Universidad de los Andes - Colombia
Systems and Computing Engineering Department

Adaptive architecture for transient IoT systems

A thesis submitted in partial fulfillment of the requirements for a degree of Magister en Ingeniería de Información

By: Jairo Andrés Ariza Castañeda

Promoter(s): Prof. Nicolás Cardozo (Ph.D.)

Prof. Kelly Garcés (Ph.D.)

May 2019

Abstract

Adaptation is crucial for the success of IoT systems since these are part of continuously changing environments. Changes may come from different elements of the architecture underlying an IoT system, as modification to services, or inclusion/exclusion of devices. Existing literature pays special attention to changes in the service layer using evolution agents or context-aware approaches to manage adaptations to said changes. In this Tesis, we present eight challenges that developers face when building adaptive IoT systems. Such challenges take into account changes at the service, middleware, and physical layers. These challenges serve us as a research agenda to foster IoT systems. As a starting point, we design an architecture to deal with the posited challenges. Various of the architectural components are inspired on a reference architecture, and complemented by new components to manage dynamic adaptations in response to the identified challenges. We show how the proposed architecture is used to deal with the first two challenges, inclusion of new devices and devices' transient connection through a service matching mechanism. To demonstrate the feasibility of our solution, we use a domain knowledge database matching instance of services with each other. Our results show a higher match precision than existing approaches.

Contents

Chapter 1

Introduction

IoT systems propose new means for users' interaction by means of software systems augmenting users' physical surrounding environment with a multitude of interconnected sensor and actuator devices [16]. Software systems available in the environment are supported by information fathered from sensors, which is then analyzed and exploited to interact with users through the services or actuators.
The IoT naturally fosters collaboration between different systems. In particular, IoT aided critical systems present a continuous communication between deployed sensors and services in an environment (e.g., to alert users about sensed abnormal situations). Under normal conditions, sensors provide continuous values within a given range dictating normal situations. Anomalous situations, present fluctuations of sensed variables, as a signal to trigger an alarm about the unexpected abnormal situation.
Moreover, IoT devices are present in a multitude of application domains for critical systems, as transport systems, smart grids, smart homes, or Industry 4.0. In these domains, software services interact with and relay on the IoT infrastructure to function properly. As other software-based systems, IoT systems are in continuous evolution, be that due to software updates to the services running on top of the infrastructure, or to hardware updates on devices. Three main problems are identified when developing software systems for such evolving IoT environments:

1. IoT systems have a highly dynamic nature due to the great amount of devices that can be simultaneously connected and the fact that these devices can appear or disappear from the system, in a quick, continuous and unannounced way.

2. Without description of IoT devices properties, external web services are unable to discover suitable IoT devices to establish a connection with their data flows. Even with a proper IoT devices description, it can be problematic for web services to find the right device for their requirements, forbidding seamless connection between web services to IoT devices.

3. The system most cope with its ever-changing surrounding environment, sometimes caused by faulty conditions [10, 36].

In the first case, new running services and IoT devices should be described, to enable their discovery and identification by external web services, through the system. This description should provide information about the functional, performance attributes and availability of the devices, which can be of interest to external users. The system should be able to take such newly collected information seamlessly, to ensure that services are working in accordance to their surrounding environment. The second case, requires a way for the web services to establish an automatic connection with a particular IoT device, that partially or fully comply with specific requirements, supplied by the service. This operation will be supported by the IoT devices description described in the first problem. Similarly, in the third case, new devices may be added to, or removed from the environment unannounced. The environment in which an IoT system is deployed may change for multiple reasons: changes in services' behavior, mobility of services and/or devices, devices o communications failure or multiplicity of information sources. The system should be able to continue working in face of all these situations.
The aforementioned problems evidence that changes in the surrounding environment of IoT systems (both internal and external), require the system to adapt to said changes to continue working seamlessly. However, current (reference) IoT architectures [2] are not equipped to deal with these problems (Chapter 3). The specific requirements to solved the three defined problems are the following and are listed accordingly:

1. **Requirement 1:** A way to describe the functional and non functional (performance) properties of IoT devices and web services connected to the middleware.

2. **Requirement 2:** An algorithm to perform matching between web services and IoT devices.

3. **Requirement 3:** An algorithm to adapt existing connections between IoT devices and web services, when one of them is unavailable.

This Tesis proposes a service-based adaptive architecture for IoT systems (Chapter 5), as an extension of Ariza et al. [1], where we posit the challenges, and present an initial approach of the adaptive architecture as a road-map for adaptive IoT systems. The main concern addressed in such architecture is the adaptation to Inclusion of new IoT devices and resiliency and means of adaptation to transient devices. The architecture consists of two main components (i.e., Instance matching, and Execution Guidepost) that integrate with classical components needed in an IoT architecture (e.g., resolution, fault and state manager components). All these components are distributed among the three layers of an IoT architecture: devices, middleware, and services. The proposed architecture is motivated by a list of challenges (Chapter 2), identified from our experience working in both industry and academic projects, and a literature review of current issues in IoT adaptation. We do not pretend the list of challenges to be exhaustive, but it is an

initial approach to issues when adapting the system, that may be refined in the future.
We put the proposed architecture into practice (Chapter 6) for a particular case-study of a Sustainable Urban Drainage System (SUDS) (Chapter 4), demonstrating the feasibility of our approach in managing introduction and removal of IoT services and devices. The preliminary results for the components associated to address the selected challenges, show an increase in the matching precision with respect to the state of the art, from 0.67 to 0.95. We conclude the Tesis and outline future directions in Chapter 7.

1.1 Main Goal

Develop and implement an architecture to address the issues related with the dynamic nature of IoT environments, focusing on the system's adaptation to inclusion of new IoT devices and transient devices. This architecture must be able to operate within the limits of typical IoT environments, like smart buildings and smart cities.

Chapter 2

Challenges in IoT Systems

This section puts forward eight challenges identified from our experience in developing IoT systems, and the state-of-the-art [10, 36]. Rather than focusing on the technicalities of building a system that uses *IoT devices*, (for example, artifact provisioning) the challenges hereinafter focus on the adaptation aspect of the system. Therefore, the challenges take into account, the actions required to ensure the system's continuous functioning in the face of faults or changes to any of the connected components across the different layers of the IoT architecture. Note that there are existing solutions for some of the proposed challenges. Our proposed architecture is designed to integrate such solutions, with the primary purpose of provide a solution to challenge 1 and 2.

1. **Inclusion of new IoT devices.** As new IoT devices appear in the environment, they should be seamlessly incorporated in the system. If the system cannot recognize new devices, services will be agnostic to new available information. Changes in the topology should be integrated dynamically.

2. **Resiliency to transient IoT devices.** IoT devices may become temporally or permanently unavailable due to a (hardware or software) failure, or users/devices mobility. In such cases, services would become unresponsive, as readings from the surrounding environment are not received. The system should dynamically adapt to deal with these situations.

3. **Data format change in a device.** IoT devices may change the format in which measured data is sent as response to devices' updates, or the replacement of an existing device by another one. Modifying service's format would break the service, as the received data cannot be interpreted, or the service would yield wrong results. Both entities should change in unison.

4. **Middleware Unavailability.** The *Middleware* layer may become unavailable due to maintenance or malfunction. If the *Middleware* is not available, communication between devices and services is lost. In such a case, the communication mechanism between services and devices must adapt to search

for new components (re-)enabling the interaction between physical and virtual entities.

5. **Authorization and authentication.** As discussed in Challenge 2, new IoT devices should be incorporated in the system. In addition to functionality, security aspects should be taken into account too. The devices should be authorized and authenticated prior to establishing new connections, to monitor and regulate all the devices in the system.

6. **Service connectivity loss.** There are multiple situations in which connectivity with a service might be lost. For example, due to connectivity problems or services leaving the environment. All of these situations cause data sent by IoT devices not to reach their associated services. The system should adapt to discontinue the service.

7. **Abnormal information from different devices.** Due to the nature of collaborative systems, a variation of the measured variable is expected in case of abnormalities. In such cases, the IoT device must adapt its behavior, to properly analyze the environment.

8. **Services' API change.** Services evolve over time. As developers modify services' APIs, these may become unresponsive as the data sent from the IoT devices may no longer be consumable. In such cases, the Middleware should be able to search for other potential suitable devices to supply services' data.

2.1 Challenges Adapting IoT Systems

Each of the eight challenges, are mapped to specific tasks to execute in our IoT architecture in Figure 5.7. The following adaptation challenges present the technical characteristics to identify and tackle each of the challenges, from an dynamic adaptation perspective.

2.1.1 AC.1 —Adaptation to Inclusion of New *IoT Devices*

Each IoT device has an identification mechanism sent to the Status Manager (Resource discovery) in the *Middleware*. This mechanism contains information describing the IoT device functions and performance properties, enabling discovery and registry of the device. With device's description, the *Middleware* can match external services with the most suitable group of IoT devices, based on their performance properties and the external users/services' needs.

2.1.2 AC.2—Adaptation to Transient *IoT Devices*

The IoT devices connected to the *Middleware*, must send a status signal in a regular basis. This way, the Status Manager (Healthcheck controller) in the *Middleware*, can identify the device and its current state. By monitoring the registry

of suitable IoT devices to a Service and its state, the *Middleware* can almost seamlessly reconnect the Service with the next most admissible IoT device, annotated on the registry.

2.1.3 AC.3—Adaptation to Format Changes from IoT Devices' Sent Data

There are two possible solutions to adapt in this scenario. First, the *Middleware* blocks the receiving port of the affected IoT device, to avoid the improper data format. The second possibility, consist of a format translation service developed to convert the improper data format into a suitable data for the service. The translation service includes a semantic and a syntax approach to solve this issue.

2.1.4 AC.4—Adaptation to *Middleware* Temporary or Permanent Unavailability

In this scenario there are two adapting layers due to *Middleware* unavailability. The first one is the *Physical layer* and the second one is the *Service layer*. In both cases the adaptation is similar, the *Middleware* is replicated to ensure that the IoT devices and services can connect with each other. When the associated *Middleware* is not available due to a malfunction, both devices and the services will look for a new *Middleware* to connect to.

2.1.5 AC.5—Adaptation of Services in the Event of a Cybernetic Attack

The required adaptation in this case, is to block the port associated with the IoT device generating the attack, to prevent a service malfunction due to corrupted data.

2.1.6 AC.6—Adaptation to Loss of Network Connection to the *Services*

The *Middleware* has better hardware capacities than IoT devices. These characteristic allows the *Middleware* to assume the functions of service, at least for a limited time, to mitigate temporal disconnection of services. For example, when there is no connection to a Web Service, the *Middleware* takes control of the communication by persisting the measured data. Moreover, the *Middleware* can analyze the sensed data to trigger an action in the system (*e.g.,* sounding an alarm). The extend of the *Middleware*'s intervention depends on the device's memory capacity.

2.1.7 AC.7—Adaptation to Abnormal Values of the Measured Variable

To save devices' battery and to optimize data storage, the sample rate determined to measure a physical variable is low. There are other situations, where systems are categorized as a low demand, i.e., it is not expected that variables reach a predetermined threshold in a very long time. However, service can detect measurement perturbations and it request IoT devices to increase their sample rate, allowing the service to perform better analysis of abnormal situations.

Chapter 3

Related Work

This section puts existing approaches that address ontology matching application and domain specific representations in the IoT systems perspective. Here we discuss the related work in these areas.

3.1 Systematic literature review

To develop the related work, we queried the Scopus database. We used several keywords related to the IoT domain, ontologies representation and matching, web services and Domain Specific Language (DSL). The relevance of the founded documents was determined by the following criteria:

1. Documents published in the last 5 years or less,
2. most cited documents, and
3. document relevance to the defined keywords.

The keywords used on the queries and the number of documents founded are the following:

1. "IoT ontology matching", 8 relevant over 19 documents
   founded.
2. "IoT Semantic Sensor Network (SSN)", 3 relevant over 32 documents founded.
3. "Ontology instance matching", 2 relevant over 3141 documents founded.
4. "Quality of Service (QoS) ontology matching", 2 relevant over 108 documents founded.
5. "Universal Description, Discovery, and Integration (UDDI)", 3 relevant over 20 documents founded.
6. "UDDI matching web services" 2 relevant over 173 documents founded.
7. "DSL Dustdar", 3 relevant over 64 documents founded.

3.2 Knowledge representation

Domain specific language description has extensive research with different approaches, like DSL, the UDDI and ontologies. From this approaches, Ontologies gather more attention from researchers, specially to describe IoT systems and its properties.
Ganzha et al. [8] identify the interoperability nature of heterogeneous IoT platforms and the need of a common method to represent the attributes of those platforms. In many cases the devices are represented using a format like the JavaScript Object Notation (JSON) or the Extensive Markup Language (XML). However the authors propose ontologies as a more universal representation enriched by the semantic meanings it provides. Ganzha et al. [8] recognize ontology matching as a suitable method to align concepts provided by IoT systems.
DSLs as used by Copil et al. [6] and Oberortner et al. [19] describe domains' specific knowledge. DSLs also define applicable rules with constraints to trigger specific actions when these are violated. Copil et al. [6] defines a constraint directive that stars with the keyword CONSTRAINT, and uses mathematical comparison signs (<,>,>=,<=,!=,==) to establish which values are acceptable. Moreover, Copil et al. [6] use DSL to control elasticity on cloud applications based on performance requirements like cost, resource performance and QoS. Oberortner et al. [19] also apply a QoS DSL to trigger alarms when web services' Service Level Agreements (SLA) gets violated.
Web services discovery (Curbera et al. [7]) is an existing technology, based on the UDDI directory, the Web Services Description Language (WSDL) format and Simple Object Access Protocol (SOAP). This technology allows registry, description and discovery of web services using syntactic matching. Due to the lack of expressivity, Paolucci et al. [21] and Purohit and Kumar [24] proposed a semantic extension of the WSDL and UDDI to provide semantic matching of the web services, to improve the matching results. Ran [25] and Seghir and Kazar [27] also modify the WSDL and UDDI with QoS attributes extend the discovery functionality to the non-functional domain.
In IoT systems, ontologies are nowadays, the most used method to describe this domain. The two most recognized IoT ontologies are the SSN [5] and the ETSI Smart Appliances REFerence (SAREF) [17]. Both of them provide functional properties description, but only SSN provides the required non-functional description required by our research.
Ontology matching and Ontology instance matching has raised the attention of many researchers and there is a great amount of research on this topic. Recently the raise of IoT systems originated an additional interest on applying the ontology knowledge in different contexts and applications of the IoT domain.
Castano et al. [3] describe the main ontology and instance matching techniques independent of the knowledge domain, which helps to align the functional attributes of the represented objects. The techniques include similarity-based methods, and reasoning-based methods. The similarity-based methods includes the syntactic

approach using string matching algorithms and the semantic technique involves the use of a thesaurus or a lexical system (e.g., Wordnet). The reasoning-based approach embraces deductive (i.e., description logics) and probability methods (i.e., machine learning) to achieve ontology instance matching.
Junhao et al. [13] propose an algorithm for web services selection based on a ontology that describes nonfunctional parameters (i.e, QoS). The paper states the upper and middle QoS ontology for web services and how it can be used to execute web services matching bases on specific QoS requirements. Tran [35] also developes a QoS ontology for web services, which describes the QoS information and the related attributes in detail. Junhao et al. [13] use an algorithm that receives as input the QoS requirements from the user and performs a semantic matching, data verification, value matching and a personalized adjustment (i.e, weights adjustment) to determine the degree of similarity between web services.
Tang and Meersman [32] implement an ontology based discovery and a recommender system for IoT projects, where the user specifies its requirements and the recommender systems suggests similar existing solutions that satisfy users' needs. The ontology matching solution consists on a string matching algorithm (syntactic similarity), followed by a lexical matching algorithm supported by Wordnet (semantic similarity), and a graph matching algorithm that uses a lexons (sentences represented as vectors) table. The objective is to determine suitable recommendation according to the user's needs.
Tao et al. [33] describe IoT devices data, provided on different formats, as individuals of an ontology. The individuals represent historic data of the same Smart Home environment and the authors use ontology instance matching to adapt the Smart Home to changes on the user requirements and behavior. The ontology instance matching algorithm uses semantic similarity calculation of the individuals and their parent, children and siblings' nodes.
Table 3.1 summarizes the knowledge representation state of the art. We have the main topics (DSL, WSDL, QoS and Semantic ontologies) related to this subject and which authors applies each approach on his research. In our research, we decided to apply a combination of knowledge representation, consisting of QoS ontologies and semantic ontologies to represent IoT devices and web services.

3.3 Ontology matching

Wang et al. [37] use ontology matching to align heterogeneous IoT systems. The proposed algorithm executes a semantic similarity matching, a syntactic similarity matching and weight adjustment calculation in parallel. The weight adjustment matrix is determined by calculating the similarity degree of the ontologies hierarchy. The calculated matrices are aggregated in a combined similarity matrix, which determines the final degree of alignment.
Zhou and Ma [41] present a service discovery algorithm for the IoT domain. The algorithm combines semantic similarity and semantic relativity to calculate the similarity of different concepts represented on the ontology. Zhou and Ma [41] also

	DSL	WSDL / UDDI	QoS	Semantic ontologies
Ganzha et al. [8]				✓
Copil et al. [6]	✓		✓	
Oberortner et al. [19]	✓			
Curbera et al. [7]		✓		
Paolucci et al. [21]		✓		
Purohit and Kumar [24]		✓		
Ran [25]		✓	✓	
Seghir and Kazar [27]		✓	✓	
Compton et al. [5]				✓
Moreira et al. [17]				✓
Junhao et al. [13]			✓	
Tran [35]			✓	
Tran et al. [34]				✓

Table 3.1: Knowledge representation papers.

implement weights to adjust the parameters compared on the matching algorithm.
Kibria et al. [14] use semantic ontologies to represents IoT devices as Virtual Objects (VO) and Composite Virtual Objects (CVO), and enable the reuse of existing objects to render multiple heterogeneous IoT devices. When a new device is registered, semantic similarity between the attributes of the new IoT device and the existing VO is calculated using a synsets dictionary. If there is a match with an existing VO or with parts of different VO, the algorithm instantiates the new device on the existing objects. In case there is no match, a new VO is created on the ontology.
Song et al. [31] apply ontology matching to provide a recommendation system for task and work flow based service creations on a Smart building environment. When the user wants to create a new task, the recommendation system search for similar IoT devices based on user requirement for the task. The matching is executed calculating a semantic similarity between the requirements and the existing instances of the ontology, and comparing the hierarchical structure of the ontology using a semantic approach. Additionally, if there is an unavailable IoT device, the system will reconnect the service with a new similar suitable replacement.
Moreira et al. [17] explore a semantic translation to map two specific IoT ontologies, the SSN ontology and SAREF ontology. This article states that the translation between the two ontologies, cannot be fully performed due to the fact that the SAREF ontology does not provide a similar structure to the SSN Measurement Capability and System attributes classes to describe the non-functional attributes of an IoT device.
Table 3.2 summarizes the related authors and the main topics of research on matching algorithms. These topics are semantic, syntactic, ontology, ontology

instances matching and weights adjustments on matching algorithms. We combine once again a several approaches in our research, ontology instance matching, syntactic matching and weight adjustments. We extended these approaches by incorporating SSN system properties on matching operations and weight adjustments based on the control systems domain.

	Semantic	Syntactic	Weight adjustments	Ontology	Ontology instances
Ganzha et al. [8]				✓	
Paolucci et al. [21]	✓				
Purohit and Kumar [24]	✓				
Castano et al. [3]	✓	✓		✓	✓
Junhao et al. [13]			✓		✓
Tran [35]					✓
Tang and Meersman [32]	✓	✓		✓	
Tao et al. [33]	✓				✓
Wang et al. [37]	✓	✓	✓	✓	
Zhou and Ma [41]	✓		✓		
Kibria et al. [14]	✓				✓
Song et al. [31]	✓			✓	
Moreira et al. [17]				✓	

Table 3.2: Matching algorithms papers.

3.4 IoT adaptation

Evolution Agents (EVAs) [34] consist of a notification management architecture exploiting a semantic service registry based on ontologies. Services are described and registered in the registry using semantic web ontologies. The registry's main function is to provide clients and EVAs with suitable information about a required service. This information is used by the evolution algorithm to compare different versions of a service, in order to adapt it (automatically or manually). EVAs detect and coordinate the evolution of the services available in the registry. Whenever a service changes, the EVA associated with the service reports the change to all dependent services; triggering their EVAs to evolve accordingly. Tran et al. [34] describe a scenario where several IoT devices from different suppliers provide different services. Each service in the environment is equipped with an EVA. The EVA analyzes the changes and if possible, it will automatically update the service with a suitable version, provided by its own repository or by another EVA. In case the version evolution cannot take place, the EVA notifies the developer (to perform the adaptation manually).
Context-aware adaptation is also explored for the IoT domain. Gaur et al. [9] propose a web framework to adapt IoT systems in response to context changes (*e.g.,* sensed information, networks, or power supplies). Developers can define resource-independent applications through a mT-Res. The mT-Res architecture implements modular self-contained code, that can be moved from one resource to a similar one, when a change is found in the context Gaur et al. [9]. The two main

components in this architecture are the Resource Administrator, and the Applications Manager. The Resource Administrator component monitors the context from sensors, and detects variations in the environment. Upon variations, the component informs the Applications Manager to deploy the functional blocks of code in another resource with the same capabilities of the affected one, effectively adapting the resource to the environment.
Pötter and Sztajnberg [23] provide a context-aware approach based on ontologies to describe resources provided by Physical devices or even virtual components. With the description provided by the resources' ontology, the *Middleware* handles *IoT devices*' data distribution, and connection security.
For IoT adaptation, Table 3.3 shows a summary of related authors and his research topic, EVAs and Context aware adaptation. In this case, we applied a similar approach to EVAs, the Execution Guidepost. These two are based on a registry to provide adaptation. The difference is, the Execution Guidepost uses the registry to denote how the connections are bonded and how can be adapted, nonetheless EVA describes different version of the services to provide the required adaptive updates.

	EVA	Context aware
Tran et al. [34]	✓	
Gaur et al. [9]		✓
Pötter and Sztajnberg [23]		✓

Table 3.3: IoT adaptation papers.

Chapter 4

Case study: Sustainable Urban Drainage System

A Sustainable Urban Drainage System (SUDS) aims to mitigate the negative impacts cause by the reduction of natural water infiltration in urban areas. Such reduction increments rain runoff that stresses drainage systems increasing the risk of flooding. Drainage systems are also used to improve water quality. However, the typologies of SUDS must be evaluated to assure that they are optimal for a specific site. As a consequence, monitoring water quality parameters on SUDSs is crucial to produce a solid analysis on their efficiency.

In 2016, the Universidad de los Andes started a research of SUDS technologies, caused by the need to find new drainage systems that improve the quality of rain water that flows to natural water bodies, and prevent floods in the city of BogotÃ¡. The Research Center of Environmental Engineering (Centro de Investigacion de Ingenieria Ambiental (CIIA) by its initials in Spanish) started the design, construction, and monitoring of a pilot SUDS in the Metropolitan Park of San Cristobal, located at the south of BogotÃ¡. The park is additionally surrounded by the Fucha river. The flow of the river can be diverted to the SUDS, in order to absorb the excess of water in the river during the rain seasons, therefore preventing flooding to the nearby neighbors.

The SUDS pilot aims to evaluate the reduction capacity of runoff volumes and peak flows. First the water flow is channeled by an inlet chamber. Next, its typology consists on a train treatment composed by a sequence of two typologies. The first is a grassed swale with a drainage area of 1.48 Ha, a surface width of 2.80 m and a length of 70 m. This treats and transports the captured runoff volume to the next typology. The second typology is a dry extended detention basin with a drainage area of 1.56 Ha, a mean surface width of 16.33 m and a length of 49 m. This temporarily stores and treats the runoff and releases it first throughout infiltration and the remaining flows to an outlet chamber drainage structure downstream the system.

The IoT physical layer consisted of two kinds of devices, one to report general weather conditions to the application layer of the system, and the other one has

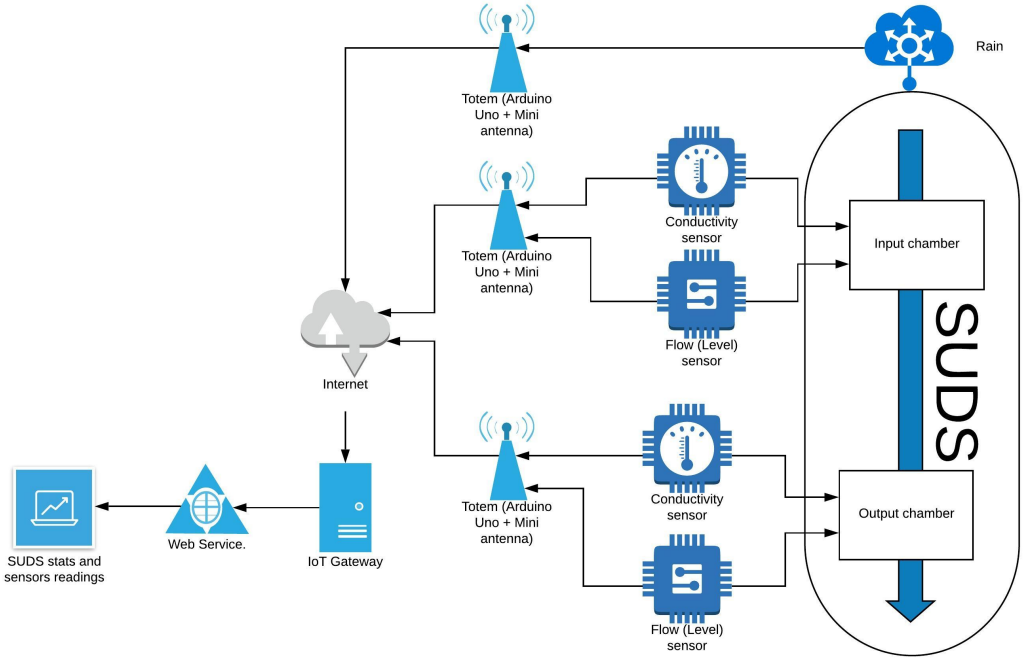

Figure 4.1: Sustainable Urban Drainage System SUDS.

been implemented twice to be located in the input/output chambers of the SUDS. The general information IoT device contains a tipping bucket rain gauge, for monitoring rain rate and total rainfall. Each chamber IoT device consists of a water level transmitter, to infer the water flow rate of the drainage system at each point and a conductivity transmitter that measures water's electrical conductivity for quality monitoring purposes.

For each node, we use an Arduino IoT device connected to the internet using wireless communication, with its corresponding power supply and the described sensors. This device also generates two rainfall alarms, to indicate the beginning and ending of a rain event to the application layer.

Figure 4.1 shows a context diagram for the implemented SUDS. The purpose of our case study is to provide the CIIA a way to remotely monitor the variables of the SUDS, persist data and analyze performance of the system. Two of the main concerns are to quantify rainfall during the year, and the volume of drained water by the SUDS. The IoT system middleware will provide connection from sensors readings to the CIIA web service, according to users' needs and requirements. However, in order to subscribe a web service with the sensors' data flows, the IoT system must deal changing environments. New devices can be installed and need to be included on the system and included devices can disappear, due to external conditions like climate(*e.g.,* rain, heat, electric discharges), voluntary or involuntary actions from persons, electric power supply and lost of communications.

Chapter 5

Solution

This chapter describes the developed solution to solve the considered challenges. Section 5.1 to Section 5.4, specify the technologies applied in the solution, and Section 5.5 depicts how this technologies are implemented on the adaptive architecture. This implementation is also accessible on the following repository[28].

5.1 IoT devices description

To solve the challenges, inclusion of new devices and resiliency to transient IoT devices, first we must describe the different entities in the environment. We investigate several methods to describe domain specific knowledge. In our state of the art development, we found three main approaches:

1. The DSL [6].
2. The WSDL format in conjunction with the UDDI directory and the SOAP protocol [7].
3. Semantic ontologies [5].

Our chosen solution is to describe IoT devices, by using semantic ontologies. The WSDL approach is discarded because the lack of semantic capabilities of this description. Compared with the DSL approach, the semantic ontologies provide a richer semantic meaning of the attributes of a domain specific language.
In addition to the selected description method, there are another two important aspects to describe an IoT device, its functional and non-functional attributes. The functional attributes describe aspects related to the behavior of the device. The non-functional attributes specify criteria used to judge the performance of a system or device. We define an ontology based on a combination of the SSN ontology [5], the Service Integration Ontology (SIO) ontology [26] and a QoS ontology [35].
The SSN ontology is an ontology implemented to describe the functional, non-functional attributes and observations of IoT devices, that covers sensors and

actuators. The SSN ontology includes a lightweight self-contained core ontology ontology called Sensor, Observation, Sample, and Actuator (SOSA), which provides a formal general-purpose specification for modeling the interaction between the entities involved in the acts of observation, actuation, and sampling. The SOSA ontology embraces most of the functional properties of the devices, and the main used classes on this Tesis are the following, extracted from World Wide Web Consortium (W3C) [38]:

1. **Actuation.** An Actuation carries out an (actuation) Procedure to change the state of the world via an Actuator.

2. **Actuator.** A device that is used by, or implements, an (actuating) Procedure. Actuators change the state of the world.

3. **Observation.** Activity of carrying out an (observation) Procedure to estimate or calculate a value of a Property of a FeatureOfInterest. Links to a Platform or Sensor to describe what made the Observation and how; links to an ObservableProperty to describe what the result is an estimate of, and to a FeatureOfInterest to detail what that property was associated with; the Result is the output.

4. **Procedure.** A workflow, protocol, plan, algorithm, or computational method specifying how to set up an Observation, take a Sample, or make a change to the state of the world (via an Actuator). A Procedure is re-usable, and might be involved in many observations, samplings, or actuations. It explains the steps to be carried out to arrive at reproducible results. Put differently, millions of observations may be created via the same Procedure the same way as millions of tables are assembled using the same instructions (as information objects, not their concrete realization).

5. **Sensor.** Device, agent (including humans), or software (simulation) involved in, or implementing, a (sensing) Procedure. Sensors responds to a stimulus, e.g., a change in the environment, and generates a Result. Sensors can be mounted on platforms, e.g., a modern smart phone hosts multiple sensors

6. **FeatureOfInterest.** The feature whose ObservableProperty is being observed by a Sensor to arrive at a Result. For example, when measuring the height of a tree, said height is the ObservedProperty, 20m may be the Result of the Observation, and the tree is the FeatureOfInterest.

7. **ObservableProperty.** An observable Quality of a thing; typically a FeatureOfInterest.

8. **Sample.** - Feature on which observations may be made, which is intended to be representative of a FeatureOfInterest that is not fully accessible. Samples are artifacts of an observational strategy, and have no significant function

outside of their role in the observation process. The physical characteristics of the features themselves are usually of little interest, except perhaps to the manager of a sampling campaign.

The most important properties of the SOSA ontology are the following, specified in World Wide Web Consortium (W3C) [38]:

1. **hasFeatureOfInterest.** A relation between an Observation and the entity whose quality was observed. For example, in an Observation of the weight of a person, the feature of interest is the person and the quality is weight.

2. **hasSample.** Relation between a FeatureOfInterest and the Sample used to represent it.

3. **hasValue.** The value of a Result, *e.g.,* 23 or true.

4. **isFeatureOfInterestOf.** - A relation between a FeatureOfInterest and an Observation about it.

5. **isObservedBy.** Relation between an ObservableProperty and the Sensor able to observe it.

6. **isSampleOf.** Relation from a Sample to the FeatureOfInterest that it is intended to be representative of.

7. **madeObservation.** Relation between a Sensor and an Observations it has made.

8. **observedProperty.** Relation linking an Observation to the Property that was observed. The observedProperty should be a Property (hasProperty) of the FeatureOfInterest (linked by featureOfInterest) of this observation.

9. **observes.** Relation between a Sensor and an ObservableProperty.

The non-functional properties contained on the SSN ontology are the System Properties. These properties describe the performance measurements or actions provided by the IoT devices. The main used classes are described bellow, using excerpts from World Wide Web Consortium (W3C) [39], and Figure 5.1 illustrates these classes and its relations.

1. **System.** System is a unit of abstraction for pieces of infrastructure that implement Procedures. A System may have components, its subsystems, which are other Systems.

2. **Property.** A quality of an entity. An aspect of an entity that is intrinsic to and cannot exist without the entity.

3. **SystemCapability.** Describes normal measurement, actuation, sampling properties such as accuracy, range, precision, etc. of a System under some specified Conditions such as a temperature range. The capabilities specified here are those that affect the primary purpose of the System, while those in OperatingRange represent the system's normal operating environment, including Conditions that don't affect the Observations or the Actuations.

4. **Condition.**Used to specify ranges for qualities that act as conditions on a system/sensor's operation. For example, wind speed of 10-60m/s may be used as the condition on a SystemProperty, for example, to state that a sensor has a particular accuracy in that condition.

5. **SystemProperty.** An identifiable and observable characteristic that represents the System's ability to operate its primary purpose: a Sensor to make Observations, an Actuator to make Actuations, or a Sampler to make Sampling.

6. **Measurement Range.** The set of values that the Sensor can return as the Result of an Observation under the defined Conditions with the defined system properties.

7. **Actuation Range.** The range of Property values that can be the Result of an Actuation under the defined Conditions.

8. **Accuracy.** The closeness of agreement between the Result of an Observation (resp. the command of an Actuation) and the true value of the observed ObservableProperty (resp. of the acted on ActuatableProperty) under the defined Conditions.

9. **Detection Limit.** An observed value for which the probability of falsely claiming the absence of a component in a material is beta, given a probability alpha of falsely claiming its presence.

10. **Drift.** As a Sensor Property: a continuous or incremental change in the reported values of Observations over time for an unchanging Property under the defined Conditions. As an Actuator Property: a continuous or incremental change in the true value of the acted on ActuatableProperty over time for an unchanging command under the defined Conditions.

11. **Frequency.** The smallest possible time between one Observation, Actuation, or Sampling and the next, under the defined Conditions.

12. **Latency.** The time between a command for an Observation (resp. Actuation) and the Sensor providing a Result (resp. the Actuator operating the Actuation), under the defined Conditions.

13. **Precision.** As a sensor capability: The closeness of agreement between replicate Observations on an unchanged or similar quality value: i.e., a measure of a Sensor's ability to consistently reproduce an Observation, under the defined Conditions. As an actuator capability: The closeness of agreement between replicate Actuations for an unchanged or similar command: i.e., a measure of an Actuator's ability to consistently reproduce an Actuations, under the defined Conditions.

14. **Resolution.** As a Sensor Property: the smallest difference in the value of a ObservableProperty being observed that would result in perceptably different values of Observation Results, under the defined Conditions. As an Actuator Property: the smallest difference in the value of an Actuation command that would result in a value change of the ActuatableProperty being acted on, under the defined Conditions.

15. **Response time.** As a Sensor Property: the time between a (step) change in the value of an observed ObservableProperty and a Sensor (possibly with specified error) 'settling' on an observed value, under the defined Conditions. As an Actuator Property: the time between a (step) change in the command of an Actuator and the 'settling' of the value of the acted on ActuatableProperty, under the defined Conditions.

16. **Selectivity.** As a Sensor Property: Selectivity is a Property of a Sensor whereby it provides observed values for one or more ObservableProperties such that the Result for each ObservableProperty are independent of other Properties in the FeatureOfInterest being investigated, under the defined Conditions. As an Actuator Property: Selectivity is a Property of an Actuator whereby it acts on one or more ActuatableProperties such as the Results for each ActuatableProperty are independent of other Properties in the FeatureOfInterest being acted on, under the defined Conditions.

17. **Sensitivity.** As a Sensor Property: Sensitivity is the quotient of the change in a Result of Observations and the corresponding change in a value of an ObservableProperty being observed, under the defined Conditions. As an Actuator Property: Sensitivity is the quotient of the change in a command of Actuation and the corresponding change in a value of an ActuatableProperty being acted on, under the defined Conditions.

According to World Wide Web Consortium (W3C) [39], the properties that relates the System properties are:

1. **hasProperty.** Relation between an entity and a Property of that entity.

2. **hasSystemCapability.** Relation from a System to a SystemCapability describing the capabilities of the System under certain Conditions.

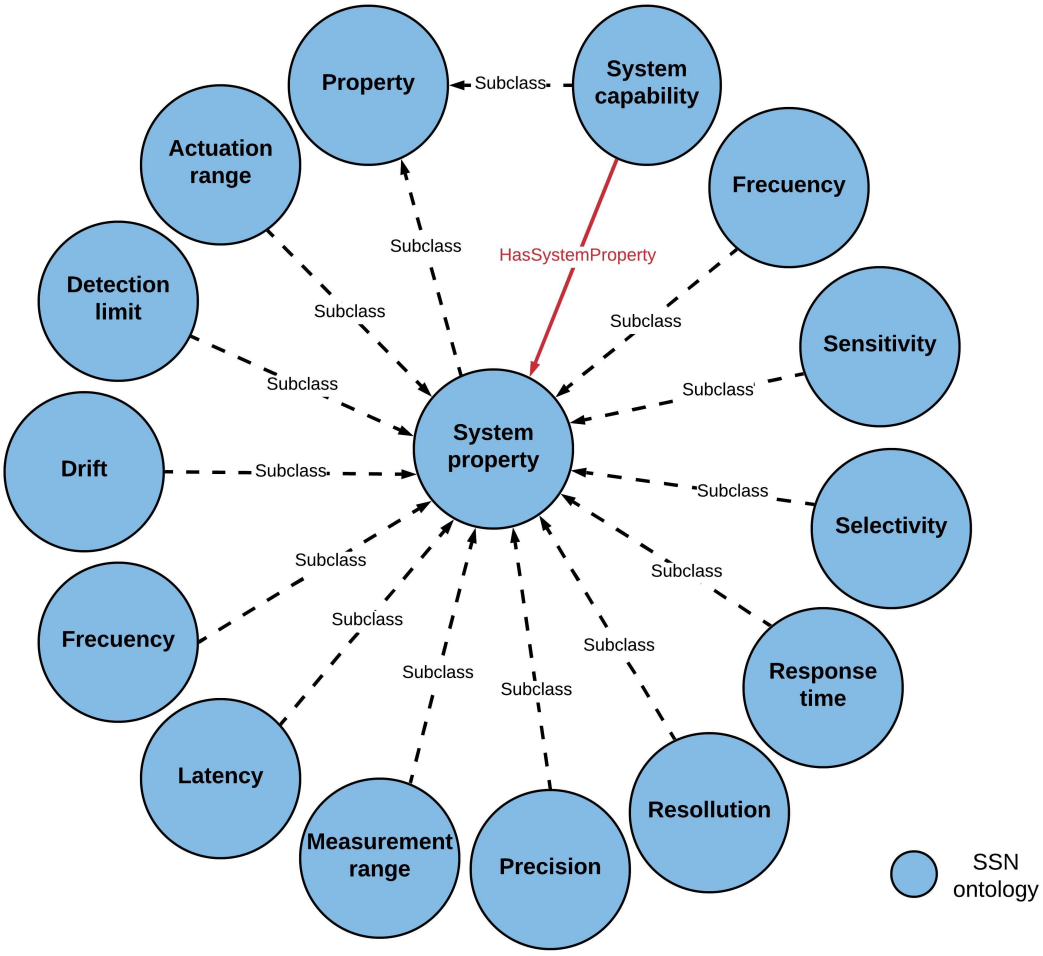

Figure 5.1: Non-functional attributes (System Property) of IoT devices, represented by the SSN ontology [39].

3. **hasSystemProperty.** Relation from an SystemCapability of a System to a SystemProperty describing the capabilities of the System.

To complete the non-functional properties, a QoS ontology [13] was integrated to describe the non-functional properties of web services. Figure 5.2 depicts the QoS ontology, and the following paragraph extracted from Yin et al. [40], provides a description of the most relevant QoS properties:

1. **Availability.** The probability of web services can access successfully. It calculates based on customer feedback.

2. **Capacity.** The number of concurrent requests a service allows. When the service goes beyond its service capacity to work, its reliability and availability will be adversely affected.

3. **Economy.** The user/service, in order to obtain services in time and space (memory), must pay for them. The main feature is the use of cost.

4. **Interoperability.** It defines such a service that is compatible with the standard. This is very difficult for consumer program or Agent and services operation.

5. **Performance.** It is able to provide services to the limit of measurement. It may be based on throughput and response time. The subclass of Performance is Throughtput.

6. **Reliability.** It indicates the ability to correctly carry out their functionalities. It is a comprehensive evaluation of the quality of service.

7. **Reputation.** Mainly depends on users' experiences of using it and evaluates the credibility of user reports.

8. **Robustness.** It is the fault tolerance when you input non-normal data and invoke service incorrectly.

9. **Scalability.** Defines whether the service capacity can increase as the consumer's requirement.

10. **Security.** Validation involves parties which is used to encrypt messages, provide access control, confidentiality and non-repudiation.

11. **Stability.** It is the change rate of service's attributes, such as its service interface and method signatures.

To complete the proposed ontology, the SIO ontology [26] is used to describe web services. Figure 5.3 illustrates an excerpt from the modified proposed ontology. In our context, we integrate this ontology to describe the data stream provided by the IoT device as a web service and the functional attributes of the entity

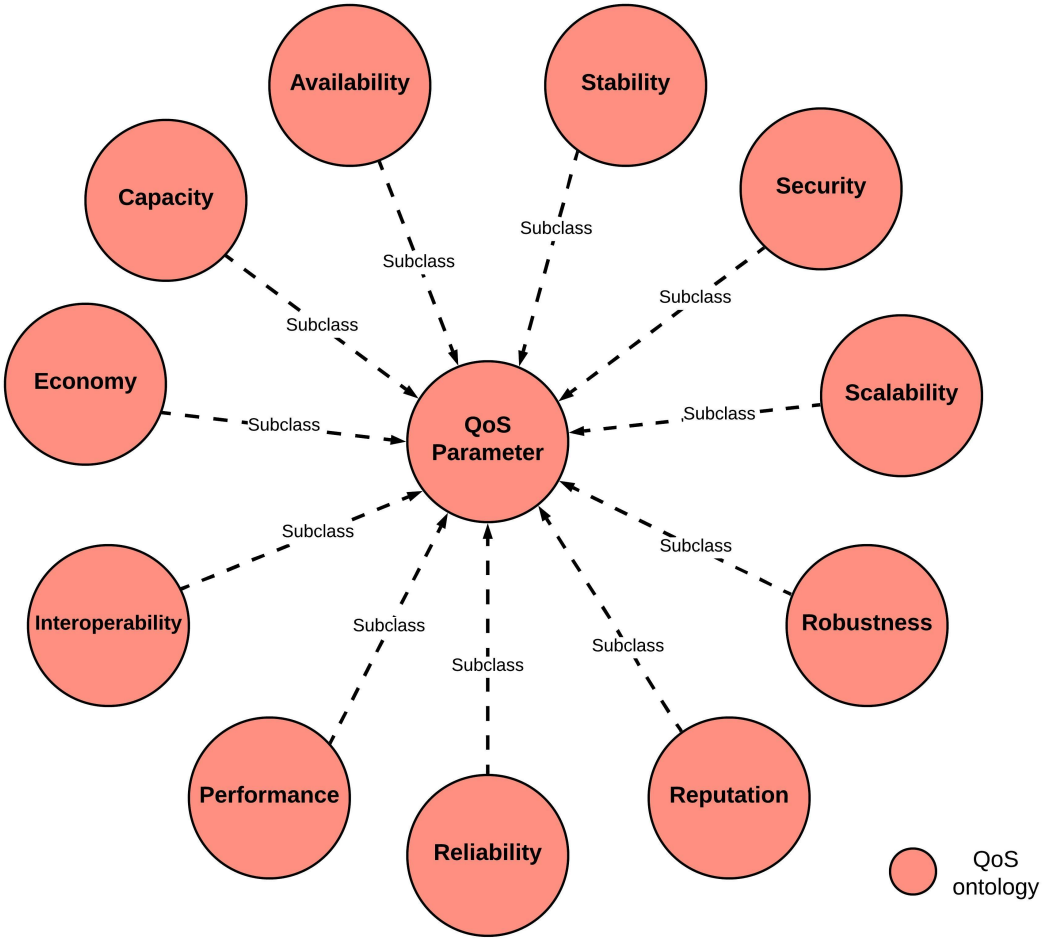

Figure 5.2: Non-functional attributes (QoS parameters) of the proposed ontology [13].

representation. The Service class is used to represent the URL associated with the IoT device dataflow. The Method class describe the Create, Retrieve, Update, Delete (CRUD) operations related to the Service class through the hasMethod property.
Figure 5.3, also shows how the SIO, QoS and SSN ontologies are combined. The Service class joint the QoSParameter class (from the QoS ontology) with the hasQoSParameter object property, and it interacts with the System class of the SSN ontology using the hasService property.

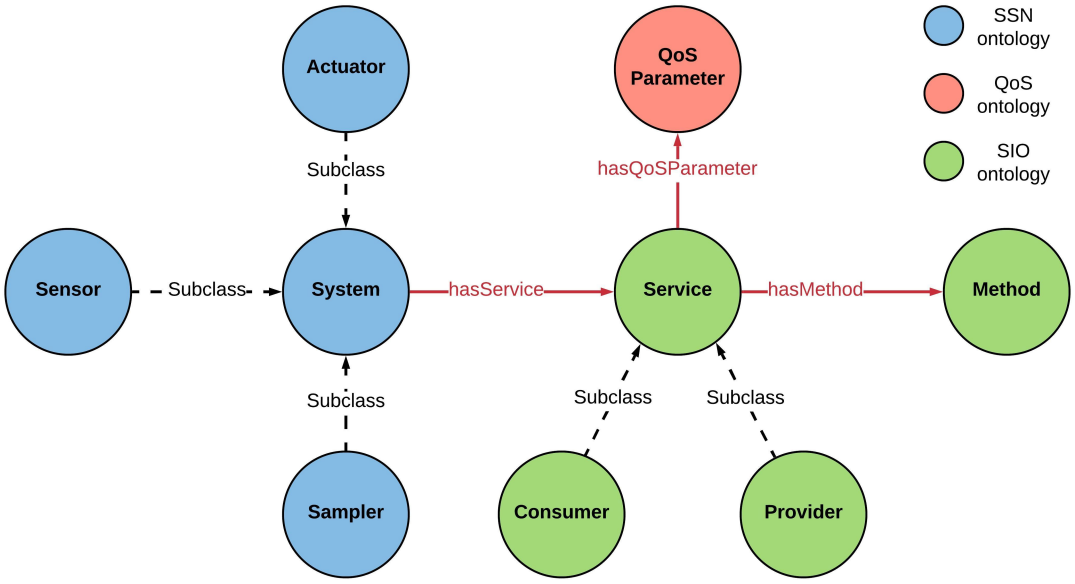

Figure 5.3: Proposed ontology.

An example of an IoT sensor representation of the proposed ontology is shown in Figure 5.4 and Figure 5.5. The figures exemplify the instantiation of the functional, System and QoS properties of the sensor on the corresponding ontology classes.

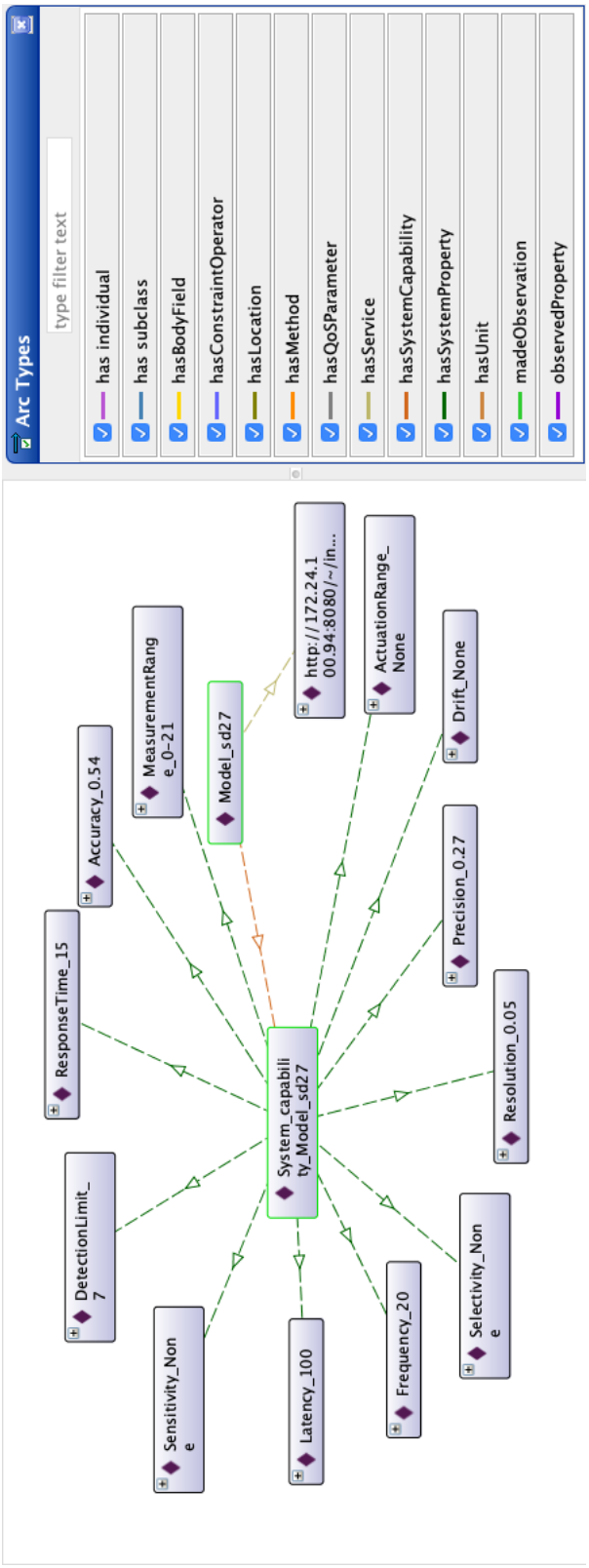

Figure 5.4: System properties of an IoT device ontology representation.

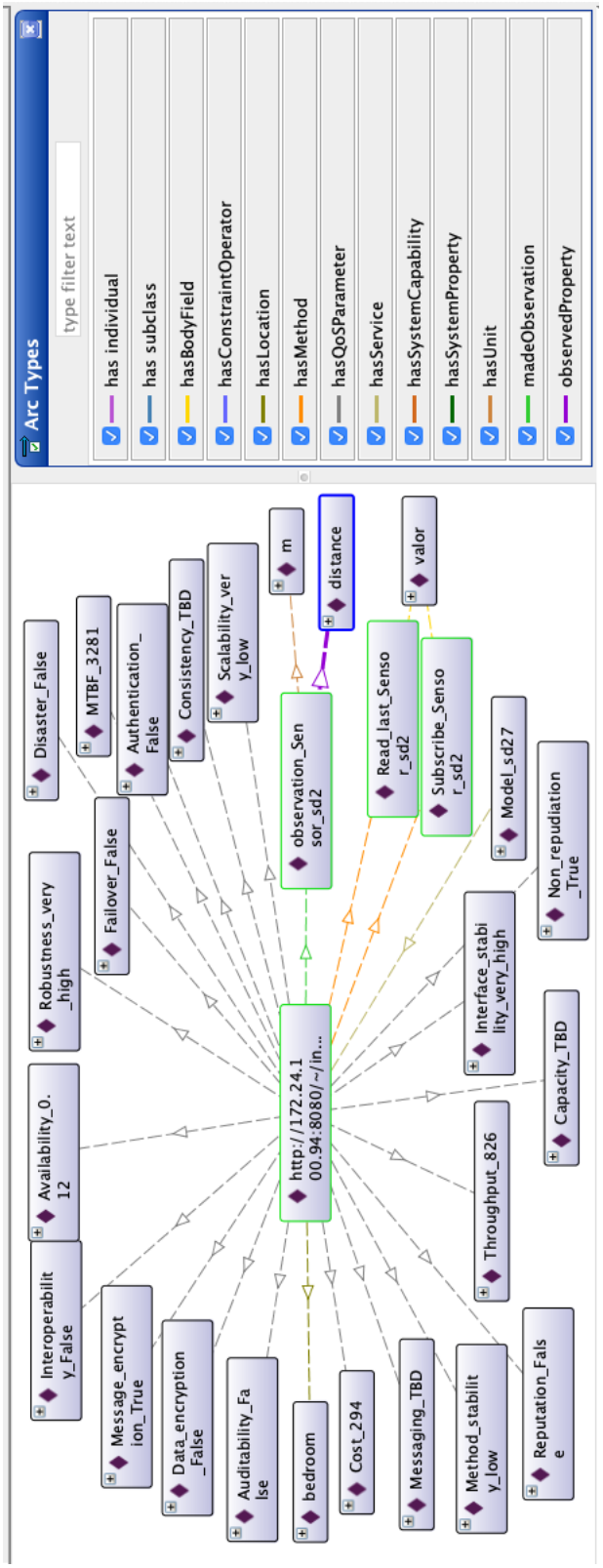

Figure 5.5: QoS properties of an IoT device ontology representation.

5.2 Matching algorithm

The solution to the first challenge continue with the autonomous connection between entities in the environment, and it is addressed by using an ontology instance matching. Shvaiko and Euzenat [29] document several matching methods in their book, which includes semantic, syntactic, taxonomy based, model-based, and graph-based methods. Shvaiko and Euzenat [29] also states that the best option is to combine different techniques in order to improve the matching results.
In our initial approach, we combine semantic, syntactic, data type and value matching. After initial experimentation, we discarded the semantic matching due to inadequate time performance response. Using these methods, we compare the functional and non-functional properties between IoT devices and web services to find the most appropriate match that fulfills the specified requirements. This matching operations also includes weight adjustments, to provide relevant results to the user according to the required intention, and based on applicable heuristics from the control systems domain.
The following subsections describe how semantic, syntactic, data type and value matching are executed, how weight adjustments are applied to our matching algorithm. The final subsection shows how these element are combined to implement the proposed matching algorithm.

5.2.1 Semantic Matching

Semantic matching is implemented to compare functional properties of the IoT devices, specifically physical location of the device, type of the physical variable and units of the measured physical variable. Semantic matching is executed by using the external lexical corpus Wordnet, included in Python Natural Language Toolkit (NLTK) library [18].
Wordnet contains Synsets which are the set of synonyms of a word. We use the Synset path_similarity function to denote how similar two word senses are, based on the shortest path that connects the senses in the is-a (hypernym/hypnoym) taxonomy. The similarity score ranges from 0 to 1, and similarity of 1 represents identity. In the proposed algorithm, we calculate the path_similarity between the user requirement and each IoT device functional property, to determine the best match available.

5.2.2 Syntactic Matching

Syntactic matching is also used to calculate matching of the functional properties of the devices. This type of matching is implemented using python Scikit-learn library [22]. To analyze the text content of the functional properties, we use the Bag of Words representation for text features extraction. The first step of the algorithm is to construct an endogenous corpus with the description of the each functional property. Then we use the CountVectorizer to extract features of the text by tokenizing and count the word occurrences. This way we have the words

represented as numeric vectors, and we proceed to calculate similarity between the user requirement and each IoT device functional property, using cosine distance. In syntactic matching, similarity score also ranges from 0 to 1.

5.2.3 Non-functional Matching

Non-functional matching is based on data type and value matching. Each System and QoS property is expressed with a numeric, text or Boolean value. When matching web services requirements with the available non-functional properties o each device, we start comparing data type of the value. If data types are equal, we determine the comparison signs (<,>,>=,<=,!=,==) applicable to the property to perform the logical value comparison. In this case similarity score is discrete, can only be 0 or 1, with 1 representing identity.

5.2.4 Weight adjustment

When comparing functional and non-functional properties, priority definition is needed to establish the relevance of a property for a specific requirement. Weight adjustments is a common approach to solve this need [13][37][41]. The numeric values of the weight can be provided by the user or by an algorithm. In our research, we determine the weight adjustments based on the specified application intention, and applicable heuristics from the control systems domain.
The intended objective of a service can range from monitoring, provide non-critical alarms, operations control or critical alarm annunciation [12] [11]. The attributes of each applications are described below:

1. **Monitoring.** Action performed from sensor to measure or sense the status or magnitude of one or more variables for the purpose of deriving useful information.

2. **Non-critical alarms.** An audible and/or visible means of indicating to the user an equipment or measured variable condition that requires awareness, but a response or action is not immediately needed.

3. **Control.** A system that responds to input signals from the variables and its associated equipment, other programmable systems, and/or from an user, and generates output signals causing the variables and its associated equipment to operate in the desired manner and within normal limits.

4. **Critical alarm annunciation.** An audible and/or visible means of indicating to the user an equipment malfunction, measured variable deviation, or abnormal condition requiring a expedited response.

Tables 5.1 and 5.2, contain the weight values for each non-functional property, according to the relevance of each property to each of the four possible intended applications. Each of these values are multiplied by the similarity result of the non functional matching operation, to assign the defined priorities.

	Monitoring	Control	Non-critical alarms	Critical alarms
Accuracy	0.5	0.75	0.75	1
Actuation range	0	0	0	0
Detection Limit	0.75	1	0.8	1
Drift	0.5	1	0.5	0.75
Frequency	0.5	1	0.75	1
Latency	0.5	1	0.5	0.75
Measurement range	1	1	1	1
Precision	0.5	1	0.5	0.75
Resolution	0.5	1	0.5	0.75
Response time	0.5	0.75	0.75	1
Selectivity	0.5	0.5	0.5	0.5
Sensitivity	0.5	0.5	0.5	0.5

Table 5.1: Weight values for system properties in control systems applications.

	Monitoring	Control	Non-critical alarms	Critical alarms
Availability	0.5	1	0.75	1
Capacity	0.5	0.8	0.5	1
Cost	1	0.5	1	0.5
Interoperability	0.5	1	0.5	0.75
Throughput	0.5	0.75	0.5	0.75
Consistency	0.5	1	0.5	1
Messaging	0.5	0.5	0.5	1
MTBF	0.5	0.75	0.5	1
Disaster	0.5	0.75	0.5	1
Failover	0.5	0.75	0.5	1
Reputation	0.5	1	0.75	1
Robustness	0.5	1	0.5	1
Scalability	0.5	0.5	0.5	0.5
Auditability	0.5	1	0.75	1
Authentication	0.5	0.75	0.5	1
Data encryption	0.75	0.75	0.75	1
Message encryption	0.75	0.75	0.75	1
Non repudiation	0.5	1	0.5	1
Method stability	0.5	0.75	0.75	1
Interface stability	0.75	1	0.75	1

Table 5.2: Weight values for QoS properties.

5.2.5 Matching algorithm description

The main steps of the matching algorithm are the following:

1. Query the ontology database to retrieve the IoT devices description.

2. Receive the user/web services requirements in the JSON format. These requirement includes functional, non-functional requirements (described in Section 5.1), and application and allowable threshold requirements. The application requirement refers to the intended use of the measured data, that includes "monitoring", "non critical alarms", "control" and "critical alarms". If no application is specified, "monitoring" is specified by default. Threshold requirements is an scale from 0 to 1 to indicate the level of thoroughness to fulfill the matching requirements. If no Threshold is specified 0.7 i set by default.

3. Calculate functional similarity (syntactic similarity) to determine a list of suitable devices. Functional similarity is performed on three functional properties, location of the device (Location), type of the physical variable (Category) and units of the measured physical variable(Unit).

4. The similarity result of each functional property (Location, Category and Unit) is aggregated. The result is compared with the maximum allowable similarity score (which is 3 on this case), scaled by the Threshold requirement. The result of the functional matching is a list of IoT devices with a similarity score above the scaled threshold.

5. Using the results provided by Functional matching, system properties similarities are calculated. Each similarity is scaled by its corresponding property weight value, specified by the application requirement.

6. The calculated system properties similarities are once again aggregated. A new maximum scaled threshold is determined by counting the number of compared properties, and multiplied by the Threshold and the average of the application weights.

7. An updated list of suitable devices is created, only the devices that comply with the functional and system properties requirements *i.e.,* devices with system properties similarity score above the new scaled threshold. Both similarities results are aggregated into one new value.

8. To complete non-functional matching, QoS matching is performed with the results of system properties matching. The same steps from systems properties matching are performed. QoS similarity weight scaling, aggregation of the similarity values, new Threshold definition and comparison and update of suitable devices list.

9. The resulting suitable devices list is ordered by similarity score in a descending order and restricted to 5 results.

The described matching algorithm is illustrated in Figure 5.6.

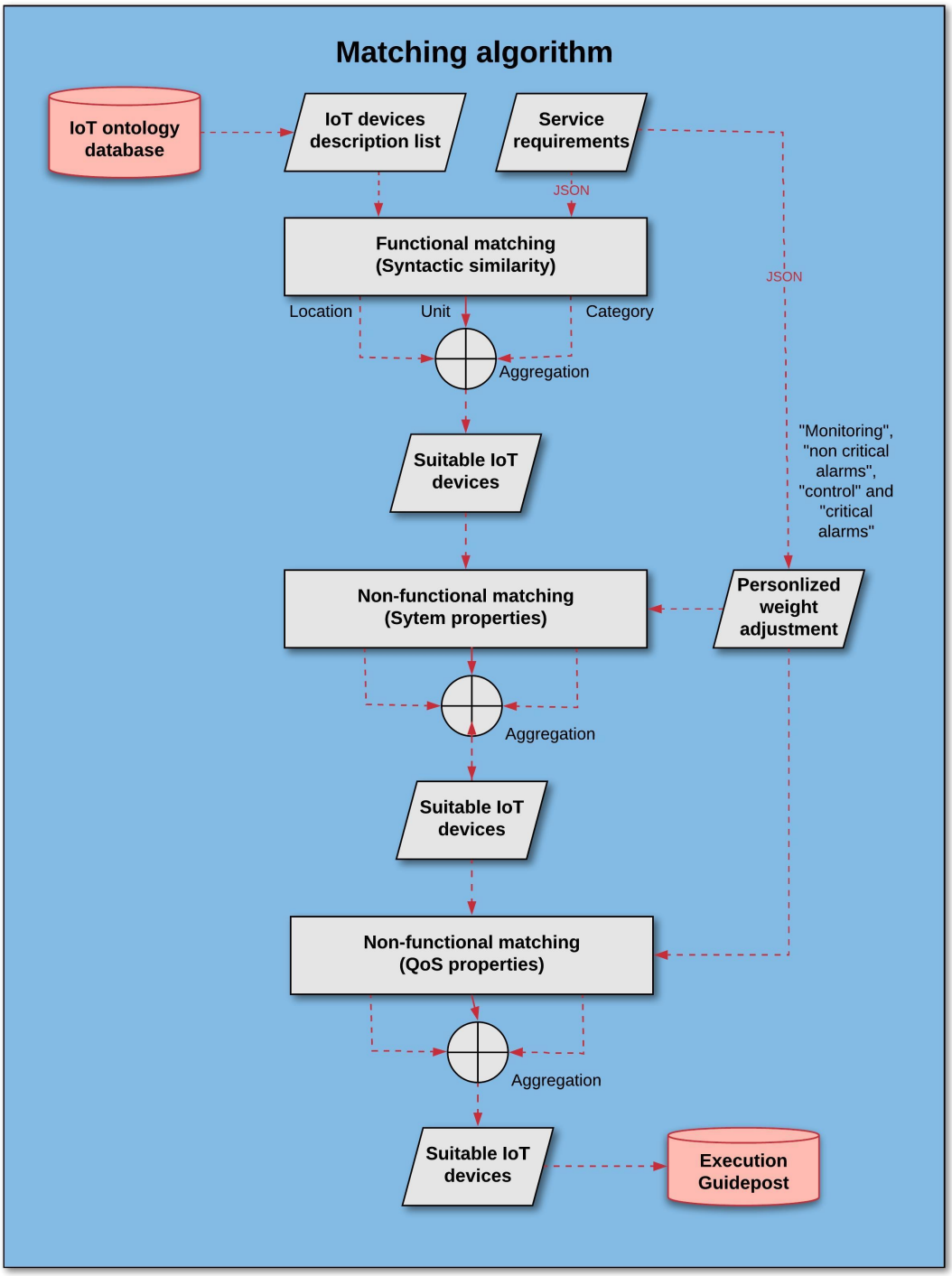

Figure 5.6: Matching algorithm

5.3 Matching registry

The result of a device and service binding, a*match*, must be stored in an easily accessible data structure. Note that such data structure should allow one to register the correspondence between a service and many suitable IoT devices.
To manage device-service bindings at run time, we use execution guideposts [4]. Guideposts work as a dictionary consisting of key-value pairs; the key represents a service and the value a the list of correspondent IoT devices, the matching score of each device and its availability status. This dictionary is presented on Table 5.1. Device-service bindings are persisted in the execution guidepost to enable fast reconfiguration whenever a device fails.

5.4 IoT devices status monitoring

The status of each IoT device is monitored and diagnosed, in order to achieve the required adaption. Devices are also be annotated on the execution guidepost registry, allowing the system to proceed with adaptation, as shown in Table 5.3. If the previously established subscription between a service and the most suitable IoT device is lost (highlighted with green on Table 5.3), the system queries the guidepost for the next suitable IoT device on the list (highlighted with yellow on Table 5.3), to execute a new subscription between the two endpoints.

Web service	IoT device	Score	Status
	Rain meter 1	11.5	True
	Rain meter 2	11	False
www.RainCheck.com	Rain meter 3	11	True
	Rain meter 4	10	False
	Rain meter 5	9	False

Table 5.3: Execution Guidepost registry.

5.5 An Adaptive IoT Architecture

This section describes our proposed architecture, shown in Figure 5.7, to deal with adaptation of IoT environments. Several architectural components are implemented by the reference IoT architecture [2]. To address the Challenges AC.1 and AC.2 described in Chapter 2, we integrate seven new components as part of the adaptive IoT architecture. The components are a server, gateway, resource discovery, instance matching, execution guidepost, and healthcheck controller, which are described in the following section.

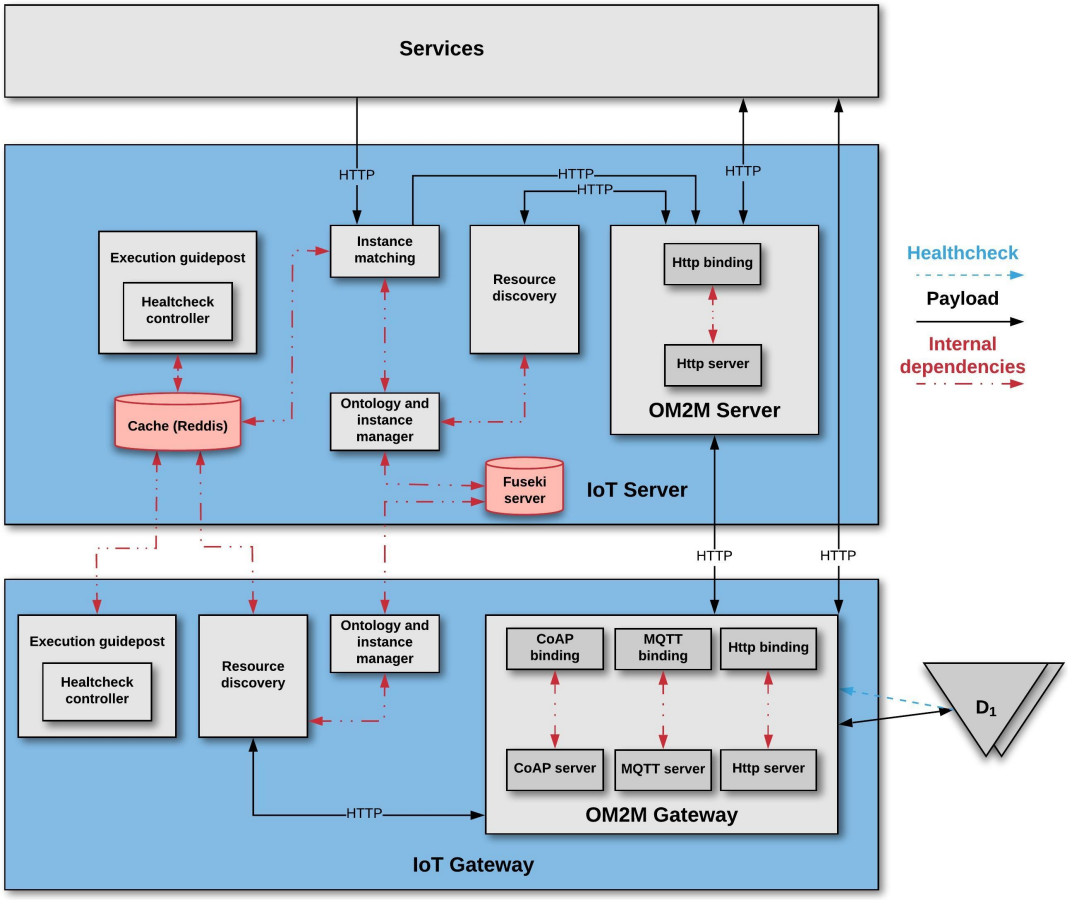

Figure 5.7: IoT architecture for adaptation.

5.5.1 OM2M server/gateway

We support our proposed architecture on top of the OM2M platform [15], this platform is used for its multiple communications. This component interacts directly with the physical IoT devices. Its primary function is to provide protocol abstraction of the IoT devices, allowing the services to connect to the IoT devices' data streams (exposed as URLs), using the HTTP protocol. Additionally, OM2M can handle a distributed architecture using gateways located on different locations, managed by a server. Figure 5.8 illustrates how data is structured on OM2M platform.

5.5.2 Resource discovery

The resource discovery component automatically search the OM2M network for all the data streams provided by IoT devices. For every data stream, the resource discovery collect metadata that describes the functional and non-functional properties of the entities. The metadata is advertised by the OM2M platform using the XML, or JSON formats.

5.5.3 Ontology and instance manager

The resource discovery results are parsed as individuals of the proposed ontology by the Ontology and instance manager component. When a new IoT device is discovered, this component also examines the non-functional properties of the new device and compares it with the existing ones. If no identical device exists, a new individual is created on the ontology. In the opposite case, the existing identical individual is reused, optimizing the use of the ontology.

5.5.4 Instance matching

The purpose of this component is to provide matching between services and data stream provided by the devices. The Instance matching will execute a combination of the matching algorithms (syntactic, data type, and value matching). Using these methods, it compares the functional and non-functional properties between IoT devices and web services to find the most appropriate match that fulfills the specified requirements. The instance matching also establish a subscription between a web service and the stream of data provide by the most suitable IoT device.

5.5.5 Execution guidepost

The execution guidepost [4] is a registry used to keep track of the subscription relationship between services and the most suitable IoT devices. In addition to a list of suitable devices, the Execution guidepost also contain the availability of the data provided by each device. This information will be used as a means of

Figure 5.8: OM2M platform data structure.

adaptation. When the connection with the device is lost, the system will search for the next most suitable an available device to establish a new subscription between the parties. Device availability will be determined by the Healthcheck Controller component.
This registry is persisted on a in-memory key-value database used as cache, to provide fast access and response. The cache technology selected by our research is Reddis.

5.5.6 Healtcheck Controller

As described in the previous section, the function of the Healthcheck controller is to determine the availability status of each available data stream. This component will subscribe to every data stream to monitor a periodic healthcheck signal. If a determined number of healthcheck signals are not received, the Controller will mark the data stream as unavailable and update the availability state of the corresponding data stream on the Execution guidepost.

5.5.7 Fuseki server

This component manages access to the proposed ontology described in Section 5.1. It stores the ontology together with its instances describing the particular devices and services of our IoT domain.

Chapter 6

Experiments and Results

This section presents preliminary results providing initial insights to the feasibility and impact of all the components of our adaptive IoT architecture. This experimentation focuses on Challenges AC.1 and AC.2. This experiments and results are also accessible on the following repository[28].

6.1 Objectives

1. Measure correctness of the matching algorithm.
2. Measure time performance of the architecture components as the system scales, by progressively increasing the number of IoT sensors from 0 up to 10000, and the number of matched web services from 0 up to 1000.

6.2 Methodology

The experiment were divided in three main parts, discovery of new IoT devices, matching of new instances, and adaptation to inclusion of new IoT devices or adaptation to transient IoT devices.
The first part of our experiment evaluated the discovery of new simulated IoT devices. Each device advertises its functional and non-functional properties. The simulation randomly created a new device with identical system properties, but with different web services properties.
When a devices connects to the OM2M gateway, the Resource discovery searches for the URLs that represents the device, in order to read the advertised functional and non-functional properties of the device. The next step is to correctly parse this information to be instantiated as a set of individuals on the proposed ontology. The first metric we measure is the amount of time taken by the system to find a new IoT device. This includes the described search and instantiation of the device properties. The performance test evaluates the variation of the previously described times, as the number of existing individuals increases. The experiment was performed to discover up to 10000 devices.

The second part is the experiment was related with the instance matching process. For this component we had two concerns: the time performance of the process and the relevance and correctness of the matching result.
We first measured the matching time performance between web services requirements and existing IoT devices data streams. The performance test quantified the time variations for the matching process as the number of existing individuals in the ontology increases. Functional and non-functional matching time performance were considered in the experiment. The test consisted on executing 70 matching operations with a fixed number of IoT devices registered on the system, varying with 1000, 2000, 3500, 5000, 7500, and 10000 devices.
We measured the correctness of our matching algorithm based on the relevance of the matching results. This matching test consisted in comparing a set of instances from two different ontologies, both describing the same knowledge domain, and measuring the precision and recall of the match. We calculated precision, recall, and F-measure for the matches between three different ontologies in perspective of three different matching algorithms. Table 6.1, describes the knowledge domain, properties, instances, and sizes of the compared ontologies.
The adaptation to new or transient IoT devices experiment is twofold. We first analyzed the inclusion of new devices. This operation is an extension of the discovery process. When there are existing matches on the system, the Resource discovery executes an additional step to its regular look-up function. It calls the instance matching, to compare the device properties of the new device against all the existing matches, in order to improve matching results. This operation updates the registry of the execution guidepost with the information of the new device. We include 100 new sensors, with a fixed number of matches registered in the system to measure the time performance of the process. Four cases were considered, 250, 500, 750, and 1000 registered matches.
The final experiment addressed the transient IoT devices adaptation. In this case, we simulated several IoT devices, transmitting their healtcheck signal. This signals were configured to operate in an erratic manner to emulate an intermittent behavior of the IoT devices and therefore induce the system to adapt. During the 10 minute execution of the experiment the system induces about 600 adaptations. We measured the amount of time taken to perform each adaptation with a fixed amount of registered matches in the range, 250, 500, 750, and 1000.

6.3 Data sets

To test the feasibility of the approach, we validated the accuracy of the instance matching component. To this end, we used the Ontology Alignment Evaluation Initiative (OAEI) ontology data set [20] as baseline. This initiative proposes ontology schema and instance matching challenges, with its corresponding data sets. We selected the Star Trek knowledge graph contained in the Knowledge Graph Track data, for our evaluation, as these sets provide a large set of pairs for instance matching. We did not find any open source ontology in the IoT domain coming

with instances and gold standard to check the algorithm correctness. Instead, we found ontologies that are out of the IoT domain but allow us to shed lights on two relevant aspects: i) the applicability of our algorithm to domains and case studies different to the IoT world; ii) the algorithm accuracy.
The campaign proposes three ontologies for the Star Trek Universe domain, namely: , Memory Alpha, Memory Beta and Star Trek Expanded Universe. Table 6.1 describes the size of ontologies with respect to their instances, properties, and classes. To test a matching system, the OAEI provides with 3 gold standards with correspondences between instances of two data sets. We have correspondences between memory-alpha/memory-beta, memory-alpha/stexpanded and memory-beta/stexpanded. The gold standards take into account the performance metrics (*i.e.,* precision, recall, F-measure, and execution time) from different matching systems that have been tested in the challenge. We chose thee benchmarks for our functional matching component, namely: Baseline, LogMapLT, and DOME.
In case of non functional matching, we didn't test the correctness of the algorithm, because it's based on logical comparisons to determine equality of data type (numeric, Boolean or text) and value similarity using comparison operators(<,>,>=,<=,!=,==).

	Memory Alpha	Star Trek Expanded Universe	Memory Beta
Instances	63240	17659	63223
Properties	326	210	413
Classes	0	3	11

Table 6.1: Star Trek knowledge domain ontology sizes.

To experiment with all the components of the adaptive architecture, we also generated a data set of simulated IoT devices. The devices were created with random functional and non-functional properties from a set of consistent values (based on the implementation of the SUDS case study), considering applicable physical variables (*e.g.,* flow, conductivity, rainfall, temperature), measuring units (*e.g.,* mm or in for rainfall), location(*e.g.,* different parks with SUDS), system properties (*e.g.,* some sensors doesn't have drift) and QoS parameters (*e.g.,* some devices data doesn't need authentication).
To determine a number of instances soundness in the IoT domain, we considered as reference the Smart Santander testbed [30]. This test facilities embrace scenarios like smart buildings and smart cities, and sizes up to 12000 sensors. In our experiments we worked with 10000 simulated sensors.

6.4 Prototype

We developed a prototype and deployed it in a private computing cloud to evaluate the contributed components of the middleware layer.

To obtain appropriate results, our case study must contain the presence of several different and identical IoT devices, which expose their data on different URLs. Due to the absence of a significant amount of physical devices, we emulate them, taking into account their inherent resource limitations. Taking into account the implementation of the three nodes for the case study, we determined empirically that the main constraint in these devices is the size of the payload sent. In our experiments, we found that the maximum allowable payload is 256 characters for the NodeMCU IoT device. Taking this constraint into consideration, we restrained the size of the payload in our emulated IoT devices.
Our simulation evaluated the response time when scaling the number of devices, by progressively increasing the number of IoT sensors from 0 up to 10000. Each device simulates, using a Python script, their inclusion in the OM2M gateway, data streaming (reporting a the physical variable), malfunction or intermittent behavior of devices, and discovery/subscription to IoT data streams by web services.
The simulated devices are subscribed to a Message Queue Telemetry Transport (MQTT) broker, which is bound to the corresponding OM2M gateway. The Ontologies Repository is deployed on top of an Apache Jena Fuseki server. The Resource discovery, Ontology instance manager, Instance matching, Healtcheck controller, and Execution guidepost are implemented using Python 3,7. The Execution guidepost and Healthcheck data are persisted in a Reddis server.
The adaptive architecture is distributed on 4 virtual machines. The first one contains the OM2M gateway along with its Resource discovery and Ontology instance manager. The second machine runs the OM2M server and also the Resource discovery and Ontology instance manager on the IoT server side. The Apache Jena Fuseki server is running on the third virtual machine. The last machine is running Reddis, the Execution Guidepost and the Healthcheck controller. All machines are equipped with 8GB of RAM and a four cores processor, Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz.

6.5 Results

As stated in Section 6.2, our first experiment is the discovery of IoT devices. As it can be seen in Figure 6.1, the response time of the Resource discovery growth is linear with respect to the number of IoT devices, up to 10000. In this process, the response time increases because the component verifies if the sensor properties exists in the registries or in the ontology. In most cases, the response time was lower than 0.8 seconds, an acceptable waiting time to include a new sensor, considering the size of the reference IoT environment, Smart Santander[30].

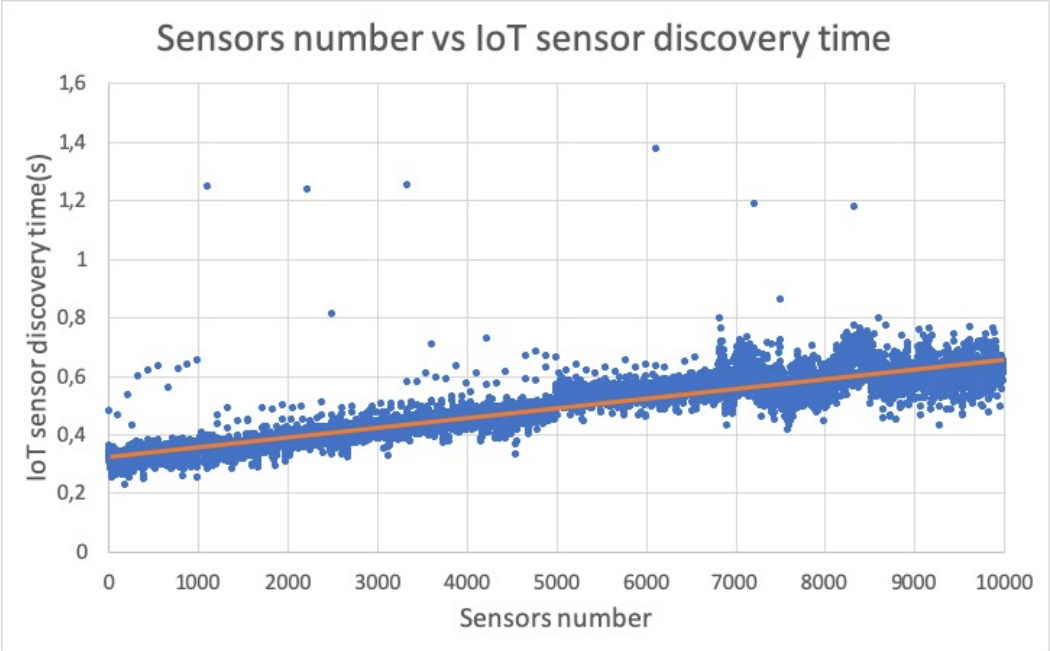

Figure 6.1: IoT device discovery time performance.

The next two figures, Figure 6.3 and Figure 6.4, illustrates the time performance for instance matching component. The instance matching time is composed by the sum of the functional and non-functional matching time. In our first experiments, the functional matching has two components, semantic and syntactic matching. Initial functional matching experiments where executed having 500 to 2000 sensors. Figure 6.2 shows the semantic matching results. This kind of matching shows poor time performance results, when executing one matching operation against existing IoT sensors. This result provide a negative suitability feedback, for semantic matching operations due to the slow response founded, ranging from 4 to 16 seconds.

Functional and non-functional matching experiments present results up to 10000 sensors, with boxplots to show the distribution of group of samples. The functional time response of the instance matching component, exhibit a exponential response in the range of 10000 sensors and most of the samples are inside the time distribution in each case. The maximum median time is about 9 seconds, when exits 10000 registered sensors in the system. The non-functional time performance satisfies our expectations as most of the samples are lower than 0.06 seconds. This is due to fact that only consist in basic data types comparisons.

With respect to the correctness and relevance of the matching operation, we compare the precision (Figure 6.5), recall (Figure 6.6), and F-measure (Figure 6.7) with three approaches participating in the OAEI 2018 contest, using the three ontologies from the Star Trek knowledge domain.

Our functional matching has a better precision for two of the ontology comparisons (Figure 6.5), performing between 10% and 27% better than the other approaches.

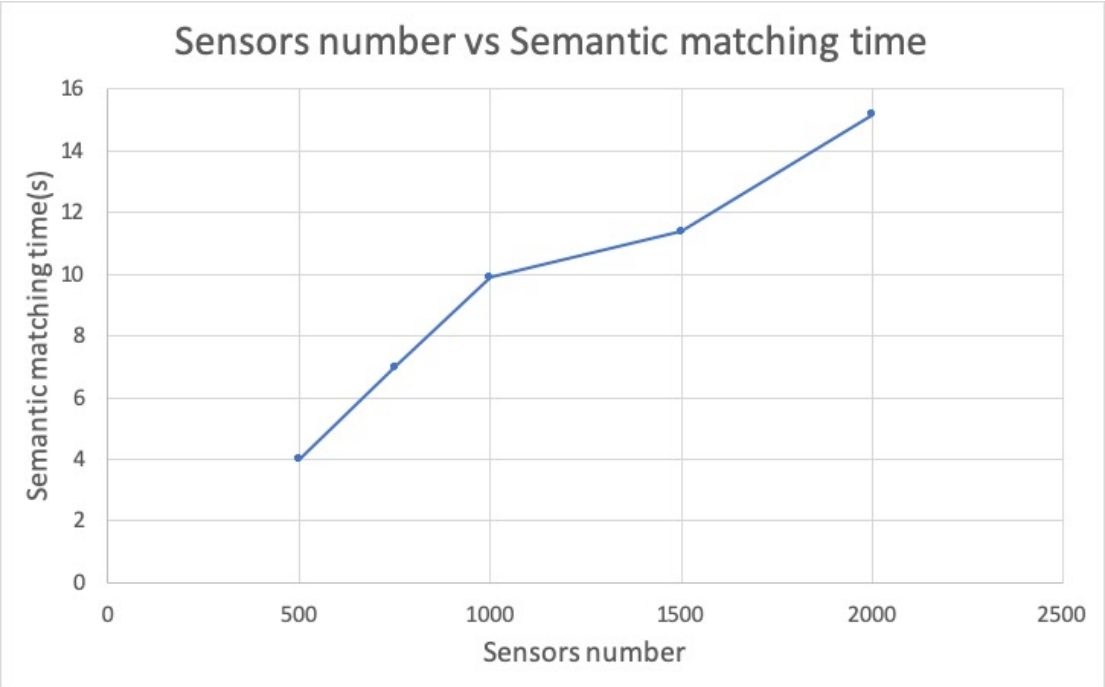

Figure 6.2: Semantic matching time performance.

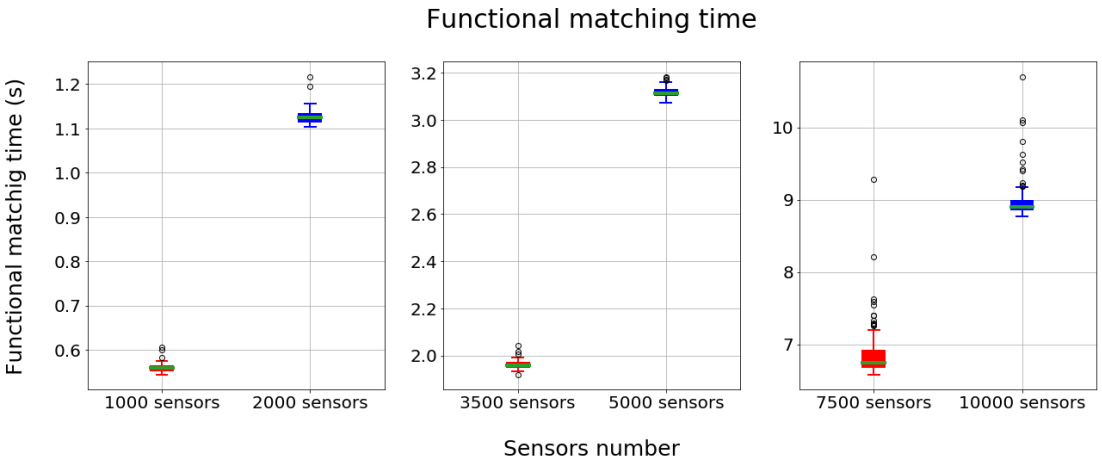

Figure 6.3: Functional matching time performance.

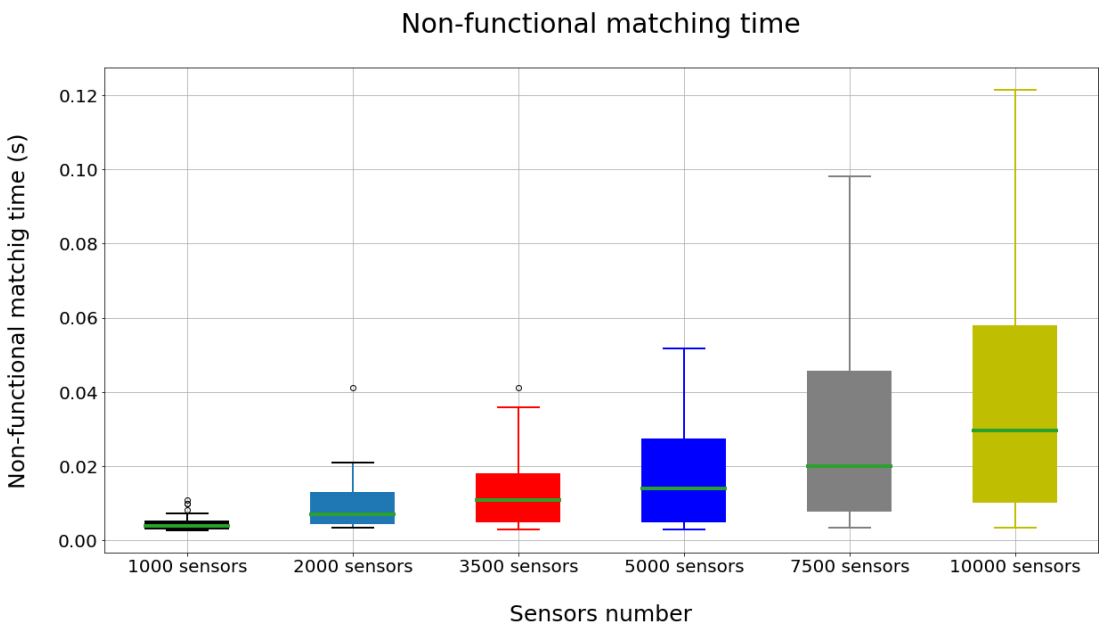

Figure 6.4: Non-functional matching time performance.

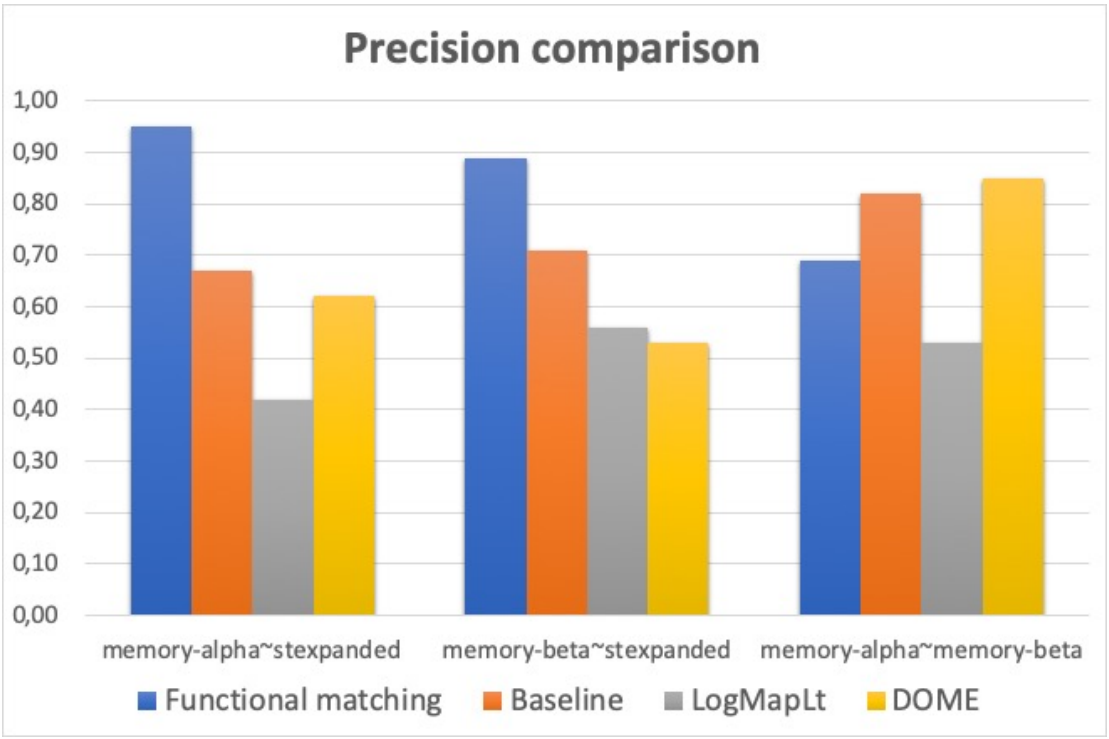

Figure 6.5: Precision comparison.

In the third ontology comparison, two of the approaches have a better precision that our functional matching. This is because the two data sets has instances with more complex descriptions, *i.e.,* more words describing the instance, and our functional matching is more suitable for words or short sentences comparison.The recall evaluation (Figure 6.6) evidences a performance in the range [0.56, 0.73]. In most cases the recall of our functional matching is lower than that of other solutions. This low performance is caused by the fact, that our matching algorithm is only based in one similarity method (syntactic matching), and in some cases, it's better to have a combination of similarity methods in order to resolve ambiguities. The F-measure (Figure 6.7) behaves similarly to the precision results, as the F-measure couples together both precision and recall results.

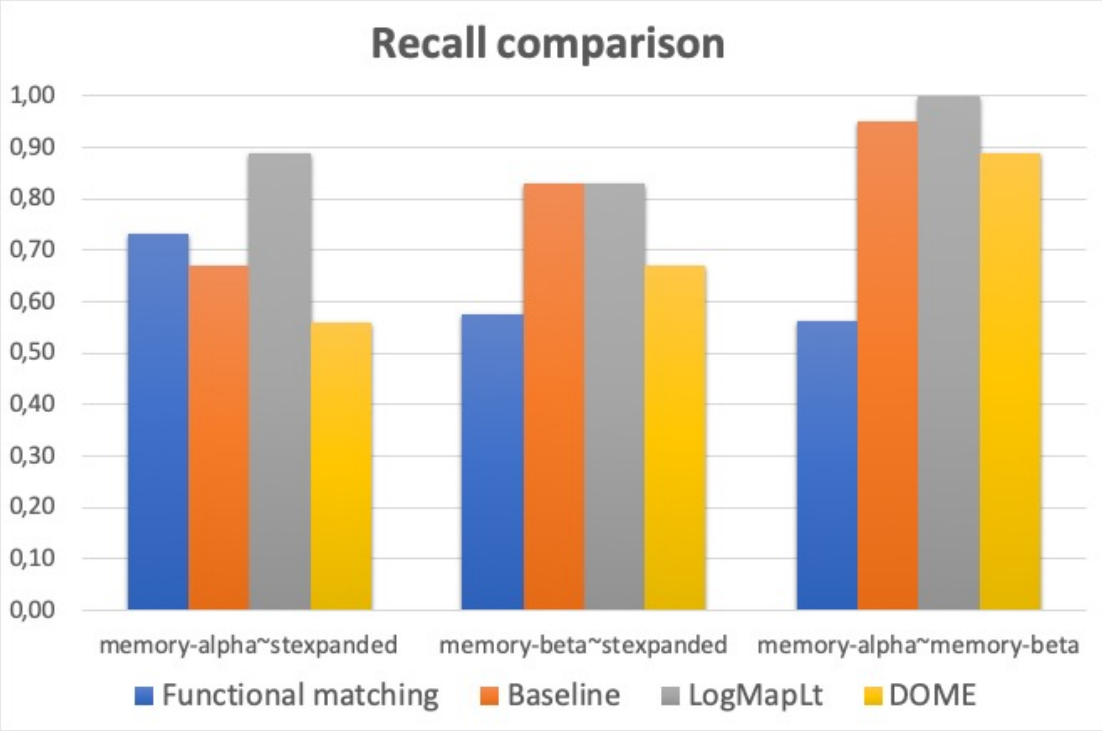

Figure 6.6: Recall comparison.

The adaptation time whenever new IoT sensors appear in the system is shown in Figure 6.8. The figure shows the maximum median sensor inclusion time is $4s$ for a sensor matching 1000 services in the system.
Figure 6.9 shows the behavior of adaptation to transient IoT devices. In this particular case, there is a great number of outliers in all samples. This is due to the fact that the matches are randomly generated, and there is no guarantee that all sensors are equally matched with external web services. It can also be observed that all sample values are less than 0.9 seconds.

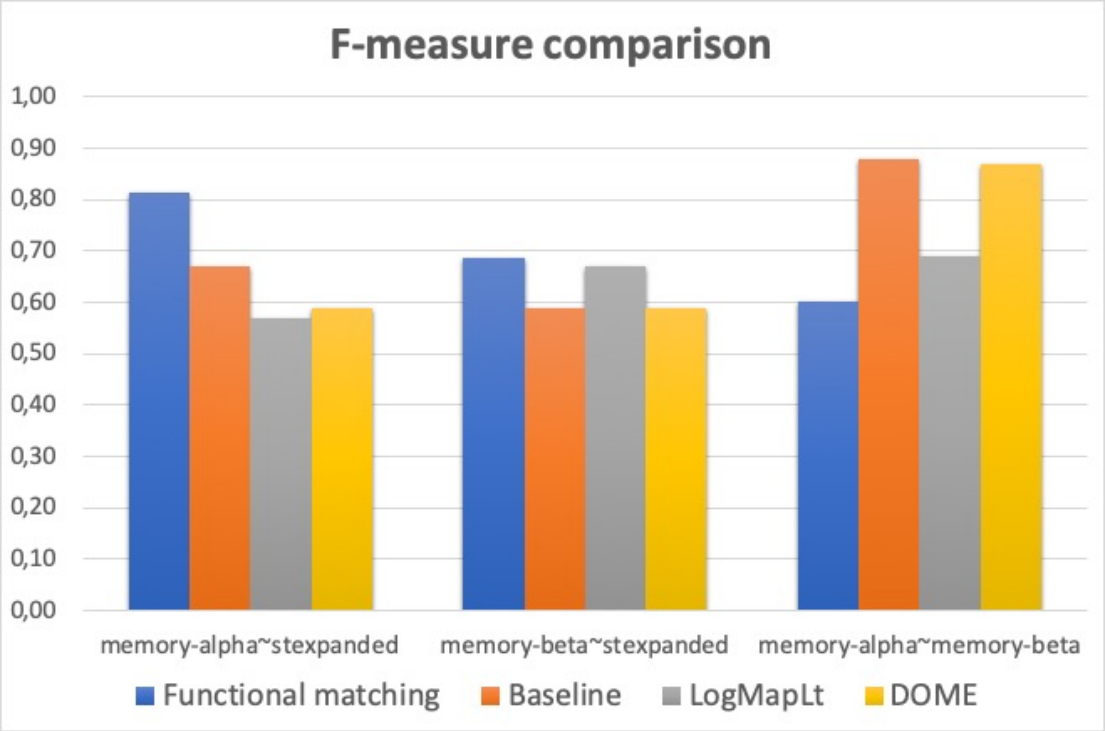

Figure 6.7: F-measure comparison.

6.6 Discussion

The experiments results described in the last section, have allowed us to evaluate the feasibility of many components of the adaptive architecture.
The Resource discovery and Ontology instance manager have exposed a satisfactory response time performance within the considered IoT environment size. AS we expected, there is a linear grow in the IoT devices discovery time as more devices are registered on the system. This behavior is caused by the verification of existing system properties of a sensor when the IoT is registered.
Currently, the Instance Matching is the component with the lowest time performance in the adaptive architecture, as shown in Figure 6.3 and Figure 6.4. For the Instance matching process, Functional matching represents essentially all time of the process. Non-functional matching time is negligible, because is almost 10000 times lower than its functional counterpart. This performance result is not suitable for the selected size of IoT environment, but it present us positive feasibility of the Functional matching component. This element was conceived as a simple matching algorithm and there are many aspects that can be improved. One refinement can be, organizing the IoT devices in similar groups based its properties, to reduce the number of elements involved in the similarity calculations.
As correctness and relevance concerns, the Instance matching have exposed a satisfactory results, as is can be seen in figures 6.5, 6.6 and 6.7. Our actual requirements involve the description of the functional properties as words or short

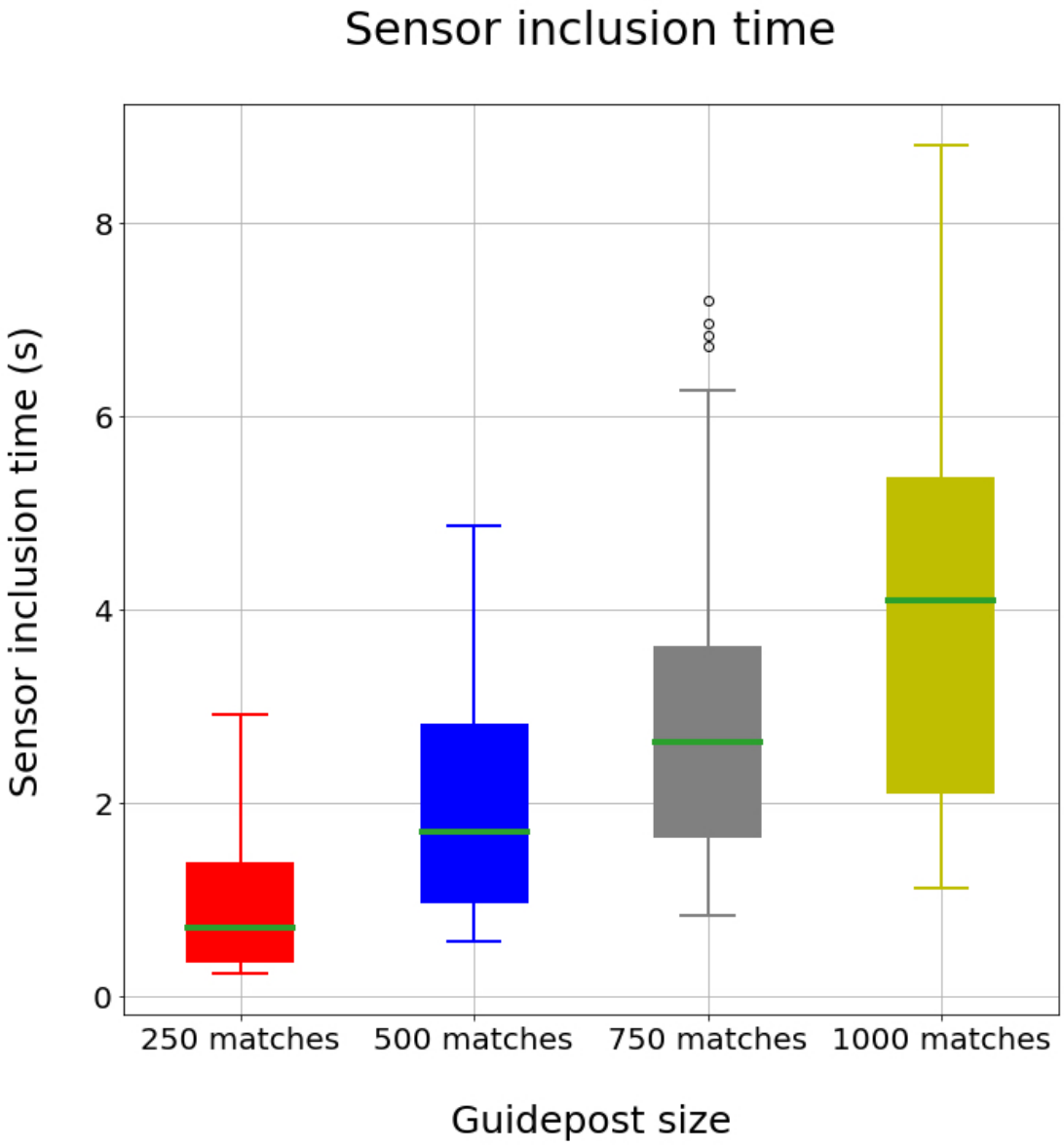

Figure 6.8: Sensor inclusion.

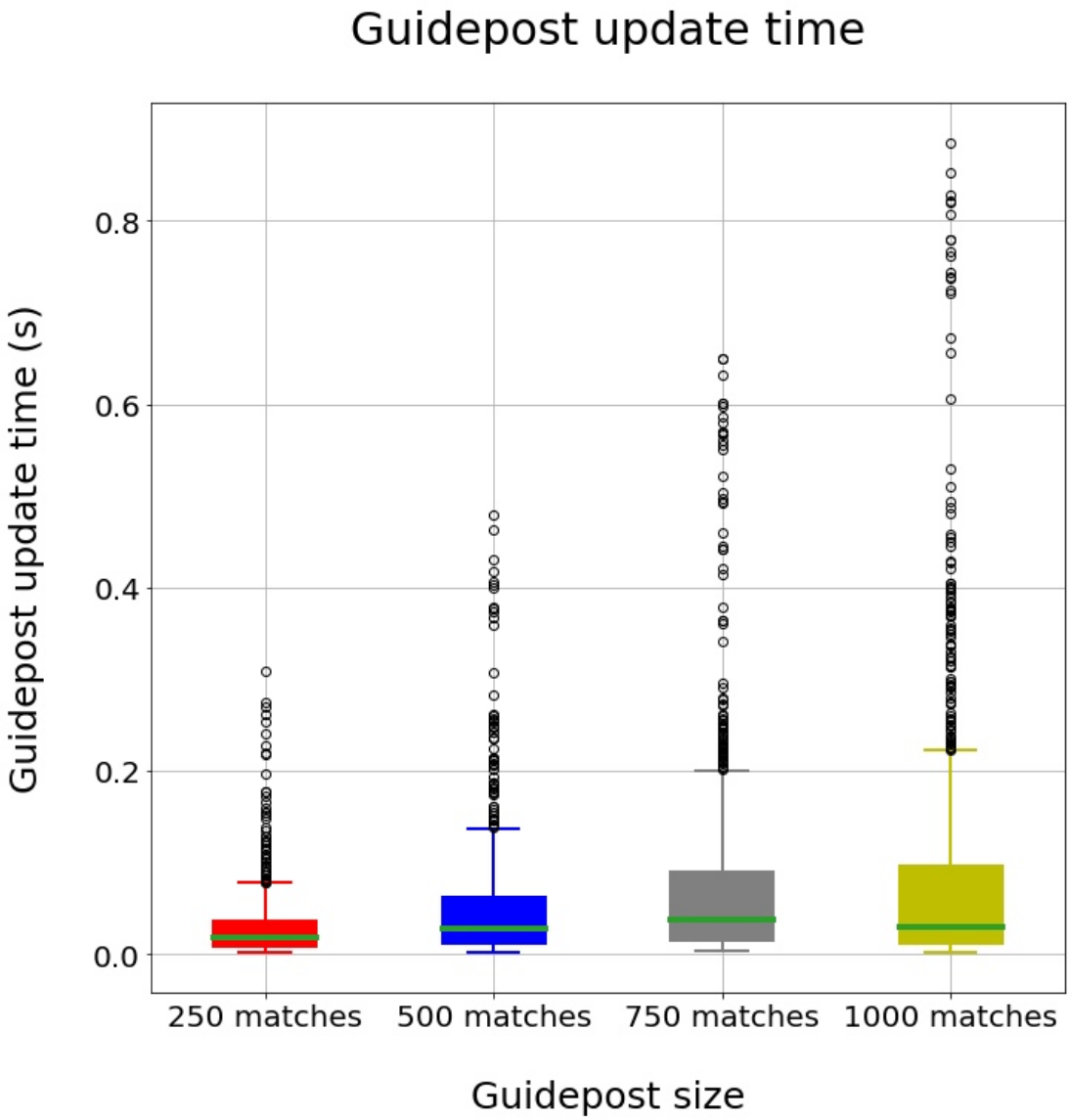

Figure 6.9: Guidepost update.

sentences, and the current Functional matching algorithm exhibits good performance to our requirements and expectations.
Inclusion of new IoT devices adaptation is a combination of the discovery of devices and instance matching with services, as stated in Section 6.2. In this case, the Functional matching is also the element with the lowest time performance of the process. The results of the test (Figure 6.8) demonstrates that this adaptation can be implemented within reasonable time responses. However, it needs great improvements in order to increase the the number of subscribed services and still provide a rapid update of the compatible IoT devices available. Improving the Functional matching time is a fundamental issue to solve in future work.
The execution guidepost adaptation to transient IoT devices, manifest a satisfactory performance in the insight test. All adaptations are executed in less than 0.2 seconds, with some cases ranges from 0,4 to 0,9 seconds. This results enable further development of the component to improve performance of this relevant adaptive element.

6.7 Threats to validity

The validity of our experiments is delimited by three aspects. Two of these are related with external tools adopted in the development of the adaptive architecture, and the last one is defined by the IoT environment test size.
In our adaptive architecture we have used the Open source service platform for Machine to machine (M2M) (OM2M) platform to enable protocol abstraction and a distributed architecture. A MQTT broker is bonded with the OM2M platform to enable connectivity with the sensors using its corresponding protocol. Performance test of these two components are not yet covered by our research, and may impose restrictions due to the size of our IoT environment. This may be specially true, when all the 10000 sensors are transmitting continuous data flows from the measured physical variable.
The last concern to validity is the size of the selected IoT environment. We have considered in our experiments, 10000 sensors and 1000 subscribed services. The size of real scenarios can be bigger than the contemplated in the experiments, and the obtained performance results may not be as suitable as expected. This issue is specially true, for the considered number of subscribed services.

Chapter 7

Conclusion and Future Work

This Tesis enumerates the different adaptation challenges found in IoT applications, according to our experience. The adaptation challenges can take place at different layers *(Physical, Middleware, or Services)* of the IoT reference architecture using specific adaptive components. To cope with these challenges we propose an adaptive architecture, based on the integration of four technologies, IoT ontologies, Ontolgies instance matching, Execution Guidepost and IoT devices status monitoring.

The main contribution of this Tesis, is the development and implementation of an adaptive architecture to address the selected Challenges, Adaptation to Inclusion of New IoT Devices and Adaptation to Transient IoT Devices. To include an IoT device in a environment, the use of ontologies enable semantic description of devices to enable their discovery by external users or web services. Next, ontogology instance matching is the first step to provide adaptation to transient devices. When comparing user requirements with IoT devices functional and non-functional properties, a set of most suitable devices is determined and persisted on registry. The final step to provide adaptive connections, is by monitoring the IoT devices data flows availability. The Execution Guidepost will connect web services with the most suitable available device annotated on registry or adapt the connection when the actual device is now longer available.

The second contribution of our work is a greater inclusion of IoT devices physical properties on devices description and matching operations. The system properties describe measurement performance of IoT devices and play a key role on the application objective *e.g.,* monitoring, provide non-critical alarms, operations control or critical alarm annunciation. From the users' point of view, this situation is not evident, and for that reason we provide awareness of these properties to the user, without adding technical complexity. The user only have to choose between the 4 types of application objectives. Based on the selection, we used weight adjustment on the matching algorithm, to assign priorities on system properties according to the application objective, therefore providing more relevant results to web services.

The architecture is validated with a series of experiments focusing on the specified

challenges. This experiments evidence the relevance of the proposed components, as they are aligned with the challenges. Nonetheless, we noticed this implementation needs to be improved, to satisfy scalability requirements of IoT systems, especially with the number web services subscribed to the system.
Another works are in progress to address the remaining challenges. A key factor in future development, is the integration of the proposed architecture with other components that solve those challenges preserving or improving the performance of the system.

Bibliography

[1] Jairo Ariza, Camilo Mendoza, Kelly Garcés, and Nicolás Cardozo. A research agenda for iot adaptive architectures. *Proceedings*, 2:1229, 10 2018. doi: 10.3390/proceedings2191229.

[2] Martin Bauer, Mathieu Boussard, Nicola Bui, Jourik De Loof, Carsten Magerkurth, Stefan Meissner, Andreas Nettsträter, Julinda Stefa, Matthias Thoma, and Joachim W. Walewski. *IoT Reference Architecture*, pages 163–211. Springer Berlin Heidelberg, Berlin, Heidelberg, Germany, 2013. ISBN 978-3-642-40403-0. doi: 10.1007/978-3-642-40403-0_8. URL `https://doi.org/10.1007/978-3-642-40403-0_8`.

[3] Silvana Castano, Alfio Ferrara, Stefano Montanelli, and Gaia Varese. Ontology and instance matching. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6050(Section 3):167–195, 2011. ISSN 03029743. doi: 10.1007/978-3-642-20795-2_7.

[4] Nanxi Chen, Nicolas Cardozo, and Siobhan Clarke. Goal-Driven Service Composition in Mobile and Pervasive Computing. *IEEE Transactions on Services Computing*, 11(1):49–62, 2018. ISSN 19391374. doi: 10.1109/TSC.2016.2533348.

[5] Michael Compton, Payam Barnaghi, Luis Bermudez, Raúl García-Castro, Oscar Corcho, Simon Cox, John Graybeal, Manfred Hauswirth, Cory Henson, Arthur Herzog, Vincent Huang, Krzysztof Janowicz, W David Kelsey, Danh Le Phuoc, Laurent Lefort, Myriam Leggieri, Holger Neuhaus, Andriy Nikolov, Kevin Page, Alexandre Passant, Amit Sheth, and Kerry Taylor. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25–32, 12 2012.

[6] Georgiana Copil, Daniel Moldovan, Hong Linh Truong, and Schahram Dustdar. SYBL: An extensible language for controlling elasticity in cloud applications. *Proceedings - 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2013*, pages 112–119, 2013. doi: 10.1109/CCGrid.2013.42.

[7] Francisco Curbera, Matthew Duftler, Rania Khalaf, and William Nagy. Unraveling the Communication : SOAP. *Ieee Internet Computing*, (April):86–93, 2002. ISSN 1089-7801. doi: 10.1109/4236.991449. URL `http://tele1.dee.fct.unl.pt/rit2_2009_2010/teo/soap.tutorial.pdf`.

[8] Maria Ganzha, Marcin Paprzycki, Wiesław Pawłowski, Katarzyna Szmeja Paweł and Wasielewska, and Giancarlo Fortino. Tools for Ontology Matching—Practical Considerations from INTER-IoT Perspective. In Wenfeng Li, Shawkat Ali, Gabriel Lodewijks, Giancarlo Fortino, Giuseppe Di Fatta, Zhouping Yin, Mukaddim Pathan, Antonio Guerrieri, and Qiang Wang, editors, *Internet and Distributed Computing Systems*, pages 296–307, Cham, 2016. Springer International Publishing. ISBN 978-3-319-45940-0.

[9] S. Gaur, R. Rangarajan, and E. Tovar. Extending t-res with mobility for context-aware iot. In *IEEE First International Conference on Internet-of-Things Design and Implementation*, volume 00 of *IoTDI'16*, pages 293–296, 04 2016. doi: 10.1109/IoTDI.2015.16. URL `doi.ieeecomputersociety.org/10.1109/IoTDI.2015.16`.

[10] V. Gazis, M. Goertz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, and F. Zeiger. Short paper: Iot: Challenges, projects, architectures. In *2015 18th International Conference on Intelligence in Next Generation Networks, Paris, France*, pages 145–147, Feb 2015. doi: 10.1109/ICIN.2015.7073822.

[11] The Instrumentation Systems ISA and Automation Society. *ANSI/ISA-18.2-2009 Management of Alarm Systems for the Process Industries*. ISA, The Instrumentation Systems and Automation Society., 2009.

[12] The Instrumentation Systems ISA and Automation Society. *ANSI/ISA-5.1-2009, Instrumentation Symbols and Identification*. ISA, The Instrumentation Systems and Automation Society., 2009.

[13] Wen Junhao, Gu Jianan, Jiang Zhuo, and Zhu Yijiao. Semantic Web Service Selection Algorithm Based on QoS Ontology. *2011 International Joint Conference on Service Sciences*, (1):163–167, 2011. doi: 10.1109/IJCSS.2011.39.

[14] Muhammad Golam Kibria, Sajjad Ali, Muhammad Aslam Jarwar, Sunil Kumar, and Ilyoung Chong. Logistic model to support service modularity for the promotion of reusability in a Web Objects-Enabled IoT environment. *Sensors (Switzerland)*, 17(10), 2017. ISSN 14248220. doi: 10.3390/s17102180.

[15] Laboratory for Analysis and Architecture of Systems (LAAS). The eclipse om2m project. URL: http://www.eclipse.org/om2m/, 1 2015.

[16] Friedemann Mattern and Christian Floerkemeier. From the internet of computers to the internet of things. *Informatik-Spektrum*, 33(2):107–121, 2010.

[17] João Luiz Moreira, L.M. Daniele, Luis Ferreira Pires, Marten J. van Sinderen, Katarzyna Wasielewska, Pawel Szmeja, Wieslaw Pawlowski, Maria Ganzha, and Marcin Paprzycki. Towards iot platforms integration: Semantic translations between w3c ssn and etsi saref. 11 2017. SEMANTiCS conference 2017 ; Conference date: 11-09-2017 Through 14-09-2017.

[18] NLTK Project. Natural language toolkit. URL: http://www.nltk.org/howto/wordnet.html, 04 2019.

[19] E. Oberortner, U. Zdun, and S. Dustdar. Tailoring a model-driven quality-of-service dsl for various stakeholders. In *2009 ICSE Workshop on Modeling in Software Engineering*, pages 20–25, May 2009. doi: 10.1109/MISE.2009.5069892.

[20] Ontology Alignment Evaluation Initiative. Ontology alignment evaluation initiative - oaei 2018 campaign. URL: http://oaei.ontologymatching.org/2018/, 01 2018.

[21] Massimo Paolucci, Takahiro Kawamura, Terry R Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities. In Ian Horrocks and James Hendler, editors, *The Semantic Web — ISWC 2002*, pages 333–347, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-48005-1.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

[23] Henrique Brittes Pötter and Alexandre Sztajnberg. Adapting heterogeneous devices into an iot context-aware infrastructure. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS'16, pages 64–74, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4187-5. doi: 10.1145/2897053.2897072. URL `http://doi.acm.org/10.1145/2897053.2897072`.

[24] Lalit Purohit and Sandeep Kumar. Web Service Selection Using Semantic Matching. In *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, AICTC '16, pages 16:1—-16:5, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4213-1. doi: 10.1145/2979779.2979795. URL `http://doi.acm.org/10.1145/2979779.2979795`.

[25] Shuping Ran. A Model for Web Services Discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003. ISSN 1551-9031. doi: 10.1145/844357.844360. URL `http://doi.acm.org/10.1145/844357.844360`.

[26] Minwoo Ryu, Jaeho Kim, and Jaeseok Yun. Integrated semantics service platform for the internet of things: a case study of a smart office. *Sensors*, 15(1):2137–2160, 01 2015.

[27] N Ben Seghir and O Kazar. A new framework for web service discovery in distributed environments. In *2017 First International Conference on Embedded Distributed Systems (EDiS)*, pages 1–6, 2017. doi: 10.1109/EDIS.2017.8284046.

[28] SELF Software Evolution Lab. Self software evolution lab. URL: https://github.com/SELF-Software-Evolution-Lab/Adaptive-Architecture-for-Transient-IoT-Systems, 06 2018.

[29] P. Shvaiko and J. Euzenat. *Ontology Matching*. 2013. ISBN 9783642387203. URL `http://www.ontologymatching.org/index.html`.

[30] Smart Santander. Smart santander. URL: http://www.smartsantander.eu, 01 2018.

[31] Seheon Song, Sang Oh Park, Sang Il Lee, and Jae Hyun Park. Mission-oriented service development using capability-based semantic recommendation for the internet of things. *Multimedia Tools and Applications*, pages 1–23, 2017. ISSN 15737721. doi: 10.1007/s11042-017-4889-1.

[32] Yan Tang and Robert Meersman. DIY-CDR: an ontology-based, Do-It-Yourself component discoverer and recommender. *Personal and Ubiquitous Computing*, 16(5):581–595, jun 2012. ISSN 1617-4917. doi: 10.1007/s00779-011-0416-y. URL `https://doi.org/10.1007/s00779-011-0416-y`.

[33] Ming Tao, Kaoru Ota, and Mianxiong Dong. Ontology-based data semantic management and application in IoT- and cloud-enabled smart homes. *Future Generation Computer Systems*, 76:528–539, 2017. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2016.11.012. URL `http://www.sciencedirect.com/science/article/pii/S0167739X1630615X`.

[34] H. T. Tran, H. Baraki, R. Kuppili, A. Taherkordi, and K. Geihs. A notification management architecture for service co-evolution in the internet of things. In *IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Environments*, MESOCA'16, pages 9–15, 10 2016. doi: 10.1109/MESOCA.2016.8.

[35] Vuong Xuan Tran. WS-QoSOnto: A QoS ontology for Web services. *Proceedings of the 4th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2008*, pages 233–238, 2008. doi: 10.1109/SOSE.2008.17.

[36] Rob van Kranenburg and Alex Bassi. Iot challenges. *Communications in Mobile Computing*, 1(1):9, 11 2012. ISSN 2192-1121. doi: 10.1186/2192-1121-1-9. URL `https://doi.org/10.1186/2192-1121-1-9`.

[37] Z Wang, R Bie, and M Zhou. Hybrid Ontology Matching for Solving the Heterogeneous Problem of the IoT. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1799–1804, 2012. doi: 10.1109/TrustCom.2012.177.

[38] World Wide Web Consortium (W3C). Sosa ontology. URL: https://www.w3.org/2015/spatial/wiki/SOSA_Ontology, 11 2016.

[39] World Wide Web Consortium (W3C). Semantic sensor network ontology. URL: https://www.w3.org/TR/2017/REC-vocab-ssn-20171019/, 10 2017.

[40] Baocai Yin, Huirong Yang, Pengbin Fu, and Xiaobo Chen. A semantic web services discovery algorithm based on qos ontology. In Aijun An, Pawan Lingras, Sheila Petty, and Runhe Huang, editors, *Active Media Technology*, pages 166–173, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[41] M Zhou and Y Ma. A web service discovery computational method for IOT system. In *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, volume 03, pages 1009–1012, oct 2012. doi: 10.1109/C-CIS.2012.6664533.