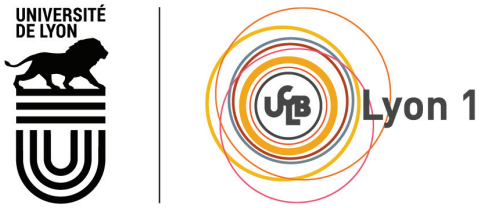
Semantics-Based Multi-Purpose Contextual Adaptation in the Web of Things

Mehdi Terdjimi



N° d'ordre NNT : 2017LYSE1315

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
l'Université Claude Bernard Lyon 1

École Doctorale ED512
Infomaths

Spécialité de doctorat : Informatique

Soutenue publiquement le 18/12/2017, par :
Mehdi Terdjimi

Adaptation Contextuelle Multi-Préoccupations Orientée Sémantique dans le Web des Objets

---

Devant le jury composé de :

Taconet Chantal, Maître de conférences (HDR), Télécom SudParis	Rapporteure
Molli Pascal, Professeur, Université de Nantes	Rapporteur
Laforest Frédérique, Professeure, Télécom Saint-Etienne	Examinatrice
Gyrard Amélie, Chercheure Post-Doctorale, École des Mines de Saint-Étienne	Examinatrice
Monteil Thierry, Professeur, INSA Toulouse	Examinateur
Champin Pierre-Antoine, Maître de conférences (HDR), Université Lyon 1	Examinateur
Mrissa Michaël, Professeur, Université de Pau et des Pays de l'Adour	Directeur de thèse
Médini Lionel, Maître de Conférences, Université Lyon 1	Co-directeur

UNIVERSITE CLAUDE BERNARD - LYON 1

Président de l'Université	**M. le Professeur Frédéric FLEURY**
Président du Conseil Académique	M. le Professeur Hamda BEN HADID
Vice-président du Conseil d'Administration	M. le Professeur Didier REVEL
Vice-président du Conseil Formation et Vie Universitaire	M. le Professeur Philippe CHEVALIER
Vice-président de la Commission Recherche	M. Fabrice VALLÉE
Directrice Générale des Services	Mme Dominique MARCHAND

COMPOSANTES SANTE

Faculté de Médecine Lyon Est – Claude Bernard	Directeur : M. le Professeur G.RODE
Faculté de Médecine et de Maïeutique Lyon Sud – Charles Mérieux	Directeur : Mme la Professeure C. BURILLON
Faculté d'Odontologie	Directeur : M. le Professeur D. BOURGEOIS
Institut des Sciences Pharmaceutiques et Biologiques	Directeur : Mme la Professeure C. VINCIGUERRA
Institut des Sciences et Techniques de la Réadaptation	Directeur : M. X. PERROT
Département de formation et Centre de Recherche en Biologie Humaine	Directeur : Mme la Professeure A-M. SCHOTT

COMPOSANTES ET DEPARTEMENTS DE SCIENCES ET TECHNOLOGIE

Faculté des Sciences et Technologies	Directeur : M. F. DE MARCHI
Département Biologie	Directeur : M. le Professeur F. THEVENARD
Département Chimie Biochimie	Directeur : Mme C. FELIX
Département GEP	Directeur : M. Hassan HAMMOURI
Département Informatique	Directeur : M. le Professeur S. AKKOUCHE
Département Mathématiques	Directeur : M. le Professeur G. TOMANOV
Département Mécanique	Directeur : M. le Professeur H. BEN HADID
Département Physique	Directeur : M. le Professeur J-C PLENET
UFR Sciences et Techniques des Activités Physiques et Sportives	Directeur : M. Y.VANPOULLE
Observatoire des Sciences de l'Univers de Lyon	Directeur : M. B. GUIDERDONI
Polytech Lyon	Directeur : M. le Professeur E.PERRIN
Ecole Supérieure de Chimie Physique Electronique	Directeur : M. G. PIGNAULT
Institut Universitaire de Technologie de Lyon 1	Directeur : M. le Professeur C. VITON
Ecole Supérieure du Professorat et de l'Education	Directeur : M. le Professeur A. MOUGNIOTTE
Institut de Science Financière et d'Assurances	Directeur : M. N. LEBOISNE

Remerciements

Cette thèse a été financée par l'Agence Nationale de la Recherche sous le numéro <ANR-13-INFR-012>. Je tiens à remercier particulièrement Michaël Mrissa et Lionel Médini pour m'avoir donné l'opportunité de faire cette thèse, qui a été pour moi le fruit d'échanges et d'expériences enrichissantes.

J'adresse mes sincères remerciements à Chantal Taconet et Pascal Molli, qui ont accepté de juger et d'évaluer le travail que j'ai produit durant ces trois années de thèse. Je souhaite aussi remercier mes examinateurs Frédérique Laforest, Amélie Gyrard, Thierry Monteil, ainsi que Pierre-Antoine Champin.

Enfin, je tiens à remercier mes proches, mon père, mon frère, ma soeur, Perrine, ainsi que Teemo, pour m'avoir soutenu non seulement durant toute cette thèse, mais aussi durant toute ma vie. Cette thèse est dédiée a ma mère, qui n'aura jamais douté de moi une seule seconde.

Semantics-Based Multi-Purpose Contextual Adaptation in the Web of Things

Mehdi Terdjimi
PhD Thesis

Contents

Résumé

Le Web des Objets s'inscrit dans divers domaines d'application, tels que la domotique, les entreprises, l'industrie, la médecine, la ville, et l'agriculture. Il se présente comme une couche uniforme placée au-dessus de l'Internet des Objets, afin de surmonter l'hétérogénéité des protocoles présents dans ces réseaux.

Une valeur ajoutée des applications Web des Objets est de pouvoir combiner l'accès à divers objets connectés et sources de données externes avec des techniques standards de raisonnement sémantique (RDF-S, OWL). Cela leur permet alors d'interpréter et de manipuler de ces données en tant qu'informations contextuelles. Ces informations contextuelles peuvent être exploitées par ces applications afin d'adapter leurs composants en fonction des changements dans leur environnement.

L'adaptation contextuelle est un défi majeur pour le Web des Objets. En effet, les solutions d'adaptation existantes sont soit fortement couplées avec leur domaine d'application (étant donné qu'elles reposent sur des modèles de contexte spécifiques au domaine), soit proposées comme composant logiciels autonomes, difficiles à intégrer dans des architectures Web et orientées sémantique. Cela mène alors à des problèmes d'intégration, de performance et de maintenance.

Dans cette thèse, nous proposons une solution d'adaptation contextuelle multi-préoccupations pour les applications Web des Objets, répondant à des besoins d'utilisabilité, de flexibilité, de pertinence et de performance. Notre travail se base sur un scénario pour l'agriculture numérique et se place dans le cadre de la plateforme orientée-avatar ASAWoO. Premièrement, nous proposons un méta-modèle générique permettant de conçevoir des modèles contextuels standards, interopérables et réutilisables. Deuxièmement, nous présentons un cycle de vie du contexte et un *workflow* d'adaptation contextuelle, permettant la sémantisation de données brutes, ainsi que la contextualisation en parallèle durant l'exécution de l'application. Ce workflow combine des données issues de sources hétérogènes, telles que l'expertise du domaine, les documentations techniques des objets, les données de capteurs et de services Web, etc. Troisièmement, nous présentons une méthode de génération de règles d'adaptations basées sur des situations contextuelles, permettant de limiter l'effort des experts et concepteurs lors de l'élaboration d'applications adaptatives. Quatrièmement, nous proposons deux optimisations pour le raisonnement contextuel : la première adapte la localisation des tâches de raisonnement en fonction du contexte, la seconde améliore le processus de maintenance incrémentale d'informations contextuelles.

Abstract

The Web of Things (WoT) takes place in a variety of application domains (*e.g.* homes, enterprises, industry, healthcare, city, agriculture...). It builds a Web-based uniform layer on top of the Internet of Things (IoT) to overcome the heterogeneity of protocols present in the IoT networks.

WoT applications provide added value by combining access to connected objects and external data sources, as well as standard-based reasoning (RDF-S, OWL 2) to allow for interpretation and manipulation of gathered data as contextual information. Contextual information is then exploited to allow these applications to adapt their components to changes in their environment. Yet, contextual adaptation is a major challenge for the WoT. Existing adaptation solutions are either tightly coupled with their application domains (as they rely on domain-specific context models) or offered as standalone software components that hardly fit in Web-based and semantic architectures. This leads to integration, performance and maintainability problems.

In this thesis, we propose a multi-purpose contextual adaptation solution for WoT applications that addresses usability, flexibility, relevance, and performance issues in such applications. Our work is based on a smart agriculture scenario running inside the avatar-based platform ASAWoO. First, we provide a generic context meta-model to build standard, interoperable et reusable context models. Second, we present a context lifecycle and a contextual adaptation workflow that provide parallel raw data semantization and contextualization at runtime, using heterogeneous sources (expert knowledge, device documentation, sensors, Web services, etc.). Third, we present a situation-driven adaptation rule design and generation at design time that eases experts and WoT application designers' work. Fourth, we provide two optimizations of contextual reasoning for the Web: the first adapts the location of reasoning tasks depending on the context, and the second improves incremental maintenance of contextual information.

Chapter 1

Introduction

Decades ago, the outbreak of the first connected devices (computers, personal digital assistants, smartphones, tablets, and other appliances) has marked the emergence of the Internet of Things (IoT). This initiative was pushed by both academics and the industry, in response to the increasing need for integrating various appliances on both the Internet and local networks. By these means, applications are provided fast and substantial knowledge that emerge from interactions with the users and connected devices, through data gathering, sharing and publishing. Historically, the terms "Internet of Things" (IoT) came in 1999 from the Auto-ID Labs[1], an international network of academic research laboratories whose purpose was to connect various devices to the Internet through the usage of RFID, QR codes, barcodes, and other means to identify objects. Since its advent, the term *thing* gained a broad variety of synonyms. A thing could be either an *object*, a *device*, an *appliance*, or an *agent*, depending on both the applicative context and the expertise field of IoT solutions designers.

Despite the fact that devices are subject to hardware and software constraints, they are able to augment their basic capabilities (memory, CPU, sensors and actuators) through knowledge processing and sharing within connected devices networks. Augmented devices have extended capabilities to realize complex tasks. This augmentation is achieved through the composition and abstraction of basic capabilities, which are initially provided on physical objects by the device manufacturers. Cyber-physical systems (CPS) tackles limitations of devices to provide efficient real-time interactions, through the integration of physical devices (the physical part) and software components (the virtual part) [Lee, 2008]. They rely on *feedback loops* in order to recursively impact physical and software pro-

[1] `http://www.autoidlabs.org/`

cesses all along the application lifecycle. CPS usually answer specific needs in particular domains, such as biomedical and healthcare systems, smart grids and renewable energy solutions [Baheti and Gill, 2011].

However, according to the Levels of Conceptual Interoperability Model (LCIM) described in [Tolk and Muguira, 2003], most IoT solutions lack technical, syntactic, and semantic interoperability. First, they use different data transfer protocols and data formats, which are sometimes both proprietary. Second, they have different meanings and interpretations for the data they manipulate. This leads to proprietary silos, *i.e.* to complex and hardly reusable applicative solutions. The Web of Things (WoT) was recently designed to deal with such issues. It relies on Web technologies and standards to support interactions between things. The ambition of the WoT is to provide a layer on top of the IoT, where things can be involved in software applications in a standardized, interoperable and secure manner[2] through Web standards.

1.1 The Web of Things

The number of connected devices grew considerably since their advent. According to [Gubbi et al., 2013], the number of interconnected devices is expected to reach 24 billions by 2020. Although this is an opportunity for companies to rapidly gain profit through IoT technologies, the increasing number of connected devices leads to even more heterogeneity within appliances, in both syntactic and semantic terms. This ends up with complex device integration problems on IoT platforms, due to the lack of standardization and proprietary silos. Hence, these problems greatly weakens the (re)usability potential of devices and IoT application architectures, as application designers are forced to continuously learn how to operate similar devices from different manufacturers for identical tasks. For these reasons, recent advances in IoT research aim to exploit the potential of the Web 1) to avoid "reinventing the wheel", by taking advantage of already proven standards and technologies (HTTP protocol and verbs, REST architectural principles [Fielding, 2000], hypermedia, Semantic Web representation and reasoning, Web RTC, Websockets, JSON serialization, etc.), 2) to solve the interoperability issues caused by IoT silos, and 3) to gather information from the Web, which in turn produces additional knowledge for both application users and companies.

[2] `https://www.w3.org/WoT/`

Integrating things on the Web. In 2002, the authors of the CoolTown project [Kindberg et al., 2002] proposed to link physical objects with a Web page through Uniform Resource Identifiers (URIs). Following this, the year 2005 has been marked with several efforts and propositions on using Web standards such as SOAP [Thompson, 2005] for the IoT. Solutions like OASIS [Dolin, 2006] have been proposed, with the intent to resolve interoperability issues. OASIS relies on the SOAP/XML specification to allow for communication between devices. Some other following projects and research in the field have attempted to exploit the potential of REST to provide generic, reusable and flexible Web-based architectures. For instance, Luckenbach et al. designed TinyREST [Luckenbach et al., 2005], a Protocol for Integrating Sensor Networks into the Internet. TinyREST is based on an REST-like architectural concept, applied to sensors and actuators, using limited verbs: GET (to access device status and component values), POST (to command sensors and accuators) and SUBSCRIBE (to register to specific services, for further event notification). In [Wilde, 2007], Wilde proposed to integrate things into RESTful architectures, to provide substantial advantages over state-based applications. Using REST as the toolbox to build universal APIs for embedded devices illustrates Guinard and Trifa's view of the WoT [Guinard et al., 2010].

Things may also provide different representations of themselves through HTTP content negotiation, depending on the needs: it could be HTML for human browsing of things, or JSON for data exchange, as proposed in Sun SPOT [Guinard et al., 2011]. In this project, resources are accessed and manipulated via RESTful Web Services using four HTTP verbs: GET for resource representation, PUT for resource updating, POST for subscribing to a rule (for change notifications) and DELETE to shutdown a node or unsuscribe to a rule. More recently, Mrissa et al. have proposed in [Mrissa et al., 2014c] a classification of physical objects, from the basic object with no sensor nor resource to the resourceful object, capable of embedding his virtual representation, without the requirement to be connected to the Internet. This consideration allows things to be referenced by an URI even though they do not possess any RFID tag, barcode, nor QR-code. Hence, the representation of the thing is obtainable through an HTTP GET request on its URI.

The Semantic Web and the WoT. The Semantic Web is a strong opportunity for the WoT, as semantics and reasoning capabilities brings additional information and provides "intelligent" behavior for things, in a standardized manner. In WoT applications, anything (including things) could be semantically annotated using description logics (DL) such as RDF [Manola et al., 2004], RDF Schema [Brickley et al., 2014] and OWL 2 [Hitzler et al., 2009]). On the whole, OWL – which is built over RDF – provides more expressivity than

RDF and RDFS by means of the concepts of classes and property types, and allows representing any knowledge as ontologies. Hence, current applications could solve many semantic interoperability issues by the means of ontologies to describe the domain, the participants and the processes, using recommended ontologies, following ontology design patterns [Gangemi and Presutti, 2009], and relying on ontology mappings [Euzenat et al., 2007, Kalfoglou and Schorlemmer, 2003].

The WoT nowadays. As of today, the WoT offers numerous benefits. First, it provides reusable platforms, technologies and user interaction techniques. Second, it eases application development and reduces time-to-market for applications. Third, it bridges the gap between the virtual and the physical worlds by providing a representation and access to any kind of thing in a generic way, as Web resources. To provide such benefits, existing WoT applications rely on Web standards, among which the exposition of things as RESTful resources [Fielding, 2000] and semantic descriptions of these resources with DL as explained previously. WoT standards thus define semantic vocabularies (Thing Description[3]) and programming interfaces (servient[4], scripting API[5]) reused in projects to design actual WoT applications. Recently, [Mrissa et al., 2015] proposed a WoT application framework called ASAWoO. ASAWoO comes with the notion of avatar, to allow for the integration of semantic reasoning within servient-like interfaces [Médini et al., 2016]. By these means, ASAWoO claims its intention to bridge the gap between the Web and Semantic Web.

1.2 The ASAWoO Platform

The current issues with CPS and classic IoT Machine-to-Machine (M2M) paradigms is the lack of flexibility and reusability of their architectures. To this end, ASAWoO – which stands for "Adaptive Supervision of Avatar/Object Links for the Web of Objects" – consists in a WoT platform that aims to connect devices (objects) together in reusable, comprehensive and well-architectured WoT applications. The purpose of the ASAWoO platform is to answer different concerns in WoT applications. These concerns include discovering functionalities from physical devices API, composing of complex applications, filtering of inade-

[3] https://www.w3.org/WoT/IG/wiki/Thing_Description

[4] https://w3c.github.io/wot/architecture/wot-architecture.html#general-description-of-wot-servient

[5] https://www.w3.org/WoT/IG/wiki/Web_of_Things_scripting_API

quate functionalities (with respect to various QoS concerns), providing disruption-tolerant networks and relying on multi-agent organization and coalitions to achieve collaboration between objects.

ASAWoO is based on the concept of avatar, which represents the software part attached to an object in a CPS-inspired approach. It embeds the components that implement the previous concerns required for the thing to participate in WoT applications through standard Web interfaces. Avatars expose the objects high-level functionalities to clients (users or other avatars) inferred from the physical capabilities of objects (*i.e.* their APIs) using a semantic approach. The functionalities directly inferred from capabilities are called *atomic functionalities*, whereas functionalities that are composed by other ones are called *composite functionalities*. Some functionalities may be local to an avatar (*i.e.* directly implemented by the device) or collaborative (*i.e.* requires other avatars to complete the missing functionalities it is composed of). The avatar architecture includes various components and managers detailed in Appendix A, which handle specific concerns.

To perform the tasks described hereafter, avatar managers have specific responsibilities within the WoT application lifecycle. The Local Functionality Manager must choose the suitable capabilities to implement an atomic functionality, the Collaborative Functionality Manager sets up collaborative functionalities with suitable avatars, the Functionality Deployment Manager has the responsibility of migrating functionality code modules in suitable location, and the Applicative and Network Protocol Managers are in charge of switching appropriate protocols. The choices made by each of these managers (*i.e.* functionality implementation, composition and exposition, code deployment, communication protocols) depends on the context. For this reason, the ASAWoO platform includes an additional component – the Context Manager – which is in charge of filtering these choices to provide adaptation for several concerns.

Contextual adaptation in ASAWoO. For managers to take appropriate decisions to answer a concern is not straightforward: many options coexist, and some options should not be chosen due to user preferences, security (which includes both physical and cyber threats), quality-of-service (QoS), or other policies. To both find the optimal choices and block inconvenient functionalities at runtime, avatars must be aware of the context. As such, WoT applications are strongly dependent to the context, and require adaptation in order to provide the most favorable decisions all along their lifecycle. For these reasons, avatars require an additional component – the Context Manager – to reason about contextual information, in order to give optimal and viable decisions to other managers, providing

adaptation for WoT applications for each concern. To perform all these adaptation tasks, we aim to provide contextual adaptation in WoT applications by means of Semantic Web concepts and technologies.

1.3 Objectives & Plan

WoT applications are built in a same fashion. Hence, they have common adaptation requirements: they need to adapt their communications protocols, their functionalities[6], as well as their collaborative setups. In ASAWoO, the avatars managers are in charge of dealing with various concerns in conjunction with the Context Manager, which allows suggesting the optimal decisions for these concerns.

In this respect, the contextual adaptation for WoT applications requires a novel approach, which raises several research questions. The objectives to the thesis are the following:

- Provide standard, interoperable and reusable adapative WoT solutions, by fully taking advantage of Web technologies
- Provide generic, extensible and multi-concern adaptation processes with the multitude of applications and scenarios
- Ease and speed up the design process of adaptive solutions in WoT applications
- Provide efficient adaptation processes in WoT environments at runtime

In this thesis, we deal with the objectives enumerated previously through the following scientific contributions:

1) **A generic context meta-model for WoT applications.** This context meta-model aims to unite state-of-the-art context modeling with cross-cutting adaptation concerns, using semantic annotations to allow for standard-based RDF-S and OWL reasoning on contextual information.
2) **A context lifecycle and an adaptation workflow that provide parallel raw data semantization and contextualization at runtime.** The context lifecycle deals with data

[6] Although the notion of functionality differs across platforms, the issue of adapting the them to several changes in the environment is common to any application.

transformation, from the raw data transmission to the adaptation decision. The adaptation workflow deals with data integration (*i.e.* when contextual data is pushed to the avatar) and query answering (*i.e.* when the avatar managers send a purpose-based adaptation question to retrieve the optimal adaptation possibilities), by using simple SPARQL queries in conjunction with an incremental reasoner.

3) **An implementation that generates adaptation rules at design time.** This solution relies on information about the context model components and the application infrastructure to generate adaptation rules at design time. These adaptation rules infer adaptation scores to ensure optimal adaptation decision.

4) **Two optimizations of contextual reasoning for the Web.** The first optimization aims to take advantage of contextual information and the purpose-based adaptation methodology to provide an adaptive reasoning process, using a location-agnostic incremental reasoner. The second optimization aims to reduce the overhead of the incremental maintenance overdeletion step through *fact-tagging*.

This thesis is organized as follows. In Chapter 2, we present a general state-of-the art on context modeling, detail our contribution on multi-purpose contextual modeling through the concept of *context meta-modeling*, and present a sustainable agriculture scenario that will be used thorough this thesis to illustrate our contributions. In Chapter 3, we present our contribution on multi-purpose contextual adaptation and detail a contextual lifecycle along with its adaptation workflow to be used in WoT applications. In Chapter 4, we describe how to build and generate ranked adaptation rules using information about the context model. In Chapter 5, we tackle optimization issues in semantic reasoning that impact the contextual adaptation process. We present the two contributions to improve contextual reasoning described above. We conclude this thesis in Chapter 6 and discuss future challenges in semantics-based contextual adaptation for the WoT.

Chapter 2

Multi-Purpose Context Modeling

Contents

2.1 Introduction

The Web of Things (WoT) takes place in various domains such as homes, enterprises, industry, healthcare, city or agriculture. It builds a Web-based uniform layer on top of the Internet of Things (IoT) to overcome the heterogeneity of protocols present in the IoT networks. To react to changes in their environment and exhibit context-adaptive behavior, WoT applications need relevant context models. WoT applications provide added value by combining access to connected objects and external data sources (i.e. Web services), as well a standard-based reasoning (RDF-S, OWL 2) in order to interpret and manipulate the

gathered data as contextual information. As a consequence of the diversity of use-cases and applications, numerous domain-specific models relying on different formalisms and reasoning mechanisms have been designed [Perera et al., 2014]. However, a single context modeling framework for WoT apps is still missing. To provide cross-domain and interoperable context models, an accurate description and exploitation of WoT application requirements and adaptation purposes is needed. In this chapter, we present a state of the art on context modeling, propose a context meta-model that allows designing context models to solve purposes that are common to any WoT application domain, and illustrate this contribution in a smart agriculture scenario.

2.2 State of the art on context modeling

Related work in context awareness rely on various context models. In the literature, authors usually group context information of the same type together, and some of them categorize these groups as *contextual dimensions*. The following state of the art on context modeling explores and analyses different concerns, from concrete representations to abstract and high-level considerations about context.

Context in the physical world. In [Abowd et al., 1999], the authors identify the context as any information that answers the questions *Where*, *Who*, *When* and *What*, which compose the physical world of entities (*i.e.* things and users). The physical world is one of the main components of former and current state of the art context models. It includes the environment, the physical characteristics of entities, their activities and their surroundings. Such context models rely on geospatial data and information about physical entities. Various applications exploit these aspects, from user assistants to custom web content based services, but use different representations of context.

PARCTAB [Schilit et al., 1993] uses three pieces of contextual information: Date and Time, Location and Co-location (*i.e.* what is nearby). [Schmidt, 2003] presents a three-dimensional context, composed of the Environment (physical and social), Self (device state, physiological and cognitive considerations) and the Activity (behavior and task). In [Zimmermann et al., 2007], the authors object to the three-dimensional context proposed in [Schmidt, 2003] as "*the Self dimension introduces a relation of the context to one specific entity [...] which lacks an approach of how his model would capture a setting comprised of many interact-*

ing entities", according to them. For these reasons, they extend this model by adding the Relations dimension. They also use the Location and the Time dimensions instead of the Environment proposed in [Schmidt, 2003]. In [Abowd et al., 1999], Dey and Abowd consider the Environment dimension as a redundant information, as they see it as a synonym for context. For this reason, they propose to replace Environment by Activity. They also exclude Schilit's *nearby* consideration [Schilit et al., 1994], because it overlaps the Location and the Identity dimensions. This way, these two dimensions can be used separately, so that information used to either locate or identify a thing can be realized independently.

More recent works rely on the Environment and the Activity (*i.e.* user-related) dimensions. This the case for the Context-Aware Web Browser [Cop, 2010], which allows automatic retrieval and constant update of the contextual information gathered from the physical world. They rely on context descriptors using the Location, Time, Activity, Posture and Privacy dimensions, as well as information that characterize the context itself (Probability, Importance, Description and Name).

Context at the communication layer. Quality of service, privacy and security concerns motivate the usage of context for the communication between objects. Indeed, such information allow to use the suitable protocols, network topologies and access control policies, to deal with resource-constrained networks [Raverdy et al., 2006]. Gold and Mascolo [Gold and Mascolo, 2001] use context for mobile peer-to-peer (P2P) networks. They use computing resources (availability, remaining battery power) and network information (services in reach, distances) to optimize the P2P network routing structure. The Context-aware Adaptive Routing (CAR) system [Musolesi and Mascolo, 2009] includes dynamic information in its context model, such as the change rate of connectivity, i.e. the number of (dis)connections a host experienced over a certain time.

In distributed architectures, Mascolo et al. [Mascolo et al., 2002] characterize two types of network connection: permanent, via continuous high-bandwidth links, or intermittent, when encountering disconnections due to unpredictable failures. In intermittent networks, the performance of wireless networks may vary depending on the protocols and technologies being used.

In a similar manner, Wei et al. [Wei et al., 2006] identify and separate static context from dynamic context. Static information is the user's profile and history, the location of network access points, the capacities and the services of the network, as well as its policies. Dynamic information is related to location prediction and status, and current load of the network. Static context and dynamic context is also used in service discovery applications,

such as the Multi-Protocol Service Discovery and Access (MSDA) middleware [Raverdy et al., 2006]. This middleware provides accurate routing for service discovery, using static parameters such as the type of network, the supported protocols, the security levels, etc. It also uses dynamic parameters, such as the number of active users and available services, the current data load, and the control policies (*i.e.* incoming and outgoing messages in the network).

Context in the application architecture. The choice and usage of context strongly depends on the application itself. Designed context models are therefore highly specific [Abowd et al., 1999], yet necessary to answer its functional needs accordingly. In the literature, the context is used to describe various application architectures, from Web-service based to groupware and collaborative systems. They sometimes use aspect-oriented models as a solution to properly separate the information concerning the application-core from its business logic.

According to both [Gensel et al., 2008] and [Chaari et al., 2005], the application core must be designed separately from the context information and its processing engine. In [Chaari et al., 2005], Chaari et al. define the context for applications as "the application's external parameters which impacts its behavior, defining new views for its data and services". They propose five dimensions: Communication, User, Terminal, Location and Environment. The modularized approach from [Munnelly et al., 2007] address the problem of "tangled code", using aspect-oriented approach and context information. They context model includes eight dimensions: Device, Location, User, Social, Environmental, System, Temporal and Application-specific. In this approach, Device, Location, Temporal, Environment and Social dimensions are similar to Schmidt's [Schmidt, 2003] as they are based on his earlier work [Schmidt et al., 1999].

In their survey [Truong and Dustdar, 2009], Truong and Dustdar show that Web-service based applications use similar dimensions as groupware and collaborative systems, such as the Activity/Task, the Team and the Machine/Device dimensions. ESCAPE [Truong et al., 2007] and inContext [Truong et al., 2008] separate the device from the application by using the Service/Application dimensions.

According to Euzenat et al., [Euzenat et al., 2008], context may have different representations for a same situation. They believe that context models must enable the aggregation and separation of context, to allow for context sharing between the application components. Coutaz et al. [Coutaz et al., 2005] categorize the context in different layers: the Sensing layer (numeric observables), the Perception layer (symbolic observables) and the Situation and

Context Identification layer (conditions for moving between situations and contexts). The Thing-REST architecture [He et al., 2012] separates the context in two categories: the Semantic context (human knowledge about the thing, static and predictable) and the Sensing context (dynamically changeable and unpredictable knowledge, gathered from sensors). In their system, each piece of information is reused across Web services. This supports the view from [Euzenat et al., 2008, Coutaz et al., 2005, He et al., 2012] of a shared, accessible, separated and aggregated context within the application.

Context for application users. Some of the literature relies on context information about the user to adapt the applications behavior. This type of information is usually combined with information about the physical world to extract additional useful information about the user's context, to improve the applications features. This is the case for the Conference Assistant [Dey et al., 1999], which uses different context information, depending on their level of privacy: the public context (location, time, presentation's keywords, people presenting, media used) the user's context (time they entered or exited a room, their location, the questions they asked, and the elements from slides that are pointed out), as well as the other users' context (their presence and the question they asked in presentations).

Context has also been used to provide accurate and adapted content to the user itself. This is the case for the query recommendation system from [Cao et al., 2008, Cao et al., 2009], which store user sessions (search queries and click-through) and use it as context information. The query-ranking systems from [Xiang et al., 2010] relies on the consecutive user query reformulations and query specialization/generalization. The query autocompletion system from [Arias, 2008] uses human and device related dimensions (User profile, Device and Browser) as well as physical-related dimensions (Geospatial, Environment and Date/Time) to provide accurate suggestions. The recommendation system from [Yu et al., 2006] relies on three contextual information: the user media preferences, the user situation and the terminal capability (which contains the bandwidth status and the device supported media). The work from Alti et al. in [Alti et al., 2012] consists in a environment-based media content adaptation in the context of mobility. They propose a context model based on four dimensions: User (profile, preferences), Mobile Device, Document (format) and Service (QoS, role).

Groupware systems also rely on the user's context to adapt their behavior. In [Kirsch-Pinheiro et al., 2004], the authors rely on the user's preference. They propose a context model for groupware systems, using five dimensions: Space (physical location), Tool (device and application), Time (group calendar), Community (referring to the concepts of

group, users and roles) and the Process viewpoint (activities with shared objects, handled by the group). They also represent user profiles, using the user's preferences and the constraints to be satisfied by the system for a group member, a role or a device. This approach confronts the current user context and the profiles and situations in which they are valid, i.e. the application context [Kirsch-Pinheiro et al., 2006].

Context in social computing. In social computing, entities that interact with the application are not only the devices, but also the users themselves. Related work relies on the cognitive and organizational aspects of the context, to improve decision making in multi-agent systems.

In the domain of artificial intelligence, the lack of an operational definition of context explains several failures in knowledge based systems, according to Brézillon and Pomerol [Brézillon and Pomerol, 1996]. In [Brézillon, 1999], the author state that "*these problems concern the exclusion of the user from the problem solving, the misuse and lack of knowledge-based systems and the impossibility to generate relevant explanations*". He explains the lack of consensus behind the definition of context in the literature. To deal with this problem in context modeling, Brézillon describes the context at different levels: *static or dynamic knowledge*, respectively any constant knowledge or changing knowledge through the whole interaction, and *contextual or contextualized knowledge*, respectively the explicit knowledge or the implicit knowledge that intervenes in the problem solving. He defines context as a shared knowledge space that is explored and exploited by agents in the interaction. Contexts can then be organized into a hierarchy, by creating a higher-level context from already existing ones. Shared knowledge includes elements from the domain, the users, their environment and their interaction with the system.

In [Bucur et al., 2005], the authors identify the Environmental context of an agent and the Organizational context resulting from their interactions. They define context as "a finality, and the set of attributes that are relevant for that finality". Much like Brézillon [Brézillon, 1999], each agent has its own context and can share his knowledge through its interactions. Brézillon and Pomerol [Brézillon and Pomerol, 1999] separate this knowledge in three categories: *proceduralized knowledge*, the shared knowledge between agents involved in the decision making step that is used directly for the problem solving, *contextual knowledge*, the implicit knowledge that influences the problem solving and *external knowledge*, the knowledge known by involved agents but that is not used in the current decision making step. The dynamic context comes from switching between *proceduralized* and *contextual knowledge*. Hence, between each decision making step, pieces of *contextual knowledge* becomes

either *procedural* or *external*, and vice versa. Brézillon identifies in [Brézillon, 2003] the following information to build this knowledge: the Domain, the User, the Environment and the Interactions. In [Bazire and Brézillon, 2005], Bazire and Brézillon propose a context model that represents the components of a situation. A situation is defined by an User, an Item in a particular Environment, and an Observer, where each of them interferes with a related context.

Synthesis on context modeling state of the art. The related work highlights the diversity and complexity of context. The purpose of this classification is to illustrate the variety of each type of contextual information, to exhibit their characteristics in a more comprehensive manner. In this state of the art, each work proposes a unique combination of device, user, network, and application-specific context elements.

First, the distinction between static and dynamic contexts [Wei et al., 2006, Raverdy et al., 2006, Brézillon, 1999] or semantic and sensing contexts [He et al., 2012] is mostly related to the intrinsic characteristics of context data and its processing, rather than to the data itself and its semantics. Second, some dimensions can be derived from other dimensions, such as the "Nearby" dimension proposed in [Schilit et al., 1994] because of its redundancy discussed in [Abowd et al., 1999]. There is also a redundancy between the Tool, Item and Device dimensions. Thus, even though the current applications solve similar issues (communication, organization, software architecture, functional aspects...), the traditional semantic heterogeneities can still be found between context models, such as polysemy and heteronymy.

Hence, reusing information from the literature to build context models and provide adaptation for applications is not straightforward. In the following sections, we propose a solution to this issue by formalizing the components of context-based models through a meta-model, which allows for reusing and combining contextual information with *adaptation purposes*.

2.3 A Meta-Model for Context

In the previous state-of-the-art, we saw that applications use different set of contextual dimensions, but may require adaptation for similar concerns. We can assume that contextual

dimensions vary across application domains, while concerns vary across platforms but are domain-independent. Hence, it could be useful to consider these concerns as part of the context model.

To avoid the redefinition of specific contextual dimensions, allow the instantiation of reusable domain-specific context models, and solve the current polysemy and heteronymy issues in context modeling, we propose a meta-model for context applications running inside component-based platforms. This meta-model relies on domain-independent concerns and promotes the usage of identical reasoning mechanisms for any application domain, through common ontological concepts. The meta-model produces multi-purpose context models using both contextual dimensions and adaptation purposes (*i.e.* concerns) defined hereafter. This work has been published in [Terdjimi et al., 2016b].

Contextual Instance

A *contextual instance* i is a high-level piece of contextual information. It is a fact about a particular context parameter, chosen among a predefined set of instances. In our work, contextual instances can be either inserted at design time as semantic information by WoT application designers (*e.g.* user preferences, regional settings, device static information, etc.) or inferred at design time from raw sensor data using rule-based semantic reasoning. We group these contextual instances in thematic sets called *contextual dimensions*.

Definition 1 (Contextual Instance) $i \in \mathcal{I}$ *where* I *is a predefined set of instances.*

Contextual Dimension

A contextual dimension d represents the set of contextual instances needed for any adaptation purpose, regarding a given type of observation (temperature, location, etc.).

Definition 2 (Contextual dimension) $d = \{i_d\}, d \in \mathcal{D}$

where D *is the set of available observations that are relevant for the application.*

Adaptation Purpose

WoT application execution rely on different types of adaptation we call *adaptation purposes*. Adaptation purposes are designed to be domain-independent, to perform all types of adap-

tations required by the platform components. An adaptation purpose ap represents the set of contextual instances related to a certain type of adaptation, for a given concern. The set $\mathcal{AP}$ of adaptation purposes covers the different concerns to adapt, among which the application domain, the users' preferences, the platform architecture, etc. In the ASAWoO project, the different managers that compose the avatar architecture require five adaptation purposes, discussed in Section 2.4 hereafter.

Definition 3 (Adaptation purpose) $ap = \{i_{ap}\}, ap \in \mathcal{AP}$

where $\mathcal{AP}$ is the set of adaptation purposes required by the platform components.

Contextual Situation

Contextual situations consist in sets of contextual instances that may belong to different adaptation purposes, *i.e.* subsets of an instantiated context model.

Let i be a contextual instance, $ap \in \mathcal{AP}$ be an adaptation purpose, and d be a contextual dimension. A contextual situation ς is a subset of an instantiated context model that characterizes a salient situation identified by domain experts.

Definition 4 (Contextual situation) $\varsigma = \{i_{j,k}\}$

where $j \in \mathcal{AP} \cup \emptyset$ and $k \in d \cup \emptyset$

Multi-purpose Context Model

Contextual dimensions and adaptation purposes are the main components of the context meta-model, which are used to build multi-purpose context models. At adaptation solution design time, WoT application designers discuss with domain experts to determine the appropriate contextual dimensions and adaptation purposes composing the context model, as well as the contextual instances that will populate the model at runtime. A multi-purpose context model (or simply "context model") $\mathcal{M}$ is a two-dimensional set of contextual instances corresponding to both adaptation purposes $\mathcal{AP}_\mathcal{M}$ and contextual dimensions $\mathcal{D}_\mathcal{M}$. Within a given model $\mathcal{M}$, dimensions and adaptation purposes are respectively disjoint in $\mathcal{D}_\mathcal{M}$ and $\mathcal{AP}_\mathcal{M}$.

Definition 5 (Multi-Purpose Context model) $\mathcal{M} = \mathcal{AP}_\mathcal{M} \times \mathcal{D}_\mathcal{M} = \{i_{ap_\mathcal{M}, d_\mathcal{M}}\}$

where $ap_\mathcal{M} \in \mathcal{AP}_\mathcal{M}$ and $d_\mathcal{M} \in \mathcal{D}_\mathcal{M}$.

The context meta-model presents the following advantages. First, it allows high flexibility for WoT application designers when building their adaptive solution, as there is no restriction in the number of dimensions to compose the context model. Second, all the models will follow the same description language (DL) thus allowing any standard-based semantic reasoner to be applicable for the adaptation. Third, the purpose/dimension view can be reused differently according to a WoT application settings, or even between several WoT applications.

2.4 Building a Multi-Purpose Context Model in the Web of Things

WoT applications are composed of various connected devices that communicate between each other to achieve a common goal through different (sub-)tasks. They require an appropriate contextual adaptation process that must be executed at runtime. This process consists in answering adaptation questions based to concerns that are common to all domains, such as communication protocols, computing resources, etc. In ASAWoO, the adaptation engine relies on semantic reasoning to answer such questions. We symbolize ASAWoO specific concerns by five adaptation purposes listed hereafter. These purposes are illustrated in the scenario described in the next section (Section 2.5).

1. (Imp) **Which appliance local capability should be involved in a given high-level functionality?** Physical devices provide several low-level capabilities that can be used by its avatar to compose high-level functionalities [Mrissa et al., 2015]. In case several possible compositions can fulfill the same functionality, the avatar must choose the best way to achieve this functionality.

2. ($Comp$) **Which functionalities should be involved in a given high-level functionality composition?** When several avatars manage to reach an agreement and propose a to compose a functionality, issues similar to the previous concern arise. If a given avatar proposes several times the same functionality, it must choose which one is the best candidate to achieve a composite functionality. A composite functionality can either be a local functionality (composed by a single object), or a collaborative functionality (*i.e.* composed by different objects).

3. (Exp) **Which functionality should be exposed to clients (in applications) and other avatars (for collaboration)?** Deducing that several capabilities or functionalities can technically be composed to achieve a higher-level functionality does not mean that this functionality should actually be proposed to the user or to other avatars. It may not be physically suitabke for several reasons (cyber and physical security, privacy, etc.). This is an adaptation decision that avatars have to take and update at every context change.

4. ($Prtcl$) **Which protocols should the application use to communicate with connected devices?** The adaptation of communication protocols depends on the task to perform. They must be adapted for wirelessly connected and mobile objects. In disruptive environments, adequate protocols must be chosen for avatars to fulfill a given functionality [Médini et al., 2016].

5. ($Code$) **Where should the application code be executed?** Application code modules may be executed on the device for resourceful devices (*i.e.* that have processing capabilities) or on a cloud infrastructure (*e.g.* for intensive calculations). Before executing them, avatars must choose to load code modules on the appropriate locations.

The different types of contextual information needed by the application impacts the choice of contextual dimensions and instances; however, each information type cannot be formally linked with adaptation purposes. Indeed, the linkage of purposes and types of information is subjective, as it depends on the application domain and may vary according to the actors of the considered application. Still, they provide directions to choose the contextual dimensions appropriately, depending on the application.

2.5 A Multi-Purpose Context Model for Smart Agriculture

In the ASAWoO platform, each physical object functionality is exposed as a RESTful service. This platform should be able to activate or deactivate functionalities at runtime, depending on the context. Along with the choice of exposition, the context influences the choice of communication protocols, the location of applicative modules and the possible functionality compositions for collaborative setups. To illustrate all these concerns, we base our work

on the following sustainable agriculture scenario.

2.5.1 Illustrative scenario

In [Médini et al., 2016], we have presented a vineyard-watering application, which is the basis of our work in the ASAWoO platform. This application allows detecting parts of the field that need to be watered, while taking environmental conditions into account. A WoT infrastructure hosts this application, which includes a cloud infrastructure, several wireless gateways as well as an irrigation system composed of geolocated watering agribots (*i.e.* robots used in smart agriculture) and drones that embed a GPS sensor and a thermal camera. Drones and agribots have the ability to move over (respectively across) the field to detect watering needs (respectively to water) given parts of the field, as illustrated in Figure 2.1

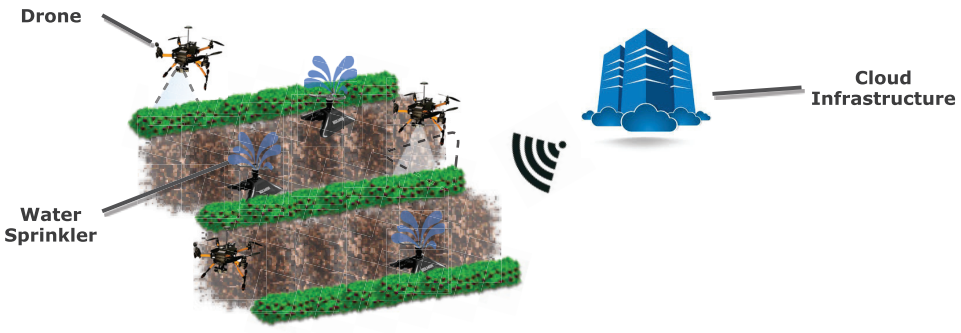


Figure 2.1: Illustration of the vineyard-watering application.

Application setup. Several sensors are placed on each part of the field. These sensors consist in an anemometer, a thermometer and a pluviometer to sense actual weather conditions. Drones possess a thermo-sensing camera, a CPU to process field images, and are equipped with a GPS sensor, a Wifi and a Bluetooth network interfaces. Drones are able to sense their hardware status, such as battery level, storage capacity, CPU usage, and storage space. In the WoT platform, complex processing tasks can be be executed either on the cloud platform, or on the device itself. Thus, if a drone cannot process a picture due to

limited memory or high CPU usage, it sends the picture to the platform to fulfill the task. The platform then communicates with the suitable agribots to take care of the parts that lack watering. Other devices consist in a central computer connected to the Internet to host the WoT platform, as well as a tablet allowing users to remotely monitor the system.

Functionality hierarchy. A WoT application is designed as a hierarchy of functionalities. In this scenario, the WateringApp functionality is on top of it. The latter is composed of Watering, an atomic functionality implemented by the Sprinkler capability, and WateringNeedsDetection, a composite functionality composed of the atomic functionalities PictureTaking, OutdoorMotion and PictureProcessing. The PictureProcessing functionality includes image processing algorithms, and is in charge of transferring pictures when needed. that pictures, and allows transferring picture . PictureProcessing allows While OutdoorMotion and PictureProcessing can be implemented only with respectively the Motor and the PictureProcessor capabilities, the PictureTaking functionality can be implemented either by a high-definition camera (HDCamera capability) or a cheaper but lower definition camera (LowResCamera capability).

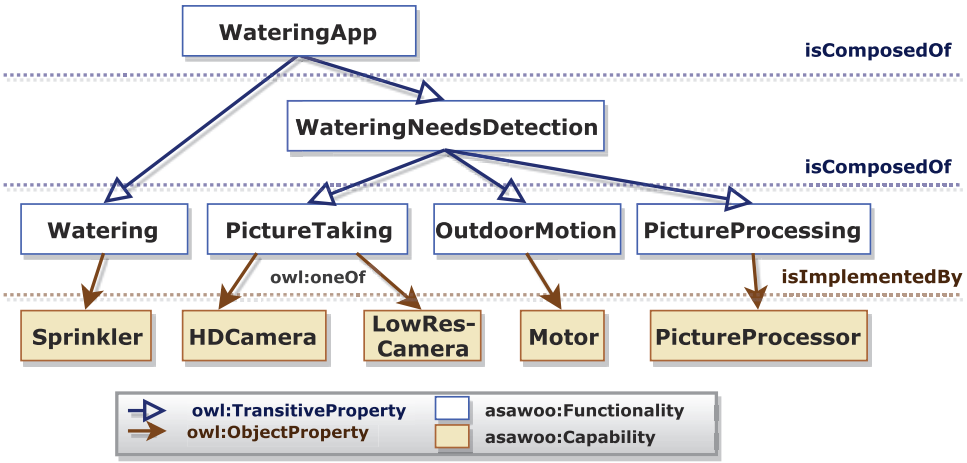


Figure 2.2: Functionalities composition and implementation for the watering application.

The hierarchy of functionalities is depicted in Figure 2.2[1]. The ASAWoO vocabulary used for functionality implementation and composition is shown in Appendix B and detailed in [Mrissa et al., 2014c, Mrissa et al., 2014b]. The implementation and composition of functionalities relies on two object properties: `asawoo:isImplementedBy`

[1]The *asawoo* prefix corresponds to the namespace `http://liris.cnrs.fr/asawoo/vocab#`

and asawoo:isComposedOf, respectively. asawoo:isImplementedBy has a owl:oneOf -restricted range on the class asawoo:Capability, *i.e.* a functionality can be implemented using exactly one capability at a time. asawoo:isComposedOf has a owl:allValuesFrom-restricted range on the class asawoo:Functionality, *i.e.* a high-level functionality is strictly composed by each low-level functionalities it is linked to. Composites functionalites are expressed using the owl:unionOf predicate on a rdf:List that contains the sub-functionalities they are composed of. An example of the WateringNeedsDetection composition is expressed in Listing B.1 below.

Listing 2.1: JSON-LD semantic representation of WateringNeedsDetection composition.

```
{
    {
        "@id": "asawoo:WateringNeedsDetection ",
        "@type": "owl:Class",
        "asawoo:isComposedOf": {
            "@id": "_:detection_comp"
        }
    },
    {
        "@id": "_:video_surveillance_comp",
        "http://www.w3.org/2002/07/owl#unionOf": {
            "@list": [
                { "@id": "asawoo:PictureTaking " },
                { "@id": "asawoo:OutdoorMotion " },
                { "@id": "asawoo:PictureProcessing " }
            ]
        }
    }
}
```

2.5.2 Answering adaptation needs through context modeling

In the vineyard-watering application, several functionalities are achievable in multiple ways. The adaptation solution should be able to choose the appropriate drones, picture resolutions, network protocols, to allow the application to adapt its behavior. The most acceptable composition can be determined using contextual information. We consider hereafter several cases in which an avatar has to answer the five adaptation questions described

previously, and present the contextual instances we use to solve these purposes. We further propose an appropriate multi-purpose context model for smart agriculture that will include these instances in the corresponding contextual dimensions. However, we will not explain how to populate the context model, as this process will be detailed in the next chapters.

(Imp) High quality pictures are preferable when implementing the functionality PictureTaking, to provide accurate watering needs detection. However, this requires a high-resolution camera and sufficient storage capacity. If a drone has a HDCamera capability and 2.5 Gb of free internal storage, the picture can be taken and stored in high definition; in this case, the HDCamera capability should implement the PictureTaking functionality instead of the LowResCamera capability. Proposed contextual instances: $\{LowQualityForPictureTaking, HighDefinitionPictureTaking,$
$HighStorageForPictureTaking, LowStorageForPictureTaking\}$

$(Comp)$ Choosing the right drone to identify if a part of field needs to be watered depends on the remaining battery power, the storage capacity, and the distance from the part of the field of each drone. Amongst the drones that have at least half-battery left, the closer should take part of the WateringNeedsDetection functionality composition, considering it has sufficient storage capacity to host the picture. Proposed contextual instances: $\{HighStorageForDetection, LowStorageForDetection,$
$FarFromFieldForDetection, CloseToFieldForDetection,$
$LowBatteryForDetection, HighBatteryForDetection\}$

(Exp) Drones may deteriorate if they are exposed to strong wind or to the rain. They should not be able to go outside if the weather is inconvenient. Hence, drones should not expose the OutdoorMotion functionality (which then disable the WateringNeedsDetection functionality) to clients in this case. Proposed contextual instances: $\{StrongWind, Breeze, NoWind, Dry, Wet, Flooded\}$

$(Prtcl)$ Choosing a network interface to transfer a picture depends on the distance between the drones and their remaining battery. On the one hand, the Wifi has a wider range but can rapidly become congested and consume lots of battery. On the other hand, high distances are problematic for low-range interfaces such as Bluetooth; in this case, the Wifi is the most suitable protocol when transferring pictures. Proposed contextual instances: $\{HighBatteryForTransfer, LowBatteryForTransfer, CloseToDroneForTransfer\}$

$(Code)$ The application module that processes pictures to determine water needs may be executed either on the drone or on the cloud. It requires high CPU availability and a minimum battery level. In addition, executing it on the cloud re-

quires high bandwidth to transfer the picture in acceptable time, and more battery. Thus, if the CPU level is sufficient to do the processing on the drone and the battery level is sufficient for both the cloud and the drone, the PictureProcessing functionality should be executed on the drone. Proposed contextual instances: $\{HighCPUAvailabilityForProcessing, LowCPUAvailabilityForProcessing, HighBatteryForProcessing, LowBatteryForProcessing, HighBandwidthForProcessing, LowBandwidthForProcessing\}$

The expression of the adaptation purposes in the vineyard-watering application above require the following contextual dimensions: Resolution, Storage, Battery, Distance, Wind, Rain, Temperature, CPU and Bandwidth. Table 2.1 details the context model considered for this scenario.

	(Imp)	$(Comp)$	(Exp)	$(Prtcl)$	$(Code)$
Resolution	LowQualityFor PictureTaking, HighDefinitionFor PictureTaking				
Storage	HighStorageFor PictureTaking, LowStorage PictureTaking	LowStorage ForDetection, HighStorage ForDetection			
Battery		LowBattery ForDetection, HighBattery ForDetection		LowBattery ForTransfer, HighBattery ForTransfer	HighBattery ForPicture, LowBattery ForPicture
Distance		FarFromField ForDetection, CloseToField ForDetection		CloseToDrone ForTransfer, FarFromDrone ForTransfer	
Wind			StrongWind, Breeze, NoWind		
Rain			Dry, Wet, Flooded		
CPU					HighCPUAvailability ForProcessing, LowCPUAvailability ForProcessing
Bandwidth					HighBandwidth ForProcessing, LowBandwidth ForProcessing

Table 2.1: The context model with each possible contextual instances for the vineyard-watering application. Adaptation purposes are displayed horizontally; contextual dimensions are displayed vertically.

This context model denotes the relation between adaptation purposes and contextual dimensions using the contextual instances; it will be used through this thesis to illustrate our next contributions. In the following chapters, we will present how to determine these contextual instances with experts and users, and how to populate the model to provide multi-purpose adaptation, at runtime.

2.6 Synthesis and discussion

In this chapter, we presented a meta-model to build multi-purpose context models. They consist in a combination of domain-specific contextual dimensions with domain-independent, platform-based adaptation purposes. Using multi-purpose context models offers the following benefits: 1) they promote the reuse of various contextual information as dimensions (including those already existing in the literature), 2) they allows combining domain-specific contextual information with adaptation purposes related to the WoT application architecture, hence helping WoT application designers and device manufacturers identify, organize and reason about context information, and 3) they encourage the reusability of reasoning mechanisms across application domains while leaving flexibility for developers to design their application-specific context models. We identified five adaptation purposes for the ASAWoO platform and illustrated our contribution through a vineyard-watering scenario.

In this work, the representation of each element composing multi-purpose context models in OWL brings many advantages. It allows processing contextual information and providing further adaptation using standard, semantic reasoning, as each element of the model is semantically-annotated. This avoids re-creating adaptation engines as any standard reasoner would be compatible with the solution, as long as it is fully OWL 2 compliant. It also improves the reusability potential of context models using shared ontologies and vocabularies. Hence, multi-purpose context models – as well as the meta-model itself – can be validated using ontology-based methodologies. The criteria suggested in [Gómez-Pérez, 1998] are relevant for them, as they include (S1) Purpose and scope, (S2) Intended uses, (S3) Intended users, (S4) Requirements, (S5) Competency Questions (CQs), and (S6) Validation of Competency Questions. The meta-model validates those criteria as presented in Tables 2.6 and 2.6. Table 2.6 shows the validation the competency questions using the metrics proposed in [Hlomani and Stacey, 2014].

Criteria	Validation
(S1)	Multi-purpose context modeling targets any domain, such as smart homes, smart farms, healthcare, etc. Its purpose is to provide context-awareness with reusable and relevant models.
(S2)	Any WoT scenario is applicable to multi-purpose context modeling, as purposes provide domain-independency for each dimension.
(S3)	This solution targets WoT application designers.
(S4)	Building multi-purpose context models require relevant contextual information coverage (contextual dimensions and instances). Needed contextual information is identified through discussion with experts, and using technical documentation of appliances.

Table 2.2: Multi-purpose Context Meta-modeling validation (S1 to S4)

Competency Question	(S5)	(S6)
(CQ1)	Does the model cover required context information?	The meta-model allows any contextual dimension to be created for complete coverage.
(CQ2)	Does the DL language allow for complete and sound results in a finite time?	The DL used (OWL) provides 3 profiles (EL, QL, RL) that are able to answer any reasoning problem (conjunctive query answering, class expression subsumption...) in a finite time.
(CQ3)	Does the DL language provide the logical constructs required by the reasoner?	OWL provides expressive relationships (object, datatype properties), concepts (classes, individuals, data values) and constructs (oneOf, allValuesFrom, unionOf, etc.) allowing the reasoner to correctly answer queries.
(CQ4)	Are the adaptation purposes sufficient to describe a WoT app context ?	Adaptation purposes provide different sets of contextual instances (including empty sets) for each dimension. Moreover, the meta-model itself do not limit the number of contextual dimensions to be used.
(CQ5)	Are the adaptation purposes redundant or overlapping?	The adaptation purposes are based on WoT application platform adaptation needs, and do not overlap by design.

Table 2.3: Multi-purpose Context Meta-modeling validation (Competency questions)

The step that follows multi-purpose context modeling is providing relevant adaptation decisions depending on contextual instances, at runtime. In the next chapter, we propose a solution to infer contextual instances from static and dynamic data sources, and to provide adaptation decisions using sets of contextual instances.

Chapter 3

Multi-Purpose Contextual Adaptation

Contents

3.1 Introduction

The WoT aims at manipulating and accessing to various things from applications in a standardized, interoperable and secure manner. The variety of technical constraints, applications and tasks imposes WoT standards (e.g. resource-oriented architectures, semantic Web, WoT specifications) and their implementations to cope with numbers of situations: an application must be able to run on different things, use different data sources and perform several tasks. Yet, contextual adaptation in such applications is a major challenge. Existing adaptation solutions are either tightly coupled with their application domains (as they rely on domain-specific context models) or offered as standalone software components that hardly fit in Web-based and semantic architectures. This leads to integration, performance and maintainability problems. Hence, there is a need for an adaptation approach able to remain independent from application domains and to comply with Web standards by design.

In Chapter 2, we introduced the notions of contextual instance, contextual dimension and adaptation purpose, and detailed how to design context models that allow for multi-purpose adaptation in WoT applications. In this work, contextual instances are high-level, semantic information that can be either static (user preferences, expert knowledge, technical documentations) or dynamic (sensors, Web services). Moreover, the right sets of contextual instances must be identified to answer the appropriate adaptation purposes, and to further take the optimal decision at runtime, simultaneously for each purpose.

In this chapter, we propose a comprehensive contextual adaptation approach, able for any WoT application to simultaneously answer domain-independent adaptation questions with data originating from various sources. To do so, it turns raw data into semantically-explicit contextual instances using *transformation rules*, and infers *ranked adaptation possibilities* using *adaptation rules*. Ii is implemented in a contextual adaptation middleware, executed at runtime on the ASAWoO platform.

3.2 State of the art on Contextual Adaptation

There are various ways to acquire contextual information, and so are the ways to deal with contextual information interpretation and processing. Based on the literature, [Perera et al., 2014] propose to combine common context lifecycle steps as context acquisition, context modeling, context reasoning, and context dissemination to other devices. The workflow proposed in [Bernardos et al., 2008] (acquisition, processing, reasoning and decision) separates context processing from reasoning. Ferscha et al. propose in [Ferscha et al., 2001] the sensing, transformation, representation, rule-base and actuation steps.

It can be seen that state-of-the-art context management lifecycles share similarities with adaptation loops such as MAPE-K [IBM, 2004]. The diversity amongst lifecycles arise from the different types of adaptation needed by the application, as well as its infrastructure. The context lifecycle we propose hereafter aims to fit Web-based decentralized architectures (in particular, avatar-based ones such as ASAWoO) to allow capitalizing on the reasoning process in terms of response times. Unlike [Perera et al., 2014], we separate context modeling from context instantiation. Context modeling is provided at design time, while context instances vary at runtime to further suggest possibilities for the application to adapt its behavior.

3.2.1 Contextual adaptation

The main notion of adaptation is related to the field of biology, where the Oxford Dictionary defines it as "*the process of change by which an organism or species becomes better suited to its environment*". In [Da et al., 2011], the authors extend this definition to the field of software engineering and identify two adaptation levels: environmental (related to the structure of the application) and user-defined (related to functional aspects of the adaptation). To provide such adaptation, state-of-the art solutions usually rely on adaptation loops based on similar steps: gather data, process information, decide the adaptation choice, and proceed the actual adaptation ("act"). Those steps however slightly differ across solutions, depending on their adaptation type and approach.

The classification of adaptation approaches described in [Barbier et al., 2015] includes

several characteristics: functional *vs.* nonfunctional, designed *vs.* unanticipated, predictable *vs.* unpredictable, third-party *vs.* self-adaptive (*i.e.* provoked by the adaptation software itself) and business-specific *vs.* generic. Another adaptation classification from [Fox and Clarke, 2009] includes the notions of anticipation (as the degree of anticipation to changes) and domain-specificity (as the level of parametrization of the adaptation solution) as well. It also introduces two additional characteristics to position adaptation solutions: tools (denotes whether the adaptation solution comes with a dedicated development environment or a runtime monitoring environment) and scope (describes the extent of the adaptation process over the application components and services). Several techniques exist in the literature to adapt a software system to contextual parameters: service configuration, service substitution and adaptation planning [Hanney et al., 1995, Barbier et al., 2015] [Fox and Clarke, 2009, 4.4]. The following adaptation frameworks illustrate these techniques and stress their advantages and drawbacks for generic adaptation approaches.

Adaptation solutions and frameworks

In the past decades, numerous self-adaptive and autonomic systems have been proposed to support dynamic contextual adaptations in pervasive and mobile computing environments. Some of these systems are based on a so-called control loop, such as in the autonomic computing architecture MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) defined by IBM [IBM, 2004] or the Rainbow framework [Garlan et al., 2009]. Some middleware platforms, such as ReMMoC [Grace et al., 2003] and CARISMA [Capra et al., 2002] implement reflexive mechanisms for dynamic adaptation purposes. The main drawback of these solutions is that they are not adapted for Internet of Things because they only present an isolated centralized autonomic manager addressing a set of specific problems.

The INCOME project [Arcangeli et al., 2012] is a multi-scale context management framework for the IoT. The authors aims to provide self-adaptation for the deployment of context management components, as well as for the dissemination of contextual information, based on the detection of situations of interest. They also address quality of context and privacy concerns.

In [Becker et al., 2007], the authors provide a service configuration adaptation approach that is generic and based on reference models. It aims to ease the design of adaptation solutions for business processes that are shared by various participants, based on a multidimensional context model and complex transformation rules. The fact that a service configuration engine has to generate numerous parameters makes such an approach strongly para-

metric and does not guarantee optimal adaptation decisions. Moreover, it is more difficult for a domain expert to completely formalize the transformation function than to express its behavior in natural language.

In the Adaptive CORBA Template (ACT) [Sadjadi and McKinley, 2004], the authors propose a service substitution approach for dynamic adaptation based on CORBA middleware interceptors. Such interceptors can be registered, unregistered and enhanced at runtime, thus producing adaptation cases that are not known in advance. This framework is domain-independent, partially anticipatory as it preconfigures so-called *adaptive CORBA templates* and its adaptation scope can address different tasks through the use of rule-based interceptors. In this sense – and even though it requires the CORBA middleware, which is neither tailored for resource-limited devices and nor compliant with Web standards - it is close to more recent approaches such as aspect-oriented adaptation [Kongdenfha et al., 2006], as well as to substitution of service-based applications [Zeginis and Plexousakis, 2010], or more generally to adaptation in dynamic, component-based middlewares.

The Mobility and ADaptation enAbling Middleware (MADAM) framework [Mikalsen et al., 2006] applies an adaptation planning approach to ease the development of adaptive solutions. At request time, it chooses the combination of available services to compose a response to a particular user's need. To do so, it provides several managers (context, configuration and adaptation managers) that allow for decoupled adaptation processes. Moreover, it relies on "context reasoners", which are generic means to locate sensor data processing in the adaptation workflow. In the perspectives, the authors foresee to reuse this approach using Web services and semantic Web technologies.

WComp [Tigli et al., 2009] is an aspect-oriented, Web service-based middleware that relies on the Aspect of Assembly (AA) aproach to provide compositional adaptation of event-based services, depending on context changes. The adaptation decision is specific to the aspect-oriented paradigm; it requires knowledge of advanced concepts such as joint points, pointcuts, advices, etc. and is hardly reusable for integration in standard, rule-based inference engines. The FraSCAti platform [Seinturier et al., 2009] aims to extend the Service Component Architecture (SCA) [Beisiegel et al., 2007] by providing reflexive behavior for service oriented architecture components. To do so, each component is associated to a generic container (at a *meta* level of the architecture) that includes several services such as component identity, lifecycle, hierarchy, wiring, etc.

More recent works, such as [Stehr et al., 2011], give some guidelines on how networked cyber-physical systems could be used with fractionated software to build reliable systems. However, such work does not include a Web-based perspective. Systems dedicated to the

IoT that can perform multi-level adaptation in environments composed of cloud infrastructure and physical objects have been proposed in [Athreya et al., 2013] and [Alaya et al., 2012]. However, in contrast to the avatars [Jamont et al., 2014], these solutions do not define an abstraction layer able to actually extend physical objects on the Web, and neither support collaboration between these objects.

In [Gyrard et al., 2014], the authors present M3, an architecture that enriches sensor data to enable its use in cross-domain applications. This work uses the Semantic Sensor Network (SSN)[1] ontology that provides a vocabulary to describe sensors and sensor data, to enable their use in semantics-aware applications. However, this work has not yet explored the adaptation problem in terms of cross-domain adaptation purposes.

Semantics-based adaptation approaches

The definition of context by Anind Dey [Dey, 2001] "*Context is any information that can be used to characterize the situation of an entity[...]*" emerges the notion of information annotation and therefore of generating graphs of interrelated pieces of data. The promise of the semantic Web is to provide easier and standard creation, handling, querying and transformation of such graphs [Berners-Lee et al., 2001]. Hence, it appears quite relevant to use semantic web languages (RDF, RDF-S, OWL)[2] to model contextual information, as well as to reason about contextual data to make adaptation decisions, as context is metadata by essence. While there are numerous domain-specific studies that produce semantic models (and sometimes sets of reasoning rules) in application fields among which video streaming [Bertini et al., 2006, Zufferey and Kosch, 2006] or e-learning [Baldoni et al., 2004, Sampson et al., 2004], few generic adaptation systems take advantage of semantic web standards and tools to make adaptation decisions.

[Laborie et al., 2011] present a semantic adaptation framework for multimedia documents in which they describe documents as sets of parts related by constraints. Although they only apply adaptation to this application domain and use an on-purpose graph matching algorithm rather than classical reasoning tools, their formalization makes their approach quite reusable. The work presented in [Baladron et al., 2012] proposes a generic approach based on the semantization of sensor data and a *context intelligence module*, that partly relies on a semantic reasoner to provide adaptation propositions. However,it seems that [Wang et al., 2004] describes the first attempt to use both rule-based reasoning for con-

[1] `http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/`
[2] `https://www.w3.org/2001/sw/interest/`

textual adaptation, as well as the expressivity of high-level description logics constructs available in OWL relations.

Hence, the potential offered by Web-based reasoning and semantic technologies could provide adaptation in WoT applications in a generic, flexible and reusable manner, for multiple and high-level purposes. We intend to reuse Web standards and provide compliance with the WoT Interest Group specifications (Thing Description[3], Servient[4], Device API[5], protocol binding[6]...), to allow for adaptive WoT applications design.

Semantics-based adaptive WoT solutions

To reason about contextual data, semantic adaptation tools must first gather semantized data. In the WoT application field, not all platforms provide semantic data to their components. Actually, only recent advances in the WoT aim to bring together the Semantic Web with Web standards, on top of the Internet of Things. UBIWARE [Katasonov et al., 2008] uses semantic annotations to describe agents and behavioral tasks, to provide interoperability and reusability of these definitions. SPITFIRE [Pfisterer et al., 2011] unites RESTful approaches using CoAP [Shelby et al., 2014] with OWL/RDFS and SPARQL[7] for constraint devices. Sense2Web [Barnaghi and Presser, 2010] allows publishing sensor data and measurements on the Web through a SPARQL endpoint, but still has to be coupled with a functional solution to provide a complete WoT application capable of managing any type of thing (sensor or actuator). The M3 framework provides a more comprehensive and domain-independent approach through a dedicated vocabulary to describe sensors, together with the tools to reason about these descriptions and deduce application templates [Gyrard et al., 2015a]. While these approaches facilitate things interoperability and WoT application development, they do not tackle the adaptation concern.

ASAWoO provides a WoT platform that includes a component-based architecture to allow each element (aka "managers") to exchange semantic data. Each of them deals with a specific purpose, and these purposes may require adaptation. In the next section, we propose a context lifecycle that achieves *multi-purpose adaptation* from semantized contextual information, using an OWL 2 reasoner for adaptation planning.

[3] https://w3c.github.io/wot-thing-description/

[4] https://w3c.github.io/wot-architecture/#sec-servient-architecture

[5] https://www.w3.org/2009/dap/

[6] https://w3c.github.io/wot/current-practices/protocol-binding-templates/html/protocolbindings.html

[7] https://www.w3.org/TR/rdf-sparql-query/

3.3 Multi-purpose adaptation in WoT applications

Multi-purpose contextual adaptation in WoT applications originates from several processes in which various actors take part. WoT platform designers create complex execution environments that need to handle several concerns to support a variety of use cases and applications; they document these concerns and the corresponding adaptation purposes. Appliance manufacturers describe device characteristics (QoS) in their documentations. Domain experts identify application concepts and processes, along with all the environmental data able to provide useful contextual data. Users specify their preferences (*e.g.* preferred devices, privacy levels, etc.). WoT application designers then need to interpret pieces of contextual information and to integrate them in a comprehensive adaptation process. Their work consists in designing a context model and two sets of rules: *transformation* and *adaptation*. Then, at runtime, the platform uses the first set to wire the context model to the available data sources and executes the second to run the adaptation process.

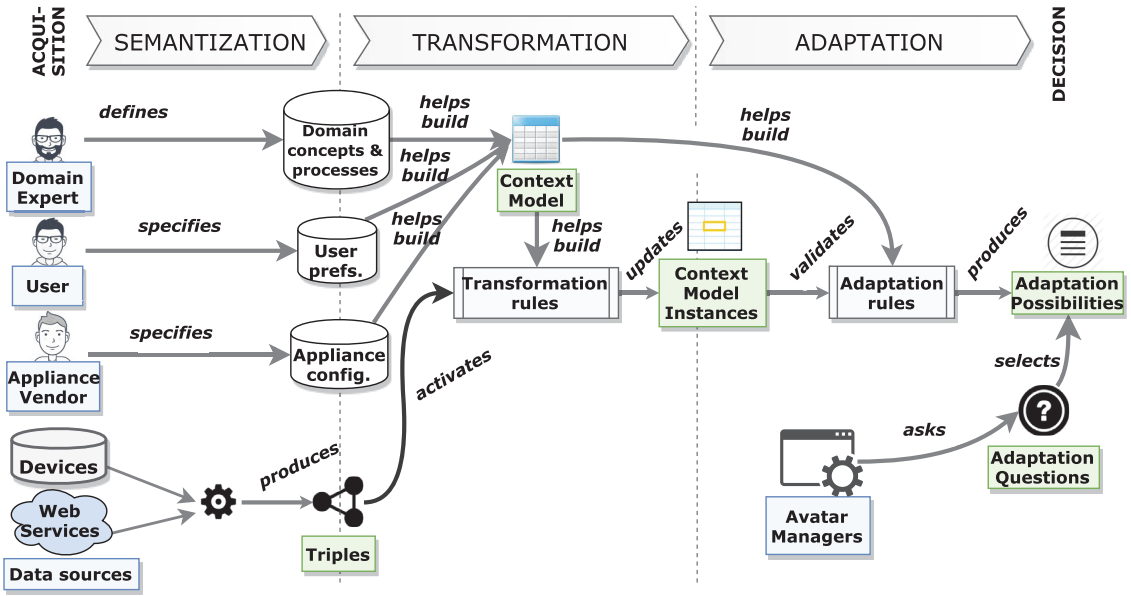


Figure 3.1: The adaptation process and its actors.

The proposed adaptation process works as follows. First, at configuration time, static

data (e.g. application context model, appliance configuration, user preferences) are stored in semantic repositories [Mrissa et al., 2015]. Then, at runtime, an avatar receives raw data from various sources, including devices and Web services. These data are semantically annotated and transformed into instances of the context model, using transformation rules. Adaptation rules are then applied to the instantiated context model to infer each possible adaptation choice. When an adaptation decision is required, an adaptation request is sent to the context manager, which retrieves the best candidate. The querying process is the same, regardless of whether the request relies on filtering (e.g. can we expose a given functionality) or on ranking (e.g. which communication protocol is the most suitable). The global process is depicted in Figure 3.1 and is detailed in the following subsections through the vineyard-watering application. This work has been published in [Terdjimi et al., 2017].

3.3.1 Semantization step

WoT application aims to manipulate semantically-annotated data that are independent from both the application domain and the adaptation solution. This include different types of data coming from various sources, such as domain expert knowledge, user profiles, application description and requirements, Web services, sensors, etc. Unlike the other data sources (for which the semantization is realized at design time), the information coming from sensors requires semantization at runtime, through appropriate semantic annotation algorithms (*e.g.* by converting numeric data into triples, depending on the type of the data). Yet, these algorithms are not part of this thesis contribution, hence will not be detailed here.

Some device manufacturers are provide semantically-annotated based on linked open vocabularies[8]. They include information from data sharing vocabularies, such as units of measure, types of data, quantities, which are both domain-independent and adaptation-independent. This is the case for the new version of SSN[9], which includes information from QUDT[10]. In the transformation step, we bridge these independent information with our adaptation solution using transformation rules.

[8] LOV: `http://lov.okfn.org/dataset/lov/` and LOV4IoT: `http://sensormeasurement.appspot.com/?p=ontologies`

[9] `http://w3c.github.io/sdw/ssn/`

[10] `http://www.qudt.org/`

3.3.2 Transformation step

The transformation step relies on a set of transformation rules that pre-processes semantized contextual data and transforms them into discrete contextual instances, in order to enable the adaptation step. To design such rules, WoT application designers confront the expert's assessments to the appliance technical constraints. They identify the type of data sent using either service descriptors or device specifications from this source, and determine the range values of contextual instances for each dimension using their thresholds. Thresholds are determined using appliance documentation and domain expert knowledge, and are expressed using comparison operators. In transformation rules, thresholds associate data sources to their value using two triple patterns, where the first pattern describes the carried value and the second pattern describes the nature of the data source. Each rule produce a contextual instance suitable to a given functionality to be adapted (which we detail in the adaptation step).

Transformation rules

Transformation rules are triggered at runtime, when the condition on semantically annotated data is met (*i.e.* when a given value threshold is reached). Some data cannot be directly compared to a threshold and require a pre-calculation: this is the case for the calculation of distances between two entities (GPS coordinates of entities must be subtracted to obtain the distance to be compared to a threshold).

A transformation rule t is a conjunctive rule, where its antecedents are 1) a set of numerical values $\{\psi_\Psi\}$ sent by different data sources $\{\Psi\}$, which can be either sensors or a Web services, 2) a calculation function $cf()$ that computes all numerical values to produce an interpretable value for the threshold, and 3) a contextualization threshold τ, *i.e.* a numerical condition that allow for contextual instantiation using the interpretable value. The consequent of the rule is a contextual instance i deduced from the comparison between the value sent by the source and the threshold set by designers.

Definition 6 (Contextualization Threshold) $\tau = (\square_\tau, \psi_\tau)$

where $\square_\tau \in \{>, <, >=, <=, =\}$ *is a comparison operator, and* $\psi_\tau \in \mathbb{R}$ *the threshold value to be compared with.*

Definition 7 (Transformation Rule) $t = cf(\{\psi_\Psi\})\ \square_\tau\ \psi_\tau \rightarrow i$

Vineyard-watering contextualization thresholds

Contextual Dimension	Data Source(s)	Value Type	Contextual Instances Thresholds
Resolution	Drone Camera	Integer (number of horizontal lines)	HighDefinitionForPictureTaking: $x \geq 720p$ LowQualityForPictureTaking: $x < 720p$
Storage	Drone SD Card	Decimal (remaining capacity in bytes)	HighStorageForPictureTaking: $x \geq 500Mb$ LowStorageForPictureTaking: $x < 500Mb$ HighStorageForDetection: $x \geq 600Mb$ LowStorageForDetection: $x < 600Mb$
Battery	Drone Battery	Decimal (current battery voltage in volts)	HighBatteryForDetection: $x \geq 50\%$ LowBatteryForDetection: $x < 50\%$ HighBatteryForTransfer: $x \geq 40\%$ LowBatteryForTransfer: $x < 40\%$ HighBatteryForPictureTaking: $x \geq 40\%$ LowBatteryForPictureTaking: $x < 40\%$
Distance	Drone GPS	Integer (GPS coordinates of both the device and the field)	CloseToFieldForDetection: $x \geq 100m$ FarFromFieldForDetection: $x < 100m$ CloseToDrone_n_ForTransfer: $x < 1m$
Wind	Anemometer	Integer (wind speed in km/h)	StrongWind: $x \geq 10km/h$ Breeze: $x \geq 1km/h$ and $x < 10km/h$ NoWind: $x < 1km/h$
Rain	Web service Pluviometer	Integer (precipitation rate in mm/h)	Dry: $x = 0mm/h$ Wet: $x > 0mm/h$ and $x < 10mm/h$ Flooded: $x \geq 10mm/h$
CPU	Device CPU	Decimal (% of CPU load)	HighCPUAvailability-ForProcessing: $x \geq 50\%$ LowCPUAvailability-ForProcessing: $x < 50\%$
Bandwidth	WoT Infrastructure	Integer (ping in milliseconds)	HighBandwidth-ForProcessing: $x < 100ms$ LowBandwidth-ForProcessing: $x \geq 100ms$

Table 3.1: The relation between contextual dimensions, data sources, raw value types and contextual instance thresholds.

The contextual thresholds of the vineyard-watering application are detailed in Figure 3.3.2. To provide instances, the Distance and the Battery require calculation function to obtain an interpretable value for the threshold (GPS coordinates of drones to substract, battery voltage to compare to a fully charged battery).

We justify the contextual instance thesholds as follows. In the Resolution dimension, only high definition pictures (720p) are suitable for watering needs detection. Hence, the threshold is fixed to this value. The contextual instances from the Storage dimension have different initial requirements: storing a picture requires less storage capacity than both storing and processing it. For the Battery dimension, detection has also higher power requirements than any other process. The Distance dimension also contains instances with different thresholds: to travel to a part of the field, 100 meters is a reasonable limit; to avoid drone collisions, 1 meter is the minimum. The Wind dimension fixes a threshold of 10 km/h to avoid drone deviation, based on technical documentation. The Rain dimension rely on thresholds based on meteorology domain expertise. The CPU dimension provides a general threshold of 50%, based on application benchmarks. The Bandwidth threshold is fixed to a limit of 100ms, considered as the maximum response time to take a decision. This value is confirmed on the performance evaluation (Section 3.5.2).

3.3.3 Adaptation and Decision steps

The adaptation step consists in inferring several *ranked adaptation possibilities* with respect to a given *contextual situation* through *adaptation rules*, while the decision step consists in querying the semantic reasoner to obtain the adaptation possibilities currently maintained. Both steps require defining the following necessary components: the contextual situations, the adaptation possibilities, and the adaptation rules.

Adaptation possibility

Adaptation possibilities are inferred using *adaptation rules* at the insertion of contextual instances, and removed at their deletion. An adaptation possibility p represents an adaptation candidate c to a functionality f, with respect to an adaptation purpose ap. The candidate varies according to the adaptation purpose we consider. In our setup, it could be

either a capacity (Imp), a functionality ($Comp, Exp$), a protocol ($Prtcl$) or a code location ($Code$). The set of possibilities for the same adaptation purpose is denoted P.

Definition 8 (Adaptation possibility) $p_{f,ap} = (c, ap, f)$

where f is a functionality, c is an adaptation candidate and $p_{f,ap} \in P_{f,ap}$.

For instance, in our scenario, we consider the adaptation possibility composed of the candidate $Cloud$ for the adaptation purpose $Code$, to adapt the location of the PictureProcessing functionality.

Adaptation rule

Adaptation rules allows inferring adaptation possibilities in response to a contextual situation, for a given adaptation purposes. They have similar patterns, regardless of their purposes. The antecedent of an adaptation rule α is a conjunction of contextual instances, *i.e.* a conjunctive contextual situation. The body of the rule is a set of adaptation possibilities P. In the adaptation solution, each purpose $ap \in \mathcal{AP}$ is associated to a set $\mathcal{R}_{ap}$ of adaptation rules.

Definition 9 (Adaptation rule) $\alpha_{\varsigma,ap} = \bigwedge_{k \leq |\varsigma|} i_k \ \rightarrow \ P_{\varsigma,ap}$

where $i_k \in \varsigma$ is a contextual instance and $P_{\varsigma,ap}$ a set of inferred adaptation possibilities for a given adaptation purpose ap, with respect to ς[11].

Scoring of adaptation possibilities

To provide optimal adaptation decisions, we score each adaptation possibility, depending on contextual situations: the more a candidate is favorable for a given adaptation request, the higher its score value is.

The score of a possibility is determined as follows. At design time, each possible contextual situation is presented to the domain expert. The expert then suggests an appropriate response to this situation, to allow the WoT application designer to determine which capabilities/functionalities are the most/least appropriate to implement/compose a functionality, which functionalities should not be exposed to clients, which protocols shoud be used, or where the functionality code should be located (as in the ASAWoO platform). The

[11] In our solution, we provide a rule for each $p \in P_{\varsigma,ap}$.

"most/least" degree is thereafter interpreted as the score for this adaptation possibility with respect to the observed situation.

In ASAWoO, while the adaptation possibilities are either to expose a functionality or not for the Exp adaptation purpose, the possibilities for other purposes may be multiple. Hence, we merged these two types of solutions using a *scoring* approach: adaptation possibilities for all purposes are ranked in a common, normalized manner. Making binary decisions then consists in selecting an adaptation possibility if its score equals 1. Finding the best candidate for another purpose consists in selecting the possibility with the highest score.[12]

Score functions. A score function attributes the impact of a contextual instances with respect to several contextual situations. Hence, each instance of a context model is associated to a score function sf that depends on its situation and allows to weight several adaptation possibilities[13].

Definition 10 (Score function) $sf : i_{ap,d,f}, \varsigma \rightarrow \{s_{i_{ap,d,f},p_n}\}, \forall n \in \mathbb{N}, p_n \in P_{f,ap}$

with $s_{p_n} \in [0; 1]$

Adaptation score calculation. The scores provided to adaptation possibilities using score functions are specific to a contextual instance, itself part of a contextual situation. Hence, to calculate the *total score* of an adaptation possibility regarding a contextual situation, the scores of this possibility must be added regarding each contextual instance confronted with a given situation. The total score of this possibility is its *adaptation score*.

An adaptation score s is a numeric value that allows to weight an adaptation possibility for an adaptation purpose. Scores are normalized by their coefficients ϵ, are situation-dependent and obtained using score functions.

Definition 11 (Adaptation score) $s_{\varsigma,f,ap,p} = \frac{\sum_{k=1}^{|\mathcal{D}|}(s_{i_{ap,d_k,f,p_n}}) \times \omega_k}{\sum_{k=1}^{|\mathcal{D}|} \omega_k}$

where $0 \leq s_{i_{ap,d_k,f,p_n}} \leq 1$.

In ASAWoO, to determine the score of a composition candidate for the $Comp$ purpose, we calculate the average of each functionality scores that are part of the composition. Hence, if a functionality can be composed in several ways, the composition with the highest score would be chosen.

[12] In the case two (or more) possibilities have equal, highest scores, both are considered "valid" in our case.
[13] An example of scoring function is provided in Chapter 4, Section 4.3.2

Adaptation request

An adaptation request is sent at decision time, to obtain the optimal adaptation choice for a given purpose. An adaptation request q is formulated by specifying the functionality f to be adapted and the related adaptation purpose $ap \in \mathcal{AP}$. The answer to q is the *optimal* adaptation possibility $\hat{p}$, *i.e.* the possibility with the highest score for this adaptation request.

Definition 12 (Adaptation request) $q = f \wedge ap \rightarrow \hat{p}$

where $\hat{p} \in P_{ap,f} \cup \varnothing$, and

- $s_p = max(s_{p_n}), \forall p_n \in P_{ap,f}$
  for purposes with multiple possibilities (e.g. Imp, $Comp$, $Prtcl$, $Code$)
- $\hat{p} = p \in P_{ap,f}$ if $s_p = 1$, $p = \varnothing$ otherwise
  for purposes resulting in binary decisions (e.g. Exp).

3.4 Avatar-based Contextual Adaptation Workflow

In ASAWoO, the avatar architecture includes various managers to handle different concerns such as communicating with objects, composing functionalities from object capabilities, collaborating with other avatars or managing contextual adaptation (Chapter 1). During the avatar lifecycle, these managers interact to identify/expose local and collaborative functionalities, and respond to functionality requests.

To perform these tasks, each manager has a specific responsibility in the WoT application adaptation workflow. Figure 3.2 shows, once the different WoT infrastructure components have been populated and the avatar managers instantiated, the details of the adaptation workflow. Steps 1 to 5 of this workflow relate to the data integration task described above, steps 6 and 7 to the query answering one. This workflow has been published in [Terdjimi et al., 2016c].

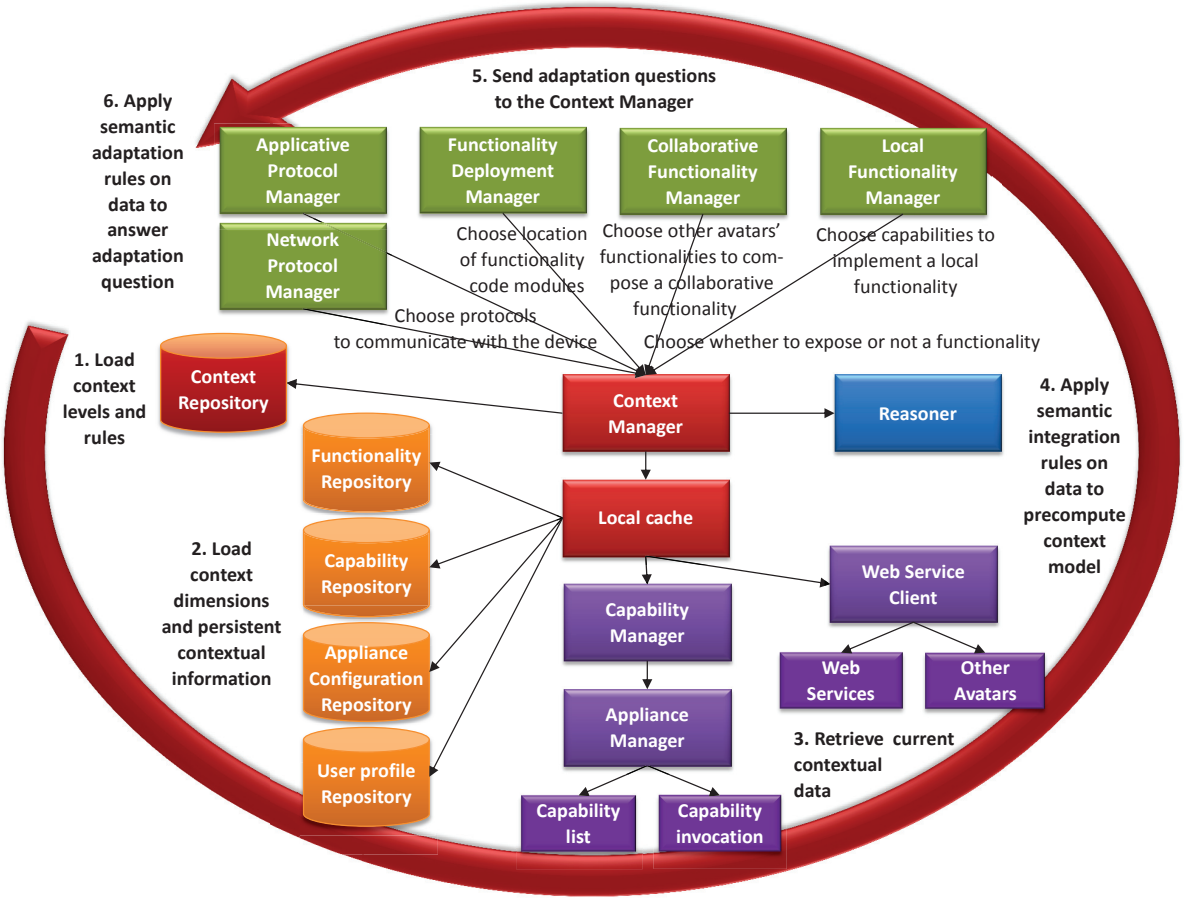


Figure 3.2: Avatar-based contextual adaptation workflow.

Step 1: Initialization. At initialization time, the context manager loads the context model. This model comprises two sets of rules, applied respectively at the transformation step and at the adaptation step. Transformation rules allow populating a stable context model with contextual instances. Adaptation rules allow generating ranked adaptation possibilities.

Step 2: Static data loading. The context manager loads static data from the different semantic repositories that store information related to the application domain, device, user and environment. Such information is disseminated in four repositories. Each repository is structured according to the type of contextual information for WoT applications identified in Chapter 2, Section 2.4.

- Functionality repository. It stores application-related information. It is populated when installing a WoT application into the WoT infrastructure and contains details about the functionalities that compose this application: semantic descriptions, as well as QoS and application domain data. These two latter are specifically intended for adaptation purposes. In particular, functionality QoS is mandatory to respond to almost all adaptation questions.

- Capability and appliance configuration repositories. These repositories store device-related information. They are populated when installing and configuring a new device. The capability repository contains semantic capability descriptions and QoS information. How to actually call these capabilities is described in the appliance configuration repository: it contains information about how to access to the device (address, protocols) and parameter format. These repositories are queried by the context manager to perform low-level adaptation tasks, such as composing a local functionality or identifying a suitable protocol to communicate with the device.

- User profile repository. It stores users' preferences. For each user, they are instantiated with a default profile that can be updated by the user.

Steps 3 and 4: Context update and semantic integration. The context manager updates the avatar context with contextual data instances. To do so, it sends queries or receives change notifications from two kinds of dynamic data sources. In order to get data from the device, the context manager queries the avatar capability manager, which in turn queries the appliance manager. It can then gather device-related contextual data by retrieving its currently available capabilities (sensors, actuators, processing, communicating capabilities). When external sources (other avatars, Web services) are required, the context manager queries them using the Web service client. This allows retrieving contextual information related to the user, application or environment.

All these data comprise numerical values from sensors or service responses. They must be transformed into contextual instances in order to fit in the context model. To do so, the context manager queries a reasoner that applies transformation rules to populate the model with contextual instances. A particular contextual situation would then trigger one or more adaptation rules, to produce sets of adaptation possibilities that will be maintained in the reasoner.

Steps 5 and 6: Purpose-based adaptation request answering. At request time, one of the five possible source managers sends an adaptation request to the context manager.

This can be done to identify the functionalities that the avatar will expose (what to do), or to actually execute a functionality invoked by a user or another avatar (how to do it).

When the context manager receives a adaptation request, it queries the reasoner to obtain the optimal adaptation possibility for a functionality to be adapted, regarding one of the five adaptation purposes. The adaptation purpose related to the adaptation request depends on the manager that sent the adaptation request. The context manager then sends back the optimal adaptation possibility to the manager, which will take the appropriate action regarding the answer.

3.5 Evaluation

We conducted two evaluations using the WoT runtime environment of the ASAWoO platform, according to criteria based on the work of Bass et al [Bass, 2007]. First, we evaluated its accuracy, i.e. its ability to do the work for which it was intended. Second, we evaluated its performance for both the data integration and query answering tasks. All experiments were performed using the HyLAR semantic reasoner [Terdjimi et al., 2016a]. Both evaluations take place in the vineyard-watering scenario, and their parametrization is detailed below.

3.5.1 Accuracy

We evaluated the accuracy of the adaptation solution by simulating the scenario from Section 2.5. We mocked three drones to the WoT platform, with the objective of realizing the WateringApp functionality. The environmental setup includes a vineyard field seperated in three parts, namely FieldPart 1, 2 and 3. The evaluation focused on the tasks that require drones and did not consider other appliances such as the automatic irrigation system, for the sake of clarity. The experimental setup was the ASAWoO platform, ran in an Ubuntu 16.04 VM with 2 VCPUs and 4Gb of RAM, in an OpenStack cloud infrastructure.

We varied the contextual parameters identified in Section 3.3.2 in a 2-days time interval. In this interval, the managers ask the five adaptation questions to each drone avatars, at

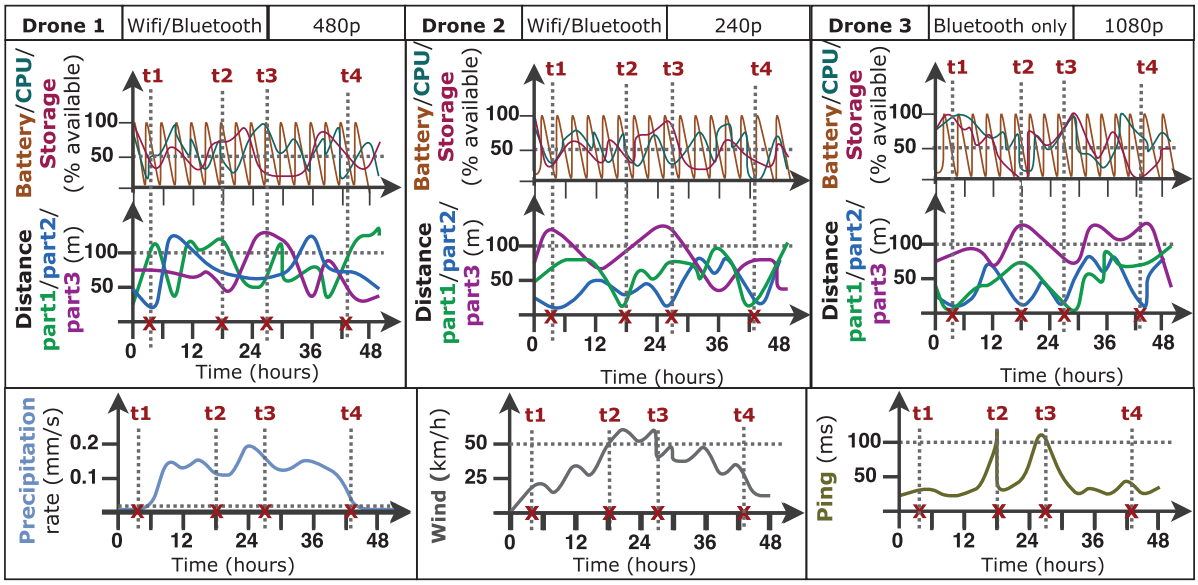


Figure 3.3: Variation of contextual data during a simulated 2-days time interval.

different times $t1$, $t2$, $t3$ and $t4$, as depicted in Figure 3.3. The answers to these questions are expected to correspond to the adaptation rules described below. Table 3.2 shows the answers returned to the adaptation questions by the five managers.

(Imp) The system must determine the best drone candidate to implement the PictureTaking functionality. We see at $t2$ that drone 1 capabilities are preferred as drone 3 storage capacity is too low. On the whole, we also see that drone 2 is not a good choice for detecting parts of field to water due to its insufficient camera resolution.

($Comp$) The system must determine the best drone candidate to compose the WateringNeedsDetection functionality for FieldPart1. In particular, we see that drone 3 is the optimal choice to take pictures at both $t1$ and $t3$, due to its proximity to FieldPart1. However, drone 1 is preferred at $t2$ due to its high battery level.

(Exp) The system is expected to determine the exposability of OutdoorMotion, for any drone. The results show that at both $t2$ and $t3$, these two functionalities are not exposed as either the wind is too strong or it rains.

($Prtcl$) The system must determine the best candidate protocol (either Wifi or Bluetooth) for the PictureProcessing functionality, between drone 1 and drone 2. At $t2$, we see that the optimal protocol to transfer a picture between drone 1 and drone 2 is Wifi as both drones are in different parts of the field (*i.e.* they are considered far from each other).

($Code$) The system must determine the best code location candidate (either the drone itself

– the device – or the cloud) for executing the PictureProcessing functionality. This part of the evaluation focus on drone 3. In particular, we see that executing PictureProcessing in devices is preferable at $t2$ and $t3$, due to ping timeouts or long delays caused by the weather conditions (even though drone 2 CPU availability is acceptable, which is the case at $t2$).

Time	(Imp) Implementation of PictureTaking	($Comp$) Composition of WateringNeedsD. (FieldPart1)	(Exp) Exposability of PictureTaking & WateringNeedsD.	($Prtcl$) Protocol to transfer a picture (Drones 1 & 2)	($Code$) Location of PictureProc. (Drone 2)
t1	Drone 3	Drone 3	Exposable	Bluetooth	Cloud
t2	Drone 1	Drone 2	Not exposable	Wifi	Device
t3	Drone 3	Drone 3	Not exposable	Bluetooth	Device
t4	Drone 3	Drone 1	Exposable	Bluetooth	Cloud

Table 3.2: Answers to five adaptation questions in four different times.

Summary. At all times, the results of this evaluation verify that all avatar context managers provide the expected answers to all adaptation questions. The correctness of the adaptation system is enforced by the use of a standard semantic reasoner, which ensures that any other rule-based solution would have given the same answers. In addition, this evaluation shows that the adaptation solution allows performing accurate multi-purpose adaptation from a common semantic contextual model.

3.5.2 Performance

In this evaluation, we evaluate the performance of the adaptation solution in terms of processing times. These experiments were performed on a Dell OptiPlex 780 - Core 2 Duo E8400 @ 3 GHz. As the integration (semantization, transformation and application of adaptation rules) and adaptation request answering processes can run in parallel, we evaluated them in separate runs, and with different goals (i.e. maximum processing times). Contextual data integration runs as a background task, but must not monopolize all computing capabilities allocated to the avatar, especially if this avatar runs on a constraint device. Hence, it must be lightweight but does not need to be immediate. We then chose a threshold of 1 second as success for this experiment. Adaptation request answering, however, is time-critical, as it is required for the current functioning of the avatar and of the WoT application.

For this reason, we limited its acceptable response time to 100ms. For both processes, we ran two experiments varying the number of rules and of triples in the knowledge base.

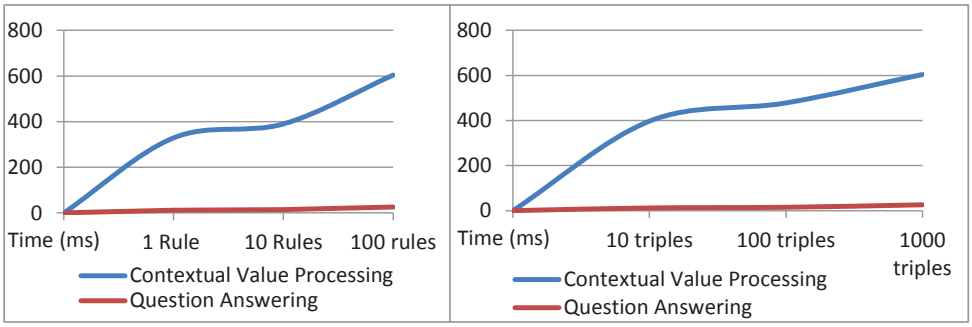


Figure 3.4: Context processing and adaptation request answering times on different situations.

The results of these experiments are depicted in Figure 3.4. They show that the adaptation solution reaches by far the two initial goals as the respective processing times for integration and answering do not exceed 650ms and 30ms.

3.6 Discussion

The multi-purpose adaptation solution we propose allows for semi-anticipated adaptation planning. At design time, the application designer and experts pave the way for generating adaptation possibilities and situation-based adaptation rules. These possibilities are inferred at runtime and the rules are triggered by context change events. In this sense, this approach can be considered anticipated, as it pre-processes some (most of the) necessary element on which the adaptation planning is based. However, plans are not yet available at context change. Actual adaptation planning corresponds to the selection of an adaptation possibility (if any), which depends on the scores of these possibilities that are computed at request time (*i.e.* unanticipatedly).

This strategy allows reconciling the cost and benefits of relying on an inference engine. Indeed, while inferences are processed at runtime, they are not triggered when an adaptation request arises, but can be processed in parallel. The inference results are only integrated in the context graph when they have all been inferred, so that the graph that is

queried for adaptation always keep consistent. Moreover, performance also comes from the fact that this solution uses an incremental reasoner that only recomputes the parts of the graph that are impacted by context changes. Indeed, incremental reasoning allows capitalizing on previous reasoning process. The inferred information is continuously maintained in a graph, and the algorithm only processes the knowledge impacted by new insertions and deletions. Thus, contextual information can be exported and shared to other component on the infrastructure, using simple SPARQL CONSTRUCT and SPARQL Update queries.

Adaptation rules provide optimal decisions at runtime by attaching scores to adaptation possibilities. Yet, designing such rules by hand is tedious. Besides the fact that these scores should be clearly identified and motivated, the process of writing adaptation rules with scores by hand takes a significant time. This process may also be subject to erroneous, duplicated or contradicting rules. To reduce the time and errors while designing such rules, we propose in the next chapter a component to generate adaptation rules with little effort from WoT application designers, at design time.

3.7 Conclusion

The multi-purpose adaptation solution we propose relies on a semantic WoT platform and is able to incrementally reason about contextual information to support situation-based adaptation for multiple adaptation purposes. In addition to easing the WoT application designers' work by pooling contextual data collection for various adaptation purposes (which would anyway have been collected and processed separately otherwise), the adaptation solution design cycle is iterative and incremental, similarly to agile methods used in software development. It supports changes in domain knowledge, as well as in appliances and actors' description through the use of generic semantic methods. For instance, in the vineyard-watering scenario, if the user buys new drones with different characteristics (in terms of battery, storage or computing capabilities), the platform will seamlessly integrate them and adapt the application to these new devices, to the cost of a simple configuration task.

In the next chapter, we propose an implementation that allows generating adaptation rules using information about the context model, at design time.

Chapter 4

Multi-Purpose Adaptation Engine

Contents

4.1 Introduction

The multi-purpose adaptation solution presented in the previous chapter relies on Semantic Web technologies to process and reason about semantically-annotated contextual information.

In this chapter, we propose an implementation that generates adaptation rules, using information about multi-purpose context models. We present a state-of-the-art of adaptation rule design, which includes the process of generating rules, as well as the assignment of values to RDF triples inferred from these rules. We present a *meta-adaptation rule engine*, which generates adaptation rules at design time. We evaluate our solution in terms of correctness for various contextual situations, discuss our results and provide some insights regarding this contribution.

4.2 State of the art of adaptation rules design

Dynamic context-aware environments involve a large number of parameters (system resources, perceived environment, user preferences, etc.). Defining a workflow starting from all these parameters and resulting in an adaptation decision is a complex task [Chuang and Chan, 2008, Kakousis et al., 2010, Floch et al., 2006] that is often achieved using rules. The challenge when designing such rules is to provide accurate scores to semantically-annotated possibilities. Attaching values to adaptation possibilities is not straightforward, as in that case they are triples in the form *(functionality, adaptation purpose, candidate)*, which prevents attaching additional values. The following literature review presents some approaches to facilitate the design of rules, and to provide enriched information about triples issued from these rules.

4.2.1 Generation of rules.

The first approach is based on reutilization. S-LOR [Gyrard et al., 2017] allows sharing rules as Linked Data using a Linked Open Data vocabulary, and provides means to integrate those rules in semantic reasoners through the Jena API. Though this approach is promising, the rules that have been shared yet are designed for the interpretation of sensor data as semantic information rather than for adaptation. Still, challenges remain in the domain of rules generation. [Boussadi et al., 2011] propose a business rule design framework that relies the Semantics of Business Vocabulary and Business Rules (SBVR) formalism[1].

[1]http://www.omg.org/spec/SBVR/

The method described in [Emani, 2014] generates such rules in the same manner, while providing their semantic formalization as SPARQL queries. However, both methods requires application users and domain experts to express each rule in natural language and are intended for functional purposes only.

The reflexive method presented in [Andersson et al., 2008] allows generating on-the-fly adaptation rules based to support dynamic changes on a system. However, this adaptation solution is exclusively nonfunctional, centralized, and may not scale in environments with numerous connected objects. Their method also does not provide adaptation possibilities ranking. The solution presented in [Hong et al., 2009] infers rules with respect to contextual information, referred as the user's behavior. This however requires sufficient data to provide significant accuracy. Such method is appropriate for recommender systems, but not for adaptive systems on critical domains (such as security, healthcare, driving, etc.), as they require highly accurate adaptation rules as soon as the application starts.

4.2.2 Reification and related techniques.

In the literature, many techniques exist to attach additional information and metadata to RDF triples. A common method is the usage of named graphs [Carroll et al., 2005], which allows the representation of *quads*, *i.e.* triples with an additional atom denoting the graphs they belong to. Quads can be easily represented using the N-Quads language[2], which provides clarity and understandability for application designers. RDF+ [Schueler et al., 2008] extends named graphs by providing a additional atom to identify triples. Singleton properties [Nguyen et al., 2014] allow contextualizing triples using specific properties, in order to represent unique relationships. They are derived from generic properties and carry contextual information about the relationships, usually in terms of periods of time or locations. N-ary relations [3] are based on a standard-based pattern that provides extension of the classic RDF triple binary relation. This pattern consists in creating an individual, which is an instance of the former binary relation. This individual relates the things that are involved in that instance of the relation, as well as any other information, using additional properties having the instance class as their domain. NdFluents [Giménez-García et al., 2017] is an ontology for RDF statements annotation through the notion of *contextual extent*, which provides a unique identification for RDF statements and brings additional context-based

[2]N-Quads is a W3C recommendation. `https://www.w3.org/TR/n-quads/#n-quads-language`
[3]http://www.w3.org/TR/2006/NOTE-swbp-n-aryRelations-20060412/

information about this statement. Triple reification [Manola et al., 2004, Section 4.3] allows declaring RDF statements composed of the subject, predicate and object of specific triples, through standard RDF properties. The statement can then be linked to additional properties, to allow for unique identification of the triple $(subject, predicate, object)$ associated to additional values. RDF* [Hartig and Thompson, 2014] is an alternative approach to reification that allows representing RDF statements directly as subject of triples. This is a metadata extension of RDF, which provides additional syntax for SPARQL and Turtle.

Yet, some of the techniques enumerated above are not appropriate for adaptation possibility scoring, for the following reasons. First, the usage of named graphs to attach a score to a given adaptation possibility implies that the score is the named graph itself. Furthermore, there is no semantic relation between a triple and the named graph it belongs to (i.e. the meaning of a triple does not imply its belonging to a given named graph), whereas a semantic relation actually exist between scores and adaptation (as adaptation possibilities are ranked with respect to their scores). RDF+, which is based on and extends named graphs, has similar concerns. Second, the singleton properties imply the contextualization of relationships using specific values. In the case of multi-purpose adaptation, this would imply that adaptation possibilities (*i.e.* the relationship between an adapted functionality, an adaptation purpose and an adaptation candidate) are contextualized by their score. This is semantically incorrect as, in our work, scores are implied by the context, rather than being part of the context. Similarly, n-ary relationships between adaptation possibilities and scores would be semantically incorrect, as they imply that the adaptation purposes are directly related to the contextualization of the adaptation possibilities. By definition, contextual situations – and therefore specific sets of contextual instances – actually contextualize adaptation possibilities and their score, attributed by rule-based inference. Hence, the integration of ranked adaptation possibilities into the NdFluents ontology would make the instantiated model even more complex, and would lead to duplicated information about contextual situation-based extents, for each possible score. This would cause knowledge base inflation and decrease the contextual reasoning performance in terms of processing times. Third, RDF+, RDF* and singleton properties are non-standard extensions of RDF (and SPARQL for RDF*), making them difficult to be integrated into existing, standard-based solutions.

In the next section, we propose a generic way to design adaptation rules and determine scores by relying on a *meta-adaptation rule engine*.

4.3 Meta-adaptation rule engine

In this section, we present the implementation of our meta-adaptation rule engine, which generates rules at application design time. This engine 1) generates contextual situations, 2) generates adaptation possibilities, 3) combines these elements with score functions to apply a score on each adaptation possibility, and 4) outputs a set of adaptation rules using the situations and the ranked adaptation possibilities. This engine has been implemented in the ASAWoO platform.

The meta adaptation rule engine generates a set of adaptation rules. It relies on a triplestore that supports SPARQL SELECT and CONSTRUCT queries, and that takes semantically annotated information as input. The architecture of the meta adaptation rule engine is depicted in Figure 4.1 and is detailed in the subsections below.

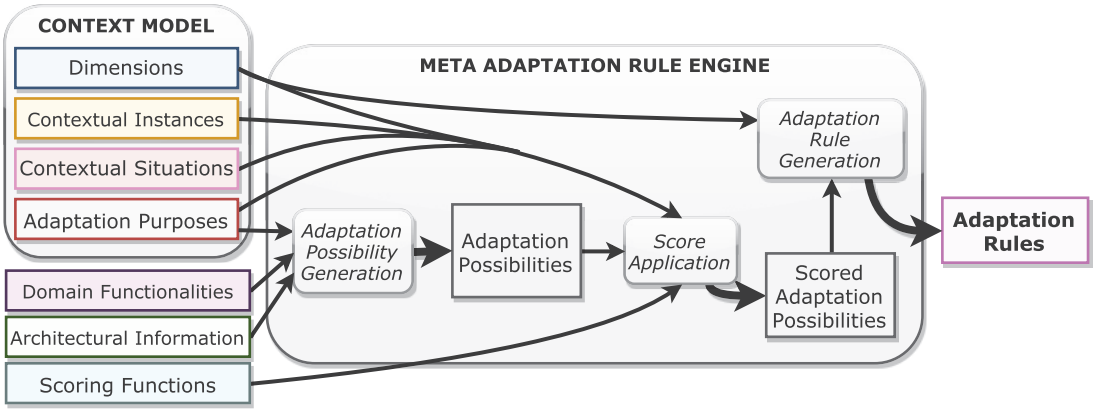


Figure 4.1: The meta adaptation rule engine, its components and inputs.

4.3.1 Generation of adaptation possibilities

The *adaptation possibility generation* step produces adaptation possibilities using the application domain functionalities, the architecture information (protocols, code hosts), and the adaptation purposes. To do so, the engine populates its triplestore with these semantically-annotated information and generates a graph of adaptation possibilities for each purpose,

using the query from Listing 4.1.

Listing 4.1: CONSTRUCT query to retrieve the graph of adaptation possibilities.

```
1  PREFIX asawoo-vocab: <http://liris.cnrs.fr/asawoo/vocab#>
2  PREFIX asawoo-ctx: <http://liris.cnrs.fr/asawoo/context/>
3  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4  CONSTRUCT { ?adapted ?possibilityPred ?candidate }
5  WHERE {
6          ?purpose asawoo-ctx:purposePredicate ?possibilityPred
               .
7          ?possibilityPred rdfs:domain ?adaptedClass .
8          ?possibilityPred rdfs:range ?candidateClass .
9          ?adapted rdf:type ?adaptedClass .
10         ?candidate rdf:type ?candidateClass .
11 }
```

In Listing 4.1, the triple patterns at lines 6-8 allow retrieving the adaptation purposes domains and ranges, so that each possibility is generated through the CONSTRUCT pattern (line 4). The $adapted/candidate$ and their classes from lines 9 and 10 refer to the subjects and objects enumerated in Table 4.1 below, which refer to the adaptation possibility triple patterns.

Purpose	Subject Type	Predicate	Object Type
Imp	Functionality	$hasSuitableCapabilityForImplementation$	Capability
$Comp$	Functionality	$hasSuitableFunctionalityForComposition$	Functionality
Exp	Functionality	$hasExposability$	Exposability
$Prtcl$	Functionality	$hasSuitableProtocol$	Protocol
$Code$	Functionality	$hasSuitableCodeLocation$	CodeLocation

Table 4.1: Triple patterns of adaptation possibilities, for each adaptation purpose.

4.3.2 Score management

To attach scores to adaptation possibility triples, we choose to rely on triple reification. Adaptation possibilities are derived into RDF statements, and are provided a score using the standard `rdf:value` predicate. Reified statements are derived using existential rules,

and the uniqueness or their URI is guaranteed by skolemization (*i.e.* by removing existential quantifiers from statements), preventing infinite loops at inference time. Reification is standard-based (properties are part of the RDF specification) and provides appropriate semantics for adaptation possibility scoring, as it does not imply that possibilities are contextualized by their score. The choice of reification is also validated by the quantitative evaluation from Chapter 3 – Section 3.5.2, which shows acceptable processing times.

To calculate possibility scores for each situation, the *score application* step combines contextual situations with adaptation possibilities and their score functions. A score function is represented as triples, as follows:

- their *rdf:type* is *asawoo-ctx:ScoringFunction*,
- they are linked to the contextual instance they apply to using the object property *asawoo-ctx:scoresInstance*,
- they are linked to a contextual dimension through the object property *asawoo-ctx:forDimension*
- they are applicable to an adaptation purpose through the object property *asawoo-ctx:applicableTo*,
- they are linked to a blank node using the object property *asawoo-ctx:scores*,
- this blank node consists in a reification through the following three object properties
  - *asawoo-ctx:forSituation* for the considered situation,
  - *asawoo-ctx:forCandidate* for the adaptation possibility candidate,
  - *asawoo-ctx:forAdapted* for the adapted functionality,
  - *rdf:value* for the actual score.

Listing 4.2 below shows an example of score function for the Bluetooth candidate, to adapt the communication protocol to transfer a picture the drone has a high battery level while in a *LowCPU_LowMemory* situation.

Listing 4.2: score function example (*Battery* dimension and *Protocol* purpose) in JSON-LD.

```
{
  "@context": {
    "asawoo-ctx": "http://liris.cnrs.fr/asawoo/context/",
    "asawoo-vocab": "http://liris.cnrs.fr/asawoo/vocab#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "@graph": [
    {
      "@id": "asawoo-ctx:F1",
      "@type": "asawoo-ctx:ScoringFunction",
```

```
13          "asawoo-ctx:scores": {
14            "@id": "_:F1_score_bn"
15          },
16          "asawoo-ctx:scoresInstance": {
17            "@id": "asawoo-ctx:HighBatteryForTransfer"
18          },
19          "asawoo-ctx:forDimension": {
20            "@id": "asawoo-ctx:Battery"
21          },
22          "asawoo-ctx:applicableTo": {
23            "@id": "asawoo-ctx:ProtocolPurpose"
24          }
25       },
26       {
27          "@id": "_:F1_score_bn",
28          "asawoo-ctx:forAdapted": {
29            "@id": "asawoo-vocab:PictureProcessing "
30          },
31          "asawoo-ctx:forCandidate": {
32            "@id": "asawoo-vocab:Bluetooth"
33          },
34          "asawoo-ctx:forSituation": {
35            "@id": "asawoo-ctx:LowCPU_LowMemory"
36          },
37          "rdf:value": 0.5
38       }
39    ]
40 }
```

The engine loads contextual instances, score functions and possibilities, and processes the SPARQL query from Listing 4.3 below in the triplestore. This query calculates scores for each group (contextual situation, adapted functionality, candidate possibility). Line 4-6 select the atoms that will compose each respective adaptation rule, compute the total score of the possibility candidate with the SUM aggregate (line 5), and retrieve each contextual instance from the current contextual situation using the GROUP_CONCAT aggregate at line 6 (these two aggregates come with the corresponding GROUP BY at line 19). Lines 8-10 link the adaptation possibility with the contextual situation. Line 12 retrieves the score functions applicable to the current purpose. Lines 13-17 then retrieve the score of each contextual instance that belongs to the current contextual situation, by reification.

Listing 4.3: SPARQL query that retrieves each groups of atoms composing the adaptation rules.

```
1 PREFIX asawoo-ctx: <http://liris.cnrs.fr/asawoo/context/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3
4 SELECT DISTINCT ?adapted ?purposePred ?candidate
5   (SUM(?score) AS ?candidateScore)
6   (GROUP_CONCAT(?contextInstance) AS ?instances) {
7
```

```
8       ?adapted ?purposePred ?candidate .
9       ?purpose asawoo-ctx:purposePredicate ?purposePred .
10
11      ?scoringFunction asawoo-ctx:applicableTo ?purpose .
12      ?scoringFunction asawoo-ctx:scoresInstance ?
            contextInstance .
13      ?scoringFunction asawoo-ctx:scores [
14          asawoo-ctx:forSituation ?contextSituation ;
15          asawoo-ctx:forCandidate ?candidate ;
16          asawoo-ctx:forAdapted ?adapted
17          rdf:value ?score ]
18
19  } GROUP BY ?adapted ?purposePred ?candidate ?contextSituation
```

Afterwards, the engine divides each score by the number of effective contextual instances from each contextual situation, to avoid invalid score computation if a contextual instance is missing (*e.g.* when a sensor is unable to send data).

4.3.3 Generation of adaptation rules

The *adaptation rule generation* step produces a set of adaptation rules using the SELECT bindings returned by the scoring application step. Algorithm 1 below details this process.

In Algorithm 1, causes and consequences are generated using the function $addConjunctiveAtom()$, which adds an atom (triple) in the conjunction. The cause is a conjunction of contextual instances, and the consequence is a set of ranked adaptation possibilities (*i.e.* a set of reified blank nodes, as described in the Definition 8 from Section 5.4.2). At each loop, if a rule with the generated cause already exists in the set ($getRuleWithCause()$ function), the generated consequence is added to this already existing rule ($addConsequence()$ function). Otherwise, a new adaptation rule is created and added to the set of adaptation rules. After this step, the meta-adaptation rule engine finally returns the set of adaptation rules, to be integrated into an avatar reasoner.

ALGORITHM 1: Adaptation rule generation

Data: A set B of SPARQL bindings with the variables { $adapted$, $purposePredicate$, $candidate$, $candidateScore$, $instances$ } corresponding to the scoring application output.

Result: A set R of conjunctive adaptation rules.

$R \leftarrow [\,]$
foreach $b \in B$ **do**
  $cause \leftarrow new\ Conjunction()$
  foreach $atom \in b.instances$ **do**
    $cause.addConjunctiveAtom(\text{"}atom$ rdf:type asawoo-ctx:ContextualInstance"$)$
  $consequence \leftarrow new\ Conjunction()$
  $consequence.addConjunctiveAtom(\text{" \_:blankNode rdf:subject } b.adapted\text{"})$
  $consequence.addConjunctiveAtom(\text{" \_:blankNode rdf:predicate } b.purposePredicate\text{"})$
  $consequence.addConjunctiveAtom(\text{" \_:blankNode rdf:object } b.candidate\text{"})$
  $consequence.addConjunctiveAtom(\text{" \_:blankNode rdf:value } b.candidateScore\text{"})$
  $tmpRule \leftarrow R.getRuleWithCause(cause)$
  if $tmpRule\ != null$ **then**
    $tmpRule.addConsequence(consequence)$
  else
    $R.push(new\ Rule(cause, consequence))$
return R

4.4 Querying ranked adaptation possibilities

Once generated, the set adaptation rules is integrated into the WoT platform reasoner, to infer and maintain adaptation possibilities. At decision time (Chapter 3, Section 3.3.3), these adaptation possibilities are queried using adaptation questions. These questions are formulated as SPARQL SELECT queries. Their pattern is based on the vocabulary described in Definition 8: subjects are the components to adapt (functionalities), predicates are based on the adaptation purposes, and objects are the adaptation possibilities. The pattern of a generic SPARQL-based adaptation question is described below (Listing 4.4).

Listing 4.4: SPARQL-based adaptation question.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?adaptationPossibility ?score {
        [] rdf:subject ?componentToBeAdapted ;
           rdf:predicate ?adaptationPurpose ;
           rdf:object ?adaptationPossibility ;
           rdf:value ?score .
} ORDER BY DESC(?score) LIMIT 1
```

The question consists in a SPARQL SELECT query that links adaptation possibilities with their reified score through the `rdf:value` predicate. The `ORDER BY DESC()` clause allows ordering the adaptation possibilities from the highest to the lowest score. Then, the `LIMIT 1` clause restricts the results to the optimal possibility. The adaptation question for the Exp adaptation purpose requires an additional FILTER clause on the score, as it must be strictly positive to allow exposability.

4.5 Evaluation

This section presents a correctness evaluation of the adaptation scores generated by the meta-adaptation rule engine. The goal is to ensure the adaptation results correspond to the expected adaptation results for different situations. We applied our experiments on the sustainable agriculture scenario described in Chapter 2 Section 2.5 and its context model (Table 2.1), and conducted this evaluation on a Dell OptiPlex 780 - Core 2 Duo E8400 @ 3 GHz. Stardog 4.2.3[4] has been used as a triplestore.

The correctness evaluation checks that the adaptation rules generated by the solution produces the expected ranked adaptation possibilities for the five ASAWoO purposes (Imp, $Comp$, Exp, $Prtcl$, $Code$). We consider the following situations: 1a) and 1b) are two different situations for the implementation of PictureTaking by different drones (distinct storage capacities and camera resolutions); 2a) and 2b) are two different drone situations for the composition of WateringNeedsDetection (distinct locations and storage capacities); 3) queries the OutdoorMotion exposability during a flood; 4) requires the PictureProcessing to choose a protocol in low-battery conditions for a far drone; and 5) requires to find the accurate PictureProcessing code location in a situation with a low CPU availability and a low battery level.

Situations 1a and 1b: Heterogeneous drones in terms of storage and camera resolution

The functionality PictureTaking can be implemented by two different drones with distinct situations. An optimal decision is expected, otherwise the watering needs detection would

[4]http://stardog.com/

fail. Let us consider drone 1, which has high storage capacity but has a low quality camera, and drone 2, which has limited storage capacity but is equipped with a HD camera.

Situation 1a (Drone #1)	HighStorageForPictureTaking	LowQualityForPictureTaking
Situation 1b (Drone #2)	LowStorageForPictureTaking	HighDefinitionForPictureTaking

Drone	CPU availability	Score
#1	High	0.6
#1	Low	0.3
#2	High	0.7
#2	Low	0.4

Table 4.2: Situations 1a/1b implementation scores.

Table 4.2 shows the scores obtained for the implementation of PictureTaking for each drone, with varying CPU availabilities. Results shows that the camera quality is the most impacting contextual information to determine the implementation of PictureTaking. However, a low quality drone will always be preferred over a high-definition one if it has both higher storage capacity and high CPU availability. This means that taking a high definition picture when having low storage capacity with a busy CPU does not ensure acceptable response times, and can even cause disruptions in worst case situations.

Situations 2a and 2b: Heterogeneous drones in terms of storage, located over different parts of the field

Two drones can take pictures to detect the watering needs. As these drones experience different situations, an optimal choice is expected to limit the energy and time consumptions on drones. Let us consider drone 3, which is far from the field but has high storage capacity, and drone 4, which is closer from the field but has a lower storage capacity.

Situation 2a (Drone #3)	FarFromFieldForDetection	HighStorageForDetection
Situation 2b (Drone #4)	CloseToFieldForDetection	LowStorageForDetection

Drone	Battery level	Score
#3	High	0.6
#3	Low	0.3
#4	High	0.7
#4	Low	0.4

Table 4.3: Situations 2a/2b composition scores.

Let us choose to vary the battery levels of drones. Table 4.3 details the scores obtained for the composition of WateringNeedsDetection for each drone. As expected, results shows that drones closer from the field are always preferred to take pictures. It also shows that, on this adaptation configuration, having a low battery level on a drone significantly lowers its composition score (regardless of its location).

Situation 3: Flood

The system must determine if the weather condition is satisfiable to expose the OutdoorMotion functionality. In a flood situation, the Wind dimension has no impact; as long as the contextual instance "Flooded" is present, the score of the possibility $(OutdoorMotion\ hasExposability\ Exposable)$ would always lower than 1. Thus, the functionality of OutdoorMotion is never exposed while in this situation.

Wind	**Score**
StrongWind	0.4
Breeze	0.7
NoWind	0.7

Table 4.4: Situation 3 exposability scores.

The scores obtained in Table 4.4 also show that the Breeze and NoWind instances from the Wind dimension produce identical scores, which means that this adaptation configuration (*i.e.* not exposing the OutdoorMotion functionality) also takes into account stormy weather situations. We can deduce that sending drones to fly over the field is only possible if the environmental conditions are neither stormy nor flooded.

Situation 4: Picture transmission to a far drone with low battery

In this situation, a drone must send a picture of the field to an another one in order to process it. However, the only drone able to receive is far, and the one that sends the picture has a low battery level. The system has to choose the best protocol to use, to save as much battery level as possible and to send the picture with acceptable delays.

Situation 4	FarFromDrone_n_ForTransfer	LowBatteryForTransfer

Table 4.5 shows that the adaptation configuration strongly favors Wifi if the picture receiver is too far. Indeed, using Bluetooth has a limited range. However, Bluetooth has a

Protocol	Far Score	Close Score
Bluetooth	0.1	0.7
Wifi	0.6	0.6

Table 4.5: Situation 4 protocols scores.

slightly higher score if drones are close from each other (regardless of the battery level) as it has a lower power consumption than Wifi, both when sending or receiving data [Perrucci et al., 2011].

Situation 5: Code hosting and execution for a drone with both limited battery and CPU, on high bandwidth conditions

In this situation, the system must determine the best solution to provide picture processing on a drone with limited capacities, while having a good network quality. It must determine the best location to host and execute this functionality (either on the drone itself or onto the cloud), in order to process pictures as fast as possible.

Situation 5	LowBatteryForTransfer	LowCPUAvailability ForProcessing	HighBandwidth ForProcessing

Storage capacity	Drone score	Cloud score
Low	0	1
High	0.3	0.9

Table 4.6: Situation 5 code locations scores.

Results from Table 4.6 show that the adaptation configuration prevents from locating the PictureProcessing functionality code on the device, as long as the storage capacity is not sufficient to host both the functionality module and the picture. It also shows that a high storage capacity is not enough for the device to be the optimal code location, as the adaptation solution would suggest taking advantage of the high bandwidth. In this configuration, we expect at least one more resource (either CPU or battery level) to be "High", to allow a score $>= 0.7$ for locating the code on the device. For all these reasons, the current situation always favors the cloud as the functionality code location.

4.6 Synthesis and discussion

This chapter presented a meta-adaptation rule engine that allows for generic, situation-based adaptation rules generation in WoT applications, based on domain-specific context models. We detailed how this process generates adaptation rules in a declarative way, through the use of the meta-adaptation rule engine. We presented the implementation of this engine and evaluated its correctness on the vineyard-watering application, for several situations.

In this work, we make the choice to rely on a semantic infrastructure to perform the adaptation. The choice of using semantic reasoning could be questionable. Indeed, the whole adaptive solution takes raw sensor data – a number – as input, infers contextual instances from these data, and attributes a score to adaptation possibilities regarding a set of contextual instances, which is also a number. However, the meaning of the score number is not the same. In a sense, scoring allows each type of information to be compared to each other, without considering the data units and ranges. Semantics also provide reusability through linked open vocabularies[5] and domain knowledge expertise ontologies[6], in an interoperable manner.

Genericity and "evolutivity"

In this approach, adaptation rules always infer "positive" answers, *i.e.* this adaptation solution is not built upon negative assumptions. This way, the system reasons about contextual information in an open-world assumption[7] that avoids to unexpectedly block application functionalities because a data is missing, or not available, or if it takes longer to transfer. Moreover, the score functions can take into account this imprecision by normalizing scores according to the number of actually available observations (aka dimensions). This makes this approach itself dynamically adaptive to contextual conditions.

On the long run, this approach can also be adapted throughout projects and platforms. While dimensions can be removed or added at design time, adaptation purposes can vary

[5]Linked Open Vocabularies (LOV) – `http://lov.okfn.org/dataset/lov/`

[6]Linked Open Vocabularies for Internet of Things (LOV4IoT) – `http://sensormeasurement.appspot.com/?p=ontologies`

[7]The absence of data does not imply that the contextual information is false or invalid, but rather is unknown.

according to the platform needs. For instance, at some point of the software maintenance cycle, the adaptation solution may require an additional adaptation purpose (corresponding to a new identified adaptation need). In that case, the application designers have to identify the possible candidates for this purpose as well as their nature (identified as objects in Section 3.3.3 – Table 4.1). They must also identify the contextual dimensions and their set of contextual instances, as well as the score functions required to provide adaptation for this purpose.

Reasoning capabilities

The adaptation solution we propose relies on an incremental, rule-based reasoner. This choice has a direct impact on reasoning performance, and therefore on the whole adaptation process. First, having incremental reasoning allows maintaining the *intentional* knowledge base, *i.e.* it maintains implicit facts in order to make SELECT query straightforward, for the cost of incremental updates. Second, using rule-based reasoning allows the integration of business rules in conjunction with sets of RDF-S and OWL entailment rules. In such system, both business rules and standard entailments can be removed or added, to tune the reasoning process with respect to the application needs. Adaptation rules are those business rules. Currently, they allow contextual adaptation in regular WoT applications (healthcare, smart spaces, agriculture, etc.).

In the next chapter, we study how the contextual reasoning task itself can be adapted, in order to improve the processing times of our adaptation solution. We explore the possibility to extend the original set of adaptation rules for the application and provide contextual adaptation for the reasoning task itself. Hence, the application would be aware of its reasoning tasks costs, and can adapt its behavior to improve time performances.

Chapter 5

Web Reasoning Performance

Contents

5.1 Introduction

In the previous chapters, we adressed multi-purpose contextual adaptation in the WoT. We use the ASAWoO platform, an avatar-based WoT infrastructure, to support our claims. In this platform, avatars rely on semantic reasoning to combine capabilities from the devices, data gathered by sensors, and information offered by Web services, in order to perform different tasks, among which adapt their functionalities for several purposes.

Still, research questions are open to provide optimization of the reasoning tasks. First, the reasoning task is a costly process. In the WoT application infrastructure, the reasoner could be located either on servers or on client devices. However, the choice of its location strongly depends on the context. Second, WoT applications usually face frequent reoccuring data (e.g. temperature, GPS coordinates). Reasoners in WoT applications would then frequently handle facts that have already been processed before.

In this chapter, we propose two solutions to optimize the contextual reasoning process. The first is to provide adaptive reasoning based on a location-agnostic reasoning architecture, able to execute different reasoning tasks either on the client side (*i.e.* directly on devices) or on the server-side (*i.e.* on the cloud). The second solution is to improve the computation times of deletions and re-insertion tasks by tagging (*i.e.* annotating) KB facts with their provenance and validity.

The contextual reasoning engine we used in Chapter 3 for the evaluation is HyLAR, a rule-based reasoner called HyLAR [Terdjimi et al., 2016a], which has been designed according to the RL profile. HyLAR is an appropriate choice for multi-purpose contexual adaptation in the WoT, as it provides straightforward SELECT queries and allows answering adaptation queries in less than 100ms.

In the following sections, we present a state-of-the-art on Web reasoning and detail our contributions and validate them on diverse evaluation setups, for each respective reasoning optimization.

5.2 State-of-the art on Web reasoning

Our contribution aims at designing reasoning processes that "bridge the gap between the Web and the Semantic Web"[1]. Making better use of standard Web mechanisms (such as HTTP caching and proxying) is the first mean to tackle this problem.

5.2.1 Reasoning in Web applications with OWL profiles.

OWL 2 profiles[2] help adjusting the trade-off between expressivity and efficiency. Each profile (EL, QL, RL) has its own specificities and targets different reasoning tasks, such as classification (*i.e.* computing the transitive closure of a graph when loading an ontology) and query answering (SELECT/UPDATE queries). Reasoning tasks differ in terms of data, query and taxonomic complexity [Motik et al., 2009]. The choice of the appropriate OWL profile is crucial to reduce reasoning overheads, but not always sufficient as reasoners mostly rely on materialization (e.g. pre-compute and store inferences [Motik et al., 2015a]) which is computationally intensive. EL is suitable for very large TBoxes and would not fit Web applications that expect their ABoxes to be contain more data than their TBoxes.

QL is appropriate for applications that manipulate high volumes of instances. It relies on query rewriting, which is not appropriate for Web applications that require fast query answering such as in WoT application scenarios. RL is more suitable, as it allows all axioms to be represented as logical implications and rules to be constructed as needed: to enable reasoning about OWL constructs, one can define both entailment rules corresponding to the expressive power expected for the application, and application-specific rules. RL reasoners can involve a large amount of explicit facts [Krötzsch, 2012], and inferences are pre-computed and explicitly stored, so that queries can be answered simply by querying the store [Horrocks and Patel-Schneider, 2010]. This makes this profile suitable for Web applications that require flexibility and need fast query answering.

[1] Phil Archer, W3C, SemWeb.Pro Paris, Nov. 2014
[2] http://www.w3.org/TR/owl2-profiles/

5.2.2 Reasoning optimization approaches

In [Krishnaswamy and Li, 2014], the author discuss challenges in mobile OWL reasoning. They describe how to reduce load by configuring reasoners for precise tasks using limited description logics. Kollia and Glimm [Kollia and Glimm, 2014] propose to rewrite axiom templates into smaller templates to reduce the query evaluation time cost. The Triple Pattern Fragments [Verborgh et al., 2014a] (TPF) interface is a Web API to RDF data where clients can ask for triples matching a certain triple pattern. This approach relies on intelligent clients that query TPF servers to address the problem of scalability and availability of SPARQL endpoints. However, the use of a LDF (Linked Data Fragments) server is necessary.

Current existing mobile reasoners are based on first-order logic (FOL), managing Tbox (schema), Rbox (roles) and Abox (assertions). Sinner and Kleemann's KRHyper [Sinner and Kleemann, 2005] is a novel-tableaux based algorithm for FOL. However, according to [Krishnaswamy and Li, 2014], KRHyper encounters memory exhausting problems when the reasoning task becomes too large for the device. Based on $\mathcal{ALCN}$, Mine-ME 2.0 from [Ruta et al., 2014] runs on Android devices. Embedded reasoners such as the $\mathcal{EL}+$ reasoner proposed in [Grimm et al., 2012] are capable of reasoning on large Tboxes due to the limitations of $\mathcal{EL}$ (no individuals nor concept disjointness). But neither [Ruta et al., 2014] nor [Grimm et al., 2012] provide access to Web clients.

5.2.3 Incremental reasoning in RL.

Web applications also need to handle frequent data updates. Reasoners embedded in those applications can then rely on incremental reasoning (IR) [Motik et al., 2012] to avoid entire recomputations. Several improvements of IR currently exist. The *fact-dependency tracking* from [Goasdoué et al., 2013] traces the origin of facts (*i.e.* the other facts they have been inferred with) and relies on query reformulation to provide the appropriate query result. The *counting* method [Gupta et al., 1993] also tracks alternative derivations for each fact but does not support recursive rules. However, even *counting* algorithms that support recursion such as in [Dewan et al., 1992] do not reduce the re-insertion cost as alternative derivations are not explicitly stated but rather counted. Nowadays, most incremental main-

tenance algorithms are based on Gupta et al.'s delete-rederive (DRed) [Gupta et al., 1993]. DRed improves performance as it ensures that the rules apply only to modified facts and thus prevents complete and successive recalculations of the KB on each update. The solution presented in [Kazakov and Klinov, 2013] relies on DRed for the classification task, but it exclusively targets $\mathcal{EL}+$ ontologies with complex and changing TBoxes to tackle reclassification issues.

In [Motik et al., 2015b], the authors tackle the derivation redundancy issue encountered in the overdeletion step using a semi-naive materialization approach that combines backward and forward (BF) chaining. This improvement however still relies on rule matching and evaluation at deletion, and does not completely avoid overdeletion and re-derivation. They implemented this approach in the RDFox triplestore [Nenov et al., 2015] that targets highly scalable applications. Yet, we aim to build a JavaScript solution that targets Web browsers, which is currently not possible with RDFox.

5.2.4 Web-based reasoners

An approach to embed a reasoner in mobile devices is to rely on Web standards and run it in a Web browser in Javascript. Such works are oriented towards Web-based technologies and rely on reasoners written in Javascript that can be embedded in mobile devices and ran on the device browser, independently of their operating system. EYE[3] is a NodeJS[4]-compatible reasoner capable of inferring on FOL rules, performing server-side reasoning while a client widget renders a graphical interface for SPARQL querying. As far as we know, the reasoner has not been ported onto the client side. Based on the JSW Toolkit, OWLReasoner[5] allows client-side processing of SPARQL queries on OWL 2 EL ontologies. After parsing an ontology, a classification step performs its deductive closure to return its Tbox and Abox and converts them into a relational database. SPARQL queries sent to the reasoner are rewritten into SQL queries, and processed on the database. Yet, the OWLReasoner SPARQL engine is limited to basic rule assertions, and its reasoning engine cannot be provided additional rule-based entailments without modifying the reasoner code. The Constraint Handling Rules [Frühwirth, 2015] language also allows efficient rule-based reasoning and has been implemented in JavaScript. However, it does not provide incremental

[3]http://reasoning.restdesc.org/
[4]https://nodejs.org/
[5]https://code.google.com/p/owlreasoner/

maintenance.

5.3 Hybrid Location-Agnostic Reasoning

To address performance concerns that arise with high numbers of simultaneous requests, Web application designers dispose of several tools, among which caching static data and deferring code execution from the server to the client side. But even if in average, client processing resources augment at a fast pace, they remain heterogeneous and in some cases, too limited to execute heavy calculation processes.

In some way, solving SPARQL queries for a large number of clients can require heavy reasoning processes and cause server unavailabilities. Client-side reasoning is therefore to consider to lighten server charge, since current mobile devices and smart appliances sometimes have enough computing capabilities to execute one or more reasoning tasks. Yet, their diversity require the ability to defer reasoning tasks on either side (client device or more powerful server) depending on the context. For such a possibility to work in the WoT paradigm, reasoning solutions must provide the possibility to reason on the Web and to be flexible enough to locate reasoning tasks on the appropriate sides.

We focus on taking in consideration the recent advances in Web technologies to exploit client resources by deferring code execution on the client. We therefore focus on JavaScript-based reasoning, so that the same parts of code can both be deployed on the client and server sides, to provide an adaptable reasoning task. We build this reasoning architecture with respect to W3C standards, using semantics-based description logics (DL) over FOL.

5.3.1 Study of the influence of location on the reasoning process performance

To improve the reasoning performance, we propose to separate and locate the reasoning tasks executed once (which can be preprocessed on the server side, such as parsing and classification steps) from the reasoning tasks executed when a query is sent to the reasoner

(SPARQL query parsing and answering). In this study, we consider stable ontologies (*i.e.* we do not consider the re-classification of the schema).

To identify the relevant pieces of contextual information to consider for the reasoning task adaptation, we propose a preliminary evaluation to benchmark the execution of the steps in different reasoning code location configurations. The following subsections characterize the most suitable architecture by evaluating the reasoning efficiency wrt. several parameters: client resource limitation, number of simultaneous clients requesting the SPARQL endpoint, size of the processed ontology and network latency. This work has been published in [Terdjimi, 2015, Terdjimi et al., 2016a].

Benchmark of influence of contextual parameters

We consider four tasks, representing all possible steps of the reasoning process: (0) for loading client scripts; (1) for loading a raw ontology; (2) for performing ontology parsing, classification and loading the parsed ontology; (3) for SPARQL query processing through the reasoner. These are depicted in Figure 5.1 below.

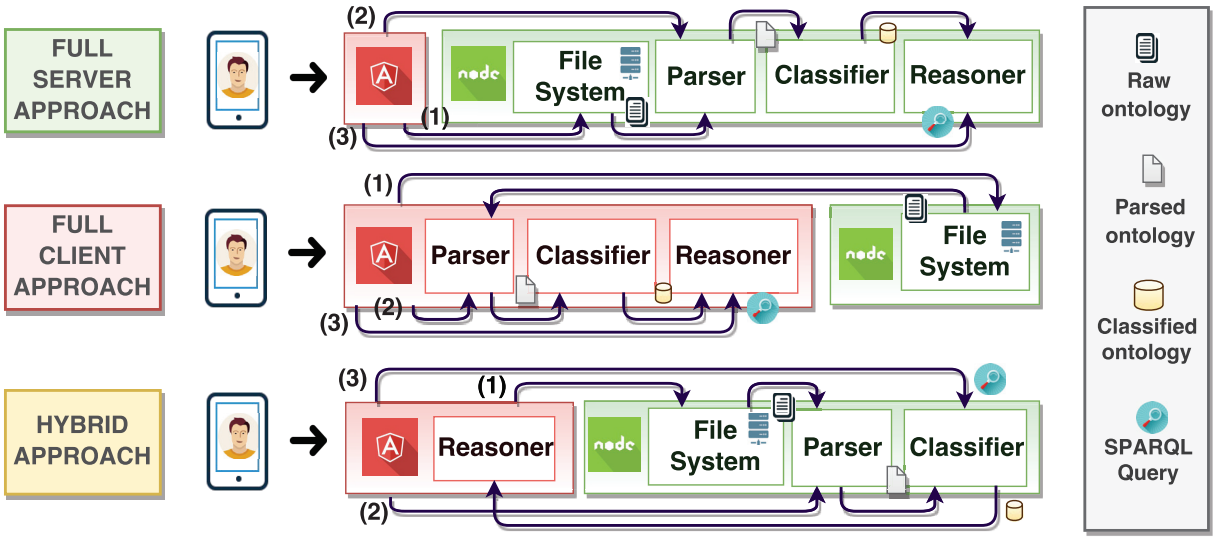


Figure 5.1: Architectures used for our evaluation

We used the architecture presented in Section 5.1 to evaluate the overall reasoning process times in three situations: full server-side, full client-side and hybrid (server-side pars-

ing and classification, and client-side query processing). Figure 5.1 shows (1), (2) and (3) for each situation. Additionally, for the hybrid and full client-side variants, client-side parts are evaluated both with and without a Web worker. We assume that scripts and ontologies are available on the server. All scenarios conform to a query-processing-response pattern. In the result tables, we noted [Q] the time for the client's request to reach the server; [P] the processing time and [R] the time for the server response to reach the client. Depending on the scenario and location of the calculations, some parts of this steps/patterns are considered immediate (e.g. querying the local reasoner to process a query). They are noted in the result tables as not applicable. Each evaluation is tested on two ontologies[6]: A (1801 class assertions and 924 object property assertions) and B (12621 class and no object property assertions).

The tests ran above show network request and response delays for each scenario. It is realized by simulating a remote server with Clumsy 0.2[7]. [R0] is the time for the client to load scripts and following are the respective query/response times for [Q1]/[R1] retrieving the raw ontlogy, [Q2]/[R2] retrieving the classification result and [Q3]/[R3] sending the SPARQL query and retrieving results. A second evaluation compares processing times for [P2] classification and [P3] reasoning in three different configurations: a Dell Inspiron (with Chrome), a Nokia Lumia 1320 (Snapdragon S4 @ 1700 MHz, with Internet Explorer), a Samsung Galaxy Note (ARM cortex A9 Dual-Core @ 1,4 GHz, with Firefox) and a Node.js server set up in the Inspiron.

Ontologies A/B	**[R0]**	**[Q1]**	**[R1]**	**[Q2]**	**[R2]**	**[Q3]**	**[R3]**
Remote server	334	54	110/275	119/120	167/647	146/154	61/85

Table 5.1: Network delays (in ms)

As expected, Table 5.2 shows that the server has the best results for the classification processing time and can use caching. Even if the raw ontology is faster to load than the classification results, loading scripts and data on the client is much faster than performing the same classification step on each client. Therefore, it makes no sense to defer and duplicate heavy calculations onto clients, rather than pre-calculating them on the server and caching results. Table 5.2 shows an important difference between configurations: it keeps reasonable processing time for the query answering task in good to average configurations (e.g. Inspiron and Lumia), but the older Galaxy Note is ten times slower than the server. For such limited resource devices, the server could therefore take over the answering pro-

[6]We chose ontologies of "reasonable" sizes, representing datasets that a Web application can require. For instance, ontology B has actually been used to perform client-side recommendation in [Médini et al., 2013]

[7]http://jagt.github.io/clumsy/

Ontologies A/B	**[P2] (no worker)**	**[P2] (worker)**	**[P3](no worker)**	**[P3] (worker)**
Inspiron (Chrome)	790 /27612	764/26464	28/101	24/88
Lumia (IE)	1989/54702	1883/53801	156/198	144/185
Galaxy Note (Firefox)	2954/81255	2872/79752	465/2988	440/2872
Server (Node.js)	780/20972	n/a	35/37	n/a

Table 5.2: Classification [P2] and reasoning [P3] times (in ms)

cess. More generally, for M clients and N queries/client, the three configuration calculation times can be calculated as follows[8]:

Full Server-side	$P2_{server} + M \times N \times (Q3 + P3_{server} + R3)$
Full Client-side	$M \times (R0 + Q1 + R1) + P2_{client} + N \times P3_{client}$
Hybrid	$P2_{server} + M \times (R0 + Q2 + R2) + N \times P3_{client}$

Synthesis

Globally, the evaluation shows that choosing a location for the query answering process is not as simple as for the classification step. For clients with limited resources, it can be more efficient to perform this step on the server. But as the ontology usage (number of queries per client) and the server load (number of clients) grow, it appears that relocating query processing on the client can be a good strategy, since queries can be processed autonomously on each client. A more powerful server would shorten server-side response times, resulting in shifting the strategy switching point. Still, higher performance – and therefore scalability – can be achieved by deferring this step on clients.

[8]Server-side classification (performed once and then cached) and client-side calculations (performed in parallel) are only counted once.

5.3.2 Reasoner code location adaptation

The preliminary evaluation above introduced the possibility to improve the reasoning process by deferring appropriate tasks to clients. We use this strategy to answer the adaptation purpose $Code$ described in Chapter 2 to locate the reasoning task at runtime, according to contextual information. We aim to identify the best code execution locations for reasoning tasks in a transparent manner. In this setup, semantic reasoning is divided in two tasks. The first is the domain ontology classification, which is the most complex but is only executed when the application starts. The second task is query answering, which consists in processing SPARQL queries on the knowledge base. This task is more straightforward but more frequent. We evaluate the efficiency of the code execution location strategy for these tasks. In order to evaluate tasks with different complexity levels, we chose to only evaluate SELECT queries.

The context model we consider is composed of the following set of dimensions { *Privacy*, *OntologySize*, *Ping*, *Battery*}. Their respective set of contextual instances are detailed in the transformation setup section. The adaptation purpose is $Code$. The set of adaptation possibilities is { *Device*, *Cloud* }, i.e. the possible code locations. The adapted components are the reasoning tasks { *Classification*, *QueryAnswering* }. The scores are detailed in the adaptation setup section.

Transformation setup

The contextual model for the reasoning task includes the following dimensions: ontology size, battery level, ping duration and user privacy preferences. Data in these dimensions come from different sources: the cloud and the device. The contextual model is built by transforming numerical values in the dimensions into level-specific DL facts. In this example, this transformation relies on the following rules:

- If the user has set her privacy preference in her profile as “important”, then the contextual instance for this dimension is $HighPrivacy$. If she has chosen to set it as “not important”, it is $LowPrivacy$.
- If the ontology contains less than 200 entities, then the inferred contextual instance is $SmallOntology$. If not, it is $BigOntology$.

- If the ping duration exceeds 150 ms, then the infered contextual instane is $LongPing$. If not, it is $ShortPing$.
- If the battery level exceeds 30%, then the inferred contextual instance is $HighBattery$. If not, it is $LowBattery$.

This transformation setup limits the inference of contextual instances to four simultaneous instances using four rules, which corresponds to the necessary contextual information used to adapt the code location.

Adaptation setup

In this adaptation setup, the two functionalities to be adapted are the ontology classification and the query answering tasks, the adapation purpose is $Code$, and the possible code execution locations are either on drones or on the cloud. The contextual instances are Big or $Small$ for the ontology size, $High$ or Low for the battery level, and $Long$ or $Short$ for the ping duration. The entailments below describe the rules related to this configuration. They have been written in a simplified manner, where a rule atom antecedent $CtxtInst(I)$ denotes the presence of the inferred contextual instance I and a rule atom consequent $X(Y)$ denotes the location Y of the reasoning task X. Adaptation possibilities explicitely inferred from the rules below always have their score $s = 1$, while their opposite possibilities (*i.e.* Device for Cloud, and vice versa) will have their score $s = 0$. For instance, HighPrivacy will produce the possibility "Classification(Device)" with $s = 1$, thus the score of "Classification(Cloud)" would be $s = 0$.

$$
\begin{aligned}
CtxtInst(HighPrivacy) &\rightarrow Classification(Device)\\
CtxtInst(BigOntology) &\rightarrow Classification(Cloud)\\
CtxtInst(LowBattery) &\rightarrow Classification(Cloud)\\
CtxtInst(SmallOntology) \wedge CtxtInst(HighBattery) &\rightarrow Classification(Device)\\
CtxtInst(LongPing) &\rightarrow QueryAnswering(Device)\\
CtxtInst(HighBattery) &\rightarrow QueryAnswering(Device)\\
CtxtInst(ShortPing) \wedge CtxtInst(LowBattery) &\rightarrow QueryAnswering(Cloud)
\end{aligned}
$$

The classification task is executed directly on the device only if the ontology size is small and the battery level is high. Otherwise, it is executed on the cloud. For the query answering task, it is executed on the device except if the ping duration is short and the

battery level is low, in which case it is executed in the cloud. At request time, the context manager sends the code location adaptation query question (Listing 5.1) to the semantic reasoner.

Listing 5.1: SPARQL code location adaptation question.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX asawoo-ctx: <http://liris.cnrs.fr/asawoo/context/>
SELECT ?codeLocation ?score {
        [] rdf:subject ?reasoningTask ;
           rdf:predicate asawoo-ctx:hasSuitableCodeLocation ;
           rdf:object ?codeLocation ;
           rdf:value ?score .
} ORDER BY ?score
```

Practical evaluation

We evaluate this adaptation on a Pentium Dual-Core CPU E5500 @ 2.80 GHz with 4 Gb RAM that acts as the device, and a virtual machine with two VCPU and 2 Gb RAM hosted on a cloud infrastructure based on Intel E52680 @ 2.50 GHz processors. The server on the cloud runs a Node[9] / Express[10] engine. The device itself is not a drone (to simplify the measures) but acts as such. We simulate network throttling using Google Chrome Canary V.50.0.2651.0 on the 'Regular 2G' mode (250 kb/s down, 50 kb/s up, 300 ms RTT). We use two ontologies: (O1) (Fipa-Device[11]) with 126 entities (schema + axioms) and (O2) (IoT-O[12]) with 328 entities. The experiment relies on HyLAR to migrate the code between the object and the cloud, and to measure reasoning times.

The process of this evaluation consists in (a) classifying the ontology and (b) sending a select query. We calculated for both (a) and (b): (REQ) the time for the device request to reach the cloud, (PROC) the task processing time, and (RES) the time for the cloud response to reach the device. We use the following setups both in a wired connection or in the 'Regular 2G' mode:

1. Locating the execution of (a) and (b) exclusively on the cloud (M1).
2. Locating the execution of (a) and (b) exclusively on the device (M2).

[9] https://nodejs.org/
[10] http://expressjs.com/
[11] www.fipa.org/specs/fipa00091/PC00091A.html
[12] http://www.irit.fr/recherches/MELODI/ontologies/IoT-O.owl

3. Locating the execution of (a) and (b) with respect to the adaptation engine answer.

(O1)		CLASSIFICATION (time in ms)			QUERY ANSWERING (time in ms)			Total (ms)
		(REQ)	(PROC)	(RES)	(REQ)	(PROC)	(RES)	
CLOUD	Wired	20	411	25	20	2	20	498
	Regular 2G	350	411	380	350	2	350	1843
DEVICE	Wired	20	660	20	0	11	0	711
	Regular 2G	350	660	350	0	11	0	1371
ADAPTIVE	Wired	20	660	20	0	11	0	711
	Regular 2G	350	660	350	0	11	0	1371

(O2)		CLASSIFICATION (times in ms)			QUERY ANSWERING (times in ms)			Total (ms)
		(REQ)	(PROC)	(RES)	(REQ)	(PROC)	(RES)	
CLOUD	Wired	20	7527	50	20	8	10	7635
	Regular 2G	350	7537	6427	350	8	444	15116
DEVICE	Wired	20	9389	19	0	31	0	9459
	Regular 2G	350	9389	1777	0	31	0	11547
ADAPTIVE	Wired	20	7527	50	0	31	0	7628
	Regular 2G	350	9389	1777	0	31	0	11547

Figure 5.2: Request, task processing and response times for ontologies (O1), (O2) in different network conditions.

Time saved on adaptation: (a) then (b)	CLOUD	DEVICE
Wired (O1)	-43%	0%
Regular 2G (O1)	26%	0%
Wired (O2)	1%	20%
Regular 2G (O2)	24%	0%

Time saved on adaptation: (a) then 10x(b)	CLOUD	DEVICE
Wired (O1)	8%	0%
Regular 2G (O1)	82%	0%
Wired (O2)	1%	19%
Regular 2G (O2)	48%	0%

Figure 5.3: Percentage of time saved if using an adaptive solution in comparison to full cloud or full device code execution, for two different workflows.

Figure 5.2 shows the results of our experimentations, and Figure 5.3 shows the time saved by using an adaptive solution in comparison to fully relying on the cloud or fully relying on the device. The latter also compares both the initial workflow (a), (b) and a more realistic scenario (a), 10 x (b), i.e. an initial classification followed by several queries. This

depicts how an adaptive solution can improve the reasoning performances by answering the question of code execution location, which is 82% more effective than fixed implementations in disrupted environments for an average domain-specific ontology.

Evaluation synthesis

The evaluation showed that choosing the relevant contextual information to adapt the location of reasoning tasks significantly improves the reasoner time performance. We showed that, in practice, using the adaptation solution does not cause a time overhead for reasoners handling a significant amount of queries. Instead, the adaptation solution improves the time performance in worst cases (*e.g.* bad network conditions, large ontologies...).

In the next section, we tackle the issues in UPDATE queries processing with incremental reasoning for WoT applications that face frequently re-ocurring information.

5.4 Tag-Based Reasoning

WoT applications must dynamically handle various types of contents generated by users or client sensors. Existing semantic technologies could improve these applications, but the state of the art shows that they are currently under-exploited. One reason is that full-fledged semantic stacks are perceived as costly, unreliable server-sided architectures, in opposition with current (i.e. modular and client-side) Web design practices [Verborgh et al., 2014b]. Adaptation of the reasoning can solve issues that are specific to the WoT application infrastructure and software environment. The previous section addressed this problem through code location adaptation, using HyLAR to defer specific reasoning tasks either on the server or on the client sides.

State-of-the-art in semantic reasoning research work [Motik et al., 2012] aims at improving reasoning tasks through maintenance algorithms such as DRed-based incremental reasoning (IR). However, when the updated data is cyclic (i.e. facts that re-occur periodically), applications should not only rely on IR to optimize reasoning, as they are regularly exposed to overheads caused by re-deriving implicit facts that have been already derived in the past. The objective of this section is to allow using reasoning for tasks currently located on WoT application clients, that satisfy several conditions. We focus on datasets of relatively small

size ($<$ 50k lines) and target Web applications based on stable data models (TBoxes) and more varying model instances (ABoxes).

To solve the present issues in incremental maintenance for WoT applications, we propose an approach inspired by IR that prevents successive re-derivations by tagging facts with respect to their provenance and validity. The solution we propose includes the following contributions:

1. **Faster deletions using validity tagging.** we provide validity tagging for explicit facts and do not process costly overdeletion tasks. Instead, explicit facts are tagged as invalid at deletion time and as valid at re-insertion time.
2. **Faster re-insertions using provenance tagging.** Our approach tracks the provenance of all implicit facts (i.e. all possible derivations), which avoids having to re-evaluate them if they are reinserted in the knowledge base.

In this section, we formalize and highlight the re-derivation overhead problem, in a classic WoT application setting. We propose three algorithms: implicit fact tagging, tag-based KB update and fact selection filtering, and analyse the gain and cost of tagging facts in terms of complexity. We evaluate the TB maintenance approach by comparing it with IR and discusses the results with respect to different application settings.

5.4.1 Illustration with a smart home case study

WoT applications – or even more generally, Web applications – can be subject to frequent updates. Possibly re-occurring data can be re-inserted or re-deleted, which can cause significant computational overheads. We illustrate this issue with the scenario of a mobile WoT application connected to a smart house: Julia uses this application on her smartphone to automatically regulate her house temperature when she approaches her house. The application locates her mobile phone either using its GPS sensor or by recognizing the network it is connected to. She will be considered close to her house either if her cell phone GPS coordinates correspond to her house neighborhood or if she connects the phone to the house local network. This activates temperature regulation and deactivates it otherwise. Julia's proximity from her house is the re-occurring data: the application infers or not this information as she moves back and forth with her cell phone, as she switches on and off the GPS sensor, or as she connects and disconnects her phone from the house network.

We use the following formalization, from [Motik et al., 2012]: a fact F can be explicit (*i.e.* provided at startup or update), implicit (*i.e.* derived as a rule consequence), or both implicit and explicit (*i.e.* explicitly stated and derived). A rule r has an antecedent, conjunction of facts F_i, $i \in \mathbb{N}$) and an implied consequence (a single fact I); when it applies, the consequence is derived as an implicit fact: r :- F1 $\wedge$ F2 $\wedge$ ... $\wedge$ Fx $\rightarrow$ I.

Application ontology. Julia's application in our scenario uses the following fixed ontology (Classes and Properties) and entailment rules.

```
# EO1
:PhysicalAgent rdf:type owl:Class .
# EO2
:User rdfs:subClassOf :PhysicalAgent .
# EO3
:SmartDevice rdfs:subClassOf :PhysicalAgent .
# EO4
:SmartPhone rdfs:subClassOf :SmartDevice .
# EO5
:SmartHome rdfs:subClassOf :SmartDevice .
# EO6
:Location rdf:type owl:Class .
# EO7
:TemperatureStatus rdf:type owl:Class .
```

Listing 5.2: Classes

```
# EO8
:hasLocation rdf:type owl:ObjectProperty .
# EO9
:hasLocation rdfs:domain :PhysicalAgent .
# EO10
:hasLocation rdfs:range :Location .
# EO11
:hasLocationCloseTo rdf:type owl:ObjectProperty .
# EO12
:hasLocationCloseTo rdf:type owl:TransitiveProperty .
# EO13
:hasLocationCloseTo rdfs:domain :PhysicalAgent .
# EO14
:hasLocationCloseTo rdfs:range :PhysicalAgent .
```

```
# EO15
:hasTemperatureRegulation rdf:type owl:ObjectProperty .
# EO16
:hasTemperatureRegulation rdfs:domain :SmartHome .
# EO17
:hasTemperatureRegulation rdfs:range :TemperatureStatus .
```

Listing 5.3: Properties

```
# Transitivity
(?p rdf:type owl:TransitiveProperty) ∧ (?i1 ?p ?i2) ∧ (?i2 ?p ?i3)
        → (?i1 ?p ?i3)
# Subsumption
(?c1 rdfs:subClassOf ?c2) ∧ (?s rdf:type ?c1)
        → (?s rdf:type ?c2)
```

Listing 5.4: Entailment rules

Application instances and rules. Below are the initial explicit and implicit facts inferred via the Business Rules (Listing 5.5), which drive the application behavior. The set of initial explicit facts declares Julia, her cell phone, her house and the instance that activates temperature regulation in the KB, and assumes that Julia always carries her cell phone with her. The application can reason about their locations via $r1$ (as they are inferred as physical agents), and can switch on the regulation via $r2$.

```
# r1
(?agent :hasLocation :JuliasHouseNeighborhoodLocation)
        → (?agent :hasLocationCloseTo :JuliasHouse)
# r2
(:Julia :hasLocationCloseTo :JuliasHouse)
        → (:JuliasHouse :hasTemperatureRegulation :Activated)
```

Listing 5.5: Business rules

```
# E1
:Julia rdf:type :User .
# E2
:JuliasPhone rdf:type :SmartPhone .
# E3
```

```
:JuliasHouse rdf:type :SmartHome .
# E4
:Julia :hasLocationCloseTo :JuliasPhone .
# E5
:Activated rdf:type :TemperatureStatus .
```

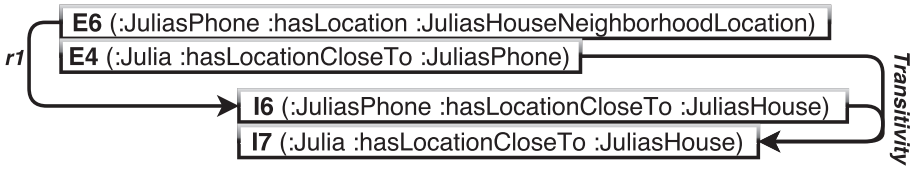
Listing 5.6: Initial explicit facts

```
# I1 (Subsumption)
:JuliasPhone rdf:type :SmartDevice .
# I2 (Subsumption)
:JuliasHouse rdf:type :SmartDevice .
# I3 (Subsumption)
:Julia rdf:type :PhysicalAgent .
# I4 (Subsumption)
:JuliasPhone rdf:type :PhysicalAgent .
# I5 (Subsumption)
:JuliasHouse rdf:type :PhysicalAgent .
```

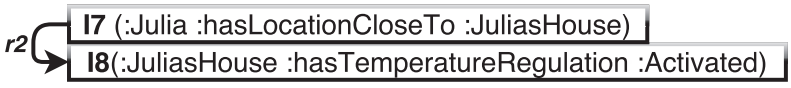
Listing 5.7: Initial implicit facts (inferred instances)

We consider the following 3-steps scenario.

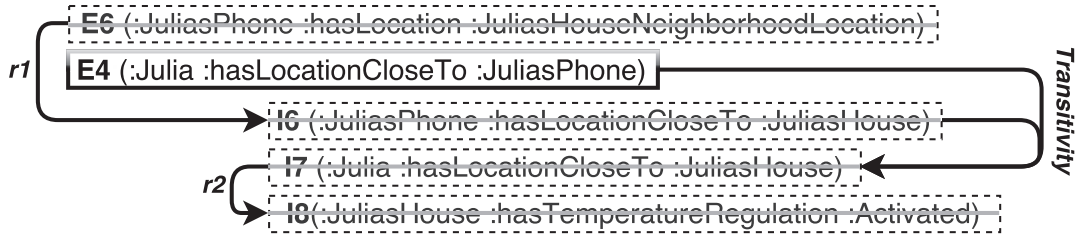
(1) Julia approaches her neighborhood with her cell phone. The application analyzes the phone GPS coordinates and adds the explicit fact E6. This allows the reasoner to infer I6 via $r1$ and I7 via $Transitivity$.



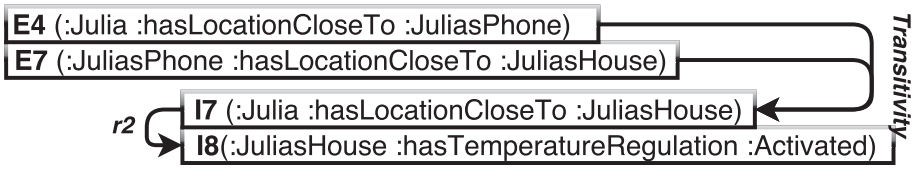
The application then enables temperature regulation as I7 triggers I8 via $r2$.



(2) Julia enters her house and cuts off the GPS to save energy. The phone position becomes unknown. The application removes E6, which also triggers the removal of I6, I7, I8, and disables temperature regulation.



(3) Julia connects her phone to the house local network. The application inserts E7, causing I7 and I8 to be re-derived respectively via $Transitivity$ and $r2$.



Step 3 highlights the re-evaluation overhead caused by over-deletion in the IR algorithm: the deletion and reinsertion of explicit facts leads to the re-derivation of two implicit facts that have already been derived at first insertion.

5.4.2 Tag-based Incremental Maintenance

To avoid recurrent re-derivations, we propose to keep the origin of previously obtained inferences so that when already known facts re-occur, the reasoner can quickly retrieve their consequences. To do so, it must *keep track of all facts*, including deleted ones, and be able to *assess their validity*: explicit facts are tagged as valid/invalid, and implicit fact validity is retrieved using those of the explicit facts they have been derived from. When the reasoner receives an INSERT query, it only runs its inference algorithm on the explicit facts that have not been inserted before and simply validates the others. Processing DELETE queries only consists in invalidating the corresponding facts instead of removing them from the knowledge base (as done in IR). At SELECT queries, the reasoner queries the knowledge base and filters the resulting facts according to their validity.

The speed of this process relies on the principle of storing explicit fact validity in memory and obtaining implicit fact validity from simple logic operations on these values: an implicit fact can originate from the disjunction of several sets of facts (explicit or implicit) that match the antecedent pattern of a same rule or from multiple rules, and rule antecedents

are defined as conjunctions.

Finally, we introduce a *fact forgetting* mechanism to avoid KB inflation. In this mechanism, each fact is tagged with the timestamp of its latest validity update (*i.e.* the last change on its validity tag), so that the oldest invalid facts are asynchronously removed when the KB size reaches a given threshold. As this mechanism only removes invalid facts (*i.e.* facts that would be deleted in the regular IR maintenance), it does not affect the inference correctness.

In the smart home case study presented in the previous section, when Julia switches the phone GPS off, the application "loses" its location and asks the reasoner to remove E6. But the reasoner only invalidates this fact. Then, the application sends a SELECT query on I8. The reasoner performs a simple logical operation (explained below) on I8 causes (E4, E6) that assesses that I8 is invalid, as E6 is invalid. It then does not return I8. When the phone connects to the house network, the application creates E7. The reasoner attaches it as alternative derivation of I7. At the next SELECT query, it deduces that I8 is valid as I7 is valid, and sends it back to the application. The next subsections detail the main elements of our Tag-Based (TB) approach: validity assessement, fact tagging, reasoning process and selection tasks.

Fact validity

Let F_e and F_i be respectively the sets of explicit and implicit facts in the KB. We propose to keep all facts (explicit and implicit) in the KB until the reasoning process is stopped or the fact forgetting mechanism triggered, and to assess their validity instead of removing them at DELETE queries. To do so, we tag explicit facts with a $valid$ boolean indicator: $f_e.valid \in \mathbb{B}, f_e \in F_e$, which is set to *true* on insertion and *false* on deletion. We tag implicit facts with a $derivedFrom$ indicator that represents the minimal set of disjoint *causes* of an implicit fact. We define a cause C as a set of explicit facts that must all be valid to validate an implicit fact[13]: $C = \{f_{ei}\}, i \in \mathbb{N}, f_{ei} \in F_e$. Hence, $\forall f_i \in F_i, f_i.derivedFrom = \{C_i\}, i \in \mathbb{N} / \forall x, y, 0 \leq x < y \leq i, C_x \nsubseteq C_y, C_y \nsubseteq C_x$. We provide an $isValid()$ function that checks the validity of an implicit fact using its $derivedFrom$ tag. It evaluates the disjunction between the tag elements and for each element, the conjunction between the $valid$ tags of the explicit facts referenced in this element: $isValid(f_i) = \vee_{C_i}\{\wedge_{f_{ej}}\{f_{ej}.valid\}\}, C_i \in f_i.derivedFrom, f_{ej} \in C_i$.

[13]To avoid recursion while assessing implicit fact validity, algorithm 3 (see below) only stores explicit facts in causes.

Implicit fact tagging

ALGORITHM 2: Implicit fact tagging

Data: A newly inferred implicit fact f and the sets F_e (resp. F_i) of explicit (resp. implicit) facts it has been derived from.
Result: f carries a $derivedFrom$ tag composed of its explicit causes only.

```
1  if F_i = ∅ then
2  |  f.derivedFrom ← {F_e}
3  |  return f
4  C' ← F_i.first().derivedFrom
5  foreach fi ∈ (F_i \ F_i.first()) do
6  |  tmp ← ∅
7  |  foreach c ∈ C' do
8  |  |  foreach δ ∈ fi.derivedFrom do
9  |  |  |  tmp ← tmp ∪ {c ∪ δ}
10 |  C' ← tmp
11 if F_e = ∅ then
12 |  f.derivedFrom ← C'
13 |  return f
14 C ← ∅
15 foreach c ∈ C' do
16 |  C ← C ∪ {c ∪ F_e}
17 f.derivedFrom ← C
18 return f
```

Each time an implicit fact is derived, Algorithm 2 is applied to set its $derivedFrom$ tag. Let F_e (resp. F_i) be the sets of explicit (resp. implicit) facts a newly inferred fact f have been derived from. In the general case, the algorithm builds the set C' of resolved explicit causes by replacing implicit facts with their explicit causes[14] and deduplicating these causes (lines 4-10). It then builds the set C of explicit causes by distributing the initial set of explicit facts F_e into C' (lines 14-16). It finally sets C as $derivedFrom$ tag of f – now tagged with a set of disjoint explicit causes – and terminates (lines 17-18).

Two optimizations allow avoiding unnecessary loops: (i) if no implicit fact is present (*i.e.* F_i is empty), the algorithm sets $f.derivedFrom$ to F_e and terminates at line 3; (ii) if no explicit fact is present (*i.e.* F_e is empty), the algorithm sets $f.derivedFrom$ to C' and terminates at line 13.

[14]These implicit facts have been inferred from prior evaluation loops; hence their derivedFrom tag is already set and stricly composed of explicit facts.

Enabling tagging in reasoning

ALGORITHM 3: Tag-based KB update

Data: Rule set R, explicit facts F_e, implicit facts F_i, added explicit facts F_e^+, removed explicit facts F_e^-

Result: The KB updates correspond to the changes caused by F_e^+ and F_e^- wrt. R.

```
1  F_i^+ ← ∅
2  foreach fact ∈ F_e do
3      if fact ∈ F_e^- then fact.valid ← false
4      else if fact ∈ F_e^+ then
5          fact.valid ← true
6          F_e^+ ← F_e^+ \ {fact}
7  if F_e^+ ≠ ∅ then
8      F_e ← F_e ∪ F_e^+
9      do
10         F_i ← F_i ∪ F_i^+
11         R_kb ← restrictRuleSet(R, F_e ∪ F_i)
12         F_i^+ ← evaluateRuleSet(R_kb, F_e ∪ F_i)
13         F_i^+ ← combine(F_i, F_i^+)
14     while F_i^+ ⊄ F_i
15 return F_e ∪ F_i
```

The KB update algorithm (Algorithm 3) performs the reasoning process while answering INSERT and DELETE queries. Let R be the set of rules and F_e^+ and F_e^- the sets of explicit facts to be respectively added and removed (from the query). It first invalidates the explicit facts to be deleted, and validates those to be inserted (lines 2-6), so that F_e^+ only contains new facts to be evaluated at line 7. Hence, for all deletions and re-insertions, our approach allows to skip the whole evaluation loop (lines 9-13).

For the remanining facts in F_e^+, the evaluation loop works very similarly to IR [Motik et al., 2012]: the reasoner restricts R to the set R_{kb} of rules that match at least one cause in the updated KB ($F_e \cup F_i$) in $restrictRuleSet()$ (line 11), evaluates R_{kb} over $F_e \cup F_i$ ($evaluateRuleSet()$, line 12) and loops as long as new implicit facts are inferred. TB reasoning requires two additional steps: (i) at each iteration, the $combine()$ function deduplicates identical facts by concatenating their causes and removes unnecessary causes[15] (line 13), and (ii) when new implicit facts have been inferred (*i.e.* in the innermost loop of the $evaluateRuleSet()$ function), it calls Algorithm 2 to set the fact causes in their $derivedFrom$

[15]For instance, if f_i can be caused by both $f_{e1} \wedge f_{e2}$ and $f_{e1} \wedge f_{e2} \wedge f_{e3}$, only the former conjunction is stored as a cause.

tags (line 12). After the evaluation loop, the algorithm terminates and returns $F_e \cup F_i$, that reflects the KB changes, namely the updates in F_e^+ and F_e^- and the $valid$ and $derivedFrom$ tags of facts.

Fact-filtering

ALGORITHM 4: Tag-based KB filtering

Data: F a set contaning explicit and implicit facts from a SELECT query answer.

Result: The returned set is composed of valid facts.

1 $V \leftarrow \emptyset$

2 **foreach** $f \in F$ **do**

3 | **if** $((hasTag(f, valid)$ ***and*** $f.valid)$ ***or*** $(isValid(f))$ **then** $V \leftarrow V \cup \{f\}$

4 **return** V

The fact-filtering algorithm (Algorithm 4) is applied after SELECT queries to filter out valid facts. As these queries are time-critical for the application and this step represents an overhead compared to other approaches, this algorithm must be kept fast. Let F be a query result set of facts. The algorithm performs a single loop over F to construct – and return – the set of valid facts V of F: $V = \{f_e \in F \cap F_e / F_e.valid = true\} \cup \{f_i \in F \cap F_i / isValid(F_e) = true\}$[16].

5.4.3 Complexity analysis and discussion

Inserting a fact in a KB requires performing a transitive closure of the graph. The number of times a rule-based reasoner executes the rule evaluation loop depends on the data and on the expressivity of the used DL[17]. As tag-based is a maintenance approach, it does not aim at reducing the whole reasoning process complexity, but at performing it as rarely as possible. Our goal hereafter is to quantify the difference between TB maintenance and regular IR for each reasoning task.

[16] For the sake of understandability, the algorithm comprises an $hasTag()$ function to filter explicit from implicit facts. This function is not implemented in practice.

[17] It is said to be EXPTIME-complete in $|KB|$ with $\mathcal{SHIQ}$ [Hustadt et al., 2005], and even untractable with other DLs [Donini, 2003] for tasks such as satisfiability or subsumption.

Algorithm analysis

Fact tagging. The algorithm loops over all facts in F_i, then over their causes, and over the causes of the fact to tag. Then it merges both types of causes in another loop. Its complexity is $O(n_f.{n_c}^3)$, where n_c represents the number of causes in a fact (bounded by $|C| = \max_{j\in\{1,\ldots,|F_i|\}} |C_{f_{ij}}|$), and n_f numbers of facts (bounded by $|KB|$). It is then $\mathcal{O}(|KB| \times |C|^3)$.

KB update. In the case of fact deletion and/or re-insertion (*i.e.* the scenario we target), the algorithm does not step through the evaluation loop; its complexity is thus limited to that of the first loop over F_e and is $\mathcal{O}(|KB|)$. The main evaluation loop (at first insertion) calls in turn at each iteration:

- $restrictRuleSet(R, F)$ checks $|R|$ rules over $|F|$ facts. Its complexity is $O(n_r.n_f)$, with n_r a number of rules (bounded by $|R|$), that is $\mathcal{O}(|R| \times |KB|)$.
- $evaluateRuleSet()$ loops over a set of rules ($n_r \in R_{kb}$), the antecedents of each rule ($n_a \in A_{r_i}$, where $r_i \in R$), and the updated KB ($F_e \cup F_i$). Its complexity is therefore $O(n_r.n_a.n_f)$, which is $\mathcal{O}(|R| \times |KB|)$ while reasoning in IR[18]. In TB maintenance, the cost of the tagging algorithm must be added: $\mathcal{O}(|R| \times |KB|^2 \times |C|^4)$.
- $combine(F1, F2)$ loops over the causes of two sets of facts. $F1 = F_i \in KB$ is the set of previously inferred facts. $F2 = F_i^+$ is the set of newly inferred facts, in which facts only have one cause. As long as each rule antecedent evaluation performed in $evaluateRuleSet()$ produced a unique (distinct) fact, the maximum complexity of $combine()$ is reached. Hence, we can factorize with $evaluateRuleSet()$ complexity, so that the additional tagging cost in TB maintenance is $\mathcal{O}(|R| \times |KB|^2 \times |C|^4 \times |F1|)$.

Hence, for both IR and TB algorithms, the complexity of a single loop is $Poly()$ in $|R|$ and $|KB|$ (with a higher degree for TB maintenance), but it is also $Poly()$ in $|C|$ for TB maintenance. As expected, TB maintenance has an additional cost at first insertion, discussed below.

Fact filtering. The algorithm loops over query result facts ($O(n_f)$), and calls $isValid()$ at each iteration. $isValid(f)$ loops over the causes of an implicit fact and over explicit facts in these causes, with an internal complexity of $O(n_c.n_f)$. The fact-filtering algorithm complexity is therefore $\mathcal{O}(|C| \times |KB|^2)$.

The above analysis shows that the time complexity of both the reasoning evaluation

[18] We do not consider the number of rule antecedent as a significant parameter. Therefore, n_a is bounded by constant $max_{r\in R}\{|A_r|\}$ and taken out of the Big-O expression.

loop at first insertion and fact filtering algorithms is $Poly()$ in $|KB|$ and $|C|$[19]. In the same way, we can calculate that space complexity is $\mathcal{O}(|KB| \times |C|)$, as it requires to store the causes of implicit facts. As causes represent sets of disjoint conjunctions of explicit facts, the set of all possible causes contains $\sum_{i=1}^{|F_e|} \binom{|Fe|}{i}$ non-permuted combinations. As the set of causes of an implicit fact does not contain redundant causes, $|C|$ is bounded by the largest term of this sum, which is $\binom{|F_e|}{\frac{|F_e|}{2}} = \frac{|F_e|!}{(\frac{|F_e|}{2}!)^2}$. Even though this function grows slower than $e^{|f_e|}$, it can still grow rapidly with $|F_e|$. This can be considered as the cost of "storing" the reasoning complexity in causes to avoid recomputing it at deletions and rederivations. We hereafter propose a method to ensure this cost stay limited, and our evaluations show that even in a common use case, it keeps affordable.

Efficient usage of TB maintenance

In order to limit both the number of causes and the inflation of the knowledge base size (which is higher in TB maintenance as explicit facts are not removed), we suggest to limit the number of explicit facts. Our underlying hypothesis is that our reasoner targets Web applications that can run on small devices such as smartphones, which is the case for most WoT applications. It is not intended for storing application history but to receive facts that will trigger rules at the application level. In our scenario, the phone GPS coordinates are raw numeric values. They are not inserted "as is" in the reasoner but are transformed into facts that fit the application requirements (the phone is located in the house neighborhood).

In these conditions, client-side reasoning can save both WoT application developers' time while constructing their datasets (by using regular Semantic Web modeling tools), and bandwidth (by leaving saturation and decision processes up to the clients). Using a known set of explicit tags, TB reasoning allows application designers to first-insert and delete these facts at bootstrap or asynchronously, to pre-compute the tagging step and ensure performance at runtime. Additionaly, the fact-forgetting method (i.e. actually delete old facts) can significantly reduce the size of the KB to deal with worst cases. Another solution, which we did not test yet, would be to discretize frequents causes patterns; i.e. replace frequent conjunctions of facts referenced as a cause by a single fact.

[19] We do not take into account the complexity in $|R|$, as we suppose it does not vary during application runtime.

5.4.4 Evaluation

We evaluate the tag-based maintenance algorithm by comparing it to Motik et al.'s incremental reasoning algorithm itself based on DRed. We chose to compare those algorithms on the same implementation rather than comparing them on different applicative solutions: our goal is not to provide the fastest reasoner, but rather to offer an optimal solution for maintaining datasets incrementally in Web browsers using JavaScript. We evaluate these algorithms for different tasks: ontology classification and initial insertion, insertion of new triples, deletion, insertion of known triples and selection. The three latter represent the cycles encountered in WoT applications such as the one we illustrated in the smart home case study. We run each algorithm in Google Chrome v.54.0.2840.99, on a Lenovo Ideapad 700-15ISK (Intel Core i5-6300HQ @2.3GHz with 4GB RAM).

Datasets and rules

We generated 3 datasets (O1, O2 and O3) using the Lehigh University Benchmark (LUBM) [Guo et al., 2005]. They are based on the Univ-Bench Ontology[20] schema, which has $\mathcal{ALEHI}+$ expressivity and contains 36 SubClassOf, 6 EquivalentClasses, 5 SubObjectPropertyOf, 1 TransitiveObjectProperty, 21 ObjectPropertyDomain, 18 ObjectPropertyRange and 4 DataPropertyDomain axioms, as well as 43 Class Assertions, 25 Object Property Assertions and 7 Data Property Assertions[21]. O1, O2 and O3 contain respectively 8824, 7394, and 5759 triples, and correspond to the initial insertion.

The evaluation uses subset of OWL 2 RL rules below (Listing 5.8). Their name (scm-sco, cax-sco...) refer the rules presented in [Motik et al., 2009] (section 4.3).

R_{sub} = {scm−sco, cax−sco, scm−spo, prp−spo1} $(Subsumption)$
$R_{trans-inv}$ = {prp−trp, prp−inv1, prp−inv2} $(Transitivity/Inverse)$
R_{equiv} = {cax−eqc1, cax−eqc2, prp−eqp1, prp−eqp2} $(Equivalence)$
R_{equal} = {eq−rep−s, eq−rep−p, eq−rep−o, eq−trans} $(SameAs)$
$R_{all} = R_{sub} \cup R_{trans-inv} \cup R_{equiv} \cup R_{equal}$

Listing 5.8: HyLAR's sets of rules

[20] http://swat.cse.lehigh.edu/onto/univ-bench.owl
[21] Those metrics are provided by Protégé 5.0.0 – `http://protege.stanford.edu/`

R_{sub} provides subsumption inferences on classes, properties, as well as their instances. $R_{trans-inv}$ provides inferences on both transitive and inverse properties. R_{equiv} provides inferences on instances that belong to equivalent classes. R_{equal} provides inferences for equality relations between subjects, predicates and objects, as well as transitive equality, through the `owl:sameAs` property.

Practical evaluation

We ran 5 evaluation tasks: classification and initial dataset insertion (CLASSIF+INIT), insertion, deletion, re-insertion and selection for both IR and TB algorithms[22]. Inserted and deleted data have also been generated with LUBM and contain 500 triples. Each task applies the five rule sets described above. The results are depicted in Figure 5.4. Processing times for each task are written in milliseconds. This table also shows the time difference betweeen IR and TB (Diff.), as well as the *performance* of TB (Perf.), i.e. the percentage of time gained if using TB instead of IR, for a particular task. The "10 CYCLES" column sums the results for (i) classification and initial insertion, (ii) insertion, and ten cycles of (iii) deletion and (iv) re-insertion. Such cycles correspond to applicative scenarios such as the one described in Section 5.4.1.

Classification and initial insertion. As expected, TB maintenance does not outperform IR for these tasks, as it adds the cost of tagging facts. Although the number of rules and the size of the schema influence these results, RL profile reasoners do not target large classification tasks (an OWL-EL reasoner would probably be more suitable). Moreover, in Web applications, the classification and initial dataset insertion usually involve shared data and their results can therefore be computed on a server and cached for all clients. As tagging time is related to the number of triggered rules and their possible recursivity, its overhead is reduced for already closed datasets.

First insertion. Again, it takes longer for TB maintenance to perform a first insertion due to the additional tagging step. In this case, it should be noted that the instance number and the expressivity influences the results, as all facts - including instances - have to be tagged while firstly inserted in the ontology. Results show that this overhead varies according to the number of activated rules (*i.e.* the number and variety of OWL constructs).

Deletion. As expected, deletion is much (more than 50%) faster on TB maintenance, as IR over-deletion is replaced with a single iteration over the KB. Instance numbers significantly

[22]The TB fact forgetting algorithm that prevents KB inflation has not been evaluated, as it is triggered and performed asynchronously, when the application is idle.

IR	CLASSIF+INIT			INSERT			DELETE			RE-INSERT			SELECT			10 CYCLES		
	O1	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*
Rsub	2 593	4 477	3 565	982	1 325	1 181	2 124	2 953	2 372	942	1 287	1 219	21	20	14	34 235	48 202	40 656
Rtrans-inv	2 224	4 129	3 204	1 139	969	767	1 561	2 155	1 592	1 110	924	743	21	15	15	30 073	35 888	27 321
Requiv	1 760	1 526	1 043	374	343	287	710	836	724	323	344	338	24	18	18	12 464	13 669	11 950
Requal	2 218	3 565	2 734	336	465	389	638	849	743	291	385	383	29	25	21	11 844	16 370	14 383
Rall	3 837	5 716	4 505	2 967	2 282	1 944	6 202	5 308	3 987	2 737	2 282	1 988	28	25	18	96 194	83 898	66 199
TB	CLASSIF+INIT			INSERT			DELETE			RE-INSERT			SELECT			10 CYCLES		
	O1	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*	*O1*	*O2*	*O3*
Rsub	5 121	4 650	3 617	1 313	1 333	1 179	827	738	665	798	656	675	39	30	25	22 684	19 923	18 196
Diff.	-2 528	-173	-52	-331	-8	2	1 297	2 215	1 707	144	631	544	-18	-10	-11	11 551	28 279	22 460
Perf. (%)	-97	-4	-1	-34	-1		**61**	**75**	**72**	**15**	**49**	**45**	-86	-50	-79	**34**	**59**	**55**
Rtrans-inv	4 302	4 222	3 305	1 156	1 028	802	674	589	533	676	649	558	41	28	27	18 958	17 630	15 017
Diff.	-2 078	-93	-101	-17	-59	-35	887	1 566	1 059	434	275	185	-20	-13	-12	11 115	18 258	12 304
Perf. (%)	-93	-2	-3	-1	-6	-5	**57**	**73**	**67**	**39**	**30**	**25**	-95	-87	-80	**37**	**51**	**45**
Requiv	1 782	1 597	1 048	375	344	360	154	187	222	180	215	247	41	32	30	5 497	5 961	6 098
Diff.	-22	-71	-5	-1	-1	-73	556	649	502	143	129	91	-17	-14	-12	6 967	7 708	5 852
Perf. (%)	-1	-5				-25	**78**	**78**	**69**	**44**	**38**	**27**	-71	-78	-67	**56**	**56**	**49**
Requal	4 050	3 812	2 877	408	480	454	160	164	241	175	168	267	46	29	33	7 808	7 612	8 411
Diff.	-1 832	-247	-143	-72	-15	-65	478	685	502	116	217	116	-17	-4	-12	4 036	8 758	5 972
Perf. (%)	-83	-7	-5	-21	-3	-17	**75**	**81**	**68**	**40**	**56**	**30**	-59	-16	-57	**34**	**54**	**42**
Rall	7 510	6 118	4 863	3 519	2 529	2 001	1 652	1 346	1 074	1 541	1 671	1 480	45	36	28	42 959	38 817	32 404
Diff.	-3 673	-402	-358	-552	-247	-57	4 550	3 962	2 913	1 196	611	508	-17	-11	-10	53 235	45 081	33 795
Perf. (%)	-96	-7	-8	-19	-11	-3	**73**	**75**	**73**	**44**	**27**	**26**	-61	-44	-56	**55**	**54**	**51**

Figure 5.4: Evaluation results

affect processing times in both algorithms.

Re-insertion. Re-inserting the same triples is also much faster on TB maintenance, as re-inserted triples do not have to be re-evaluated. TB performs particularly well with high expressivity (such as transitivity + inverse and equivalence rules). As in the deletion process, the number of instances is the most influential parameter.

Selection. Selections in IR are straightforward and give stable processing times. They are slower on TB maintenance as the algorithm checks the validity of each fact returned by the KB. With respect to IR, TB maintenance could then significantly impact SELECT queries with high numbers of triples or highly interrelated datasets. However, SELECT operations are much faster than the previous ones. Hence, despite its important value in percentage, this overhead sounds acceptable in terms of absolute times (about twenty milliseconds), as it only corresponds to a couple of frame rates of the most performant Web applications.

Multiple cycles. Here, the interest of the TB maintenance approach is clearly noticeable: the initial tagging cost at classification and first insertion is re-gained along deletions/re-insertion cycles. Due space limitation, we only show figures for 10 cycles. However, for all situations in the evaluation, TB maintenance outperforms IR from 4 cycles, and its gain in total computing time exceeds 50% for 100 cycles.

Evaluation synthesis

This evaluation shows that TB maintenance outperforms the regular IR algorithm when data are being cyclicly deleted and reinserted into the reasoner, despite its cost on first insertions and selections. For applications going through hundreds of such cycles, TB maintenance can represent a massive performance improvement. This approach can particularly fit applications that rely on constantly changing data. For instance, context-aware applications that take adaptation decisions according to environmental (sensor) data can now integrate their own Web-based reasoner, process these data in Web clients and behave autonomously.

5.4.5 Implementation

The implementation we propose is HyLAR[23]. HyLAR – which stands for Hybrid Location-Agnostic Reasoning – consists in a partial OWL 2 RL reasoner that enables rule-based incremental reasoning, with modularized reasoning steps to provide the execution of both parsing and classification, as well as query answering steps either on the client side or on the server side. HyLAR is composed of the following modules depicted in Figure 5.5.

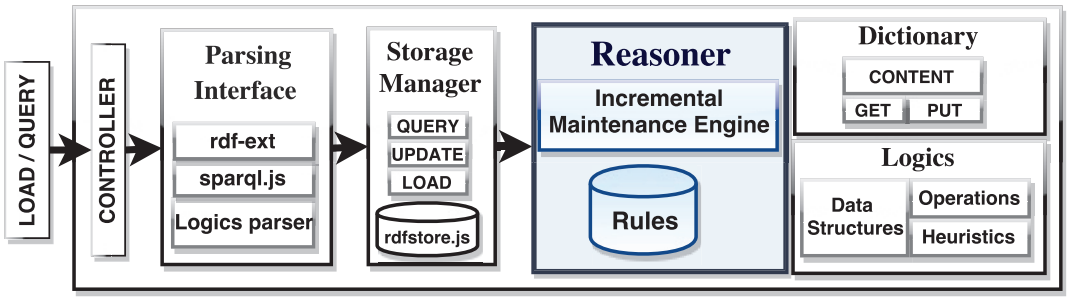


Figure 5.5: HyLAR global architecture.

- **Controller:** handles ontology loading (parsing and classification) and querying, providing inferences on INSERT and DELETE queries with incremental maintenance.

[23]Repository: https://github.com/ucbl/HyLAR-Framework

- **Parsing Interface:** integrates rdf-ext RDF/XML parser and SPARQL.js[24] library, and is able to convert triples (as described in RDF Interfaces 1.0[25]) into turtle (for direct triplestore insertion/deletion) and facts (for reasoning).
- **Storage Manager:** based on rdfstore.js[26] [Hernández, 2012], this module handles ontology loading, updates and queries.
- **Reasoner:** holds and processes rules using a pattern matching mechanism. It includes incremental maintenance algorithms (both IR and TB) and is able to process the set of rules enumerated in Section 5.4.4, Listing 5.8.
- **Dictionary:** indexes all triples registered in the store and their representations as facts in the KB; this accelerates validity checking at selection time.
- **Logics:** contains first-order logic operations: fact instantiation, fact set merging, and rule restriction.

HyLAR can both run on server (Node.js[27] 5.1.1) and client (using browserify[28]) sides. Thus, it can be integrated into a Web application, as well as a framework designed to optimize the reasoning process location. HyLAR benefits from JavaScript's asynchronous patterns (promises, callbacks). On the client side, the reasoner modules can be embedded either in a regular angular service, or in a Web worker. On servers, event emitters (on Node.js), allow for background task processing. HyLAR reasoning steps are packaged as Node.js modules and AngularJS[29] services. They are queried by an independent angular service using an asynchronous promise pattern, so that the main service is totally agnostic about the location of the reasoning modules.

5.5 Conclusion

We presented a solution to lighten the server load and reduce the network congestion by deferring part of the reasoning tasks on the client side. However, choosing a location for

[24] https://github.com/RubenVerborgh/SPARQL.js
[25] http://www.w3.org/TR/rdf-interfaces/#triples
[26] Rdfstore.js is a graph store implementation with support for SPARQL 1.0 and 1.1/Update. Available at https://github.com/antoniogarrote/rdfstore-js
[27] https://nodejs.org
[28] http://browserify.org/
[29] http://www.angularjs.org

each reasoning step is a complex task as it depends on the context. Thus, there is no optimal reasoning configuration that works for any case: the architectural setup and contextual conditions must be taken into consideration to provide the suitable adaptation. The difficulty is therefore to identify and choose the actual contextual parameters that influence the performance of each reasoning task, in order to build the reasoning context mode accordingly.

Our study on the contextual parameters influence exhibited different types of contextual information that affect processing times. An evaluation of the reasoning adaptation has been conducted with several parameters (ping, battery level, ontology size) actually affecting Web applications. The evaluation validated both the time processing efficiency of our adaptation solution and its applicability for the reasoning task, which is a crucial concern in the WoT as devices may not have sufficient capabilities to process each reasoning task.

We addressed the issue of overdeleting and re-derivating facts in DRed Incremental Reasoning (IR). We proposed a Tag-Based (TB) incremental maintenance approach that solve these issues by tagging facts with respect to their provenance and validity. We presented three algorithms: implicit fact tagging, tag-based update and fact-filtering. The first two are executed to update the KB and the third to filter valid facts out of SELECT query results. Our approach targets Web applications that face multiple cycles of data deletions and reinsertions, such as many WoT applications. Evaluation results showed that the cost of TB maintenance is slightly higher for first insertions and for selections, but significantly outperforms IR at deletion and reinsertion. This cost is also re-gained within a few cycles. We conducted a complexity analysis of TB maintenance algorithms and showed that, to the initial cost of fact tagging at first insertion and validity assessment at selection, the complexity of re-insertions and deletion operations drops to linear, regardless of the reasoning conditions.

The implementation of both IR and TB algorithms in HyLAR can stimulate the adoption of semantic technologies not only for the WoT but also in the Web community. It and is available as server-side (Node.js) or client-side (Bower) packages, this applies on HyLAR reasoning location adaptation mechanism.

Chapter 6

General Conclusion

The Web of Things (WoT) has been designed to unify the way connected objects interact, through Web technologies and standards (REST, the Semantic Web, hypermedia, the programmable Web, etc.). WoT applications require – as for most IoT applications – to adapt to the context, *i.e.* to react accordingly to changes in their environment, to comply with quality of service, privacy, and security concerns, etc. In this thesis, we explored how the Semantic Web can provide such adaptation through proven, standard semantic reasoning techniques. We provided a solution at the junction between WoT applications, the Semantic Web and adaptive systems that was lacking.

Summary of our contributions. In this thesis, we proposed a solution to allow multi-purpose contextual adaptation in the WoT, *i.e.* to provide adaptation for several concerns. This solution 1) allows designing standard, interoperable and reusable context models, 2) provides generic and cross-domain contextual adaptation, 3) optimizes the reasoning process that provides such adaptation. We presented a *context meta-model* that allows building semantically-annotated multi-purpose context models. We detailed our *context lifecycle*, which joins static (expert knowledge, technical documentations) and dynamic (sensors, Web services) information to build the context model, along with a *multi-purpose adaptation workflow* that relies on semantic reasoning to update context information and select adaptation possibilities at runtime for each purpose. We presented our *meta-adaptation rule engine*, which generates adaptation rules and scores adaptation possibilities at design time, to ensure optimal adaptation decisions and reduce the adaptive WoT application design effort. The rest of our contributions tackles Web reasoning performance issues which have an impact on the contextual adaptation process in WoT applications. Firstly HyLAR, a *hy-*

brid location-agnostic architecture, locates parts of the reasoning tasks either on devices or on the cloud depending on the context. Secondly, a *tag-based maintenance* mechanism reduces processing time of both data deletions and re-insertions – which are common situations in most WoT applications – in comparison to state-of-the-art incremental maintenance algorithms.

Open questions and perspectives. The work we presented in this thesis opens various research questions, related to the Semantic Web, the adaptive systems and the reasoning fields. We identify the following perspectives.

- Currently, transformation rules infer contextual instances without providing metadata about the quality of the information. Hence, quality of context (QoC) – as defined in [Buchholz et al., 2003] and built as part of the INCOME project [Arcangeli et al., 2012] – could be integrated into the solution to improve the relevance of the adaptation decision process. The choice of attaching this metadata either on the contextual instance itself or on the adaptation possibility issued from this instance would therefore be an open question.
- We proposed an adaptive solution design technique similar to agile development methods; the next step would be to evaluate models and solutions through a designer-centered vision. Indeed, our contributions target WoT applications designers instead of application end-users.
- To capitalize on the design of transformation rules (which are currently written by hand), the solution may rely on Sensor-based Linked Open Rules (S-LOR). This may also provide a further extension of S-LOR for adaptation rules generated from the meta-adaptation rule engine, reducing even more the time and effort from designers when building their solution.
- We envision many improvements to HyLAR, to provide distributed reasoning among several clients. First, we would look into partitioning knowledge bases into named graphs. Theses partitions would isolate data from the other they do not depend on. Hence, the reasoner would apply on reduced and separated graphs, reducing the processing times of both updates and selections. Second, we would provide caching and replication techniques using several Web clients, to capitalize on the reasoning process. For instance, if a client A has processed an ontology $\mathcal{O}$, HyLAR would cache its saturated knowledge base and replicate it on other clients that need to process $\mathcal{O}$; the same mechanism would be applicable to update and select SPARQL queries.

- Tag-based maintenance space complexity improvement is also a perspective. Open questions include the design of fact-forgetting mechanisms using pattern detection for the discretization of fact sets, or deferred caching using distributed reasoning architectures. The storage method may also be improved, by storing facts as bit vectors, and by removing duplicated facts from same $derivedFrom$ tags.

Concluding thoughts. Semantic Web technologies are usually seen as complex technologies for use in the current industry or in commercial applications. Nevertheless, we believe that the WoT is an opportunity to provide use-cases for Semantic Web technologies. This thesis allowed us to highlight the benefits of the Semantic Web to the WoT, as it provides linked, interoperable and smart applications and features. We support these views by pushing advanced adaptation techniques to the Web and to WoT applications using semantics-based, ontological reasoning methods. Our work on contextual reasoning allowed us to study reasoning not only on WoT applications, but also for the Web in general. We explored the possibility to reason on the client side, which gave us the opportunity to bring the good practices of the Web to the Semantic Web.

In order to keep WoT applications standard, interoperable and reusable, we believe that both designers, researchers and the industry should promote linked, structured, and – above all – open data.

Appendix A

ASAWoO Avatar Architecture

Some components of the avatar architecture (Fig A.1) are dedicated to thing control and others implement the autonomous, self-adaptive and collaborative behavior of avatars. The physical setup is decoupled from its logical architecture: an avatar can dynamically adapt the distribution of its components to different locations (see below) to improve their efficiency. We grouped the avatar components in 8 functional modules.

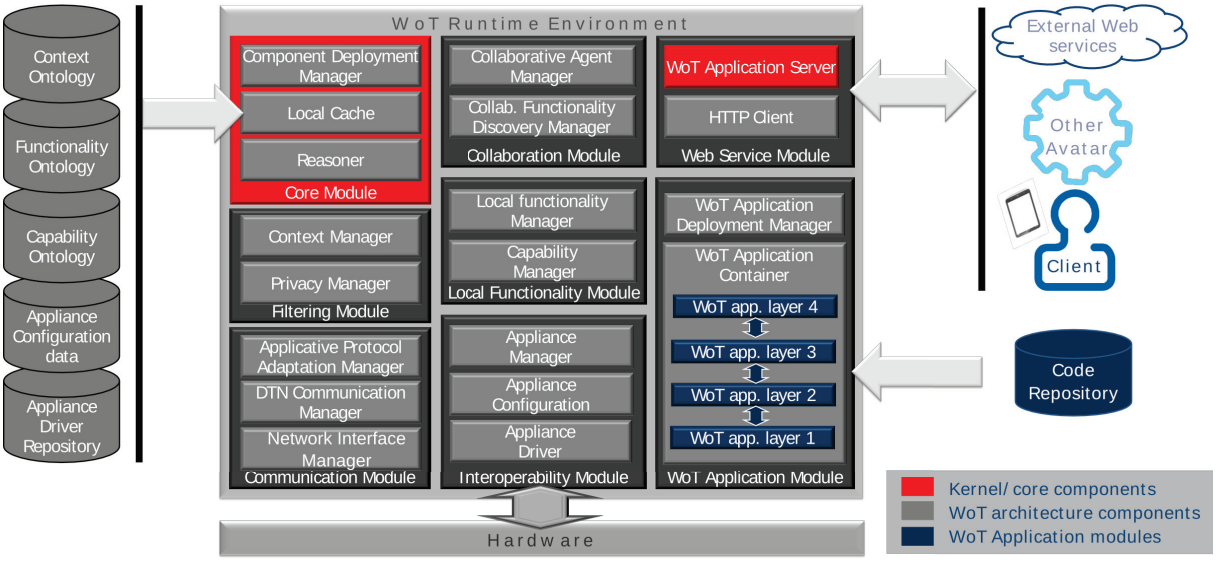


Figure A.1: The avatar architecture

- The **Core Module** includes components that are used in several steps of the avatar life-

cycle. The component deployment manager defines which avatar components will be instantiated wrt. the thing capabilities, and where[1]. Each avatar embeds a Reasoner, used by other components to process semantic information pertaining on the capabilities, functionalities and context. So is the Local Cache, that stores semantic information from diverse sources (thing, repositories, external context) and reflects the current state of the avatar. In particular, the cache loads concepts from the semantic repositories, in order to make them available to other modules through the reasoner. This module is essential to address the multiple concerns targeted by the application through the avatar, while avoiding allocating unnecessary resources. As such, it participates in addressing most of the requirements.

- The **Interoperability module** provides the other avatar modules with a uniform interface to interact with the thing it is attached to. This interface consists of a set of *capabilities* that represent the thing API. It loads drivers from a platform repository and uses them to identify the communication schemes understood by the thing; eventually, it uploads onto the thing the appropriate configuration.
- The **Filtering module** restricts functionality exposition and data exchanges. If, for privacy or security reason, some functionalities should not be achieved by the avatar, they will be filtered by the Privacy manager.
- The **Communication module** ensures reliable communication with the thing. It selects the appropriate network interface (Ethernet, Wi-Fi, Zigbee, etc.) and protocols (CoAP, HTTP, etc.) according to communication purposes and performance needs (throughput / energy consumption). It also supports connectivity disruptions.
- The **Web service module** allows avatars to communicate with other avatars and with the external world wrt. Web standards. By this means, avatars can: interact with the WoT platform to query repositories, respond to client requests regarding the functionalities they expose as RESTful resources, exchange data with other avatars to achieve collaborative functionalities and query external Web services to enrich their own data.
- The **Local Functionality module** handles high-level *functionalities* achievable using the thing capabilities[2]. It relies on semantic technologies to map the thing layer (capabilities) with the application layer (functionalities) in a declarative and loosely coupled manner, ensuring application interoperability with various things [Mrissa et al.,

[1] Avatar components can be located on the thing if it has enough computing capabilities or for time-constrained code modules, on the gateway for processes that involve inter-avatar communication, or on the cloud for calculation-intensive processes. This way, application components that model different cyber-physical systems (CPS) aspects and address different application concerns can be executed at an optimal location.

[2] Functionalities provide user-understandable compositions of capabilities. For instance, a user will prefer to tell a robot to move to another part of the field, rather than to pilot each of its wheels individually.

2014a]. When the avatar is created, the CapabilityManager queries the Interoperability module for the thing capabilities and the platform capability ontology for their semantic descriptions. It is queried by the LocalFunctionalityManager, which also loads the descriptions of functionalities and uses the reasoner to infer the avatar local functionalities[3]. For each inferred functionality, the LocalFunctionalityManager queries the Context Manager to decide if it should be exposed to clients. Exposed functionalities are bound to a registry, so that users and other avatars can find them.

- The **Collaboration module** handles functionalities that require collaboration between several avatars. The CollaborativeFunctionalityDiscoveryManager queries the reasoner to identify, from the local functionalities, in which higher-level ones it could participate. Then, it queries the platform functionality directory to search for the locally missing functionalities. If such functionalities are available from other avatars, it calls the Collaborative Agent Manager, which handles negotiation with these other avatars.
- The **WoT Application module** provides and controls "WoT application containers" that execute code modules implementing the different aspects of a WoT application. Such containers can be replicated on the thing, on the gateway and on the cloud infrastructure thanks to the deployment manager, so that modules are executed on the appropriate location.

[3]Inference processing relies on the Capability and Functionality classes, and relationships between them, expressed in our own OWL (`http://www.w3.org/TR/owl2-overview/`) vocabulary. Individuals expressed in other vocabularies [Gyrard et al., 2015b] can be used and "rdf:typed" as capabilities or functionalities.

Appendix B

ASAWoO vocabulary

Listing B.1: ASAWoO vocabulary for functionality composition and implementation in JSON-LD.

```
{
  "@context": {
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "owl": "http://www.w3.org/2002/07/owl#",
    "asawoo": "http://liris.cnrs.fr/asawoo/vocab#"
  },
  "@graph": [
    {
      "@id": "asawoo:isImplementedBy",
      "@type": "owl:ObjectProperty",
      "rdfs:domain": {
        "@id": "asawoo:Functionality"
      },
      "rdfs:range": {
        "@id": "asawoo:Capability"
      }
    },
    {
      "@id": "asawoo:isComposedOf",
      "@type": [
        "owl:ObjectProperty",
        "owl:TransitiveProperty"
      ],
      "rdfs:domain": {
          "@id": "asawoo:Functionality"
      },
      "rdfs:range": {
          "@id": "asawoo:Functionality"
      }
    },
    {
      "@id": "asawoo:Functionality",
```

```
33          "@type": "owl:Class",
34          "rdfs:subClassOf": {
35              "@id": "http://www.w3.org/ns/sosa/Procedure"
36          }
37      },
38      {
39          "@id": "asawoo:Capability",
40          "@type": "owl:Class",
41          "rdfs:subClassOf": {
42              "@id": "http://www.w3.org/ns/sosa/Procedure"
43          }
44      }
45  ]
46 }
```

Bibliography

[Cop, 2010] (2010). The context-aware browser. *IEEE Intelligent Systems*, 25(1):38–47.

[Abowd et al., 1999] Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer.

[Alaya et al., 2012] Alaya, M. B., Matoussi, S., Monteil, T., and Drira, K. (2012). Autonomic computing system for self-management of machine-to-machine networks. In *Proceedings of the 2012 International Workshop on Self-aware Internet of Things*, Self-IoT '12, pages 25–30, New York, NY, USA. ACM.

[Alti et al., 2012] Alti, a., Roose, P., Saffidine, R., and Laborie, S. (2012). Towards an automatic adaptation of heterogeneous multimedia mobile applications. *Conference Internationale: Nouvelles Technologies de la Repartition - Colloque Francophone sur l'Ingenierie des Protocoles, NOTERE/CFIP 2012*.

[Andersson et al., 2008] Andersson, J., Ericsson, M., and Lowe, W. (2008). Automatic rule derivation for adaptive architectures. In *Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on*, pages 323–326. IEEE.

[Arcangeli et al., 2012] Arcangeli, J.-P., Bouzeghoub, A., Camps, V., Canut, M.-F., Chabridon, S., Conan, D., Desprats, T., Laborde, R., Lavinal, E., Leriche, S., et al. (2012). Income–multi-scale context management for the internet of things. In *International joint conference on ambient intelligence*, pages 338–347. Springer.

[Arias, 2008] Arias, M. (2008). Context-Based Personalization for Mobile Web Search. *Context*, pages 33–39.

[Athreya et al., 2013] Athreya, A., DeBruhl, B., and Tague, P. (2013). Designing for self-configuration and self-adaptation in the internet of things. In *First International Workshop on Internet of Things (C-IOT), 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2013)*, pages 585–592.

[Baheti and Gill, 2011] Baheti, R. and Gill, H. (2011). Cyber-physical systems. *The impact of control technology*, 12:161–166.

[Baladron et al., 2012] Baladron, C., Aguiar, J. M., Carro, B., Calavia, L., Cadenas, A., and Sanchez-Esguevillas, A. (2012). Framework for intelligent service adaptation to user's context in next generation networks. *IEEE Communications Magazine*, 50(3).

[Baldoni et al., 2004] Baldoni, M., Baroglio, C., Patti, V., and Torasso, L. (2004). Reasoning about learning object metadata for adapting scorm courseware. *Engineering the Adaptive Web., CS-Report*, pages 04–18.

[Barbier et al., 2015] Barbier, F., Cariou, E., Goaer, O. L., and Pierre, S. (2015). Software adaptation: Classification and a case study with state chart xml. *IEEE Software*, 32(5).

[Barnaghi and Presser, 2010] Barnaghi, P. and Presser, M. (2010). Publishing linked sensor data. In *Proceedings of the 3rd International Conference on Semantic Sensor Networks-Volume 668*, pages 1–16. CEUR-WS. org.

[Bass, 2007] Bass, L. (2007). *Software architecture in practice*. Pearson Education India.

[Bazire and Brézillon, 2005] Bazire, M. and Brézillon, P. (2005). Understanding context before using it. *Modeling and using context*.

[Becker et al., 2007] Becker, J., Delfmann, P., and Knackstedt, R. (2007). Adaptive reference modeling: integrating configurative and generic adaptation techniques for information models. In *Reference Modeling*, pages 27–58. Springer.

[Beisiegel et al., 2007] Beisiegel, M., Blohm, H., Booz, D., Dubray, J.-J., Interface21, A. C., Edwards, M., Ferguson, D., Mischkinsky, J., Nally, M., and Pavlik, G. (2007). Service component architecture. *Building systems using a Service Oriented Architecture. BEA, IBM, Interface21, IONA, Oracle, SAP, Siebel, Sybase, white paper, version*, 9.

[Bernardos et al., 2008] Bernardos, A. M., Tarrio, P., and Casar, J. R. (2008). A data fusion framework for context-aware mobile services. In *Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on*, pages 606–613. IEEE.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., Lassila, O., et al. (2001). The semantic web. *Scientific american*, 284(5):28–37.

[Bertini et al., 2006] Bertini, M., Cucchiara, R., Bimbo, A., and Prati, A. (2006). Semantic adaptation of sport videos with user-centred performance analysis. *IEEE Transactions on Multimedia*, 8(3):433–443.

[Boussadi et al., 2011] Boussadi, A., Bousquet, C., Sabatier, B., Caruba, T., Durieux, P., Degoulet, P., et al. (2011). A business rules design framework for a pharmaceutical validation and alert system. *Methods of information in medicine*, 50(1):36.

[Brézillon, 1999] Brézillon, P. (1999). Context in Artificial Intelligence: II. Key elements of contexts. *Computers and artificial intelligence*, pages 1–27.

[Brézillon, 2003] Brézillon, P. (2003). Representation of procedures and practices in contextual graphs. *The Knowledge Engineering Review*, pages 1–26.

[Brézillon and Pomerol, 1996] Brézillon, P. and Pomerol, J. (1996). Misuse and nonuse of knowledge-based systems: The past experiences revisited. *... Systems for Supporting Management Decisions*.

[Brézillon and Pomerol, 1999] Brézillon, P. and Pomerol, J. (1999). Contextual knowledge sharing and cooperation in intelligent assistant systems. *Le Travail Humain*, pages 1–33.

[Brickley et al., 2014] Brickley, D., Guha, R. V., and McBride, B. (2014). Rdf schema 1.1. *W3C recommendation*, 25:2004–2014.

[Buchholz et al., 2003] Buchholz, T., Küpper, A., and Schiffers, M. (2003). Quality of context information: What it is and why wee need it. In *Proceedings of the 10th International Workshop of the HP OpenView University Association (HPOVUA'01)*, volume 2003.

[Bucur et al., 2005] Bucur, O., Beaune, P., and Boissier, O. (2005). Representing context in an agent architecture for context-based decision making. *Proceedings of the Workshop on ....*

[Cao et al., 2009] Cao, H., Hu, D. H., Shen, D., Jiang, D., Sun, J.-T., Chen, E., and Yang, Q. (2009). Context-aware query classification. *Acm Sigir*, 106(3):3.

[Cao et al., 2008] Cao, H., Jiang, D., Pei, J., He, Q., Liao, Z., Chen, E., and Li, H. (2008). Context-aware query suggestion by mining click-through and session data. *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '08*, page 875.

[Capra et al., 2002] Capra, L., Blair, G. S., Mascolo, C., Emmerich, W., and Grace, P. (2002). Exploiting reflection in mobile computing middleware. *SIGMOBILE Mobile Computing Communication Revue*, 6(4):34–44.

[Carroll et al., 2005] Carroll, J. J., Bizer, C., Hayes, P., and Stickler, P. (2005). Named graphs, provenance and trust. In *Proceedings of the 14th international conference on World Wide Web*, pages 613–622. ACM.

[Chaari et al., 2005] Chaari, T., Laforest, F., Flory, A., Einstein, A. A., and Cedex, V. (2005). Adaptation des applications au contexte en utilisant les services web. *Proceedings of the 2nd French-speaking conference on Mobility and uibquity computing - UbiMob '05*, pages 3–6.

[Chuang and Chan, 2008] Chuang, S.-N. and Chan, A. T. (2008). Dynamic qos adaptation for mobile middleware. *IEEE Transactions on Software Engineering*, 34(6):738–752.

[Coutaz et al., 2005] Coutaz, J., Crowley, J. L., Dobson, S., and Garlan, D. (2005). Context is key. *Communications of the ACM*, 48(3):49–53.

[Da et al., 2011] Da, K., Dalmau, M., and Roose, P. (2011). A survey of adaptation systems. *International Journal on Internet and Distributed Computing Systems*, 2(1):1–18.

[Dewan et al., 1992] Dewan, H. M., Ohsie, D., Stolfo, S. J., Wolfson, O., and Silva, S. (1992). Incremental database rule processing in paradiser. *Journal of Intelligent Information Systems*, 1(2):177–209.

[Dey, 2001] Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7.

[Dey et al., 1999] Dey, A. K., Salber, D., Abowd, G. D., and Futakawa, M. (1999). The conference assistant: Combining context-awareness with wearable computing. In *Wearable Computers, 1999. Digest of Papers. The Third International Symposium on*, pages 21–28. IEEE.

[Dolin, 2006] Dolin, R. A. (2006). Deploying the" internet of things". In *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, pages 216–219. IEEE Computer Society.

[Donini, 2003] Donini, F. M. (2003). Complexity of reasoning. In *The description logic handbook*, pages 96–136. Cambridge University Press.

[Emani, 2014] Emani, C. K. (2014). Automatic detection and semantic formalisation of business rules. In *European Semantic Web Conference*, pages 834–844. Springer.

[Euzenat et al., 2008] Euzenat, J., Pierson, J., and Ramparany, F. (2008). Dynamic context management for pervasive applications. *The Knowledge Engineering Review*, 23(01):21–49.

[Euzenat et al., 2007] Euzenat, J., Shvaiko, P., et al. (2007). *Ontology matching*, volume 18. Springer.

[Ferscha et al., 2001] Ferscha, A., Vogl, S., and Beer, W. (2001). Context sensing, aggregation, representation and exploitation in wireless networks. *Scalable Computing: Practice and Experience*, 6(2).

[Fielding, 2000] Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine.

[Floch et al., 2006] Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., and Gjorven, E. (2006). Using architecture models for runtime adaptability. *IEEE software*, 23(2):62–70.

[Fox and Clarke, 2009] Fox, J. and Clarke, S. (2009). Exploring approaches to dynamic adaptation. In *Proceedings of the 3rd International DiscCoTec Workshop on Middleware-Application Interaction*, pages 19–24. ACM.

[Frühwirth, 2015] Frühwirth, T. (2015). Constraint handling rules-what else? In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 13–34. Springer.

[Gangemi and Presutti, 2009] Gangemi, A. and Presutti, V. (2009). Ontology design patterns. In *Handbook on ontologies*, pages 221–243. Springer.

[Garlan et al., 2009] Garlan, D., Schmerl, B., and Cheng, S.-W. (2009). *Autonomic Computing and Networking*, chapter Software Architecture-Based Self-Adaptation, pages 31–55. Springer.

[Gensel et al., 2008] Gensel, J., Villanova-Oliver, M., and Kirsch-Pinheiro, M. (2008). Modèles de contexte pour l'adaptation à l'utilisateur dans des systèmes d'information web collaboratifs. In *Workshop from" 8èmes journées francophones". Sophia-Antipolis, France*, pages 5–16.

[Giménez-García et al., 2017] Giménez-García, J. M., Zimmermann, A., and Maret, P. (2017). Ndfluents: An ontology for annotated statements with inference preservation. In *European Semantic Web Conference*, pages 638–654. Springer.

[Goasdoué et al., 2013] Goasdoué, F., Manolescu, I., and Roatiş, A. (2013). Efficient query answering against dynamic rdf databases. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 299–310. ACM.

[Gold and Mascolo, 2001] Gold, R. and Mascolo, C. (2001). Use of context-awareness in mobile peer-to-peer networks. In *Distributed Computing Systems, 2001. FTDCS 2001. Proceedings. The Eighth IEEE Workshop on Future Trends of*, pages 142–147. IEEE.

[Gómez-Pérez, 1998] Gómez-Pérez, A. (1998). Knowledge sharing and reuse. *Handbook of applied expert systems*, pages 10–11.

[Grace et al., 2003] Grace, P., Blair, G. S., and Samuel, S. (2003). Remmoc: A reflective middleware to support mobile client interoperability. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 1170–1187, Catania, Sicily, Italy.

[Grimm et al., 2012] Grimm, S., Watzke, M., Hubauer, T., and Cescolini, F. (2012). Embedded $\mathcal{EL}$+ reasoning on programmable logic controllers. In *The Semantic Web–ISWC 2012*, pages 66–81. Springer.

[Gubbi et al., 2013] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660.

[Guinard et al., 2010] Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., and Savio, D. (2010). Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *Services Computing, IEEE Transactions on*, 3(3):223–235.

[Guinard et al., 2011] Guinard, D., Trifa, V., Mattern, F., and Wilde, E. (2011). From the internet of things to the web of things: Resource-oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer.

[Guo et al., 2005] Guo, Y., Pan, Z., and Heflin, J. (2005). Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):158–182.

[Gupta et al., 1993] Gupta, A., Mumick, I. S., and Subrahmanian, V. S. (1993). Maintaining views incrementally. *ACM SIGMOD Record*, 22(2):157–166.

[Gyrard et al., 2014] Gyrard, A., Datta, S. K., Bonnet, C., and Boudaoud, K. (2014). Standardizing generic cross-domain applications in internet of things. In *IEEE GLOBECOM Workshops 2014*, pages 589–594, Austin, TX, USA. IEEE.

[Gyrard et al., 2015a] Gyrard, A., Datta, S. K., Bonnet, C., and Boudaoud, K. (2015a). Cross-domain internet of things application development: M3 framework and evaluation. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 9–16. IEEE.

[Gyrard et al., 2015b] Gyrard, A., Serrano, M., and Atemezing, G. A. (2015b). Semantic web methodologies, best practices and ontology engineering applied to internet of things. In *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pages 412–417. IEEE.

[Gyrard et al., 2017] Gyrard, A., Serrano, M., Jares, J. B., Datta, S. K., and Ali, M. I. (2017). Sensor-based linked open rules (s-lor): An automated rule discovery approach for iot applications and its use in smart cities. In *Proceedings of the 26th International Conference Companion on World Wide Web*, page to be published. International World Wide Web Conferences Steering Committee.

[Hanney et al., 1995] Hanney, K., Keane, M., Smyth, B., and Cunningham, P. (1995). Systems, tasks and adaptation knowledge: Revealing some revealing dependencies. In *International Conference on Case-Based Reasoning*, pages 461–470. Springer.

[Hartig and Thompson, 2014] Hartig, O. and Thompson, B. (2014). Foundations of an alternative approach to reification in rdf. *arXiv preprint arXiv:1406.3399*.

[He et al., 2012] He, J., Zhang, Y., Huang, G., and Cao, J. (2012). A smart web service based on the context of things.

[Hernández, 2012] Hernández, A. G. (2012). A javascript rdf store and application library for linked data client applications. Citeseer.

[Hitzler et al., 2009] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., and Rudolph, S., editors (27 October 2009). *OWL 2 Web Ontology Language: Primer*. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-primer/`.

[Hlomani and Stacey, 2014] Hlomani, H. and Stacey, D. (2014). Approaches, methods, metrics, measures, and subjectivity in ontology evaluation: A survey. *Semantic Web Journal, na (na)*, pages 1–5.

[Hong et al., 2009] Hong, J., Suh, E.-H., Kim, J., and Kim, S. (2009). Context-aware system for proactive personalized service based on context history. *Expert Systems with Applications*, 36(4):7448–7457.

[Horrocks and Patel-Schneider, 2010] Horrocks, I. and Patel-Schneider, P. (2010). Knowledge representation and reasoning on the semantic web: OWL, 2010.

[Hustadt et al., 2005] Hustadt, U., Motik, B., and Sattler, U. (2005). Data complexity of reasoning in very expressive description logics. In *IJCAI*, volume 5, pages 466–471.

[IBM, 2004] IBM (2004). An architectural blueprint for autonomic computing.

[Jamont et al., 2014] Jamont, J., Médini, L., and Mrissa, M. (2014). A Web-Based Agent-Oriented Approach to Address Heterogeneity in Cooperative Embedded Systems. In *the 12th International Conference on Practical Applications of Agents and Multi-Agent Systems PAAMS 2014*, volume 293 of *Advances in Intelligent Systems and Computing*, pages 45–52, Salamanca, Spain.

[Kakousis et al., 2010] Kakousis, K., Paspallis, N., and Papadopoulos, G. A. (2010). A survey of software adaptation in mobile and ubiquitous computing. *Enterprise Information Systems*, 4(4):355–389.

[Kalfoglou and Schorlemmer, 2003] Kalfoglou, Y. and Schorlemmer, M. (2003). Ontology mapping: the state of the art. *The knowledge engineering review*, 18(1):1–31.

[Katasonov et al., 2008] Katasonov, A., Kaykova, O., Khriyenko, O., Nikitin, S., and Terziyan, V. Y. (2008). Smart semantic middleware for the internet of things.

[Kazakov and Klinov, 2013] Kazakov, Y. and Klinov, P. (2013). Incremental reasoning in owl el without bookkeeping. In *The Semantic Web–ISWC 2013*, pages 232–247. Springer.

[Kindberg et al., 2002] Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., et al. (2002). People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376.

[Kirsch-Pinheiro et al., 2004] Kirsch-Pinheiro, M., Gensel, J., and Martin, H. (2004). Representing context for an adaptative awareness mechanism. In *Groupware: Design, Implementation, and Use*, pages 339–348. Springer.

[Kirsch-Pinheiro et al., 2006] Kirsch-Pinheiro, M., Villanova-Oliver, M., Gensel, J., and Martin, H. (2006). A Personalized and Context-Aware Adaptation Process for Web-based Groupware Systems. In , pages 884–898.

[Kollia and Glimm, 2014] Kollia, I. and Glimm, B. (2014). Optimizing sparql query answering over owl ontologies. *arXiv preprint arXiv:1402.0576*.

[Kongdenfha et al., 2006] Kongdenfha, W., Saint-Paul, R., Benatallah, B., and Casati, F. (2006). An aspect-oriented framework for service adaptation. In *International Conference on Service-Oriented Computing*, pages 15–26. Springer.

[Krishnaswamy and Li, 2014] Krishnaswamy, S. and Li, Y.-F. (2014). The mobile semantic web. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 197–198. International World Wide Web Conferences Steering Committee.

[Krötzsch, 2012] Krötzsch, M. (2012). *OWL 2 Profiles: An introduction to lightweight ontology languages*. Springer.

[Laborie et al., 2011] Laborie, S., Euzenat, J., and Layaïda, N. (2011). Semantic adaptation of multimedia documents. *Multimedia tools and applications*, 55(3):379–398.

[Lee, 2008] Lee, E. A. (2008). Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE.

[Luckenbach et al., 2005] Luckenbach, T., Gober, P., Arbanowski, S., Kotsopoulos, A., and Kim, K. (2005). Tinyrest: A protocol for integrating sensor networks into the internet. In *Proc. of REALWSN*, pages 101–105. Citeseer.

[Manola et al., 2004] Manola, F., Miller, E., McBride, B., et al. (2004). Rdf primer. *W3C recommendation*, 10(1-107):6.

[Mascolo et al., 2002] Mascolo, C., Capra, L., and Emmerich, W. (2002). Mobile computing middleware. In *Advanced lectures on networking*, pages 20–58. Springer.

[Médini et al., 2013] Médini, L., Bâcle, F., and Nguyen, H. D. T. (2013). Dataconf: Enriching conference publications with a mobile mashup application. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 477–478. ACM.

[Médini et al., 2016] Médini, L., Mrissa, M., Terdjimi, M., Khalfi, E.-M., Le Sommer, N., Capdepuy, P., Jamont, J.-P., Occello, M., and Touseau, L. (2016). Building a web of things with avatars.

[Mikalsen et al., 2006] Mikalsen, M., Paspallis, N., Floch, J., Stav, E., Papadopoulos, G. A., and Chimaris, A. (2006). Distributed context management in a mobility and adaptation enabling middleware (madam). In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 733–734. ACM.

[Motik et al., 2009] Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C., et al. (2009). Owl 2 web ontology language: Profiles. *W3C recommendation*, 27:61.

[Motik et al., 2012] Motik, B., Horrocks, I., and Kim, S. M. (2012). Delta-reasoner: a semantic web reasoner for an intelligent mobile platform. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 63–72. ACM.

[Motik et al., 2015a] Motik, B., Nenov, Y., Piro, R., and Horrocks, I. (2015a). Combining rewriting and incremental materialisation maintenance for datalog programs with equality. *arXiv preprint arXiv:1505.00212*.

[Motik et al., 2015b] Motik, B., Nenov, Y., Piro, R., and Horrocks, I. (2015b). Incremental update of datalog materialisation: the backward/forward algorithm. In *Proc. AAAI*.

[Mrissa et al., 2014a] Mrissa, M., Médini, L., and Jamont, J. (2014a). Semantic discovery and invocation of functionalities for the web of things. In *2014 IEEE 23rd International WETICE Conference, WETICE 2014, Parma, Italy, 23-25 June, 2014*, pages 281–286. IEEE Computer Society.

[Mrissa et al., 2015] Mrissa, M., Médini, L., Jamont, J.-P., Le Sommer, N., and Laplace, J. (2015). An avatar architecture for the web of things. *IEEE Internet Computing*, 19(2):30–38.

[Mrissa et al., 2014b] Mrissa, M., Médini, L., and Jamont, J.-P. (2014b). Semantic discovery and invocation of functionalities for the web of things. In *IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*.

[Mrissa et al., 2014c] Mrissa, M., Médini, L., Jamont, J.-P., Le Sommer, N., and Laplace, J. (2014c). An avatar architecture for the web of things.

[Munnelly et al., 2007] Munnelly, J., Fritsch, S., and Clarke, S. (2007). An aspect-oriented approach to the modularisation of context. In *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*, pages 114–124. IEEE.

[Musolesi and Mascolo, 2009] Musolesi, M. and Mascolo, C. (2009). Car: context-aware adaptive routing for delay-tolerant mobile networks. *Mobile Computing, IEEE Transactions on*, 8(2):246–260.

[Nenov et al., 2015] Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., and Banerjee, J. (2015). RDFox: A highly-scalable RDF store. In *The Semantic Web - ISWC 2015*, volume 9367, pages 3–20. Springer International Publishing.

[Nguyen et al., 2014] Nguyen, V., Bodenreider, O., and Sheth, A. (2014). Don't like rdf reification?: making statements about statements using singleton property. In *Proceedings of the 23rd international conference on World wide web*, pages 759–770. ACM.

[Perera et al., 2014] Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *Communications Surveys & Tutorials, IEEE*, 16(1):414–454.

[Perrucci et al., 2011] Perrucci, G. P., Fitzek, F. H., and Widmer, J. (2011). Survey on energy consumption entities on the smartphone platform. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pages 1–6. IEEE.

[Pfisterer et al., 2011] Pfisterer, D., Romer, K., Bimschas, D., Kleine, O., Mietz, R., Truong, C., Hasemann, H., Kröller, A., Pagel, M., Hauswirth, M., et al. (2011). Spitfire: toward a semantic web of things. *IEEE Communications Magazine*, 49(11):40–48.

[Raverdy et al., 2006] Raverdy, P.-G., Riva, O., de La Chapelle, A., Chibout, R., and Issarny, V. (2006). Efficient context-aware service discovery in multi-protocol pervasive environments. In *Mobile Data Management, 2006. MDM 2006. 7th International Conference on*, pages 3–3. IEEE.

[Ruta et al., 2014] Ruta, M., Scioscia, F., Loseto, G., Gramegna, F., Ieva, S., and Di Sciascio, E. (2014). Mini-me 2.0: powering the semantic web of things. In *3rd OWL Reasoner Evaluation Workshop (ORE 2014)(jul 2014)*.

[Sadjadi and McKinley, 2004] Sadjadi, S. M. and McKinley, P. K. (2004). Act: An adaptive corba template to support unanticipated adaptation. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*, pages 74–83. IEEE.

[Sampson et al., 2004] Sampson, D. G., Lytras, M. D., Wagner, G., and Diaz, P. (2004). Ontologies and the semantic web for e-learning. *Educational Technology & Society*, 7(4):26–28.

[Schilit et al., 1994] Schilit, B., Adams, N., and Want, R. (1994). Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE.

[Schilit et al., 1993] Schilit, B. N., Adams, N., Gold, R., Tso, M. M., and Want, R. (1993). The parctab mobile computing system. In *Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on*, pages 34–39. IEEE.

[Schmidt, 2003] Schmidt, A. (2003). *Ubiquitous computing-computing in context*. PhD thesis, Lancaster University.

[Schmidt et al., 1999] Schmidt, A., Beigl, M., and Gellersen, H.-W. (1999). There is more to context than location. *Computers & Graphics*, 23(6):893–901.

[Schueler et al., 2008] Schueler, B., Sizov, S., Staab, S., and Tran, D. T. (2008). Querying for meta knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 625–634. ACM.

[Seinturier et al., 2009] Seinturier, L., Merle, P., Fournier, D., Dolet, N., Schiavoni, V., and Stefani, J.-B. (2009). Reconfigurable sca applications with the frascati platform. In *Services Computing, 2009. SCC'09. IEEE International Conference on*, pages 268–275. IEEE.

[Shelby et al., 2014] Shelby, Z., Hartke, K., and Bormann, C. (2014). The constrained application protocol (coap).

[Sinner and Kleemann, 2005] Sinner, A. and Kleemann, T. (2005). Krhyper–in your pocket. In *Automated Deduction–CADE-20*, pages 452–457. Springer.

[Stehr et al., 2011] Stehr, M., Talcott, C. L., Rushby, J. M., Lincoln, P., Kim, M., Cheung, S., and Poggio, A. (2011). Fractionated software for networked cyber-physical systems: Research directions and long-term vision. In *Formal Modeling: Actors, Open Systems, Biological Systems - Essays Dedicated to Carolyn Talcott on the Occasion of Her 70th Birthday*, volume 7000 of *Lecture Notes in Computer Science*, pages 110–143. Springer.

[Terdjimi, 2015] Terdjimi, M. (2015). Multi-level context adaptation in the web of things. In *Doctoral Consortium at ISWC2015*.

[Terdjimi et al., 2016a] Terdjimi, M., Médini, L., and Mrissa, M. (2016a). HyLAR+: Improving Hybrid Location-Agnostic Reasoning with Incremental Rule-based Update. In *WWW'16: 25th International World Wide Web Conference Companion*, Montreal, Québec, Canada.

[Terdjimi et al., 2016b] Terdjimi, M., Médini, L., and Mrissa, M. (2016b). Towards a meta-model for context in the web of things. In *Karlsruhe Service Summit Workshop*.

[Terdjimi et al., 2016c] Terdjimi, M., Médini, L., Mrissa, M., and Le Sommer, N. (2016c). An avatar-based adaptation workflow for the web of things. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2016 IEEE 25th International Conference on*, pages 62–67. IEEE.

[Terdjimi et al., 2017] Terdjimi, M., Médini, L., Mrissa, M., and Maleshkova, M. (2017). Multi-purpose adaptation in the web of things. In *CONTEXT-17*.

[Thompson, 2005] Thompson, C. W. (2005). Smart devices and soft controllers. *Internet Computing, IEEE*, 9(1):82–85.

[Tigli et al., 2009] Tigli, J.-Y., Lavirotte, S., Rey, G., Hourdin, V., Cheung-Foo-Wo, D., Callegari, E., and Riveill, M. (2009). Wcomp middleware for ubiquitous computing: Aspects and composite event-based web services. *annals of telecommunications-annales des télécommunications*, 64(3-4):197–214.

[Tolk and Muguira, 2003] Tolk, A. and Muguira, J. A. (2003). The levels of conceptual interoperability model. In *Proceedings of the 2003 Fall Simulation Interoperability Workshop*, volume 7. Citeseer.

[Truong and Dustdar, 2009] Truong, H.-L. and Dustdar, S. (2009). A survey on context-aware web service systems. *International Journal of Web Information Systems*, 5(1):5–31.

[Truong et al., 2008] Truong, H.-L., Dustdar, S., Baggio, D., Corlosquet, S., Dorn, C., Giuliani, G., Gombotz, R., Hong, Y., Kendal, P., Melchiorre, C., et al. (2008). Incontext: A pervasive and collaborative working environment for emerging team forms. In *Applications and the Internet, 2008. SAINT 2008. International Symposium on*, pages 118–125. IEEE.

[Truong et al., 2007] Truong, H.-L., Juszczyk, L., Manzoor, A., and Dustdar, S. (2007). *ESCAPE–an adaptive framework for managing and providing context information in emergency situations*. Springer.

[Verborgh et al., 2014a] Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., and Van de Walle, R. (2014a). Querying datasets on the web with high availability. In *The Semantic Web–ISWC 2014*, pages 180–196. Springer.

[Verborgh et al., 2014b] Verborgh, R., Sande, M. V., Colpaert, P., Coppens, S., Mannens, E., and de Walle, R. V. (2014b). Web-scale querying through linked data fragments. In *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014.*

[Wang et al., 2004] Wang, X. H., Zhang, D. Q., Gu, T., and Pung, H. K. (2004). Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. Ieee.

[Wei et al., 2006] Wei, Q., Farkas, K., Prehofer, C., Mendes, P., and Plattner, B. (2006). Context-aware handover using active network technology. *Computer Networks*, 50(15):2855–2872.

[Wilde, 2007] Wilde, E. (2007). Putting things to rest. *School of Information*.

[Xiang et al., 2010] Xiang, B., Jiang, D., Pei, J., Sun, X., Chen, E., and Li, H. (2010). Context-aware ranking in web search. *Sigir 2010*, page 451.

[Yu et al., 2006] Yu, Z., Zhou, X., Zhang, D., Chin, C.-Y., Wang, X., et al. (2006). Supporting context-aware media recommendations for smart phones. *Pervasive Computing, IEEE*, 5(3):68–75.

[Zeginis and Plexousakis, 2010] Zeginis, C. and Plexousakis, D. (2010). Web service adaptation: State of the art and research challenges. *Self*, 2:5.

[Zimmermann et al., 2007] Zimmermann, A., Lorenz, A., and Oppermann, R. (2007). An operational definition of context. In *Modeling and using context*, pages 558–571. Springer.

[Zufferey and Kosch, 2006] Zufferey, M. and Kosch, H. (2006). Semantic adaptation of multimedia content. In *Multimedia Signal Processing and Communications, 48th International Symposium ELMAR-2006 focused on*, pages 319–322. IEEE.