognition chain to indicate each step in this process. It consists of constructing recognition chains (1), filtering recognition chains whose individual services cannot fulfil the quality requirements of the recognition goal (2), calculating each quality attribute of a recognition chain as a result of quality aggregation (3), filtering recognition chains whose aggregated quality cannot fulfil the quality requirements of the recognition goal (4), selection of best fitting recognition chain after optimization (5). Algorithm 1 returns the best recognition chain after the optimization process. The details of each part in this process are provided in the subsections of this section.

Constructing recognition chains (1) and filtering the individuals elements in the recognition chain based on recognition goal requirements (2) are presented in Section 4.2.1. Calculating

Algorithm 1 Selection and Optimization of a Recognition Chain

Require: RG, $serviceDirectory$
1: $QReq \leftarrow RG$.getQualityRequirements()
2: $availableServices \leftarrow$ loadServices($serviceDirectory$)
3: $RCList \leftarrow$ construct($RG, availableServices$)
4: $RCListFiltered \leftarrow$ filter($RCList, QReq$)
5: **for** each RC $rc_i \in RCListFiltered$ **do**
6: $Qprop_i \leftarrow$ calculateEachAggregatedQualityProperty($rc_i, QReq$)
7: $rc_i \leftarrow rc_i$.setQualityProp_i($Qprop_i$)
8: **end for**
9: $RCListFiltered \leftarrow$ filter($RCListFiltered, QReq$)
10: $RCListRanked \leftarrow$ optimize($RCListFiltered$,QReq)
11: return $RCListRanked$.getFirst()

aggregated quality for recognition chain (3) and filtering based on this value against recognition goal (4) are presented in Section 4.2.2. Finally, optimal selection of recognition chain by using quality optimization (5) is presented in Section 4.2.3.

4.2.1 Recognition Chain Construction and Filtering

In order to create recognition chains from available sensors and processing functions, this problem is mapped to a service composition problem as the elements are defined as a service in the models. One method to solve service composition problems is using a classical AI backward-chaining [113]. Backward chaining starts with the goal state and searches backwards that provide the required inputs until no more unfulfiled input remains.

In order to construct recognition chains from available services, AI backward-chaining is used by extending the work by Chen et al. [114] to include the quality restrictions in each state of the chaining process. Fig. 4.2 illustrates the general model for composition with quality restrictions. The quality restrictions come from the recognition goal definition (e.g. $Qreq$). While constructing the plans we use the quality requirements retrieved from the request of an application to eliminate the services in the chains which do not satisfy the quality requirements.

As can be seen in Fig. 4.2, there are 5 different services available with the different input and output and quality criteria at the bottom of the figure. In order to reach the goal of (a->d) by not considering any quality as stated in [114], the backward search starts to solve the sub-goals by finding the services going to the output of the goal state (S1 and S4). Partial results to the sub-goals are found on the first iteration. Then the search continues to satisfy the output of the services found in the first iteration. Partial results to this sub-goal found on

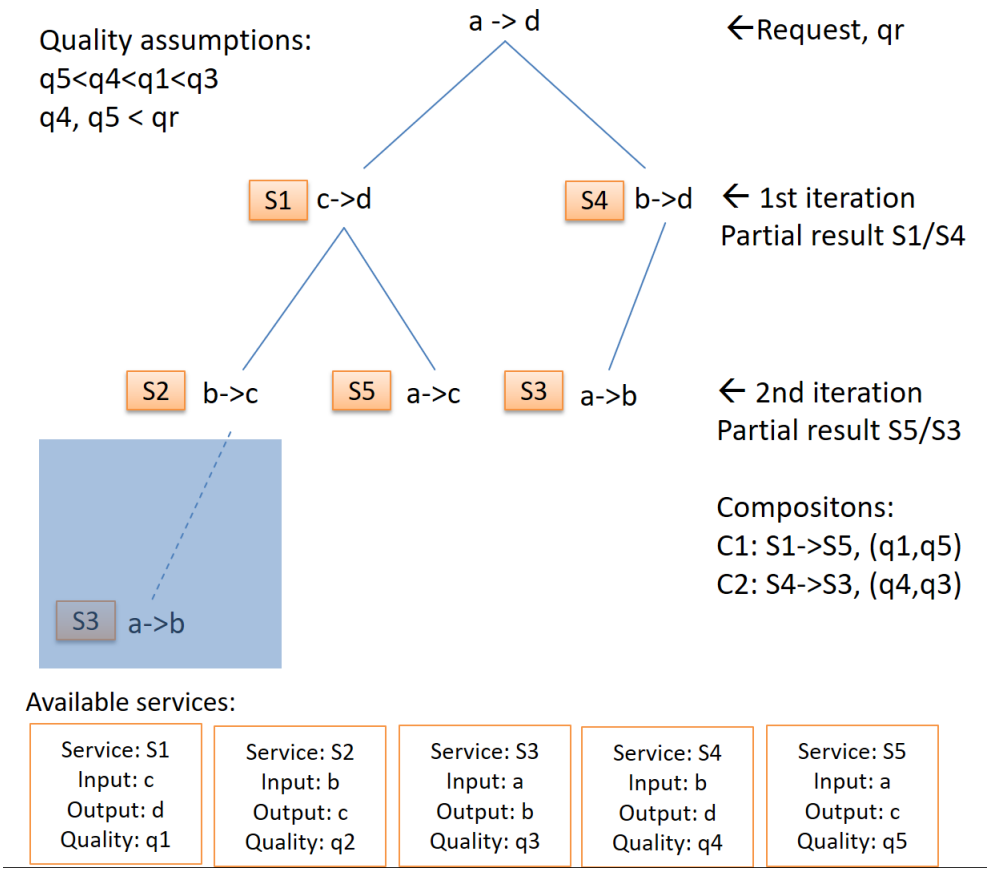


Figure 4.2: General backward chaining model with qualities (extended from [114], published in [110, 111])

the services S2, S5, and S3. In the second iteration, S5 and S3 already satisfy the given goal state with the resultant compositions C1 and C2 {S1 ->S5} or {S4 ->S3} respectively.

The extension is done by including the quality restrictions by assuming different qualities for services as depicted in Fig. 4.2 upper left-corner. Based on this assumption q4 and q5 cannot satisfy the requested quality for the goal which makes the found composites obsolete. Therefore, the back-chaining needs to continue until to find a solution of the sub-goal which also satisfy the quality requirement. Therefore, the last iterations state is saved to continue in order to find a satisfying service for the sub-goals. In the third iteration, S3 is found to satisfy the sub-goal and as its input is a, therefore the overall goal is also satisfied with the constructed composition S1 ->S2 ->S3. In the chain construction, the services which cannot fulfil the recognition goal requirement are eliminated. However, when there is no violation of requirements, the constructed chains are kept to further optimize based on overall chain quality. The overall chain quality calculation is provided in Section 4.2.2.

Within the guidance of extended backward chaining, to create a recognition chain, available data sources and processing method services in the distributed service repository can be searched. When any service violates the quality requirement from the recognition goal, it is eliminated and the chain is stopped to go further iteration from this branch. If there is no violation, the constructed chains with the iteration order can be kept to further calculate overall chain quality presented in Section 4.2.2.

4.2.2 Quality Aggregation and Filtering

In order to select the best recognition chain amongst the ones created as a result of backward chaining explained in Section 4.2.1, it is necessary to first calculate the quality of the overall chain (*Qeval* in recognition chain description). For this purpose, each quality parameter in the recognition goal (*Qreq*) should be iterated and based on the type of the quality parameter, related aggregation method, a direction of optimization and minimum and maximum boundaries of the parameter should be applied. For example, for latency in a chain of services can be aggregated by using a SUM operator to sum of all consisting services in the chain whereas for the accuracy of information MIN operator should be applied for the aggregation of different accuracy value of information to serve as a boundary. A direction of a quality parameter indicates to the favour of smaller or larger values. In the proposed quality description, negative direction indicates smaller values are favourable. Maximum and minimum boundaries from the

recognition goal description in this phase should be also taken into account for the calculation. Although single services do not violate the boundaries as a result of extended backward chaining process presented in the previous section, the aggregated quality value of a recognition chain should remain within the boundaries.

Algorithm 2 shows a quality calculation algorithm for a recognition chain based on each quality attribute as a result of aggregation. Algorithm 2 returns the aggregated quality values for each quality parameter in the recognition chain. As a result, a list of quality properties for a recognition chain including related id and quality value is found and assigned to the recognition chain quality property ($rcQProp$).

Algorithm 2 Quality Calculation for a Recognition Chain

Require: RC, RG
1: $QReq \leftarrow RG$.getQualityRequirements()
2: $SSer \leftarrow RC$.getServices();
3: $Qprop \leftarrow$ new List<>
4: **for** each quality requirement $qr_i \in QReq$ **do**
5: $qrID \leftarrow qr_i$.getID()
6: $qrType \leftarrow qr_i$.getAggregationType()
7: $qrMax \leftarrow qr_i$.getMinimum()
8: $qrMin \leftarrow qr_i$.getMaximum()
9: **for** each service $s_j \in SSer$ **do**
10: $sQProps \leftarrow s_j$.getQualityProperties()
11: $qProp \leftarrow 0.0$
12: **for** each prop $p_k \in sQProps$ **do**
13: $pID \leftarrow p_k$.getID()
14: **if** pID equals $qrID$ **then**
15: $pValue \leftarrow p_k$.getValue()
16: switch ($qrType$)
17: case: "SUM"
18: $qProp \mathrel{+}= pValue$ break
19: case: "MAX"
20: $qProp = MAX${qProp,pValue} break
21: case: "MIN"
22: $qProp = MIN${qProp,pValue} break
23: case: "MULT"
24: $qProp \mathrel{*}= pValue$ break
25: **end if**
26: **end for**
27: **end for**
28: **if** $(qProp \neq 0.0) \wedge (qProp \leq qrMax) \wedge (qProp \geq qrMin)$ **then**
29: $rcQProp \leftarrow qProp, pID$
30: Set_Qprop.add($rcQProp$);
31: **end if**
32: **end for**
33: **return** Set_Qprop

After each quality value is calculated separately for a recognition chain, they are checked against the boundaries of the $Qreq$. If there is a violation of any aggregated quality property of a recognition chain based on the recognition goal, this recognition chain is discarded. If there is no violation of any quality parameter in the aggregated values, the resultant quality values of recognition chains are kept for further optimized to use different quality properties in combination to select the best recognition chain. This process in explained in Section 4.2.3.

4.2.3 Quality-Optimal Selection

In order to use different quality values of recognition chains in a single optimization function, the values should be used independent of their metric. This requires to normalize the values in an interval preferably between 0 and 1. The main problem is having unbounded values for some of the quality properties of the services. If there is no requirement from recognition goal for this property, the boundaries of this value should be found based on the observed values. For this purpose, different statistical techniques (tanh, sigmoid, z-score, logistic regression, pnorm) can be used to find out which one is suitable for the observed values for each different quality property to find a normally distributed scale between 0 and 1. The behaviour of how different techniques work on the observed values for normalization is depicted in Figure 4.3.

As can be seen from Figure 4.3, except tanh and softmax, the other methods provide close to equal distribution for the normalization. For each observed quality variable, different technique is used fitting the with the related sample mean and standard deviation calculated based on the existing observed values for each quality property. After the normalization phase, the resultant quality values can be used in a weighted sum of different quality parameters in an optimization function to calculate the overall quality of a recognition chain.

The algorithm decides for an optimum recognition chain to select based on the recognition goal. The mathematical formulation is given as a Linear Programming optimization problem. The constraint for the objective function can be formulated as maximization function as defined in Equation 4.1.

$$\underset{rc}{\text{maximize}} \quad \sum_{i=1}^{n} w_i x_i(rc), \quad where \sum_{i=1}^{n} w_i(rc) = 1, \quad 0 \leq x_i(rc) \leq 1 \tag{4.1}$$

This function assumes that there are quality characteristics x_0 to x_n are normalized values which can be derived from a recognition chain rc after the normalization and weights for these quality characteristics w_0 to w_n. The goal is to find a recognition chain that maximizes

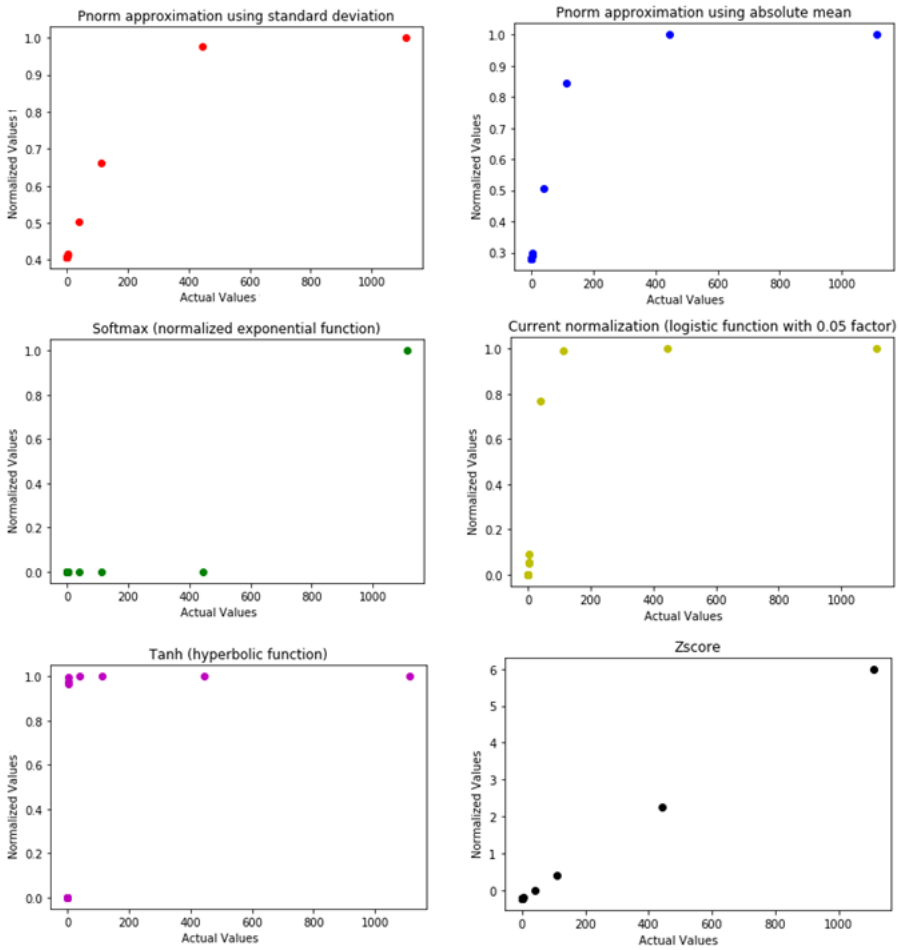


Figure 4.3: Different statistical methods for normalization

the weighted sum of these quality characteristics. In this work, Linear Programming (LP) is applied to solve the equation in order to find the optimal values for each quality parameter in the optimization function.

After finding the optimal values, those values are used to calculate the distance for each recognition chain from the optimal values. The quality values are summed up for each separate distance and ranked the recognition chains based on the minimum distance from the optimal value.

$$\underset{rcdist}{\text{minimize}} \quad \sum_{i=1}^{n} w_i(xopt - x_i(rc)). \tag{4.2}$$

where $xopt$ is the optimal value for each quality property found as a result of solving LP in the Equation 4.1. As a result of Equation 4.2 to calculate the distance, a ranked list of optimal recognition chains is created.

4.3 Summary

In this Chapter, the models and methods are presented to construct optimal recognition chains from available data sources for a recognition goal. The data sources (e.g. sensors) and processing functions are modelled as a service by using the service model proposed. The presented sections addressed the RQ1, RQ2 and RQ3 (see Section 2.3) to increase the quality of context recognition by finding the optimal recognition chain from the available services. The remaining part of the approach is making the selection process adaptive to the changes in the sensing infrastructure. In Chapter 5, the details about the adaptation approach to the changing conditions in the selection process by using feedback loops is provided.

5

Adaptation of Optimal Selection

While developing context-aware applications, there may be uncertainty with respect to the available data sources. Applications that are developed to a fixed set of data sources may not be flexible enough, to adapt to change in the sensing environment such as sensor disappearance or degradation. As mentioned in 3.2.4, Opportunity as a reference framework for human activity and context recognition, provides adaptability to the changing conditions through black-box components which makes the extension of the adaptation logic to include further quality criteria harder. In order to provide external adaptation, the applicability of different methods and architectures using feedback loops are shown in this Chapter. Section 5.1 presents simple MAPE-K loop to map the responsible components of Opportunity for adaptation. Then in Section 5.2, it is further extended for the separation of adaptation in Opportunity by using DYNAMICO a reference model for implementing self-adaptive systems. Section 5.3, presents the usage of runtime models for the implementation of external adaptation logic for the optimal recognition chain selection. The parts of this Chapter are published in the author's publications [111] and [115].

5.1 Application of MAPE-K

In this section, the applicability of a single MAPE-K feedback loop to opportunistic sensing is presented by applying such a loop for the adaptations in opportunistic sensing and then discussing the differences to the implementation in the Opportunity Framework.

In this application of the MAPE-K feedback loop the managed software system is the recognition chain and the feedback loop is responsible for dynamic configuration and autonomous evolution of the recognition chain based on the recognition goal. This combination is depicted in Fig. 5.1. This feedback loop has the same input (recognition goal, information about the recognition chain) and output (changes in the recognition chain) as the components for dynamic configuration and autonomous evolution of the Opportunity Framework and thus is compatible from an interface point of view. The phases of the feedback loop and the specific information they communicate in opportunistic sensing are discussed below.

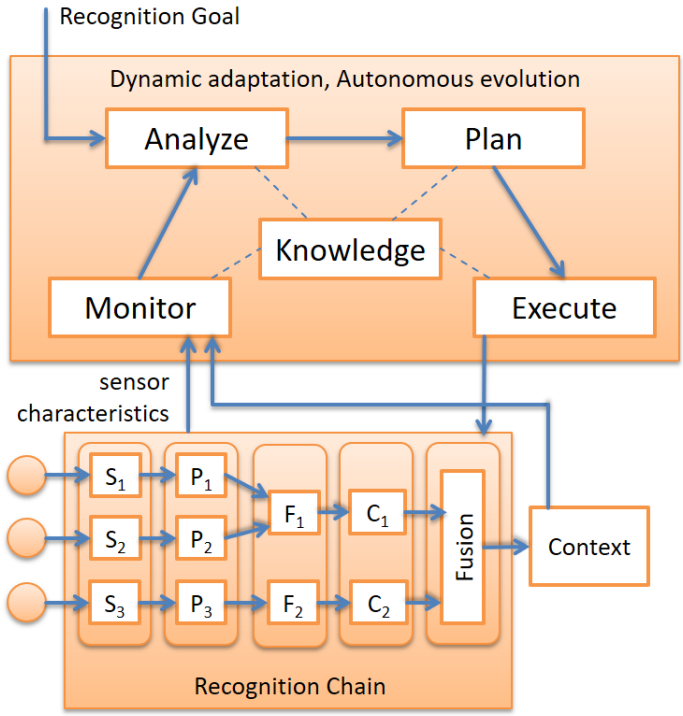


Figure 5.1: MAPE-K feedback loop in the Opportunity Framework for dynamic adaptation and autonomous evolution

During the *Monitor* phase the feedback loop collects information about the recognition chain. A part of this information concerns the properties of sensors and their signals. Events that are observed here are changes in sensor characteristics (e.g., degradation of the accuracy or change in frequency) as well as changes in the availability of sensors (e.g., sensor appearance and disappearance). In addition, this phase monitors the output of recognition chains.

The *Analyze* phase processes this information and derives information that can be used to decide whether the recognition chain needs to be updated. An update is required if the analyse phase detects that a recognition chain does not function any more or if the data it

derives indicates an opportunity to improve a recognition chain. A recognition chain ceases to function if one of its sensors disappears of if its sensors or the intermediate processing methods degrade to a point where the required information cannot be detected with the desired quality any more. An improvement opportunity is a situation in which new sensors appear or changes in the quality of existing sensors indicate that a recognition chain with better quality might be possible. The quality of a recognition chain can also be judged by monitoring information about the output of recognition chains, e.g., via comparison it to a benchmark recognition chain. Finally, the analysis phase should also take into account changes in recognition goals. Those changes concern new / removed recognition goals as well as changes in properties of existing recognition goals (e.g., a change in the required accuracy or tolerable costs).

If the *Analyze* phase determines that the recognition chain should be updated the *Plan* phase is responsible for finding an optimal recognition chain with respect to the recognition goal. The plan can change most aspects of the recognition chain. It may change the methods used for pre-processing (P), feature extraction (F) and classification (C) as well as the fusion methods to derive context information. The signals (S) represent the direct output signals of the sensors and thus cannot be changed by the plan. Adapting these individual processing functions enables the autonomous evolution of recognition chains. In addition, the adaptation may also compose a new recognition chain using other sensors and other processing methods. This facilitates the dynamic configuration of the recognition chain.

The *Execute* phase is responsible for enacting the changes to the recognition chains planned by the Plan phase. These plans, as well as the results of monitoring and analysis of this and previous cycles of the feedback loop are stored in the *Knowledge* component.

Since the dynamic configuration and autonomous evolution components in the Opportunity Framework also facilitate these adaptations they likely contain a decision process that is similar to the described MAPE-K loop, i.e., they collect information, analyse it and plan on the analysed information and enact the planned changes. However, this process is hidden in black-box components. As argued above distinguishing between these phases and making the activities in each phase explicit can help to compare and combine this framework with other approaches. E.g., the transfer learning approach by Van Kasteren et al. aims to use known recognition chains to learn new ones [116]. This could be applied in the *Plan* phase to find new recognition chains.

In the Opportunity Framework dynamic adaptation and autonomous evolution are treated as separate components. Although it is possible to handle both in a single feedback loop, as indicated above, proposed feedback loop handles two adaptations that are conceptually different. While autonomous evolution adapts individual data-processing methods in the recognition chain the dynamic configuration adapts the composition of the whole recognition chain. To enable a separation of concerns in these situations several authors in the autonomous computing community advocate the use of multiple feedback loops. In the Opportunity Framework separate loops for autonomous evolution and dynamic configuration are conceptually cleaner than a general feedback loop that merges these aspects. The three interconnected feedback loops defined in the DYNAMICO model could provide a better separation of concerns in this respect. This part is presented in Section 5.2.

5.2 Application of DYNAMICO

In this section, the DYNAMICO model is compared with opportunistic sensing. The monitoring feedback loop in the DYNAMICO model has the same purpose as opportunistic sensing: it enables an adaptive infrastructure for sensing information. Thus, in the first part of this section, it is investigated whether the Opportunity Framework could be used as implementation of the monitoring feedback loop. As stated in the Section 5.1, multiple MAPE-K loops may be required to correctly reflect the separation of adaptation concerns in the Opportunity Framework. The DYNAMICO model proposes three feedback loops that cover three adaptation concerns in a variety of adaptive systems. This distinction could provide a better conceptualization of the adaptations in the Opportunity Framework than a single feedback loop does. To test this proposal, the second part of this section applies these three feedback loops to the Opportunity Framework.

To compare opportunistic sensing with the monitoring feedback loop first the input and output of the monitoring feedback loop is matched with the input and output of the opportunistic sensing component in the Opportunity Framework. The result is depicted in Figure 5.2.

The monitoring feedback loop is configured via reference input that specifies the context information to be detected and the quality requirements for the detection. In the Opportunity Framework, recognition goal plays the role of reference input. It specifies which information to

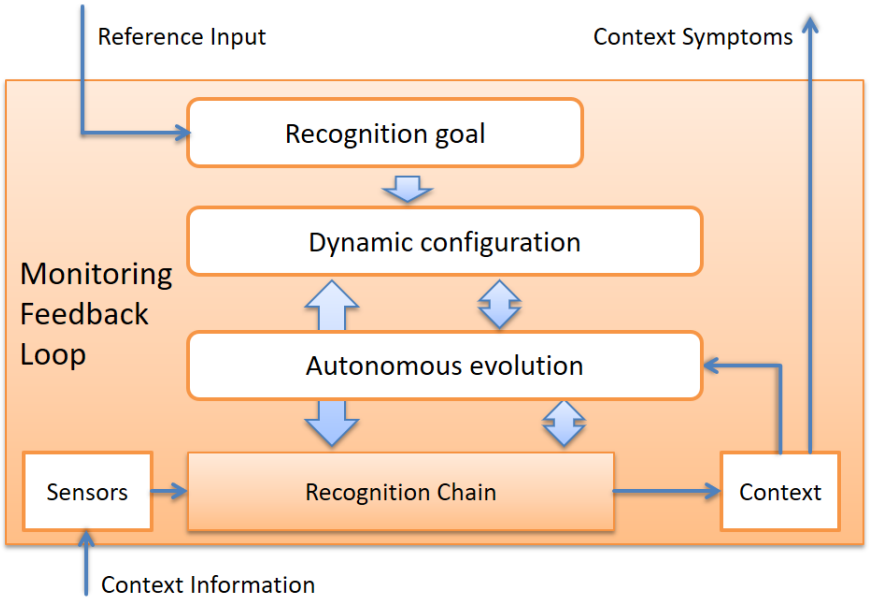


Figure 5.2: Opportunistic Sensing as implementation of the Monitoring Feedback Loop of DYNAMICO

detect. In the Opportunity Framework the quality criteria are implicit – the accuracy of the recognition is optimized [2].

Based on the recognition goal the monitoring feedback loop is responsible for detecting the required information (context symptoms). In the Opportunity Framework this is the purpose of the recognition chain. The Opportunity Framework does not discriminate between context symptoms for the adaptation or objective feedback loop. However, this is only a difference in the recipient of information.

The monitoring feedback loop receives context information from external as well as internal components. In the Opportunity Framework this information is derived from a set of sensors, each of which provides sensor signals as input for the recognition chain. The Opportunity Framework does not discriminate between internal or external information. However, both can be derived via sensors.

The Opportunity Framework and the monitoring feedback loop are similar enough in their input, output and general functionality to enable an application of opportunistic sensing for implementing a monitoring feedback loop. However, one of the goals of the DYNAMICO model is to explicitly use MAPE-K feedback loops for frequently appearing adaptation concerns and to define their interactions. A direct application of the Opportunity Framework thus would represent a step backwards as one feedback loop is substituted by the black-box adaptation

components in the Opportunity Framework. The explicit inclusion of feedback loops into the architecture Opportunity Framework could remedy this.

As discussed in Section 5.1, dynamic configuration and autonomous evolution in the Opportunity Framework are two different adaptation concerns that should also be separated in their adaptation feedback loops. While the dynamic configuration is responsible for assembling and updating recognition chains out of data-processing methods, the autonomous evolution is responsible for the reconfiguration of the data-processing methods (sensors, pre-processing, feature extraction, classification, domain knowledge) themselves.

If the recognition chain is considered to be the adapted software system, as it was in Section 5.1, then dynamic configuration can be seen on the level of the DYNAMICO adaptation feedback loop. It adapts the recognition chain directly based on the provided recognition goal and information about the chain and the potential processing methods.

Autonomous evolution can be argued to be on the level of the monitoring feedback loop. However, it does not directly adapt a monitoring infrastructure. Instead it adapts individual methods in the recognition chain. However, for this adaptation it needs to collect, analyse and derive information about the recognition chain and individual processing methods in the chain. This information is necessary for the dynamic configuration and can be seen as the context symptoms provided to the adaptation feedback loop. Thus, from the view of the dynamic reconfiguration the autonomous evolution serves the same purpose as the monitoring feedback loop. This indication can be seen in Fig. 5.3.

The third feedback loop from the DYNAMICO model is not a central component in the architecture of the Opportunity Framework. In the scope of opportunistic sensing the objectives feedback loop corresponds to a dynamic adaptation of recognition goals that have been given as reference control objectives. The recognition goal is only received in the respective component but not adapted. However, Kurz et al. [2] mention that if a high level recognition goal in the Opportunity Framework cannot be met, it can be refined into sub-goals, for which detection may be possible. However, they do not reflect this in their architecture explicitly. Following the ideas of the DYNAMICO model a dedicated objectives feedback loop could make the management of goals explicit. It can be responsible for selecting between different decompositions of complex recognition goals and dealing with situations where a recognition goal cannot be fulfilled. It could also enable the handling of more complex quality requirements for the detected information. E.g., the costs, frequency or latency of detected information

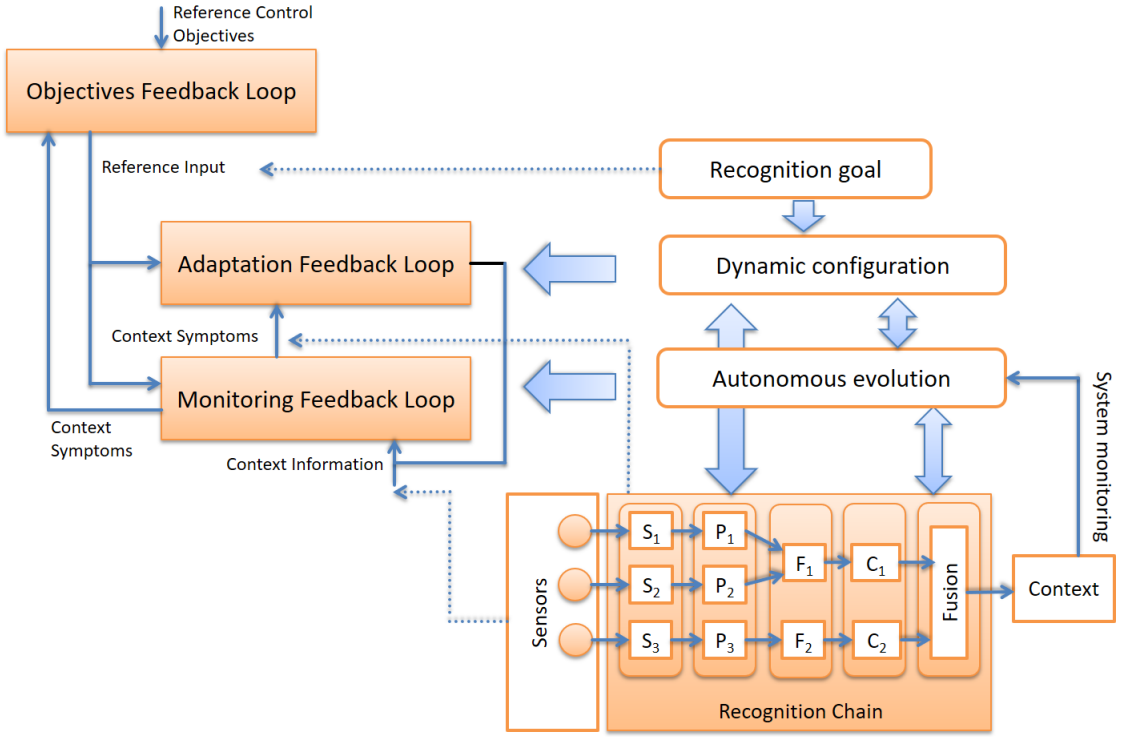


Figure 5.3: Opportunistic Sensing as implementation of the other feedback loops of DYNAMICO

may also be relevant quality of service requirements for recognition goals. The Opportunity Framework mainly deals with accuracy of the detected information and thus abstracts from issues of conflicts or trade-offs between multiple quality of service requirements. A dedicated objectives feedback loop would be a suitable place to deal with those issues.

Although the autonomous evolution only roughly corresponds to the monitoring feedback loop the three levels of DYNAMICO provide a cleaner separation of concerns between autonomous evolution and dynamic configuration than the single feedback loop discussed in Section 5.1 and also specify the interactions between these feedback loops. The application of DYNAMICO also provided the insight that a dedicated feedback loop for recognition goals could provide a conceptually clean way to reflect the decomposition of these goals and is extensible to more complex recognition goals involving multiple quality of service attributes.

One additional point that may be considered is to use more than one feedback loop for autonomous evolution. Methods in the recognition chain represent specific data-processing methods and algorithms. Adaptation practices for these algorithms can range from simple configuration parameters to more complex tasks like the selection of data sets for classification and learning purposes. Since these tasks are highly dependent on the individual algorithms it

may make sense to handle individual data-processing methods or similar classes of methods in their own dedicated feedback loop instead of having a general feedback loop for all methods.

As proposed in [115], the adaptability in opportunistic sensing can be implemented by using multiple feedback loops of the DYNAMICO model as opportunistic sensing is essentially an adaptation of monitoring infrastructure according to a recognition goal.

5.3 Application of Runtime Models

In this section, the usage of runtime models are presented for the implementation of external adaptation logic for the optimal recognition chain selection as a part of DYNAMICO. To structure the information used in the adaptation process, it will be first placed in the scope of the main adaptation loop. The purpose of this adaptation loop is to update the monitoring system so it fulfils some requirements in terms of information to be monitored and its quality. This corresponds to the monitoring feedback loop in the DYNAMICO reference framework [107].

This feedback loop is depicted in Fig. 5.4. The adaptive system consists of the deployed recognition chains. In this work, recognition chains consist of services that represent sensors and processing functions. Processing functions can be used to process information, e.g., to remove noise, normalize, or to derive high-level information from low-level sensor values. The deployed recognition chains also contain connections between sensors and processing functions that determine which information is used as input for which processing function. The parts of recognition chains are arranged in a pipeline. Thus, the deployed recognition chain represents an information processing chain that results in the information to be detected.

To further structure the explanations in this section, the extension of the MAPE-K loop proposed in the EUREMA framework [109] is chosen to use. In this extension, the knowledge base is further subdivided into three models:

- Evaluation Model: representing the goals to be reached via adaptation.
- Change Model: representing the options available for adaptations.
- Reflection Model: containing information about the state of the adaptive system and its environment.

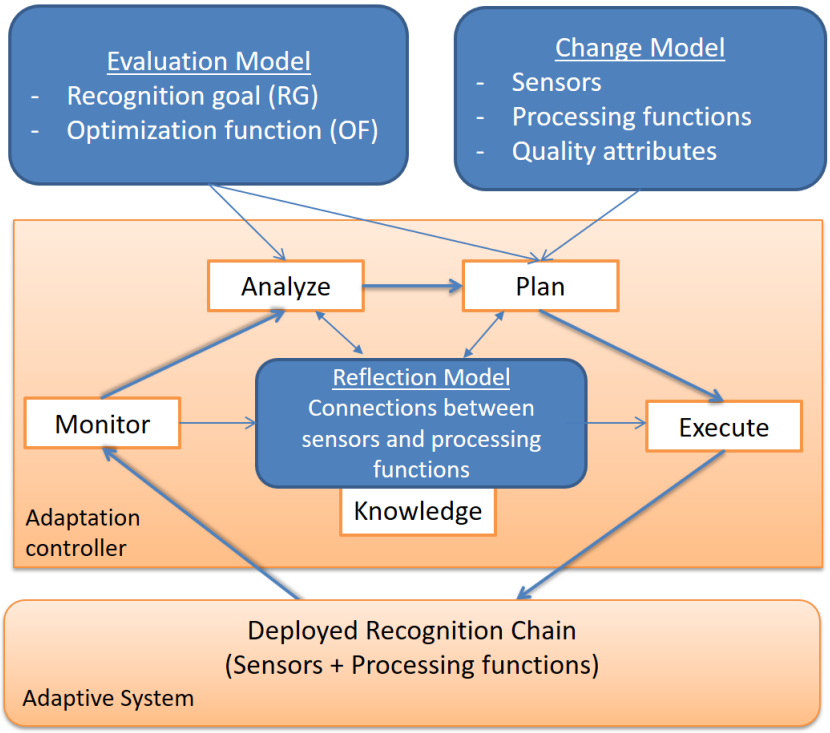


Figure 5.4: Models used for opportunistic sensing

Since this subdivision conveniently separates the knowledge about the adaptive system from the goals and options of the adaptation, it will be used to further structure descriptions presented in this section.

The evaluation model of opportunistic sensing covers the modelling of the recognition goal. This includes a definition of the required information and related objects. The evaluation model also includes the optimization function. This function is derived from different quality metrics and a trade-off between them. For this purpose, the recognition goal definition presented in Section 4.1.2.1 and optimization function presented in Section 4.2.3 in Equation 4.1 are used.

The change model consists of the descriptions of sensors and processing functions. These descriptions also contain annotations of their quality attributes. For this purpose, the service descriptions of recognition chain elements as presented in Section 4.1.2.4 based on the service model described in Section 4.1.2.2 is used. This model affects the planning phase of the feedback loop by defining the elements which are available during the construction of the recognition chain.

The reflection model covers the representation of connections between sensors and processing functions. It aims to serve as knowledge between the phases of the feedback loop. For this purpose, the recognition chain description presented in Section 4.1.2.3 is used. It contains the

current recognition chain as well as information monitored from the currently deployed chain, such as the observed performance of the used sensors.

Based on those models, phases of the MAPE-K loop are implemented as follows. The monitor phase is responsible to retrieve the information from deployed recognition chains with quality properties in case of changes in the components of the recognition chain. The analysis phase observes the information about the deployed recognition chains in the context of the goals described in the evaluation model to decide whether a recalculation of recognition chains is promising.

The planning phase constructs a new recognition chain. This is dependent on the quality of the components as well as their connection. Another parameter for the planning phase is finding the optimal recognition chain using the optimization function contained in the evaluation model. After the planning phase, the execute phase is responsible to realize the plan(e.g. recognition chain in this case) by deploying/un-deploying the components and establishing communication channels between them. This can include start/stop/register of sensors and processing functions and their services.

5.4 Summary

This section provides the proposed methods to enable the adaption in opportunistic sensing for autonomous recognition chain selection. It is shown that adaptability in opportunistic sensing can be implemented by using multiple feedback loops of the DYNAMICO model as opportunistic sensing is essentially an adaptation of monitoring infrastructure according to a recognition goal [115]. For this purpose, runtime models can be used to map different phases of MAPE-K for the adaptation [110]. The approach presented in this section is focused on run-time models of the available data sources and data processing methods that can be used to assemble context recognition chains and estimate their quality. Chapter 6 defines a case study to evaluate the proposed models and mechanisms are provided.

6

Case Study

In this Chapter, a case study is defined within the scope of a research project called *Diginet-PS*[1] in order to evaluate the proposed models and methods presented in Chapter 4 and Chapter 5. For this purpose, this project is briefly described first and an example use case from the autonomous driving domain is given in Section 6.1. Based on this use case, the architectural requirements for the implementation are provided in Section 6.2. The parts of this Chapter are published in the author's publications [110] and [117].

6.1 Domain Description and Running Example

The project Diginet-PS aims to create a test area in an urban environment for the validation of autonomous driving in conjunction with infrastructure along the streets of Berlin. The planned area serves as a test field to deploy and validate prototypes for different use cases in the scope of automated and networked driving. The main outcome of the project is to have a broader experience of autonomous driving under real-life traffic circumstances and to derive implications for further technology development and integration.

The test environment in Diginet-PS project is built around a 3.65 km long main street, which provides complex traffic situations with end-of-day traffic, government convoys, 2 large roundabouts, 11 traffic signalling systems for vehicles, bicycles, pedestrians, wheelchair users and complex parking space settings. An overview of the sensors integrated into this environment is depicted in Figure 6.1.

[1] DigiNet-PS project website: `http://diginet-ps.de`

Figure 6.1: Sensor Deployment Overview of DigiNet-PS

This test area is populated with sensors for parking space detection, environmental conditions (e.g., weather, pollution and emissions), traffic situation, light status and passers-by information (e.g. pedestrian/cyclist detection). The test infrastructure enables communication between sensors and vehicles via the Internet.

DigiNet-PS deviates from many classical activities for its philosophy of widening the visibility of vehicles by digitizing (i.e., by deploying the different types of sensors) the roads and consequently creating the perception of road segments in the extended roadside units. The sensory data captured through on-road deployed sensors may be fused for studying the correlations between different parameters of the data, processed with AI approaches, and construction of patterns for different decision making instances.

In order to apply the proposed approach, a running example for detecting traffic congestions in this testing area is considered. Thus, the recognition goal is to detect traffic congestions on a specified route. This information could help to reduce traffic jams by suggesting alternative routes or adapting speed before arriving at the congested street. For detecting traffic jams different types of recognition chains can be calculated. First, camera-based traffic analyser sensors can provide information about the number of passing vehicles and types of vehicles. Since these sensors are camera based they may suffer from inaccuracies, caused by occlusion or different luminosity values, e.g., under rainy weather conditions. One example of those situations can be seen in Figure 6.2 depicting that a problem on vehicle counter sensors providing no data between January, 9-10.

Even though the sensors would cover the same area from different directions of the road, they can provide different sensory data due to the placement or calibration. Due to communication problems, the sensors may become unavailable when this information is requested. Another problem is the number of passing vehicles in a specific location cannot provide information on traffic congestion alone. The data retrieved from sensors should be analyzed and compared

Figure 6.2: No data due to changes in the sensing infrastructure

with historical data to infer this information. Therefore, additional processing is necessary on top of the retrieved data. Additionally, the system could use information from, Traffic APIs, like HERE Maps[2] or TomTom Maps traffic APIs[3] which can e.g., provide traffic status information in Berlin. For external APIs, there can be a problem of retrieving no up-to-date data in a specific time when an application requires.

For the camera-based sensors, camera position and placement influences the quality of the image recognition (since some parts of the road may be partially covered by trees, or just further away and thus harder to recognize via image processing). Moving vehicles may change the quality provided by the camera even without the camera itself degrading. Additionally, in the running example, even if the accuracy of each camera is known and changes, the distance between the camera and the area to detect traffic instances may change. Traffic instances that are further away may be harder to detect since with increasing distance the size of the relevant part of the image captured by the camera shrinks and the relative information loss due to image resolution grows. This may lead to lower accuracy of a recognized event. If available, this may be compensated by using higher quality image recognition approaches to detect the situation, which is likely to be computationally more expensive, thus influencing the latency of detection. The quality estimation needs to cover different quality characteristics

[2]HERE Traffic API, `https://developer.here.com/documentation/traffic/topics/what-is.html`
[3]Tomtom Traffic API: `https://developer.tomtom.com/traffic-api`

in combination. In this scenario, the aim by using the proposed approach is to find the right recognition chain to retrieve information about traffic congestion (recognition goal) and update the recognition chain in case of any change caused by high mobility. This will enable dynamic context recognition in the testing environment. In order to enable dynamic context recognition by selecting adaptive optimal recognition chain, the architectural requirements for the implementation are presented in the next section.

6.2 Architectural Requirements

In this section, the architectural requirements based on the running example is provided to enable dynamic context recognition. Based on the running example presented in Section 6.1, the following architectural requirements (AR) are derived:

AR1- Deal with heterogeneity: The context recognition system should be able to deal with the fact that integrated sensors use different communication technologies, speak different protocols and use different data formats. In the running example, traffic analyser sensors and Google Maps provides REST APIs requiring to send HTTP request to their backend, whereas inertial sensors in the vehicle use the ETSI ITS-G5[4] protocol for vehicle-to-vehicle and vehicle-to-infrastructure communication.

AR2- Stream-based context recognition: The context recognition should not provide information once, but should be based on data streams that continually provide information. This way, applications can continually receive updates on the requested information.

AR3- Reflect runtime changes: Since the infrastructure of sensor systems is subject to change, it should be possible to update the recognition chain to reflect changes in the available sensors or their properties. In the scenario, a detection of traffic jams from on-vehicle inertial sensors may be accurate but is very volatile because vehicles tend to move to different locations. Consequently, the recognition chain needs to be calculated based on the currently available sensors and updated regularly to reflect changes.

AR4– Optimize quality: If there are multiple possible recognition chains then there is an opportunity for optimizing the recognition chain by selecting the best sensors. This requires mechanisms that are able to estimate the influence of the quality of sensors on the overall quality of the recognition chain and are able to select/update sensors if possible.

[4]EN 302 663 Intelligent Transport Systems (ITS); `http://www.etsi.org/deliver/etsi_en/302600_302699/302663/01.02.00_20/en_302663v010200a.pdf`

AR5– Decoupled calculation: Since the quality of a recognition chain is a concern, the calculation and update of recognition chains should be decoupled from the recognition chain itself. This means the quality of the recognition chain should not be influenced negatively by the fact that there is a mechanism that fulfils AR2 and AR3 by updating the recognition chain.

Based on those architectural requirements, an agent-based architecture is proposed in the scope of implementation. The details about this architecture and the running example implementation are provided in Chapter 7.

7

Implementation

In this Chapter, the implementation details about the proposed approach presented in Chapter 4 and Chapter 5 are provided in terms of architecture and specific technologies to enable selecting the best quality recognition chain and adaptation of it during run time. For this purpose, first the architecture for dynamic context recognition is proposed in Section 7.1 addressing the requirements mentioned in Section 6.2. The implementation of the proposed architecture is presented in Section 7.2. Section 7.3 presents the used ontologies to represent the data sources required based on the case study. Then, the implementation details of the running example are provided on the sample services from autonomous driving domain in Section 7.4. Section 7.5 provides information about implemented interfaces. The parts of this Chapter are published in the author's publications [111] and [117].

7.1 Architecture

In order to address the architectural requirements presented in Section 6.2, an agent-based dynamic context recognition architecture is proposed based on multiple abstraction and data processing layers. An overview of those layers can be seen in Fig. 7.1. It consists of an IoT Middleware, which integrates different sensors and provides their values via a unified Streaming Platform. The Data Integration layer is agent-based and relies on this event-driven Streaming Platform and is responsible for providing applications with the information requested based on the available sensor information.

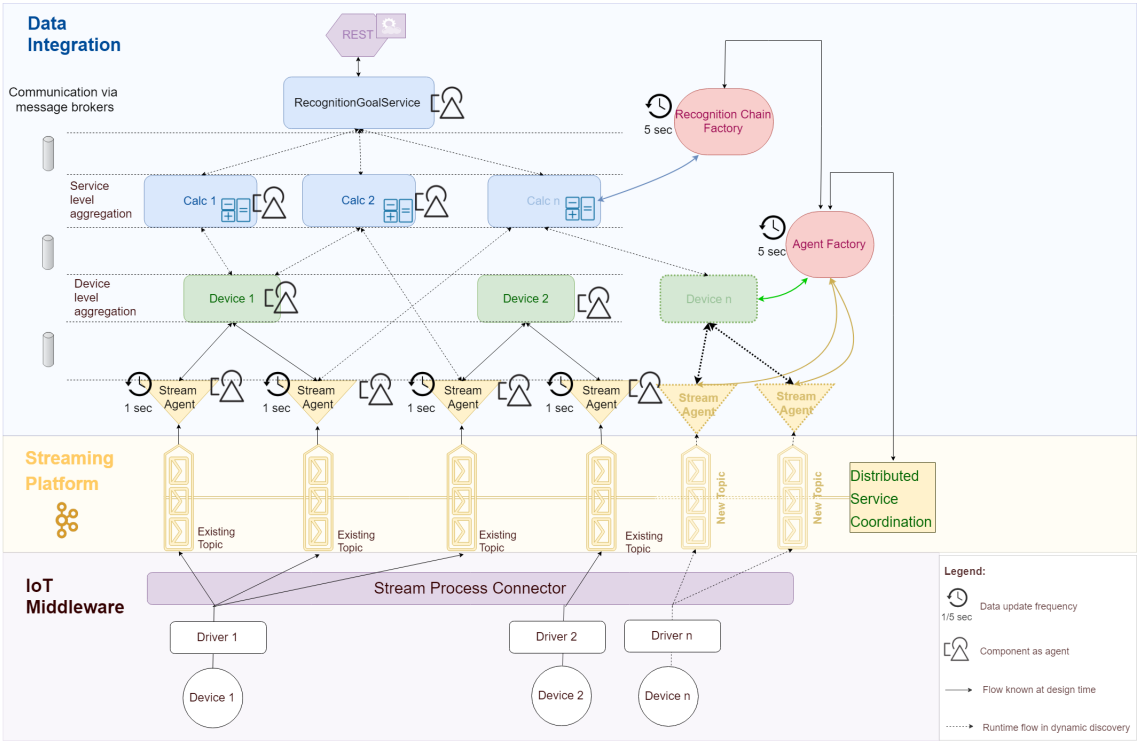


Figure 7.1: The agent-based Architecture of Data Integration Layer

The IoT Middleware is responsible for retrieving data from devices having one or more sensors attached to them and making this data available in a unified way. This directly relates to **AR1** (see Section 6.2). For this purpose, the IoT Middleware needs to be able to deal with different communication protocols and to convert data formats used by the sensors into a generalized representation used for publishing data in the Streaming Platform. It is assumed that the platform has technology-specific drivers that can fill this role. The idea is that for any new sensor type one driver needs to be implemented to integrate it into the platform. It is assumed that the metadata of each device is modelled in a so-called device profile. In this profile, each device is represented as one or more sensors attached to it and the properties that can be retrieved from those sensors. E.g., for traffic analyser sensors the related profile is called TrafficLane and specifies the availability of the following counters:

- crossingsCounter1Hour - Crossings counter of last hour, counting from now.
- crossingsCounter24Hours - Crossings counter of last 24 hours, counting from now.
- crossingsCounter5Minutes - Crossings counter of last 5 minutes

The Streaming Platform serves as a buffer between the IoT Middleware and the Data Integration layer in the form of streams that can be accessed via a publish/subscribe mechanism.

This allows asynchronous publishing and delivering of data between consumers and producers. Producers can publish data into a stream and consumers can subscribe to this stream to automatically get the published data. This mechanism is chosen to address **AR2** (see Section 6.2).

The main contribution starts with Data Integration layer which is organized in multiple sub-layers to reflect the proposed approach depicted in Chapter 4 and Chapter 5. The so-called data integration layer is implemented as an agent-based architecture where all components are agents and communication is done via service calls. The agent-based architecture enables to support capabilities like starting and stopping agents and connections between agents on the fly and to migrate agents to react to runtime changes for quality optimization.

Stream Agents form the lowest part of the Data Integration layer. The main task of these agents is to receive all new values from a certain topic which is executed every second. The execution frequency is configurable and can be adjusted to increase the frequency of information provision if required. Each Stream Agent represents separate sensor properties which are published by separate topics. The aim of modelling each sensor property instead of sending all properties once is to separate the data update for different properties as not every property needs to be updated frequently. This gives us the flexibility to adjust the frequency of the update for different sensor properties. Another advantage is it gives us the flexibility to combine different properties at runtime based on the requirements from applications. This is required as a part of the quality optimization expressed at **AR4** (see Section 6.2).

Above the Stream Agents, the architecture also allows device level aggregation to group the properties retrieved from the same device as depicted in green boxes in Figure 7.1. The main task of these agents in the device level is to combine all topics associated with one device. If there is an update on the values from Stream Agents, the Device Agents also receive the up-to-date values. These agents can be used when there is no need to have different update frequency for specific sensor properties.

On the device level, each agent exposes specific services related to the type of data that they can provide (traffic, light status, parking, weather, etc.). This also applies for the Stream Agents, although each Stream Agent only provides one type of data. In the proposed architecture, these services are modelled based on the device profile retrieved from the IoT Middleware via the device profile (e.g. TrafficLane). Both Device and Stream Agents are

enriched with the device profile information which could be used to describe the domain or the type of the information that they can provide.

To fulfil the requirements for enabling runtime changes (**AR3** in Section 6.2) and quality optimization (**AR4** in Section 6.2) of the recognition chain, two additional agents are proposed: the Agent Factory and the Recognition Chain Factory. The task of these agents is to facilitate the necessary changes in the recognition chain.

The Agent Factory is depicted in Figure 7.1 on the right-hand side. Its main task is to reflect changes in the sensor infrastructure (appearance or disappearance) in the respective Stream and Device Agents. This component is executed regularly in a configurable update frequency to compare available topics in the Streaming Platform with the Stream and Device Agents and start/stop these agents as necessary. This applies also to the device level agents.

The Recognition Chain Factory is responsible for searching available services by communicating to the Agent Factory on the stream level as well as on the device level. As depicted in Figure 7.1, the calculation of recognition chains are designed and developed as services that can be aggregated from low-level agents. Recognition chains are calculated based on the services exposed by the available agents. In addition to the sensors, it is assumed that processing methods to derive high-level information from low-level sensor values are available as services. Those can be external service API calls, signal and information processing methods as well as machine learning and prediction techniques. Recognition chains are calculated based on those services. This calculation includes the quality parameters of the recognition chain. Those could include distance to the requested location, elapsed time after the last update of the observation from the data source, battery status of the sensor if available for the energy consumption. Currently, the quality parameters are calculated separately and added as a part of quality criteria to the recognition chain. The quality calculations for those criteria are updated when there is a new observation retrieved from the related data sources. The Recognition Chain Factory can be configured with a frequency which denotes how often it is executed.

New recognition chains can be deployed by starting a respective agent wrapping the recognition chain. They can be changed by changing the communication within the agents belonging to this recognition chain or adding/removing agents. This provides the necessary flexibility to change recognition chains to accommodate runtime changes (**AR3** in Section 6.2) and quality optimization (**AR4** in Section 6.2). Both the Agent Factory and Recognition

Chain Factory are agents themselves. This enables us to decouple their execution from the recognition chains to fulfil **AR5** (see Section 6.2).

7.2 Architecture Implementation

In this Section, the details for the architecture implementation are provided in terms of specific technologies. It is exemplified how the dynamic context recognition is used by illustrating it on a test user interface.

For the implementation, specific technologies for the IoT Middleware, Streaming Platform and Agent Platform are chosen. The technology choices and required extensions are described below. However, it should be pointed out that the architecture in Section 7.1 is technology independent and could also be implemented using other architectures.

As an IoT Middleware, IOLITE[1] is used which provides a unified platform for connecting different devices and sensors by dealing with the heterogeneity of communication and information protocols. IOLITE was initially developed for smart homes at DAI-Labor and had to be extended to be usable in the IoT environment. It provides the above-mentioned drivers for smart home devices. These had to be extended to add support for additional attributes (e.g., to use a geo-location based system for locating sensors rather than rooms and coordinates within a home). In addition, IOLITE had to be extended to forward the sensor data values to the Streaming Platform. The technology choice for IOLITE has been made because it already supports heterogeneous IoT devices. It also has been developed by our research institute and thus was the pragmatic choice for compatibility and extension.

As a Streaming Platform, Apache Kafka[2] is chosen as a distributed Streaming Platform which supports the publish/subscribe mechanism. Apache Kafka is an open-source distributed stream buffer system designed for high scalability and robustness. It supports managing the sensor data streams by using different "topics" in the Kafka system. It also uses Apache ZooKeeper[3] which supports developing and maintaining an open-source server with the highly reliable distributed coordination. However, it should also be pointed out that although Apache Kafka can be used to store data persistently in a specific amount of time, it is not primarily designed to be used as a persistent data storage meaning that it is not optimized for search queries in the default configuration.

[1] IOLITE, `http://iolite.de/`
[2] Apache Kafka, `https://kafka.apache.org/`
[3] Apache ZooKeeper, `https://zookeeper.apache.org/`

Stream Agents were implemented based on Kafka data receivers to retrieve data from separate topics forwarded by the IoT Middleware.

As an Agent Platform, JIAC V, a Java-based Intelligent Agent Framework[4] is used to implement the proposed agents. JIAC is bringing service-oriented and agent-oriented architectures together, which enables easy development of complex and scalable distributed systems [118]. For the purpose of using it in the architecture, the publish/subscribe mechanism from Apache Kafka had to be implemented by extending JIAC with a subscribe-notify wrapper. The class diagrams of the agent-based architecture implementation can be seen in Appendix B.

After the implementation of the architecture enabling the connection and communication of sensors to the system via agents, they need to be semantically described for device and service level aggregation (depicted in Figure 7.1). In Section 7.3, the semantic layer to represent data sources with the related ontologies are provided.

7.3 Semantic Layer

In this section, the ontologies used as a vocabulary are provided to represent data sources required to in the case study.

High-level relationships between the entries is created via Protege[5] open-source ontology editor and can be seen in Figure 7.2. The ontology description can be found in Appendix C.

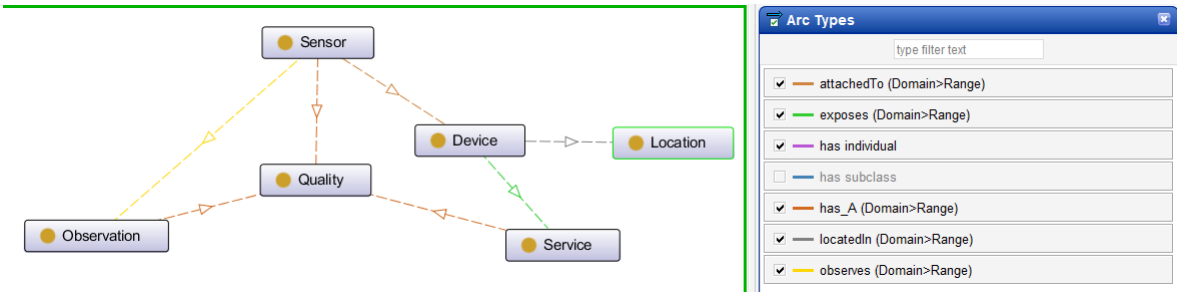


Figure 7.2: High-level relationships between entities

Section 4.1.2.4 proposed the classes and relationships for the data source model. Based on that. Figure 4.1 shows the modelled entities and relationships by using Protege and serialized by using OWL API[6].

[4]JIAC V (Java-based Intelligent Agent Componentware): `http://www.jiac.de/agent-frameworks/jiac-v/`
[5]Protege website: `https://protege.stanford.edu/`
[6]OWL API website: `https://github.com/owlcs/owlapi/`

Different than ontology presented in Section 4.1.2.4 for data source model, Figure 7.3 shows the ontologies for sensor deployment. The data is represented and stored in GraphDB a Semantic Graph Database compliant with W3C standards[7].

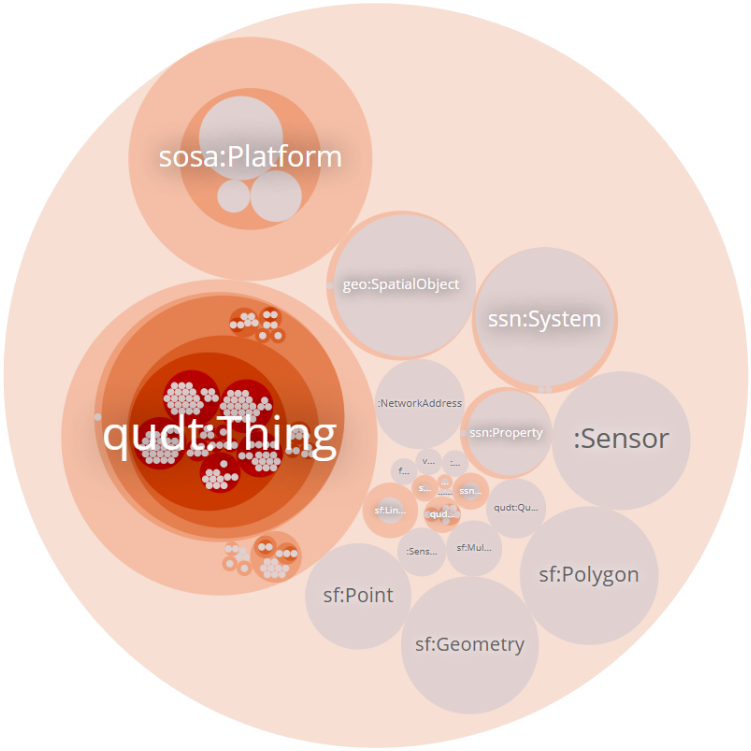


Figure 7.3: Class hierarchy used in Diginet data source deployment

The namespaces represents the following ontologies:

- sosa: Sensor, Observation, Sample, and Actuator Ontology by W3C (`http://www.w3.org/ns/sosa/`)
- ssn: Semantic Sensor Network Ontology by by W3C (`http://www.w3.org/ns/ssn/`)
- qudt: Units of Measure, Quantity Kinds, Dimensions and Types Ontology by QUDT.org, a member of W3C (`http://qudt.org/`)
- sf: Simple Features Access Ontology by OGC und ISO (`https://www.opengeospatial.org/standards/sfa`, `https://www.iso.org/standard/40114.html`)
- geo: Vocabulary for representing spatial-location by W3C interest group (`https://www.w3.org/2003/01/geo/`)

[7]GraphDB: `http://graphdb.ontotext.com/`

As can be seen, many of the vocabularies used in representing sensor deployment are the well-known ontologies by the standardization organizations. In Figure 7.4, the visual graph of deployed data source instances can be seen by using the relations from the used vocabularies.

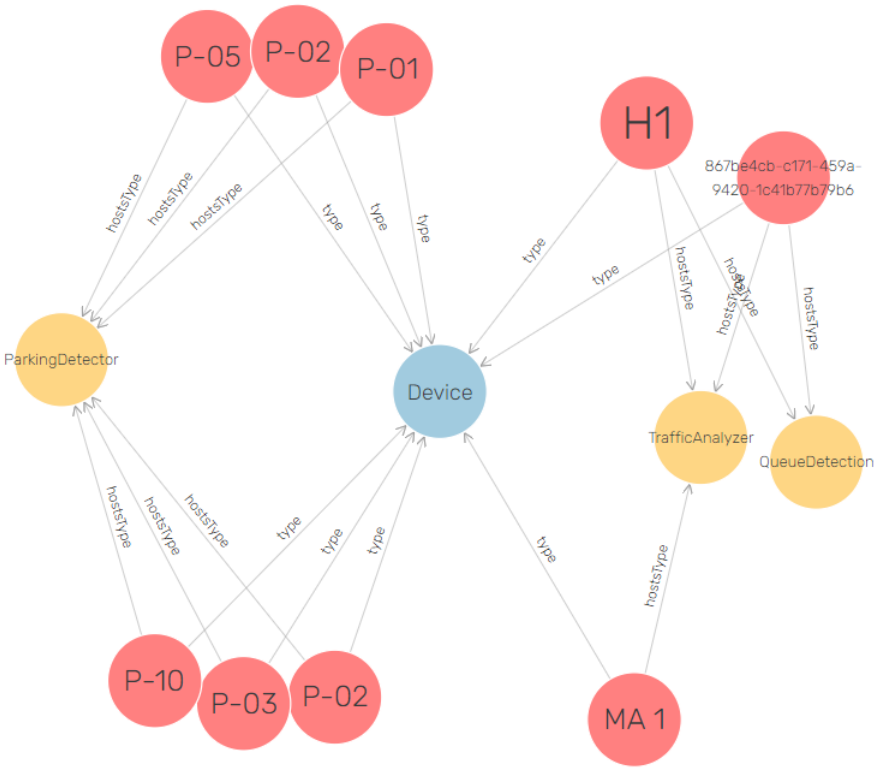


Figure 7.4: Visual graph for deployed data source instances in test road

Figure 7.4 shows the deployed devices with attached sensors in the autonomous driving testing road. Although modelling sensor data as service with an ontology as presented in Section 4.1.2.4 is a part of the approach, sensor deployment ontology is out of scope for this study. Sensor deployment ontology serves as a knowledge base for the instances of the recognition chain elements with the real sensors. The modelling of it is not discussed in the scope of this thesis. For further reference, it can be found at Diginet Project code repository by contacting from here [8].

7.4 Running Example Implementation

In order to handle runtime changes, runtime models are proposed as presented in Section 5.3. For this purpose, specific components from the architecture presented in Section 7.1 are used.

[8] `https://diginet-ps.de/corporate-creative-contact/`

In this section, the running example by using runtime models is detailed and implementation details are provided in Section 7.4.1. Then the information about setup and testing of the running example are provided in Section 7.4.2.

7.4.1 Using Runtime Models

In this section, the running example with the related runtime models are presented to enable dynamic context recognition for traffic congestion. The adaptive system is part of the control system of an autonomous vehicle. In addition, it is assumed that the vehicle can access external sensors (e.g., deployed on the road-side) in addition to its on-board sensors, to improve context detection. Thus, it has a need for sensor selection as the relevant sensors change with the position of the vehicle and the planned route.

In the running example, the aim is the detection of traffic congestions on a planned route. The model of the running example is depicted in Fig. 7.5. The adaptive monitoring infrastructure of the running example is responsible for finding and maintaining recognition chains that can provide this information.

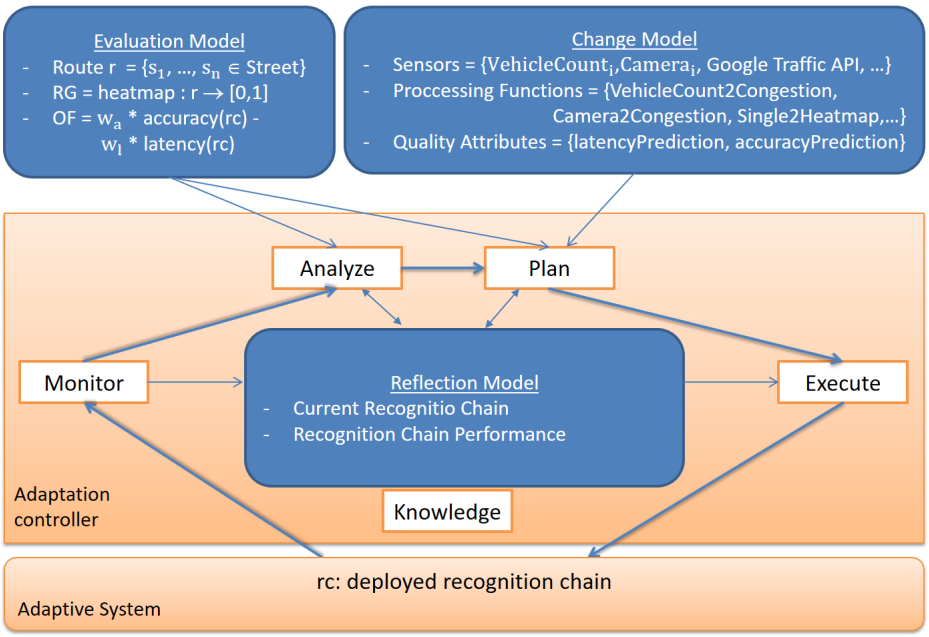


Figure 7.5: Running example

The evaluation model contains a route r along which traffic congestions should be detected. For the sake of simplicity, r represents a set of streets. The recognition goal specifies that the target of the recognition process is a heat-map of traffic congestion risk, represented as a mapping from the streets in r to real numbers between 0 and 1. This recognition goal is

accompanied by an objective function that is a weighted sum of two quality requirements the latency of the resulting recognition chain (i.e., the time it takes to detect a change in traffic jam risk) and the accuracy of the detected heat-map. The objective function is maximized, meaning that accuracy is maximized and latency is minimized. This may require a trade-off if the more accurate detection also takes longer.

The change model describes the sensors available to calculate traffic congestions from. These include roadside information sources, like vehicle counters and cameras, and online information sources from external online map APIs. The processing functions include methods to derive traffic congestion information from low-level sensors like vehicle counters ($VehicleCount2Congestion$) and cameras ($Camera2Congestion$). It also contains a function $Single2Heatmap$ that merges a set of processing functions for single streets into the heat-map for all streets on the route specified in the recognition goal. The quality attributes are given by two prediction functions that are able to predict latency and accuracy for any sensor and processing function. This information is partially derived from the $RecognitionChainPerformance$ information in the reflection model, which has been collected from running recognition chains.

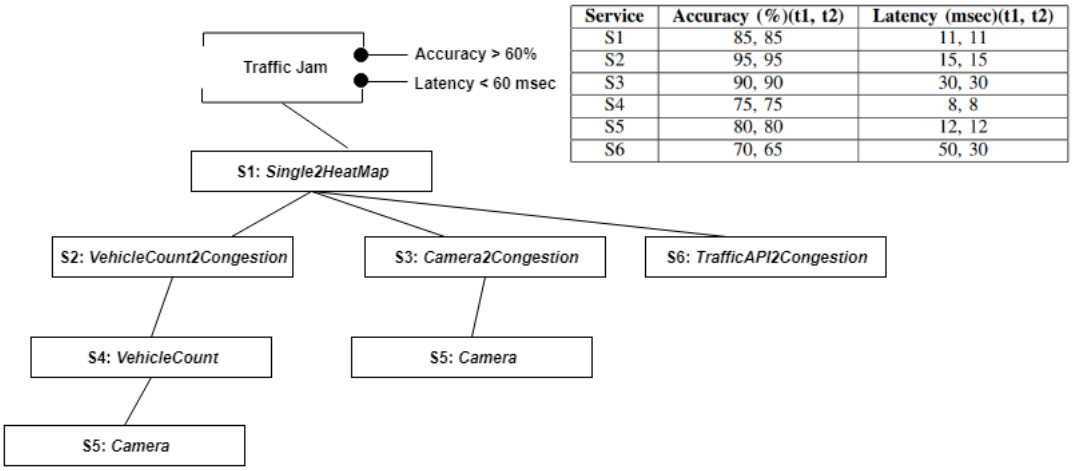
The plan phase combines the sensors and processing functions from the change model and updates the $CurrentRecognitionChain$ in the reflection model. The expected outcome of the plan phase in the running example is a recognition chain that uses service $Single2Heatmap$ as the last service and has dedicated processing functions for each street in r. These recognition chains either use the $TrafficAPI$ or combinations of $VehicleCount$ and $VehicleCount2Congestion$ or $Camera$ and $Camera2Congestion$.

The runtime models are implemented as separate agents following the model descriptions presented in Section 4.2. Change model is implemented as an agent responsible for monitoring the sensors and processing functions (AgentFactory in the architecture) whereas reflection model implemented as an agent responsible for keeping track of deployed recognition chains (RecognitionGoalFactory in the architecture) as a result of service composition and quality optimization. Evaluation model is implemented as a separate agent to store recognition goal and quality requirements for finding suitable recognition chain. Before deploying those models, Eclipse Modelling Framework (EMF)[9] is used for proof-of-concept implementation of runtime models.

[9]EMF website: `https://www.eclipse.org/modeling/emf/`

7.4.2 Setup and Testing

In order to test for dynamic behaviour of optimal recognition chain selection for traffic congestion running example, sample services are developed first. Figure 7.6 presents possible recognition chains for the running example. In order to summarize from the running example, the recognition goal is providing traffic jam information as a heat-map in the specified location. Quality requirements for this goal are that the accuracy of detected information should be more than 60% and the latency of information should be less than 60 milliseconds. The recognition chains as a result of backward-chaining are RC1 = {S1 ->S2 ->S4 ->S5}, RC2 = {S1 ->S3 ->S5} and RC3 ={S1 ->S6} which individual services satisfy the quality requirements.



Service	Accuracy (%)(t1, t2)	Latency (msec)(t1, t2)
S1	85, 85	11, 11
S2	95, 95	15, 15
S3	90, 90	30, 30
S4	75, 75	8, 8
S5	80, 80	12, 12
S6	70, 65	50, 30

Figure 7.6: Evaluation setup for the running example

This result is found using backward-chaining, as described in Section 4.2. After the related recognition chains are found, the overall quality values of the chains should be calculated before further optimization. Based on the Algorithm 2, the aggregated values for each recognition chains (RC) are depicted in Table 7.1. While aggregating the values, MIN operator is used for accuracy and SUM operator for latency in the algorithm.

Based on the requirements for accuracy and latency, RC3 does not fulfil the latency requirement after the aggregation of services and cannot be used for the recognition goal. Therefore RC1 and RC2 are kept to further optimize at time $t1$.

As mentioned in Section 4.1, when the observations arrive from the data sources, related quality properties are updated. This is enabled at run-time by using an agent-based architecture to check constantly for the new observations. Subject to the existing services for traffic jam information, the new observation for S6 effects the accuracy value by decreasing it to 65%

and latency to a 30 milliseconds. Based on this, aggregated values for recognition chains are calculated again as depicted in Table 7.1 for time $t2$. Based on new calculation as there is no requirement violation, RC1, RC2, RC3 are all taken into account to further optimize.

Table 7.1: Quality Values for Recognition Chains (at time = t1, t2)

Recognition Chain	**Accuracy(%) (t1, t2)**	**Latency(msec) (t1, t2)**
RC1	75, 75	46, 46
RC2	80, 80	53, 53
RC3	70, 65	61, 41

For further optimization in two different observation time-frames for $t1$ and $t2$, the following scenarios are tested:

a Maximize score only for accuracy (weight-acc = 1.0, weight-lat = 0.0)

b Maximize score only for latency (weight-acc = 0.0, weight-lat = 1.0)

c Maximize score for both equally (weight-acc = 0.5, weight-lat = 0.5)

d Maximize score for accuracy more than latency (weight-acc = 0.7, weight-lat = 0.3)

e Maximize score for latency more than accuracy (weight-acc= 0.3, weight-lat = 0.7)

The selected recognition chains for each scenario based on the proposed optimization function is depicted in Table 7.2.

Table 7.2: Selected Recognition Chains at t1 and t2

Scenario	**t1**	**t2**
Scenario (a)	RC2	RC2
Scenario (b)	RC1	RC3
Scenario (c)	RC1	RC1
Scenario (d)	RC2	RC2
Scenario (e)	RC1	RC1

As can be seen in the resultant recognition chain selection, the selection process can adapt to the changes in the environment (e.g. when a new observation is made) and provide different recognition chains to satisfy the recognition goal better. The mechanisms implemented by using models are able to find different solutions for different situations.

As mentioned in Section 4.2.3, statistical techniques need to be used for different quality parameters. Those statistical techniques for the optimization are implemented in Python to compare the results of each method for normal distribution. Additionally for solving the LP optimization problem, CPLEX [10] is used for the implementation. For evaluation purposes, service agents have been implemented that mimic the behaviour of real services with related quality properties.

7.5 Interfaces

In this section, the developed interfaces are presented to enable dynamic context recognition. This includes Graphical User Interface (GUI) for testing to see the change of selected data sources at runtime as presented in Section 7.5.1. Additionally, an Application Programming Interface (API) is developed for the data services to retrieve sensory data as presented in Section 7.5.2.

7.5.1 GUI for Testing

Using presented technology choices and extensions, the dynamic context recognition has been implemented for the case study presented in Chapter 6. This component can be used by applications to request context information and point out relevant quality criterion. To demonstrate the component, independent of any application, a web-based test user interface is developed as can be seen in Figure 7.7.

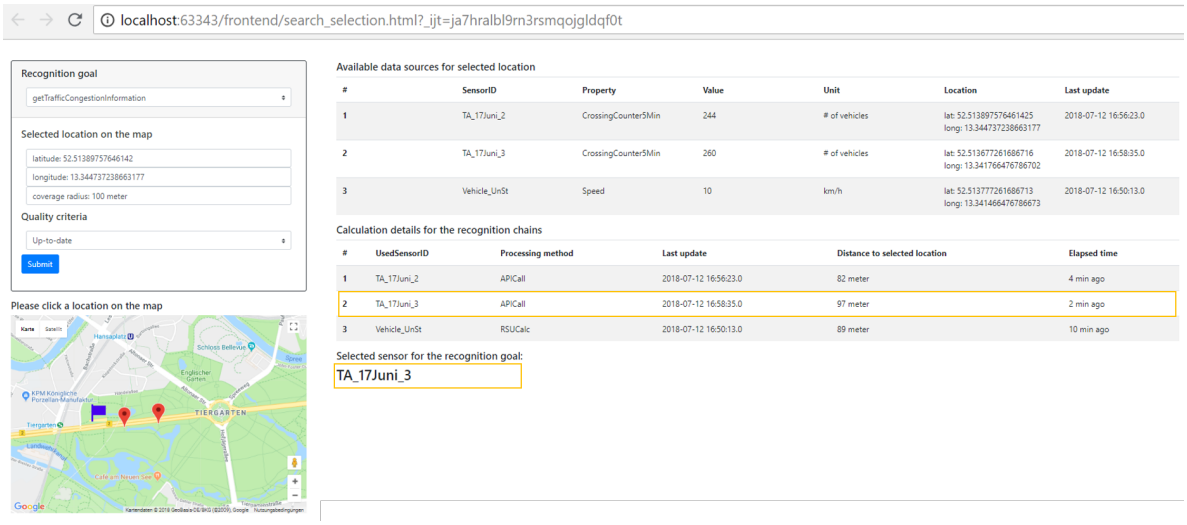


Figure 7.7: Test Interface of Dynamic Context Recognition for Traffic Congestion

[10] CPLEX Optimizer: `https://www.ibm.com/analytics/cplex-optimizer`

The recognition request is simulated on the left part of the screen. The user is able to select a recognition goal (top), a quality criterion (e.g., accuracy, up-to-dateness, low-cost)) to optimize for (middle) and a reference location (bottom) with a coverage radius. Information about possible recognition chains is visualized on the right part of the screen. The user interface displays available data sources (top) recognition chains (middle), their quality assessment and recommends one recognition chain based on its quality (bottom) based on the selected criterion. Figure 7.7 exemplifies this user interface on the running example presented in Section 6.1. There, traffic congestion status is chosen as a recognition goal and up-to-dateness as a quality criterion to be optimized for. The reference location is one location on the test road mentioned in 6.1. By using the test interface, the users can select a location (latitude, longitude) on the map. In addition, the user can specify coverage in meter based on the selected location to find the closest data sources within this coverage. For this purpose, it is specified 100 meters based on the use-case scenario. Based on all these criteria the context recognition has detected that currently, it has the options to use traffic analyser TA_17Juni_2 and TA_17Juni_3 and Vehicle called $Vehicle_UnSt$ as can be seen on the top-right of the screen. This list shows the detail information found within this coverage including sensor ID, related property for the recognition goal that can be used in the context recognition algorithm, last update time of the sensor observation and geo-location of the sensor found in the coverage. The stationary sensors deployed on the test street are also shown on the map if they are found based on the request.

Based on these three sensors the respective recognition chains have been calculated based on their quality as can be seen in the bottom-right table. The IDs of sensors that can be used for this recognition chain are presented with the detailed information including the calculated distance to the requested location, last update time and which processing method should be used. In the scenario, the calculation is based on the up-to-dateness of the retrieved information. According to these values, the $SensorTA_17Juni_3$ is judged to be the best result at the current point in time. Although the TA_17Juni_2 sensor is much closer to the requested location, it is not selected because up-to-dateness is optimized.

7.5.2 API for Data Services

In order to access sensory data in a structured way, an API is developed for data services. This API covers the descriptions of REST services with required input parameters and the

structure of the responses that the services can provide. Available REST services are described below.

- **getSensorsInArea:** This service takes a given location (latitude and longitude) and a radius as an input and returns a list of sensors which are nearby the given location and within the radius.
- **getSensorsInAreaBySensorType:** This service takes a given location (latitude and longitude), a radius and the name of a sensor type as an input and returns a list of sensors which are nearby the given location, within the radius and from a specific sensor type.
- **getSensorsInAreaBySensorTypeAndObservableProperty:** This service takes a given location (latitude and longitude), a radius,the name of a sensor type and the name of the property which the sensor type provides as an input and returns a list of sensors which are nearby the given location, within the radius, from a specific sensor type and the sensor type measures the given property.
- **getDevicesInArea:** This service takes a given location (latitude and longitude) and a radius as an input and returns a list of devices which are nearby the given location and within the radius.
- **getHistoricalData:** This service returns all the data from a certain point of time of a specific sensor from a specific device.
- **getSensorTypes:** This services provides all available sensor types within the system.
- **getObservablePropertiesBySensorType:** This services provides all available properties for a given sensor type.

Apart from those data services, a REST service is developed to return all available meta-descriptions of services as well as execution of match maker to fulfil the request. These are:

- **getServiceDescriptions:** This service provides a list with all available web services within the system. It provides the inputs, outputs, name and more for those services.
- **executeMatchMaker:** This service executes the service matchmaker with a given request to find the most fitting service composition based on quality.

Detailed information about the API provided via Postman[11] collection can be found in the following link here: `https://documenter.getpostman.com/view/9337092/SWLYAqsk?version=latest`

7.6 Summary

In this Chapter, the implementation of an abstraction layer for information provision in the context of information support is provided. It has two main abstractions as (1) from technical protocols, as (2) from specific sensors. (1) task is enabled by IoT middleware where different sensory data is integrated and the data is stored in the streaming platform.

The main contribution of this work is for the (2), where the algorithms are developed to search for optimal information channels based on extension of opportunistic sensing [5] by multi-criteria optimization as well as self-adaptation. Major functionalities of this part is providing adaptive resource/service selection based on quality parameters. It also allows dynamic discovery of resources/services by using streaming platform.

In order to provide mentioned functionalities, the architecture is implemented based on agent-based architecture [117]. It allows service and device-based aggregation/composition to use data from different sensors sending data to streaming platform. Regarding the device/sensor integration, the agents are implemented by using JIAC V to receive data from different topics. Additionally, an API for sensor data services (REST services) and sample GUI for testing regarding dynamic selection are provided.

Regarding the service part, the service composition is provided by using AI backward chaining and extended to use quality parameters of services. Quality parameters are used for the multi-criteria optimization to find and select the optimal service composition fulfilling the application request. All descriptions including service, request, service composition as a response are represented as JSON. Service description uses partly IOPE (Input, Output, Precondition, Effect) model from OWL-S [112] with the extension of representing quality.

In Chapter 8, the evaluation of the proposed algorithms and implemented functionalities with different experiments are provided.

[11] Postman API Development Platform: `https://www.getpostman.com/`

8

Evaluation

In this Chapter, the evaluation of overall approach presented in Chapter 4 and 5 is presented to provide end-to-end quality-optimal adaptive recognition chain selection. In Section 8.1, the dataset and setup are presented. Then in Section 8.2, the design and evaluation results of the experiments are provided. Some of the experiment results are presented in the author's published work [110].

8.1 Dataset and Setup

To create a dataset for evaluation, publicly available Smart City SD service descriptions dataset[1] is used. Cabrera et al. created this dataset based on OWL-S descriptions for smart city services[2][119]. This dataset is enriched by the service model proposed in this work by including quality properties of the service in this dataset. A sample description of a service in JSON[3] format can be seen in Figure 8.1.

[1]Smart City SD GitLab `https://gitlab.scss.tcd.ie/smartcitySD/data/services-dataset`
[2]OWLS-SLR Data set: `http://lpis.csd.auth.gr/systems/OWLSSLR/datasets.html`
[3]JavaScript Object Notation (JSON): `https://www.json.org/`

```
{
  "id": "s1",
  "name": "single2HeatMap",
  "url": "http://blabla/s1",
  "domains": [
    {
      "name": "Traffic",
      "url": "http://dai-labor.de/cityOntology/#Traffic"
    }
  ],
  "outputs": [
    {
      "name": "trafficJam",
      "type": "heatmap"
    }
  ],
  "inputs": [
    {
      "name": "singleHeatmap",
      "type": "heatmap"
    }
  ],
  "state": true,
  "qualityProps": [
    {
      "name": "accuracy",
      "value": "85",
      "type": "double",
      "unit": "percent"
    },
    {
      "name": "latency",
      "value": "11",
      "type": "double",
      "unit": "msec"
    }

  ]
}
```


Figure 8.1: Sample quality-enriched service model

In order to evaluate the proposed approach, this enriched dataset together with the sample service descriptions created for the running example as presented in Section 7.4 are used. Based on this dataset, the proposed approach is evaluated with the experiments presented in the next section.

As mentioned in Section 4.1.2.2, recognition goal description is modelled similar to service description used the proposed approach. A sample recognition goal description in JSON format can be seen in Figure 8.2.

```
{
  "id": "rg1",
  "name": "trafficAPI2Congestion",
  "outputs": [
    {
      "name": "trafficJam",
      "type": "heatmap"
    }
  ],
  "inputs": [
    {
      "name": "latitude",
      "type": "double"
    },
    {
      "name": "longitude",
      "type": "double"
    }
  ],
  "qualityReqs": [
    {
      "name": "accuracy",
      "direction": "positive",
      "maximum": "100",
      "minimum": "60",
      "aggregationMethod": "min",
      "unit": "percent",
      "type": "double",
      "weight": "0.5"
    },
    {
      "name": "latency",
      "direction": "negative",
      "maximum": "60",
      "minimum": "0",
      "aggregationMethod": "sum",
      "unit": "msec",
      "type": "double",
      "weight": "0.5"
    }
  ],
  "domains": [
    {
      "name": "Traffic",
      "url": "http://dai-labor.de/cityOntology/#Traffic"
    }
  ]

}
```


Figure 8.2: Sample recognition goal model

To evaluate the effectiveness of proposed approach, two experiments are designed targeting the overall performance enabling adaptive service selection at runtime and the correctness of the results after the optimization process. The first experiment is measuring the performance of the algorithm, and the second experiment is testing its adaptive behaviour in case of changes. After those experiments, the expressiveness of the proposed models are examined separately.

The experiment setup consists of a virtual machine (VM) with a two core Intel Xeon Processor E7 (2.27GHz) and 10 GB of RAM, running an UBUNTU 16.04 LTS operating system. The VM has a Apache Kafka[4] instance running on it to retrieve sensor data used in the services. As Apache Kafka is based on publish-subscribe mechanism, the changes about the service quality is forwarded directly to the agent subscribe to related topic for this specific service.

[4]Apache Kafka: `https://kafka.apache.org/`

8.2 Experiments

In this section, the proposed approach is evaluated based on two main metrics performance and correctness. Correctness is a key metric for context recognition especially in dynamic environments. Adaptation could help increasing the correctness of the context recognition task. However, this could also burden the performance of the system as the system needs also time for adaptation. Therefore, first those two metrics are looked into for the evaluation of the proposed approach. After that, the expressiveness of the models are examined based on the identified quality criteria.

8.2.1 Experiment 1- Performance

To test the performance of the algorithm for the optimization on top of backward chaining, the services in the dataset are tested incrementally with the number of 10, 20, 40, 50, 100 service deployment setup. For the performance, the setup is done via warm-up iterations with a number of 100 and then run the iterations with a number of 100 to calculate the average time. The results are depicted in Figure 8.3 and Figure 8.4.

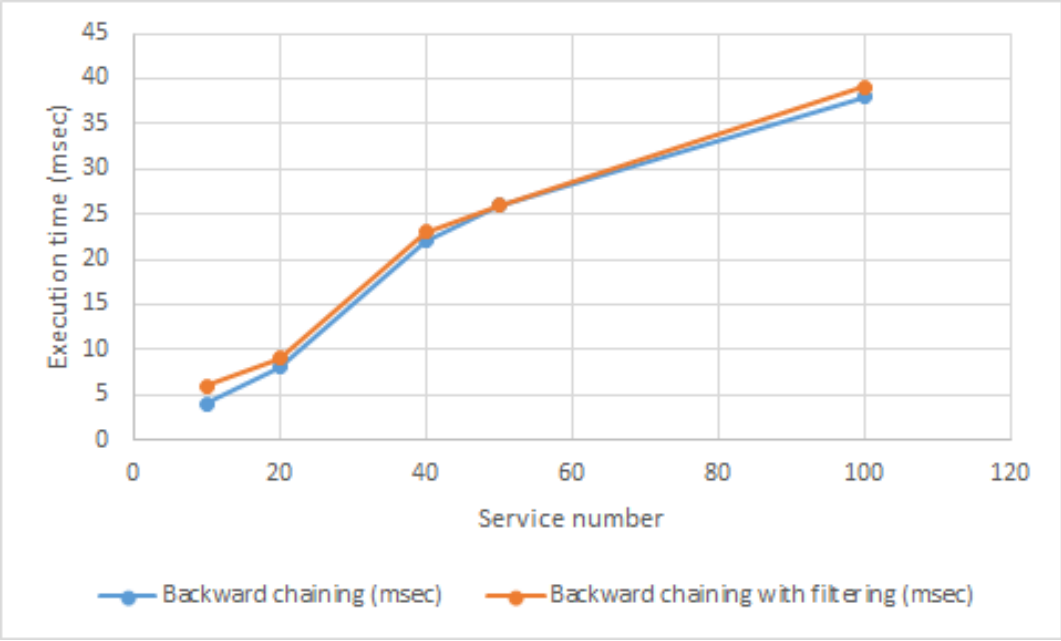


Figure 8.3: Execution times with different service numbers

As can be seen in Figure 8.3, the execution time of backward chaining and backward chaining with quality filtering is close to each other with slightly in favour of backward chaining in the low number of services and the opposite for the high number of services.

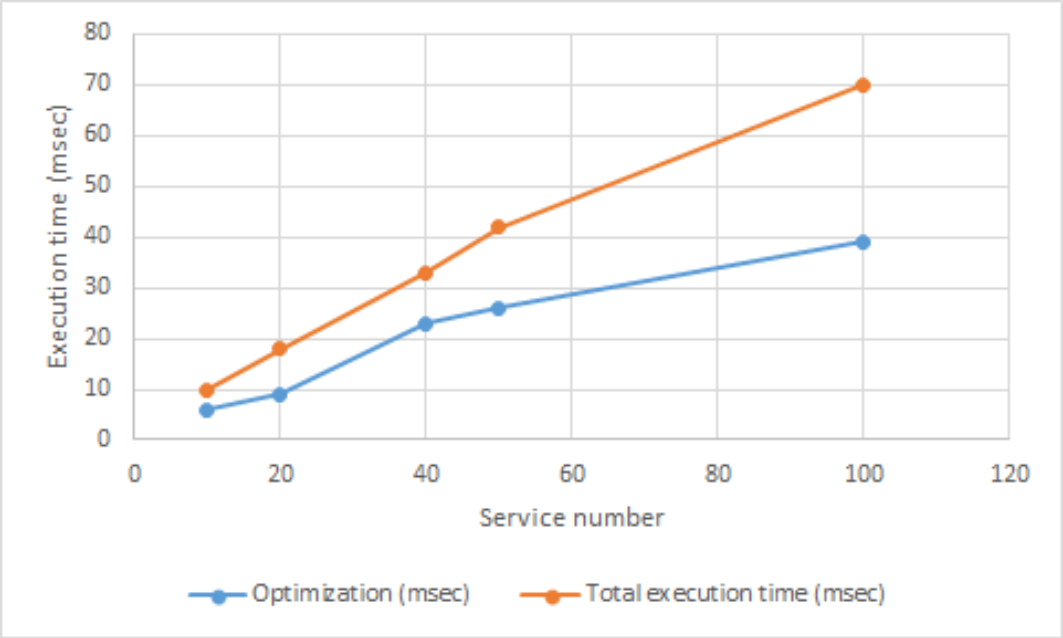


Figure 8.4: Execution times with different service numbers

Figure 8.4 shows the required optimization time with the total for finding the optimal recognition chain. Although the optimization increases the total execution time, this can be compensated in the situations where it can decrease the failure rate of a service.

8.2.2 Experiment 2- Success rate

To test the adaptive behaviour, the setup is created with 10, 20, 40, 50, 100 service deployment by randomly making some services unavailable. The success rate of the mechanism is measured in case of changes with the non-adaptive version. For this purpose, the output of the required recognition goal is compared with the result of the recognition chains. The results are depicted in Figure 8.5 and Figure 8.6 for the time required for adaptive behaviour of the system with the success rate.

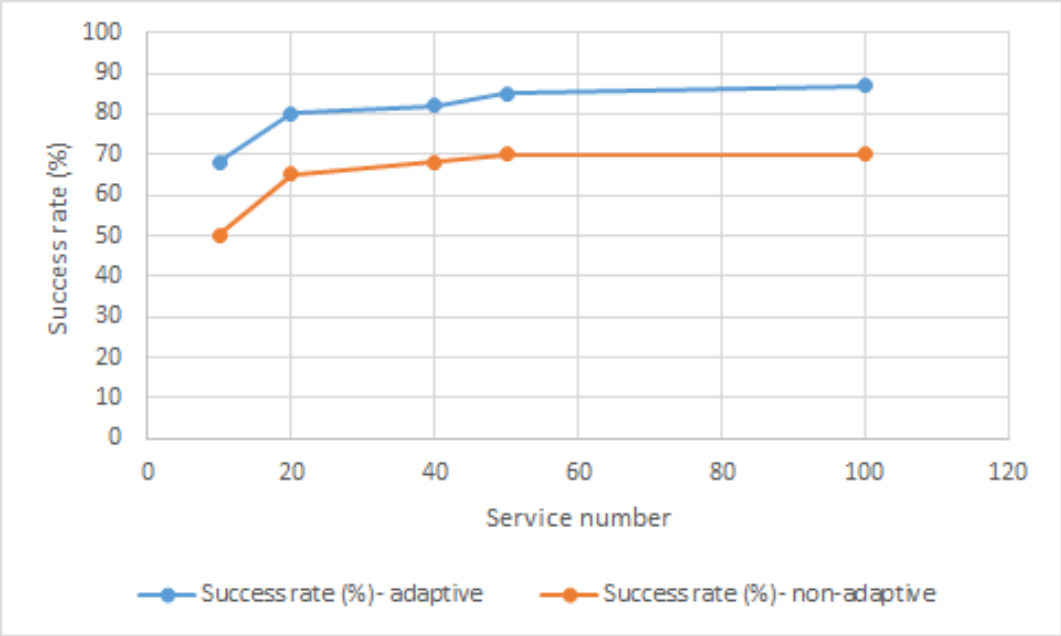


Figure 8.5: Success rate with the execution times for different services

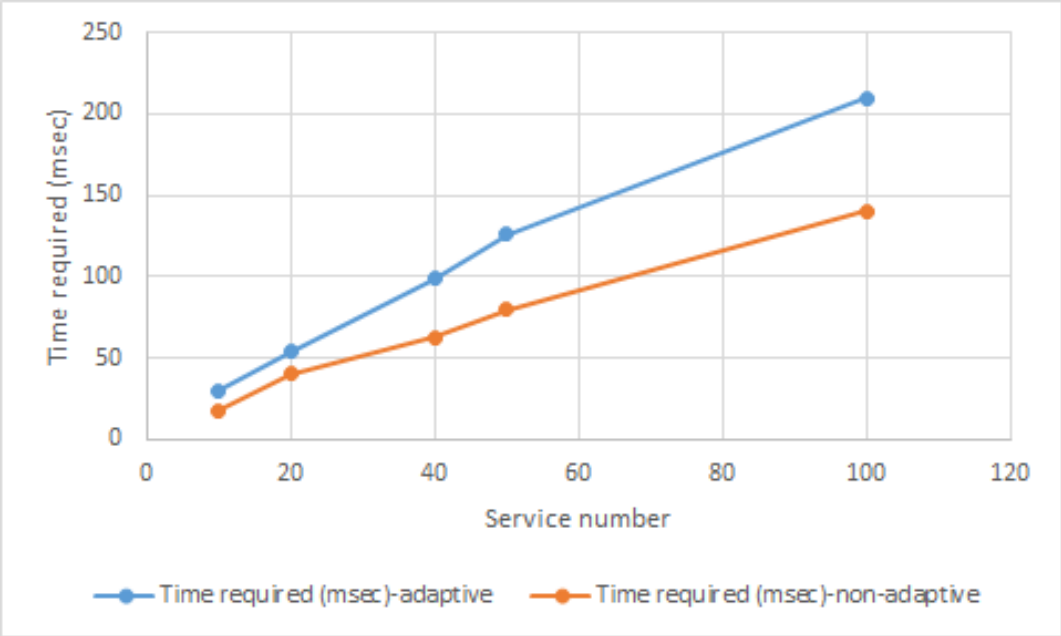


Figure 8.6: Success rate with the execution times for different services

As can be seen in Figure 8.5, success rate for adaptive selection is higher than the non-adaptive version with a execution time difference trade-off depicted in Figure 8.6. Although the adaptive behaviour increases the execution time, based on the results to increase success rate, it is still in an acceptable range for the conditions which are not time-critical in the test road compare to the correctness of the information that is more important.

8.2.3 Experiment 3- Expressiveness

For modelling the recognition goal and recognition chain elements, service model is proposed to describe the required properties as presented in 4.1. The expressiveness of this model is shown from the quality point of view in this section.

To recall the quality property definition in the service from Section 4.1.2.2, it has id, type and value properties whereas quality of recognition goal has id, maximum, minimum, direction and aggregation method properties. Based on those definitions, Quality of Context (QoC) parameters are classified from Appendix A.1 based on the state-of-the-art. Figure 8.7 shows the related quality requirement implemented based on the description provided in the proposed approach.

Parameter	**Description**	**Quality description for recognition goal (Qreq)**
Accuracy	Correctness rate of measured values to the actual values	id = accuracy max = {value} min = {value} direction = positive aggregation = min
Precision	Equality rate of measured values in repetitions	id = precision max = {value} min = {value} direction = positive aggregation = min
Freshness	Up-to-dateness of the data comparing to the current time	id = fresshness max = {value} min = {value} direction = negative aggregation = max
Confidence/Reliability	Level of certainty in the measured values	id = reliability max = {value} min = {value} direction = positive aggregation = min
Importance/Criticality	Significance level of the measured attributes	id = importance max = {value} min = {value} direction = positive aggregation = min
Correctness	Probability of matching between the measured value and real value	id = correctness max = {value} min = {value} direction = positive aggregation = min

Figure 8.7: Recognition goal requirements

This list for quality criteria is compiled from different studies in the scope of quality of context. The values for minimum and maximum are configured by the application/request depending on the requirement.

Apart from the recognition goal requirements, a list of QoS and QoD properties is also compiled from different studies as presented in Appendix A.2 and A.3. The expressiveness is depicted for each property based on the proposed quality requirement model as depicted in the Figure 8.8.

Parameter	Description	Quality description for recognition chain element (Qprop)
Availability (QoS)	Measurement of time when the service is operable	id = availability value = {value} type = percentage
Successability (QoS)	The proportion of number of response over number of request messages	id = successability value = {value} type = percentage
Reliability (QoS)	Measure of how long the service performs its expected behaviour	id = reliability value = {value} type = percentage
Response time (QoS)	Total time it takes from when a request is made until the response is received	id = responsetime value = {value} type = msec
Latency (QoS)	Delay occurred while communicating a message to a service	id = latency value = {value} type = msec
Throughput (QoS)	Total number of service invocations for a given period of time	id = throughput value = {value} type = percentage
Sensor type (QoD)	The type of the sensor	id = sensortype value = {value} type = enum
Location (QoD)	Geographical location of the sensor	id = location value = {value} type = geolocation
Battery life/Power consumption (QoD)	Battery consumption of the sensor.	id = consumption value = {value} type = percentage
Measurement/sensor range (QoD)	Spatial scope of the measured value where it is valid	id = range value = {value} type = meter
Resolution	Granularity of the data measured	id = resolution value = {value} type = percentage
Source	Identifier or indicator of the source providing measurement	id = source value = {value} type = string

Figure 8.8: Recognition chain elements quality properties

By using the models proposed, comprehensive list of QoS, QoD and QoS properties can be expressed for adaptive quality-aware recognition chain selection.

8.3 Summary

In this Chapter, the results of the evaluation for the proposed approach are provided. Three different metrics are used as performance, success rate and expressiveness to address the research goals presented in 2.3. The results show that although the adaptive behaviour increases the execution time; increased success rate provides an increase in the quality of context optimization. Based on the expressiveness evaluation depicted in Section 8.2.3, comprehensive list of quality parameters can be used to increase the quality of context optimization depending on the context-aware application requirement.

The results of evaluation show promising results for end-to-end adaptive optimal recognition chain in the scope of autonomous driving case study. In Chapter 9, additional application areas are presented where proposed approach is applied and implemented.

9

Additional Applications

In this Chapter, two use case scenarios from different research projects are presented where the proposed approach of this work is partially applied. In 9.1, a use case from an e-mobility domain is presented where the proposed selection algorithm on the service optimization for the electro-mobility services is applied. In Section 9.2, a demo application from a health domain is provided to demonstrate the importance of flexibility for context-aware applications using adaptive service selection mechanisms.

9.1 Service Optimizer for Electro-mobility

Adoption of more electro-mobility services in urban environments is becoming necessity for sustainable transportation. To enable this vision and address the challenges in real world, Hyper-Network for electro-Mobility (NeMo) project[1] aims at providing a decentralized service platform to ease the integration of electro-mobility services throughout Europe. NeMo provides different components to enable interoperability and secure data exchange between different service providers. Service Optimizer is one of them as a part of decentralized service platform for e-mobility services in the scope of NeMo project where partial results of this work is applied. Service Optimizer together with other components is presented in the published work [120].

The *Service Optimizer* component has been developed for optimizing the service search results based on non-functional criteria. Relevant Quality of Service (QoS) parameters have

[1]NeMo Project: `https://nemo-emobility.eu`

been identified for electro-mobility services based on ISO/IEC 13236[2] and W3C Web Service Architecture (WSA)[3] standards. Those QoS parameters are *average availability* of a service in percentage and *average response time* of a service in milliseconds. Service providers have to agree to provide those parameters when registering their services to the NeMo Hyper-Network. Service Optimizer functionalities are integrated to the Service Creation Tools of NeMo [120] and can be seen in Figure 9.1.

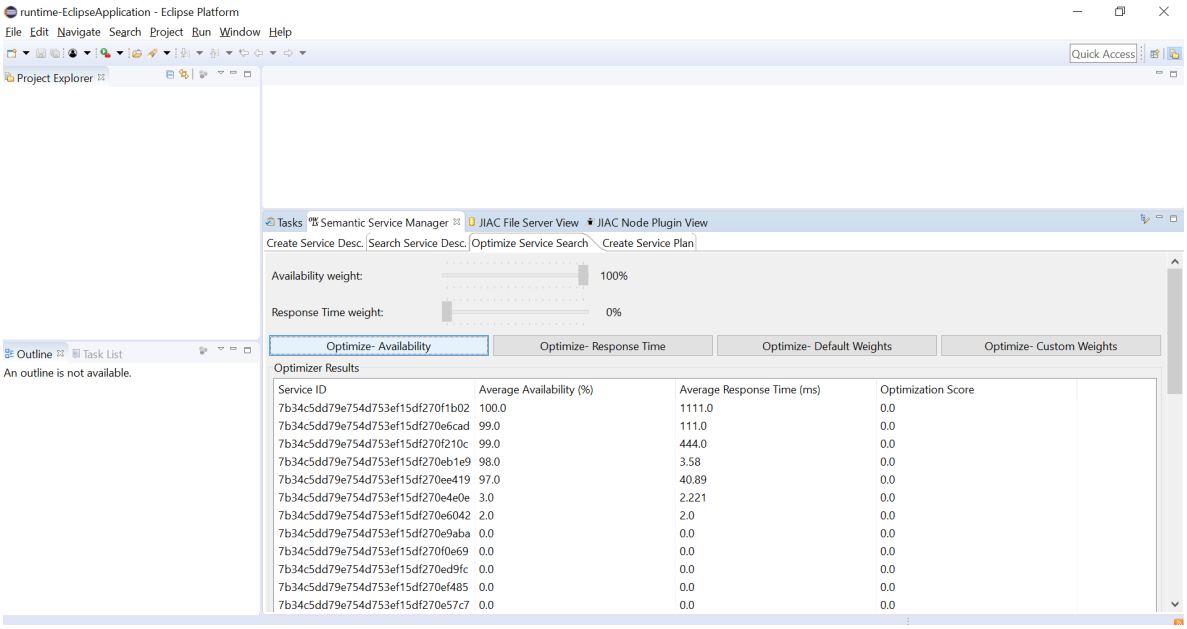


Figure 9.1: NeMo Service Optimizer

The *Service Optimizer* provides functionalities for (1) sorting the services retrieved from the Service Finder based on one QoS parameter; (2) optimizing the search results based on multiple identified QoS parameters. For the (1), the sorting depends on the type of the QoS parameter e.g. for the average availability, it sorts from high to low, for the average response time, it sorts from low to high. For the (2), Weighted Sum Model (WSM) has been implemented which is a commonly used in multi-objective optimization method [121]. By using WSM model, the following maximization function is defined to calculate the score:

$$\underset{s}{\text{maximize}} \quad \sum_{i=1}^{n} w_i x_i(s).$$

This function assumes that there are quality characteristics x_1 to x_n which can be derived from a service s and weights for these quality characteristics w_1 to w_n. In NeMo, service

[2]ISO/IEC 13236 Telecommunications and information exchange between systems standard: `https://www.iso.org/standard/27993.html`

[3]W3C WSA standard: `https://www.w3.org/TR/ws-arch/`

providers provide the average QoS values while registering their service to the NeMo Hyper-Network. The goal is to find a service that maximizes the weighted sum of these quality characteristics and sort them based on this score (from high to low). For NeMo services, average response time which is supposed to be minimized, is combined with a negative weight, to make it a maximization goal. In NeMo, two QoS parameters are taken into account to calculate the score with different customizable weights. The different score calculations can be seen in Figure 9.2.

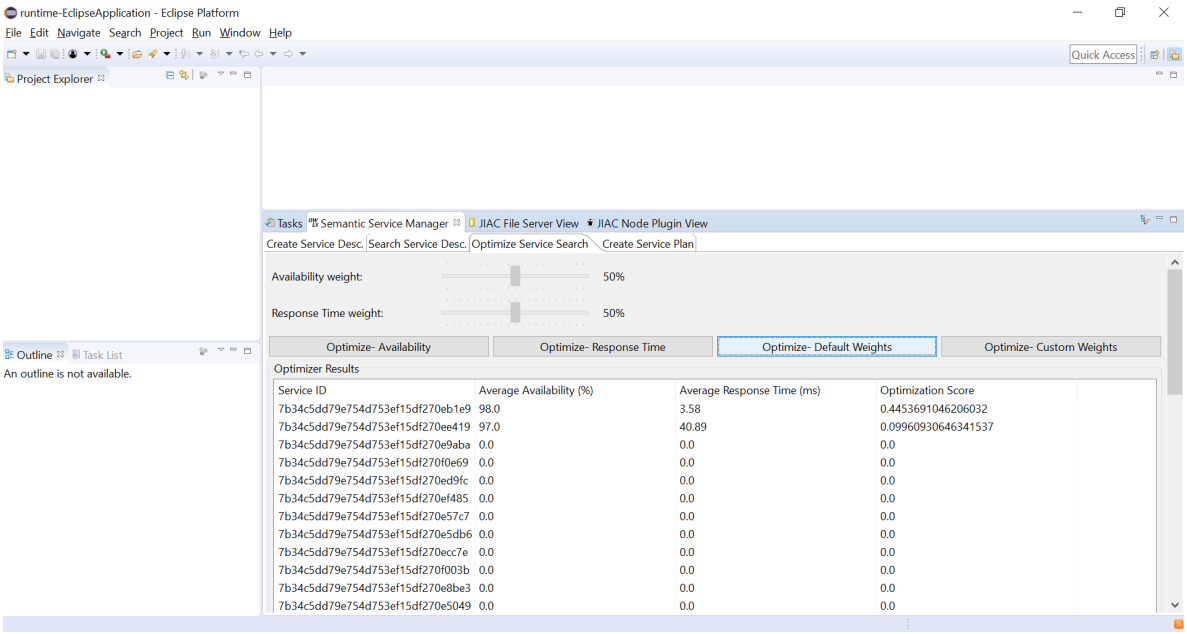


Figure 9.2: Ranking with different scores

To use different-scaled QoS parameters in the WSM, the QoS parameters need to be normalized to become in the same scale. For this purpose, different normalization techniques have been implemented, using statistical methods (e.g. *tanh*, sigmoid, z-score, logistic regression and *pnorm*). In WSM, the types of different quality parameters to maximize or to minimize are also taken into account by using positive or negative weights, respectively.

This use-case and implementation show that proposed models and methods in this dissertation can be applied to electro-mobility service domain for adaptive service selection.

9.2 Bike+: Context-aware Sensor Selection

More and more applications need to be aware of their environment (or their context of use) to support healthy lifestyle of people. However, it is not feasible to use different apps or setups for the same purpose everytime when the context has changed. With the IoT by having

the opportunity to use different sensors for the same purpose, it becomes less feasible to be dependent on one type of sensor for the context-aware applications. To demonstrate the importance of the adaptive service selection for this purpose, an application demonstration interface called Bike+[4] is designed to optimally find out sensor or sensor combination best fitting the goal of a health application by taking into account context of use. This demo with related challenges is presented in one of the earliest publication of the author related to this work [9].

Bike+ is based on two different scenarios namely biking indoor and outdoor. To prevent the burden of using different application for the same activity indoor and outdoor, Bike+ demostrates different types of sensors that can be used for context-aware sensor selection which is able to make use of existing sensors based on user requirements.

In outdoor training, it is considered that the user has a smartphone (having GPS, 3G, phone inertial sensors like accelerometer and gyroscope) while biking outside. In this context-aware application, as a default, the application is looking for the most accurate sensor (lowest mean error) to get data about user speed and calories burned while biking. This can be seen in Figure 9.3 While going through the forest, when the selected sensor becomes unavailable (e.g. GPS signal), the system demonstrates to automatically searching amongst other available sensors to keep providing data to the application. When the better sensor with respect to quality (e.g. GPS signal) becomes available again, the system demonstrates switching back to use the better sensor to provide most accurate result to the application.

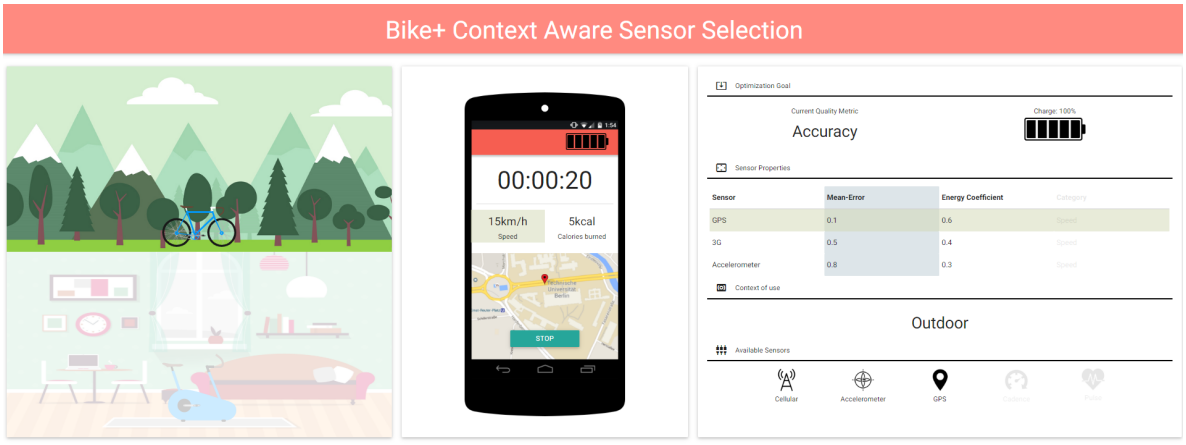


Figure 9.3: Bike+ demo with outdoor biking

However, in many cases, using most accurate sensor has a disadvantage as it consumes battery of the phone. To regulate this for better balance between accuracy and energy

[4] https://dainas.dai-labor.de/~eryilmaz@dai/bikeplus/index.html

consumption, when the battery is lower than 50%, the system demonstrates to look for a most energy efficient sensor (lowest energy coefficient) to provide required data for the application even with less accuracy. The change can be seen in Figure 9.4.

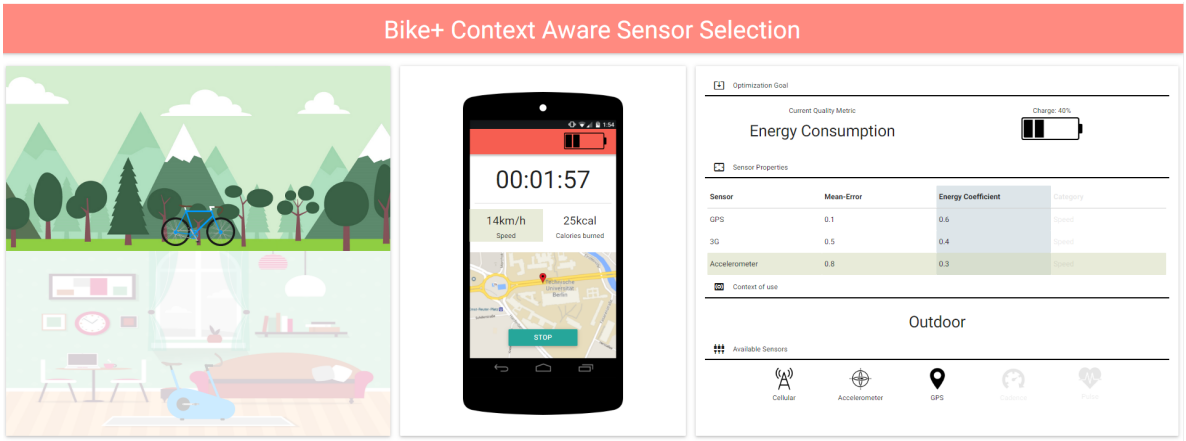


Figure 9.4: Bike+ demo with outdoor biking with changed sensor selection

Similar to outdoor training, indoor training uses different sensors and different preferences for quality requirement. When there is a context change from outdoor to indoor, the availability of the sensors has changed as some sensors are neither unavailable (such as GPS) nor cannot provide meaningful data (such as 3G, accelerometer) for speed and calorie required by the application. In this case, the system demonstrates for checking both signals of the sensors as well as their contribution to the required information. Therefore, the sensors are modelled semantically to include the contribution of sensors to the required information in specific conditions. Then, the selection algorithm can query the available and meaningful sensors for indoor biking (such as cadence sensor) to get the speed information and calculate the approximate calorie based on the speed. This can be seen in Figure 9.5. Target default property of the application is the accuracy for the indoor biking as the battery can be charged while the user is at home.

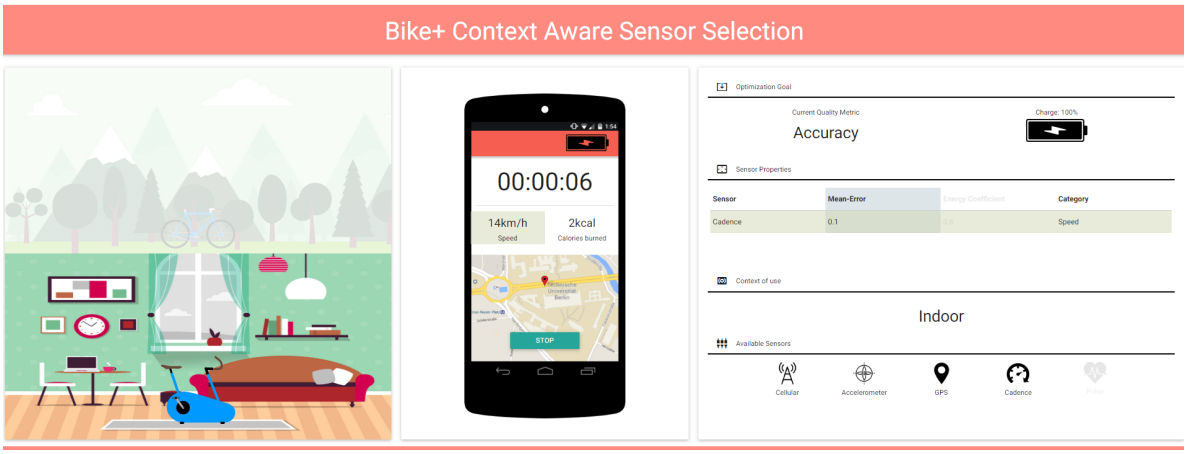


Figure 9.5: Bike+ demo with indoor biking

In this demo application, two basic quality criteria are included namely accuracy and energy consumption for the goal mean-error and energy coefficient for the sensors. There are many other parameters for the optimization which should be taken into account based on the data required by the application as well as sensor properties. All these quality parameters should be weighted to create a feasible optimization function. Secondly, every sensor/sensor combination requires different processing to provide data for the application. The impact of processing function should be also covered for the goal-based optimization. After the goal-based optimization based on extended quality of context parameters, the adaptation should take place to update the sensor selection providing the required data for the application. This application outlines how to make use of the research results of this work for a healthcare application by addressing the challenges mentioned.

10

Conclusion

This chapter concludes the presented research work with the contributions achieved and potential future work. In Section 10.1, a summary with the contributions is provided by comparing them to the State-of-the-Art approaches presented in 3.2. Then, in Section 10.2, discussion and future direction are presented as an outcome of this research with the potential impact to the ongoing research efforts.

10.1 Summary

Throughout this research work, the problem of changes in the dynamic environments with respect to sensor appearance/disappearance is addressed and an approach is proposed to increase the quality of recognition by adaptive optimal recognition chain selection. Proposed approach extends AI backward-chaining with additional quality filtering before and during recognition chain creation. For this purpose, the models for recognition goal and recognition chain elements are proposed. Those models and methods are implemented by using agent- and service-oriented architectures. To provide an autonomous control, runtime models are used. Proposed approach is evaluated with respect to expressiveness for the proposed models, performance and correctness for the proposed methods. The contributions with respect to each research question from 1-4 (see 2.3) are summarized below:

- **Modelling of recognition goal with quality requirements** First research question (RQ1) is answered by providing recognition goal model in Section 4.1.2.1.

- **Modelling of recognition chain elements and their quality impact on the recognition goal** Second research question (RQ2) is answered by providing recognition chain model and its involved elements including sensor, data processing and service descriptions in Section 4.1.2.4.
- **Method to determine optimal recognition chain given the models** Third research question (RQ3) is answered by providing algorithm for optimal selection as presented in Section 4.
- **Adaptation of optimal recognition chain in case of changes** Forth research question (RQ4) is answered by providing runtime models for the adaptation as presented in Section 5.

RQ1 and RQ2 are evaluated for the expressiveness in Section 8.2 as they are related to the proposed models. RQ3 and RQ4 are evaluated with respect to correctness/success rate and performance as they are related to the proposed methods.

To address the lack of expressiveness for quality parameters in the existing modelling techniques, an ontology and service model for the recognition chain elements as well as recognition goal is proposed. This contribution combines the efforts of semantic service description and sensor ontologies to address the quality requirements of a context recognition task.

Adequate model of recognition chain elements allow to further develop selection and ranking techniques to select the best recognition chain among the available ones. For this purpose, multi-criteria optimization is applied by using the different quality parameters in the chain elements fulfilling the recognition goal.

Last but not least contribution of this work is enabling the optimal selection dynamically in case of any changes in the environment. For this purpose, MAPE-K through run-time models is applied as an extension of opportunistic sensing. This allow us to further develop and combine opportunistic sensing frameworks with self-adaptive approaches.

Besides addressing the research questions, agent-based architecture is proposed as presented in Section 7.1 which fulfils the following requirements:

- Dealing with heterogeneity by integrating with IoT middleware
- Providing stream-based context recognition by updating the data continuously

- Reflecting on the runtime changes to select the optimal recognition chain
- Optimizing quality by adaptive selection of the recognition chain
- Providing decoupled calculation via an agent-based implementation

The results shown throughout this work are promising for the increase on the quality of context recognition by autonomous selection. The case study is done in the autonomous driving test environment and applicability of the approach is depicted in different domains like healthcare and electro-mobility as presented in Chapter 9.

10.2 Discussion and Future Work

As presented in 10.1, the work conducted in the scope of this thesis achieved main contributions in the scope of context management and self-adaptability. Industry Specification Group (ISG) addressed the need of context information together with sensory data by developing Context Information Management (CIM)[1]. The contributions achieved throughout this thesis could also be used to provide adaptive quality of context optimization in IoT.

As of every research dissertation, there is still work to do on top of the results of this thesis. To increase the performance of the proposed approach, edge computing can be used to decrease the communication time. Using the edge to minimize communication delay and thus improves the overall quality of the recognition chains could decrease the latency more for time-critical actions required in the testing environment.

As mentioned in Section 7.3, the deployment ontology is left out of scope for this work. This requires ontology and semantic matching for the modelling of the deployment environment in different application domains.

Another future direction would be application of different AI planning algorithms for service composition. Only backward chaining is considered in this work by extending with comprehensive quality description and calculation to provide end-to-end framework for adaptive optimal selection.

Last but not least, benchmarks for quality monitoring at runtime are needed to cover QoS, QoD and QoC properties. In those benchmarks, different machine learning techniques can be applied for the statistical quality analysis.

[1]CIM by ISG `https://www.etsi.org/committee/cim`

For the implementation of the proposed models and approaches within the case study can be found in code repository of Diginet project under Data Integration[2] namely under the modules of composition, optimization, jiacServices and ontology. For the access, please get in touch from here [3]. The related class diagrams and ontologies from those modules are provided in Appendix.

This work is partially funded in the DIGINET-PS project which is supported by the Federal Ministry of Transport and Digital Infrastructure (BMVI) within the framework of its funding guideline “Automated and Networked Driving at Digital Test Fields in Germany”.

[2]Data Integration Gitlab `https://gitlab.dai-labor.de/diginet-ps/data-integration`
[3]`https://diginet-ps.de/corporate-creative-contact/`

References

[1] M Weiser. "The Computer for the Twenty-First Century". In: *Scientific American* (1991).

[2] Marc Kurz et al. "The OPPORTUNITY Framework and Data Processing Ecosystem for Opportunistic Activity and Context Recognition". In: *International Journal of Sensors Wireless Communications and Controle* (2012). ISSN: 22103279. DOI: 10.2174/2210327911101020102.

[3] K. A. Dey and G. D. Abwod. "Towards a better understanding of and context awarnes". In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. 2000.

[4] Mirko Presser et al. "The SENSEI project: integrating the physical world with the digital world of the network of the future". In: *IEEE Communications Magazine* (2009). ISSN: 0163-6804. DOI: 10.1109/mcom.2009.4907403.

[5] Daniel Roggen et al. "Opportunistic human activity and context recognition". In: *Computer* (2013). ISSN: 00189162. DOI: 10.1109/MC.2012.393.

[6] Charith Perera et al. "Sensing as a service model for smart cities supported by Internet of Things". In: *Transactions on Emerging Telecommunications Technologies* (2014). ISSN: 21613915. DOI: 10.1002/ett.2704. arXiv: 1307.8198.

[7] Q. Liang et al. "Opportunistic Sensing in Wireless Sensor Networks: Theory and Application". In: *IEEE Transactions on Computers* 63.08 (2014), pp. 2002–2010. ISSN: 1557-9956. DOI: 10.1109/TC.2013.85.

[8] Younghee Kim and Keumsuk Lee. "A quality measurement method of Context information in ubiquitous environments". In: *Proceedings - 2006 International Conference on Hybrid Information Technology, ICHIT 2006*. 2006. DOI: 10.1109/ICHIT.2006.253664.

[9] Elif Eryilmaz et al. “Challenges for Adaptable Quality of Context Recognition in Opportunistic Sensing”. In: *Proceedings VDE-Kongress 2016 - Internet der Dinge*. 2016. URL: https://www.vde-verlag.de/proceedings-en/454308120.html.

[10] Elif Eryilmaz and Sahin Albayrak. “Quality of Context Optimization in Opportunistic Sensing for the Automatization of Sensor Selection over the Internet of Things”. In: *ECAI 2016, Workshop Modelling and Reasoning in Context (MRC)*. 2016, pp. 11–16.

[11] John Soldatos et al. “OpenIoT: Open Source Internet-of-Things in the Cloud”. In: *Lecture Notes in Computer Science* 9001 (2015), pp. 13–25. DOI: 10.1007/978-3-319-16546-2_3.

[12] IBM. “An architectural blueprint for autonomic computing”. In: *IBM White Paper* (2006). ISSN: 19448244. DOI: 10.1021/am900608j.

[13] Kevin Ashton. “That" Internet of things", in the real world things matter than ideas”. In: *RFID Journal, June* (2009).

[14] Ovidiu Vermesan et al. “Internet of Things: Strategic Research Roadmap”. In: *Internet of Things-Global Technological and Societal Trends*. 2011. ISBN: 978-87-92329-67-7.

[15] Payam Barnaghi et al. “Sense and sens’ ability: Semantic data modelling for sensor networks”. In: *Conference Proceedings of ICT Mobile Summit 2009* (2009).

[16] James Manyika et al. “Big data: The next frontier for innovation, competition, and productivity”. In: *McKinsey Global Institute* (2011). ISSN: 14712970. DOI: 10.1080/01443610903114527.

[17] Dave Evans. “How the Next Evolution of the Internet Is Changing Everything”. In: 2011. URL: http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[18] Andrew T. Campbell et al. “People-centric urban sensing”. In: *ACM International Conference Proceeding Series*. 2006. ISBN: 159593510X. DOI: 10.1145/1234161.1234179.

[19] Daniel Roggen et al. “OPPORTUNITY: Towards opportunistic activity and context recognition systems”. In: *2009 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks and Workshops, WOWMOM 2009*. 2009. ISBN: 9781424444397. DOI: 10.1109/WOWMOM.2009.5282442.

[20] Gerold Hölzl et al. "A Framework for Opportunistic Context and Activity Recognition". In: 2011. URL: https://www.researchgate.net/publication/235351136_A_Framework_for_Opportunistic_Context_and_Activity_Recognition.

[21] Gregory D. Abowd et al. "Towards a better understanding of context and context-awareness". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 1999. ISBN: 3540665501. DOI: 10.1007/3-540-48157-5_29.

[22] Mary Bazire and Patrick Brézillon. "Understanding context before using it". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2005. ISBN: 354026924X.

[23] Andrey Boytsov et al. "Situation awareness meets ontologies: A context spaces case study". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2015. ISBN: 9783319255903. DOI: 10.1007/978-3-319-25591-0_1.

[24] Norha M. Villegas and Hausi A. Müller. "Managing dynamic context to optimize smart interactions and services". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2010). ISSN: 03029743. DOI: 10.1007/978-3-642-16599-3_18.

[25] C. B. Anagnostopoulos, Y. Ntarladimas, and S. Hadjiefthymiades. "Situational computing: An innovative architecture with imprecise reasoning". In: *Journal of Systems and Software* (2007). ISSN: 01641212. DOI: 10.1016/j.jss.2007.03.003.

[26] Juan Ye, Simon Dobson, and Susan McKeever. *Situation identification techniques in pervasive computing: A review*. 2012. DOI: 10.1016/j.pmcj.2011.01.004.

[27] Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. 2009. ISBN: 9781420090512. DOI: 10.1201/9781420090512.

[28] Frank Manola and Eric Miller. *RDF Primer W3C Recommendation*. 2004.

[29] S Bechhofer et al. *OWL Web Ontology Language Reference*. 2004.

[30] Tim Berners-Lee et al. "Tabulator: Exploring and Analyzing linked data on the Semantic Web". In: 2006. URL: https://www.researchgate.net/publication/319393309_Tabulator_Exploring_and_Analyzing_linked_data_on_the_Semantic_Web.

[31] Eric Prud Hommeaux and Andy Seaborne. “SPARQL Query Language for RDF”. In: *W3C Recommendation* (2008). URL: http://www.w3.org/TR/rdf-sparql-query/.

[32] Thomas R. Gruber. “A translation approach to portable ontology specifications”. In: *Knowledge Acquisition* (1993). ISSN: 10428143. DOI: 10.1006/knac.1993.1008.

[33] Juan Ye et al. “Ontology-based models in pervasive computing systems”. In: *Knowledge Engineering Review* (2007). ISSN: 02698889. DOI: 10.1017/S0269888907001208.

[34] Elena Simperl. “Reusing ontologies on the Semantic Web: A feasibility study”. In: *Data and Knowledge Engineering* (2009). ISSN: 0169023X. DOI: 10.1016/j.datak.2009.02.002.

[35] Michael P. Papazoglou et al. “Service-oriented computing: State of the art and research challenges”. In: *Computer* (2007). ISSN: 00189162. DOI: 10.1109/MC.2007.400.

[36] Kishore Channabasavaiah, Kerrie Holley, and Edward M Tuggle. “Migrating to a service-oriented architecture”. In: *IBM DeveloperWorks* (2004). DOI: 10.1109/ICWS.2004.1314715.

[37] T. Erl. *SOA Principles of Service Design*. The Pearson Service Technology Series from Thomas Erl. Pearson Education, 2007. ISBN: 9780132715836. URL: https://books.google.de/books?id=mkQJvjR2sX0C.

[38] Zaigham Mahmood. “Synergies between SOA and grid computing”. In: *Innovation and Knowledge Management in Twin Track Economies Challenges and Solutions - Proceedings of the 11th International Business Information Management Association Conference, IBIMA 2009*. 2009.

[39] *W3C Web Services Activity Statement*. Accessed: 2020-05-02. 2002. URL: https://www.w3.org/2002/ws/Activity.

[40] Roy Thomas Fielding. *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. 2000.

[41] Jorge Cardoso. *Semantic web services: Theory, Tools and applications*. 2007. ISBN: 9781599040455. DOI: 10.4018/978-1-59904-045-5.

[42] J. Dominque and D. Martin. *Semantic Web Services*. http://podcast.open.ac.uk/pod/iswc08-semantic-web-intro/. Accessed: 2020-05-06.

[43] David Martin et al. “OWL-S: Semantic Markup for Web Services”. In: *W3C Member Submission* (2004).

[44] Anupriya Ankolekar et al. “DAML-S: Semantic Markup for Web Services”. In: *The First Semantic Web Working Symposium (SWWS 2001)*. Semantic Web Working Symposium. SWWS 2001 (Stanford, CA, USA, July 30–Aug. 1, 2001). 2001. URL: http://www.cs.cmu.edu/%5Ctextasciitilde%20softagents/DamlPaper/DAML-S/daml-s.pdf.

[45] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. 2002. ISBN: 0137903952. DOI: 10.1017/S0269888900007724. arXiv: arXiv:1011.1669v3.

[46] *Problem solving in Artificial Intelligence*. https://www.tutorialandexample.com/problem-solving-in-artificial-intelligence/. Accessed: 2020-05-06.

[47] *Agents in Artificial Intelligence*. https://www.geeksforgeeks.org/agents-artificial-intelligence/. Accessed: 2020-05-06.

[48] Joseph L. Hellerstein et al. *Feedback Control of Computing Systems*. 2004. DOI: 10.1002/047166880x.

[49] Jeffrey O. Kephart and David M. Chess. “The vision of autonomic computing”. In: *Computer* (2003). ISSN: 00189162. DOI: 10.1109/MC.2003.1160055.

[50] Mazeiar Salehie and Ladan Tahvildari. “Self-adaptive software: Landscape and research challenges”. In: *ACM Transactions on Autonomous and Adaptive Systems* (2009). ISSN: 15564665. DOI: 10.1145/1516533.1516538.

[51] Michael Compton et al. “A survey of the semantic specification of sensors”. In: *CEUR Workshop Proceedings*. 2009.

[52] Jean Paul Calbimonte et al. “Deriving semantic sensor metadata from raw measurements”. In: *CEUR Workshop Proceedings*. 2012.

[53] Adam Pease, Ian Niles, and John Li. “The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications”. In: *Imagine* (2002).

[54] David J. Russomanno, Cartik R. Kothari, and Omoju A. Thomas. “Building a sensor ontology: A practical approach leveraging ISO and OGC models”. In: *Proceedings*

of the 2005 International Conference on Artificial Intelligence, ICAI'05. 2005. ISBN: 9781932415667.

[55] Matt Calder, Robert A. Morris, and Francesco Peri. "Machine reasoning about anomalous sensor data". In: *Ecological Informatics* (2010). ISSN: 15749541. DOI: 10.1016/j.ecoinf.2009.08.007.

[56] Carlos Rueda et al. "The MMI Device Ontology: Enabling Sensor Integration". In: *American Geophysical Union Fall Meeting – Session* 16 (2010), pp. 44–8.

[57] Michael Compton et al. "Reasoning about sensors and compositions". In: *CEUR Workshop Proceedings*. 2009.

[58] M Botts and A Robin. *OGC® SensorML*. 2014. DOI: OGC12-000.

[59] Simon J D Cox. *Observations and Measurements - XML Implementation*. 2011.

[60] Laurent Lefort et al. *Semantic Sensor Network XG Final Report*. Tech. rep. 2011.

[61] Krzysztof Janowicz and Michael Compton. "The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology". In: *CEUR Workshop Proceedings*. 2010.

[62] Ralph Hodgson et al. "QUDT - Quantities, Units, Dimensions and Data Types Ontologies". In: *W3C* (2014).

[63] Dominique Guinard et al. "From the Internet of Things to the Web of Things: Resource-oriented Architecture and Best Practices". In: *Architecting the Internet of Things*. 2011. DOI: 10.1007/978-3-642-19157-2_5.

[64] D. Guinard and V. Trifa. *Building the Web of Things: With examples in Node.js and Raspberry Pi*. Manning Publications, 2016. ISBN: 9781617292682. URL: https://books.google.de/books?id=Q1b2jwEACAAJ.

[65] Prith Banerjee et al. "Everything as a service: Powering the new information economy". In: *Computer* (2011). ISSN: 00189162. DOI: 10.1109/MC.2011.67.

[66] Arkady B. Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. "Sensing as a Service and Big Data". In: *CoRR* abs/1301.0159 (2013). arXiv: 1301.0159. URL: http://arxiv.org/abs/1301.0159.

[67] Karl Aberer, Manfred Hauswirth, and Ali Salehi. "Infrastructure for data processing in large-scale interconnected sensor networks". In: *Proceedings - IEEE International Conference on Mobile Data Management*. 2007. ISBN: 1424412404. DOI: 10.1109/MDM.2007.36.

[68] Michael Wagner, Roland Reichle, and Kurt Geihs. "Context as a service - Requirements, design and middleware support". In: *2011 IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops 2011*. 2011. ISBN: 9781612849379. DOI: 10.1109/PERCOMW.2011.5766873.

[69] Philip Moore, Fatos Xhafa, and Leonard Barolli. "Context-as-a-Service: A Service Model for Cloud-Based Systems". In: *2014 Eighth International Conference on Complex, Intelligent and Software Intensive Systems* (2014), pp. 379–385.

[70] Suparna De et al. "Service modelling for the Internet of Things". In: *2011 Federated Conference on Computer Science and Information Systems, FedCSIS 2011*. 2011. ISBN: 9781457700415.

[71] Suparna De et al. "An internet of things platform for real-world and digital objects". In: *Scalable Computing* (2012). ISSN: 18951767. DOI: 10.12694/scpe.v13i1.766.

[72] Maria Bermudez-Edo et al. "IoT-Lite: A Lightweight Semantic Model for the Internet of Things". In: *Proceedings - 13th IEEE International Conference on Ubiquitous Intelligence and Computing, 13th IEEE International Conference on Advanced and Trusted Computing, 16th IEEE International Conference on Scalable Computing and Communications, IEEE International Conference on Cloud and Big Data Computing, IEEE International Conference on Internet of People and IEEE Smart World Congress and Workshops, UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld 2016*. 2017. ISBN: 9781509027705. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0035.

[73] Claudio Bettini et al. "A survey of context modelling and reasoning techniques". In: *Pervasive and Mobile Computing* (2010). ISSN: 15741192. DOI: 10.1016/j.pmcj.2009.06.002.

[74] Thomas Strang and Claudia Linnhoff-Popien. "A Context Modeling Survey". In: *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp*

2004 - The Sixth International Conference on Ubiquitous Computing. 2004. DOI: 10.1.1.2.2060.

[75] Xiao Hang Wang et al. "Ontology based context modeling and reasoning using OWL". In: *Proceedings - Second IEEE Annual Conference on Pervasive Computing and Communications, Workshops, PerCom*. 2004. ISBN: 0769520901. DOI: 10.1109/percomw.2004.1276898.

[76] G.Tau et al. "An ontology based context model in Intelligent environments". In: *Proc. of CNDS*. 2004.

[77] Harry Chen, Tim Finin, and Anupam Joshi. "An ontology for context-aware pervasive computing environments". In: *Knowledge Engineering Review* (2003). ISSN: 02698889. DOI: 10.1017/S0269888904000025.

[78] Harry Chen, Tim Finin, and Anupam Joshi. "The SOUPA Ontology for Pervasive Computing". In: *Ontologies for Agents: Theory and Experiences*. 2005. DOI: 10.1007/3-7643-7361-x_10.

[79] Amir Padovitz, Seng W. Loke, and Arkady Zaslavsky. "Multiple-agent perspectives in reasoning about situations for context-aware pervasive computing systems". In: *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans* (2008). ISSN: 10834427. DOI: 10.1109/TSMCA.2008.918589.

[80] Thomas Buchholz, Axel Küpper, and Michael Schiffers. "Quality of Context: What it is and why we need it". In: *Proceedings of the 10th Workshop of the OpenView University Association: OVUA'03* (2003). DOI: 10.1.1.147.565.

[81] Karen Henricksen. "A framework for context-aware pervasive computing applications". In: *The School of Information Technology and Electrical Engineering, The University of Queensland* (2003).

[82] Susan McKeever et al. "A context quality model to support transparent reasoning with uncertain context". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. ISBN: 3642045588. DOI: 10.1007/978-3-642-04559-2_6.

[83] José Bringel Filho et al. "Modeling and measuring quality of context information in pervasive environments". In: *Proceedings - International Conference on Advanced*

Information Networking and Applications, AINA. 2010. ISBN: 9780769540184. DOI: 10.1109/AINA.2010.164.

[84] Claudia Villalonga et al. "Bringing quality of context into wearable human activity recognition systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. ISBN: 3642045588. DOI: 10.1007/978-3-642-04559-2_15.

[85] Charith Perera et al. "Sensor search techniques for sensing as a service architecture for the internet of things". In: *IEEE Sensors Journal* (2014). ISSN: 1530437X. DOI: 10.1109/JSEN.2013.2282292. arXiv: 1309.3618.

[86] Pierrick Marie et al. "QoCIM: A meta-model for quality of context". In: vol. 8175. 2013. DOI: 10.1007/978-3-642-40972-1_23.

[87] Atif Manzoor, Hong Linh Truong, and Schahram Dustdar. "On the evaluation of quality of context". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2008. ISBN: 354088792X. DOI: 10.1007/978-3-540-88793-5-11.

[88] Atif Manzoor, Hong Linh Truong, and Schahram Dustdar. "Quality of context: Models and applications for context-aware systems in pervasive environments". In: *Knowledge Engineering Review* (2014). ISSN: 14698005. DOI: 10.1017/S0269888914000034.

[89] Pierrick Marie et al. "From ambient sensing to IoT-based context computing: An open framework for end to end QoC management". In: *Sensors (Switzerland)* (2015). ISSN: 14248220. DOI: 10.3390/s150614180.

[90] Sabrina Sicari et al. "A security-and quality-aware system architecture for Internet of Things". In: *Information Systems Frontiers* (2016). ISSN: 15729419. DOI: 10.1007/s10796-014-9538-x.

[91] Atif Manzoor, Hong Linh Truong, and Schahram Dustdar. "Using quality of context to resolve conflicts in context-aware systems". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. ISBN: 3642045588. DOI: 10.1007/978-3-642-04559-2_13.

[92] Benedikt Ostermaier et al. "A real-time search engine for the web of things". In: *2010 Internet of Things, IoT 2010*. 2010. ISBN: 9781424474158. DOI: 10.1109/IOT.2010.5678450.

[93] B. Maryam Elahi et al. “Sensor ranking: A primitive for efficient content-based sensor search”. In: *2009 International Conference on Information Processing in Sensor Networks, IPSN 2009*. 2009. ISBN: 9781424451081.

[94] Charith Perera et al. “Context-aware sensor search, selection and ranking model for internet of things middleware”. In: *Proceedings - IEEE International Conference on Mobile Data Management*. 2013. DOI: 10.1109/MDM.2013.46. arXiv: 1303.2447.

[95] Kyriakos Kritikos and Dimitris Plexousakis. “Semantic QoS-based web service discovery algorithms”. In: *Proceedings of the 5th IEEE European Conference on Web Services, ECOWS 07*. 2007. ISBN: 0769530443. DOI: 10.1109/ECOWS.2007.20.

[96] Marco Comuzzi and Barbara Pernici. “A framework for QoS-based Web service contracting”. In: *ACM Transactions on the Web* (2009). ISSN: 15591131. DOI: 10.1145/1541822.1541825.

[97] Wei Wang et al. “A ranking method for sensor services based on estimation of service access cost”. In: *Information Sciences* (2015). ISSN: 00200255. DOI: 10.1016/j.ins.2015.05.029.

[98] Anja Strunk. “QoS-aware service composition: A survey”. In: *Proceedings - 8th IEEE European Conference on Web Services, ECOWS 2010*. 2010. ISBN: 9780769543109. DOI: 10.1109/ECOWS.2010.16.

[99] Michael R. Garey and David S. Johnson. *A guide to the theory of NP-completeness*. 1983. ISBN: 0-7167-1045-5. DOI: 10.2307/2273574.

[100] Michael C. Jaeger, Gero Mühl, and Sebastian Golze. “QoS-aware composition of web services: A look at selection algorithms”. In: *Proceedings - 2005 IEEE International Conference on Web Services, ICWS 2005*. 2005. ISBN: 0769524095. DOI: 10.1109/ICWS.2005.95.

[101] Florian Rosenberg et al. “Metaheuristic optimization of large-scale QoS-aware service compositions”. In: *Proceedings - 2010 IEEE 7th International Conference on Services Computing, SCC 2010*. 2010. ISBN: 9780769541266. DOI: 10.1109/SCC.2010.58.

[102] Jorge Cardoso et al. “Quality of service for workflows and web service processes”. In: *Web Semantics* (2004). ISSN: 15708268. DOI: 10.1016/j.websem.2004.03.001.

[103] Yuriy Brun et al. “Engineering self-adaptive systems through feedback loops”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. ISBN: 3642021603. DOI: 10.1007/978-3-642-02161-9_3.

[104] Hausi A. Müller, Holger M. Kienle, and Ulrike Stege. “Autonomic computing now you see it, now you don’t: Design and evolution of autonomic software systems”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2009. DOI: 10.1007/978-3-540-95888-8_2.

[105] Danny Weyns et al. “On patterns for decentralized control in self-adaptive systems”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2013. ISBN: 9783642358128. DOI: 10.1007/978-3-642-35813-5_4.

[106] Pieter Vromant et al. “On interacting control loops in self-adaptive systems”. In: *Proceedings - International Conference on Software Engineering*. 2011. ISBN: 9781450305754. DOI: 10.1145/1988008.1988037.

[107] Norha M. Villegas et al. “DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2013. ISBN: 9783642358128. DOI: 10.1007/978-3-642-35813-5_11.

[108] Danny Weyns, Sam Malek, and Jesper Andersson. “FORMS: Unifying reference model for formal specification of distributed self-adaptive systems”. In: *ACM Transactions on Autonomous and Adaptive Systems* (2012). ISSN: 15564665. DOI: 10.1145/2168260.2168268.

[109] Thomas Vogel and Holger Giese. “Model-driven engineering of self-adaptive software with EUREMA”. In: *ACM Transactions on Autonomous and Adaptive Systems*. 2014. DOI: 10.1145/2555612. arXiv: 1805.07353.

[110] Elif Eryilmaz et al. “Adaptive service selection for enabling the mobility of autonomous vehicles”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2019. ISBN: 9783030342548. DOI: 10.1007/978-3-030-34255-5_14.

[111] Elif Eryilmaz, Frank Trollmann, and Sahin Albayrak. "Quality-aware service selection approach for adaptive context recognition in IoT". In: *ACM International Conference Proceeding Series*. 2019. ISBN: 9781450372077. DOI: 10.1145/3365871.3365874.

[112] Aitor Urbieta et al. "Analysis of effects-and preconditions-based service representation in ubiquitous computing environments". In: *Proceedings - IEEE International Conference on Semantic Computing 2008, ICSC 2008*. 2008. ISBN: 9780769532790. DOI: 10.1109/ICSC.2008.18.

[113] Artur Hibner and Krzysztof Zielinski. "Semantic-based dynamic service composition and adaptation". In: *Proceedings - 2007 IEEE Congress on Services, SERVICES 2007*. 2007. ISBN: 0769529267. DOI: 10.1109/SERVICES.2007.55.

[114] Nanxi Chen, Nicolas Cardozo, and Siobhan Clarke. "Goal-Driven Service Composition in Mobile and Pervasive Computing". In: *IEEE Transactions on Services Computing* (2018). ISSN: 19391374. DOI: 10.1109/TSC.2016.2533348.

[115] Elif Eryilmaz, Frank Trollmann, and Sahin Albayrak. "Conceptual application of the MAPE-K feedback loop to opportunistic sensing". In: *2015 Workshop on Sensor Data Fusion: Trends, Solutions, Applications, SDF 2015*. 2015. ISBN: 9781467371759. DOI: 10.1109/SDF.2015.7347697.

[116] T. L.M. Van Kasteren, G. Englebienne, and B. J.A. Kröse. "Transferring knowledge of activity recognition across sensor networks". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2010. ISBN: 3642126537. DOI: 10.1007/978-3-642-12654-3_17.

[117] Elif Eryilmaz, Frank Trollmann, and Sahin Albayrak. "An Architecture for Dynamic Context Recognition in an Autonomous Driving Testing Environment". In: *Proceedings - IEEE 11th International Conference on Service-Oriented Computing and Applications, SOCA 2018*. 2018, pp. 9–16. ISBN: 9781538691335. DOI: 10.1109/SOCA.2018.00009.

[118] Marco Lützenberger et al. "A multi-agent approach to professional software engineering". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2013. ISBN: 9783642453427. DOI: 10.1007/978-3-642-45343-4_9.

[119] Christian Cabrera et al. “The Right Service at the Right Place: A Service Model for Smart Cities”. In: *2018 IEEE International Conference on Pervasive Computing and Communications, PerCom 2018*. 2018. ISBN: 9781538632246. DOI: 10.1109/PERCOM.2018.8444606.

[120] Nils Masuch et al. “Decentralized Service Platform for Interoperable Electro-Mobility Services Throughout Europe”. In: *Towards User-Centric Transport in Europe 2: Enablers of Inclusive, Seamless and Sustainable Mobility*. Springer International Publishing, 2020, pp. 184–199. ISBN: 978-3-030-38027-4. DOI: 10.1007/978-3-030-38028-1_13.

[121] Evangelos Triantaphyllou. *Multi-Criteria Decision Making Methods: A Comparative Study*. Vol. 44. 2000. ISBN: 978-1-4419-4838-0. DOI: 10.1007/978-1-4757-3157-6.

[122] Eyhab Al-Masri and Qusay H. Mahmoud. “QoS-based discovery and ranking of Web services”. In: *Proceedings - International Conference on Computer Communications and Networks, ICCCN*. 2007. ISBN: 9781424412518. DOI: 10.1109/ICCCN.2007.4317873.

A

Quality Criteria List

A.1 Quality of Context (QoC) parameter list

Table A.1: QoC parameters compiled from different studies

Parameter	Description	Reference
Accuracy	Correctness rate of measured values to the actual values	[81], [8], [87], [82], [84], [83], [88], [85], [90]
Precision	Equality rate of measured values in repetitions	[80], [87], [82], [83], [88], [85], [89]
Freshness	Up-to-dateness of the data comparing to the current time	[81], [80], [8], [87], [83], [89]
Confidence/Reliability	Level of certainty in the measured values	[81], [82], [88]
Importance/Criticality	Significance level of the measured attributes	[87]
Correctness	Probability of matching between the measured value and real value	[80], [84], [83]

A.2 Quality of Device (QoD) parameter list

Table A.2: QoD parameters compiled from different studies

Parameter	Description	Reference
Sensor type	The type of the sensor such as temperature, presence, etc.	[94]
Location	Geographical location of the sensor	[87]
Battery life	Power consumption of the sensor	[84], [85]
Measurement/sensor range	Spatial scope of the measured value where it is valid	[84], [85], [88]
Resolution	Granularity of the measurements	[80], [81], [84], [83], [88], [85], [89]
Source	Identifier or indicator of the source providing measurement	[84]

A.3 Quality of Service (QoS) parameter list

QWS dataset[1] provides QoS parameters by including 2507 web services [122]. Some of the selected parameters are depicted in Table A.3.

Table A.3: Selected QoS parameters from QWS dataset

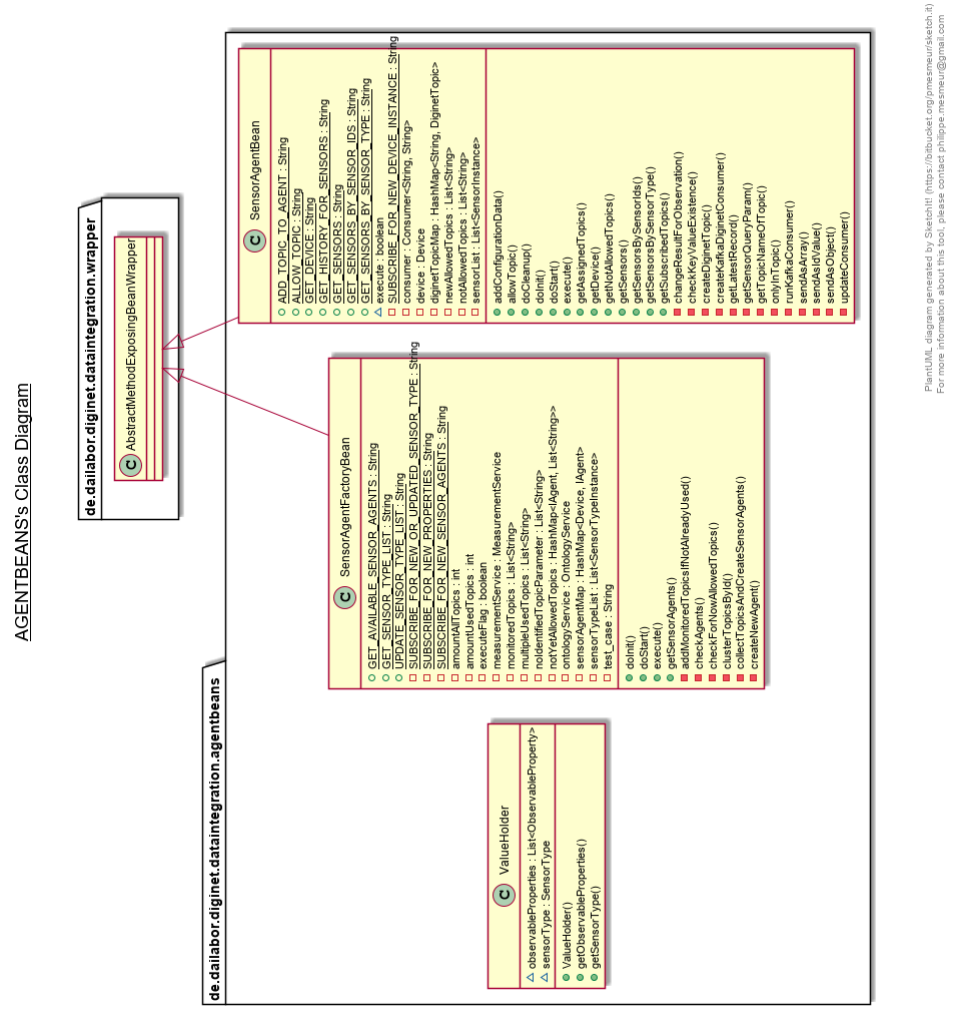
Parameter	Description
Availability	Measurement of time when the service is operable
Successability	The proportion of number of response over number of request messages
Reliability	Measure of how long the service performs its expected behaviour
Response time	Total time it takes from when a request is made until the response is received
Latency	Delay occurred while communicating a message to a service
Throughput	Total number of service invocations for a given period of time

[1]QWS website `https://qwsdata.github.io/`

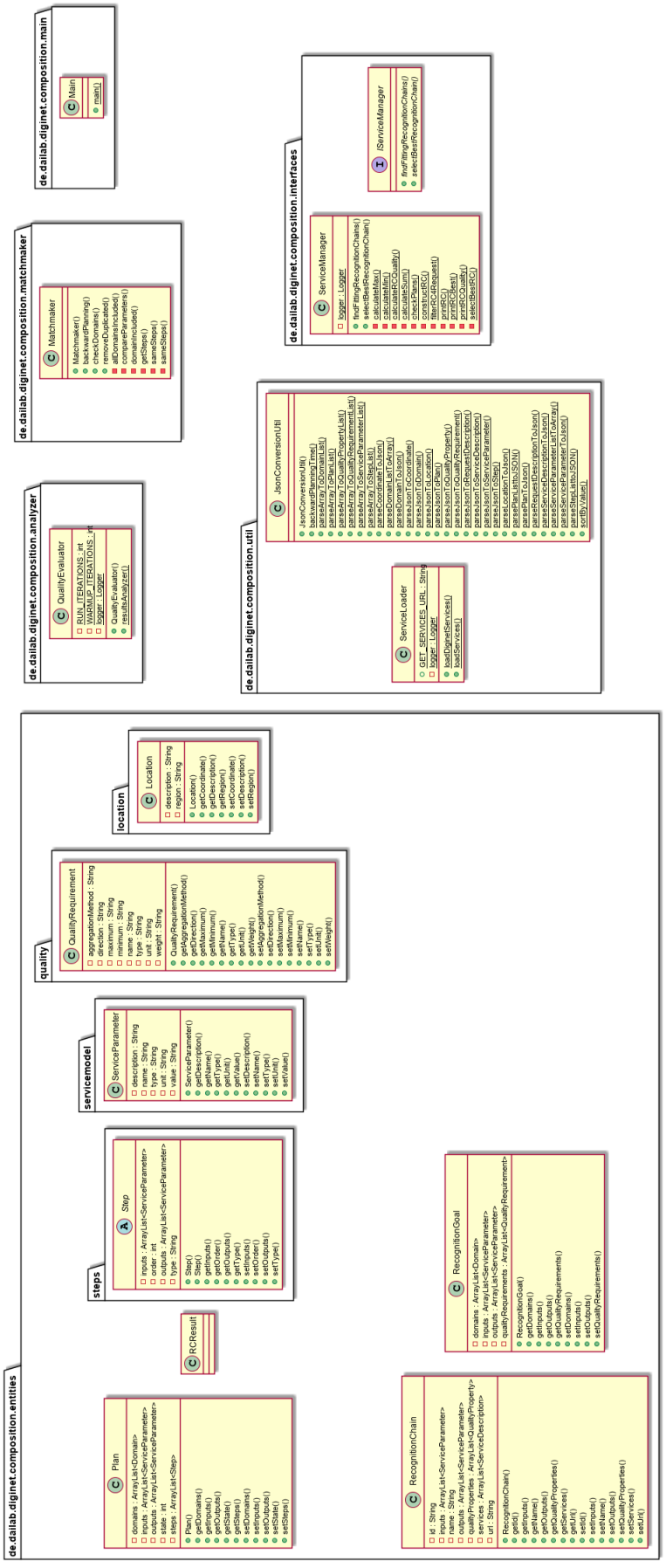
B

Class diagrams

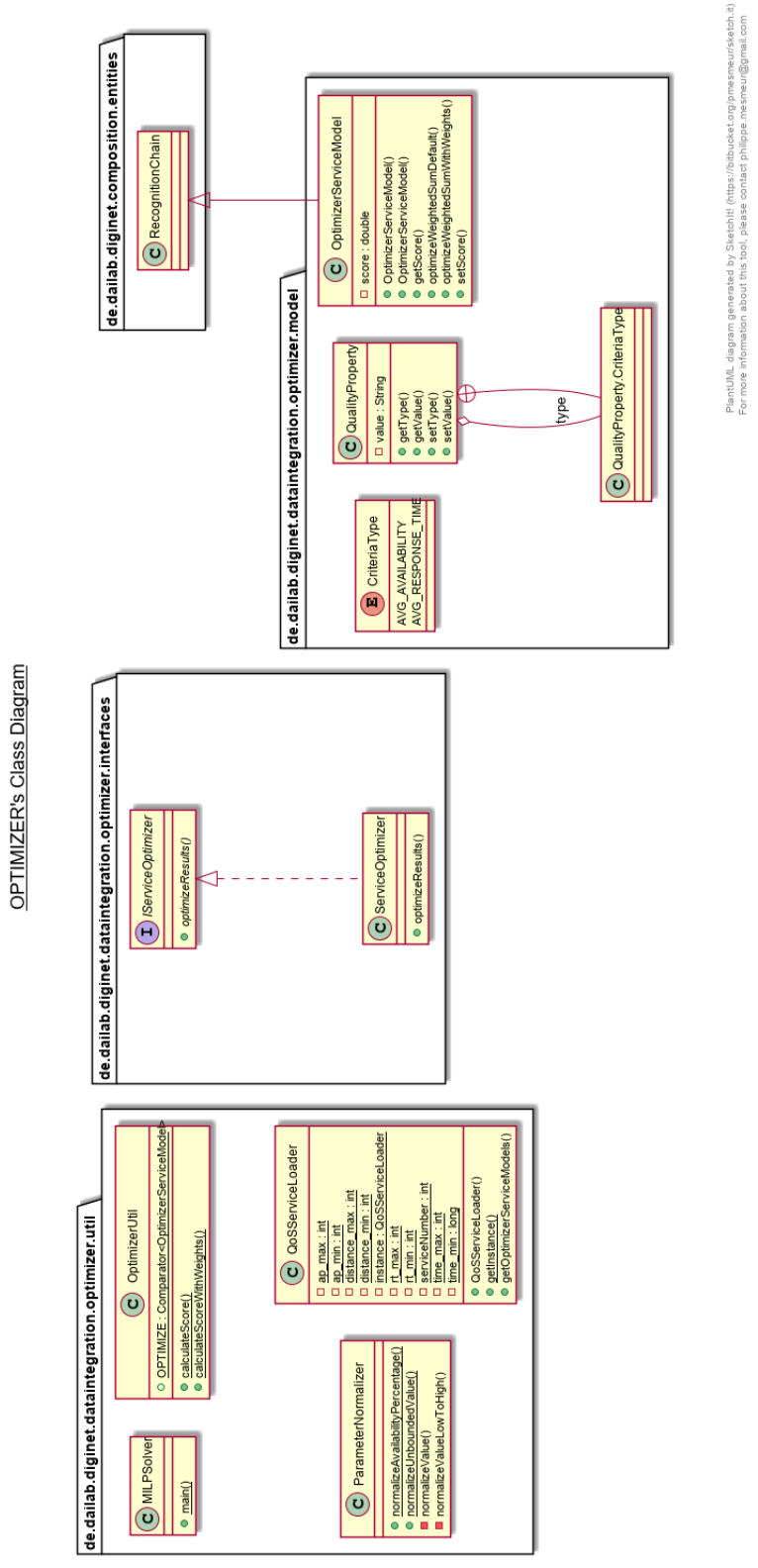
B.1 Class diagram of Agent-based Architecture



B.2 Class diagram of Recognition Chain Creation



B.3 Class diagram of Selection Optimization



C

Data Source Ontology

```
1
2 <?xml version="1.0"?>
3 <rdf:RDF xmlns="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#"
4      xml:base="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor"
5      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
6      xmlns:owl="http://www.w3.org/2002/07/owl#"
7      xmlns:xml="http://www.w3.org/XML/1998/namespace"
8      xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
9      xmlns:sensor="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#
           "
10     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
11    <owl:Ontology rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
          ontologies/sensor"/>
12
13
14
15    <!--
16    ///////////////////////////////////////////////////////////
17    //
18    // Object Properties
19    //
20    ///////////////////////////////////////////////////////////
21     -->
22
```

```
23
24
25
26      <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#attachedTo
            -->
27
28      <owl:ObjectProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#attachedTo">
29          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topObjectProperty"/>
30          <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Sensor"/>
31          <rdfs:range rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Device"/>
32      </owl:ObjectProperty>
33
34
35
36      <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#exposes --
            >
37
38      <owl:ObjectProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#exposes">
39          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topObjectProperty"/>
40          <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Device"/>
41          <rdfs:range rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Service"/>
42      </owl:ObjectProperty>
43
44
45
46      <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#locatedIn
            -->
47
48      <owl:ObjectProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#locatedIn">
```

```
49          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topObjectProperty"/>
50          <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Device"/>
51          <rdfs:range rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Location"/>
52      </owl:ObjectProperty>
53
54
55
56      <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#observes
            -->
57
58      <owl:ObjectProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#observes">
59          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topObjectProperty"/>
60          <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Sensor"/>
61          <rdfs:range rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Observation"/>
62      </owl:ObjectProperty>
63
64
65
66      <!--
67      ///////////////////////////////////////////////////////////
68      //
69      // Data properties
70      //
71      ///////////////////////////////////////////////////////////
72       -->
73
74
75
76
77      <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#deviceId
            -->
78
```

```
79      <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#deviceId">
80          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
81          <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Device"/>
82          <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
83      </owl:DatatypeProperty>
84
85
86
87      <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#effects --
            >
88
89      <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#effects">
90          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
91          <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Service"/>
92      </owl:DatatypeProperty>
93
94
95
96      <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#inputs -->
97
98      <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#inputs">
99          <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
100         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Service"/>
101     </owl:DatatypeProperty>
102
103
104
105     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#latitude
            -->
106
```

```
107     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#latitude">
108         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
109         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Location"/>
110         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
111     </owl:DatatypeProperty>
112
113
114
115     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#longitude
            -->
116
117     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#longitude">
118         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
119         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Location"/>
120         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
121     </owl:DatatypeProperty>
122
123
124
125     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#
            observationName -->
126
127     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#observationName">
128         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
129         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Observation"/>
130         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
131     </owl:DatatypeProperty>
132
133
134
```

```
135     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#
            observationTime -->
136
137     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#observationTime">
138         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
139         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Observation"/>
140         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#long"/>
141     </owl:DatatypeProperty>
142
143
144
145     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#
            observationUnit -->
146
147     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#observationUnit">
148         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
149         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Observation"/>
150         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
151     </owl:DatatypeProperty>
152
153
154
155     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#
            observationValue -->
156
157     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#observationValue">
158         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
159         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Observation"/>
160         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
161     </owl:DatatypeProperty>
```

```
162
163
164
165     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#outputs -->
166
167     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#outputs">
168         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
169         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#Service"/>
170     </owl:DatatypeProperty>
171
172
173
174     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#preconditions -->
175
176     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#preconditions">
177         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
178         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#Service"/>
179     </owl:DatatypeProperty>
180
181
182
183     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#sensorId -->
184
185     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#sensorId">
186         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topDataProperty"/>
187         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#Sensor"/>
188         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
```

```
189     </owl:DatatypeProperty>
190
191
192
193     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#sensorType
            -->
194
195     <owl:DatatypeProperty rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/
            ontologies/sensor#sensorType">
196         <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#
                topDataProperty"/>
197         <rdfs:domain rdf:resource="https://dainas.dai-labor.de/~eryilmaz@dai/
                ontologies/sensor#Sensor"/>
198         <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
199     </owl:DatatypeProperty>
200
201
202
203     <!--
204     ///////////////////////////////////////////////////////////////
205     //
206     // Classes
207     //
208     ///////////////////////////////////////////////////////////////
209      -->
210
211
212
213
214     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#Device -->
215
216     <owl:Class rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/
            sensor#Device"/>
217
218
219
220     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#Location
            -->
221
```

```
222     <owl:Class rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/
            sensor#Location"/>
223
224
225
226     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#
            Observation -->
227
228     <owl:Class rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/
            sensor#Observation"/>
229
230
231
232     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#Sensor -->
233
234     <owl:Class rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/
            sensor#Sensor"/>
235
236
237
238     <!-- https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/sensor#Service --
            >
239
240     <owl:Class rdf:about="https://dainas.dai-labor.de/~eryilmaz@dai/ontologies/
            sensor#Service"/>
241 </rdf:RDF>
242
243
244
245 <!-- Generated by the OWL API (version 4.2.8.20170104-2310) https://github.com/
        owlcs/owlapi -->
```