Ubiquitous Computing (UbiComp) envisions environments where devices interact among themselves to work seamlessly together on behalf of humans. In recent years, the emergence of the Internet of Things concept, which opts for connecting everyday objects to the Internet, and the Mobile Computing paradigm have contributed to strengthening UbiComp. For this reason, UbiComp environments are not necessarily populated by powerful computers. On the contrary, resource constrained devices (e.g., embedded and mobile devices) are the main actors in these environments. Thus, it is important for the environment to deal with their heterogeneity, unreliability, and replaceability.

In order to cope with heterogeneity, the Semantic Web has proposed several standards and models to clearly define the terms so that they can be reused across applications boundaries. Regarding unreliability and replaceability, Tuple Spaces promote the uncoupled coordination of the devices. Solutions based on semantic Tuple Spaces combine these three beneficial aspects resulting from bridging the Semantic Web and Tuple Spaces domains for UbiComp.

Most of these semantic Tuple Spaces consider embedded and mobile devices as mere clients in a space managed by more powerful devices. Such delegation helps to reduce the workload of devices with computing and energy limitations. However, this delegation moves the data away from where it is physically generated. This creates a conflict between providing updated data and generating unnecessary network traffic for unused information. In addition, this delegation makes constrained devices intrinsically dependent onother devices when it might not always be necessary. This dissertation explores how these constrained devices can act as fully fledged semantic knowledge providers to create a more decentralized space.

Aitor Gómez Goiri

Semantic Tuple Spaces for Constrained Devices: A Web-compliant Vision

Semantic Tuple Spaces for Constrained Devices: A Web-compliant Vision

Aitor Gómez Goiri

Bilbao, 2014

Supervisors:
Diego López de Ipiña
Íñigo Goiri

Deusto
University of Deusto

UNIVERSIDAD DE DEUSTO

SEMANTIC TUPLE SPACES FOR CONSTRAINED DEVICES: A WEB-COMPLIANT VISION

Tesis doctoral presentada por Aitor Gómez Goiri

dentro del Programa de Doctorado en Ingeniería Informática y Telecomunicación

Dirigida por Dr. Diego López de Ipiña
y Dr. Íñigo Goiri Presa

Bilbao, abril de 2014

Semantic Tuple Spaces for Constrained Devices: A Web-compliant Vision

Author: Aitor Gómez Goiri
Supervisor: Dr. Diego López de Ipiña
Supervisor: Dr. Íñigo Goiri Presa

The following web-page address contains up to date information about this dissertation and related topics:
`http://gomezgoiri.net`

Text printed in Bilbao
First edition, April 2014

A todo doctorando que no vea la luz al final del tunel o se sienta desbordado por la frustración.

Ánimo.

Abstract

Ubiquitous Computing (UbiComp) envisions environments where devices interact among themselves to work seamlessly together on behalf of humans. In recent years, the emergence of the Internet of Things (IoT) concept, which opts for connecting everyday objects to the Internet, and the Mobile Computing paradigm have contributed to strengthening UbiComp. For this reason, UbiComp environments are not necessarily populated by powerful computers. On the contrary, resource constrained devices (e.g., embedded and mobile devices) are the main actors in these environments. Thus, it is important for the environment to deal with their heterogeneity, unreliability, and replaceability.

In order to cope with heterogeneity, the Semantic Web has proposed several standards and models to clearly define the terms so that they can be reused across applications boundaries. Regarding unreliability and replaceability, space-based computing (or Tuple Spaces) promotes the uncoupled coordination of the devices. Solutions based on semantic tuple spaces combine these three beneficial aspects resulting from bridging the Semantic Web and Tuple Spaces domains for UbiComp.

Most of these semantic tuple spaces consider embedded and mobile devices as mere clients in a space managed by more powerful devices. Such delegation helps to reduce the workload of devices with computing and energy limitations. However, this delegation moves the data away from where it is physically generated. This creates a conflict between providing updated data and generating unnecessary network traffic for unused information. In addition,

this delegation makes constrained devices intrinsically dependent on other devices when it might not always be necessary. This dissertation explores how these constrained devices can act as fully fledged semantic knowledge providers to create a more decentralized space.

In conclusion, this dissertation presents a novel adaptation of semantic tuple space which considers the energy and computational impact on the devices. Specifically, this dissertation proposes the following contributions:

- A space model which considers the principles which have made the web flourish in the last decades, together with the uncoupling properties of space-based computing.
- An energy-aware search mechanism for autonomous constrained devices.
- An alignment of two approaches to act on the physical environment, namely a space-based indirect actuation and a web-based direct actuation.

Resumen

La computación ubicua (UbiComp) concibe entornos donde los dispositivos interactúan entre sí para trabajar en beneficio de los seres humanos, pero de forma imperceptible para los mismos. En los últimos años, la emergencia del Internet de las Cosas (IoT), que aboga por conectar objetos cotidianos a Internet, y la computación móvil han contribuido a fortalecer la idea de UbiComp. Es por ello que los entornos ubicuos no están necesariamente poblados por computadoras potentes. Por contra, los actores principales de dichos entornos suelen ser en su mayor parte dispositivos con recursos limitados (p.e. dispositivos móviles y embebidos). Es por ello que para estos entornos es de vital importancia enfrentarse a la heterogeneidad, falta de fiabilidad y facilidad de reemplazo de dichos dispositivos.

Para hacer frente a la heterogeneidad, la web semántica propone diversos estándares y modelos para proveer un significado preciso a los términos para que puedan reusarse más allá de las fronteras marcadas por las aplicaciones. En lo relativo a la falta de fiabilidad y facilidad de reemplazo, la computación basada en espacios (o espacios de tuplas o Tuple Spaces) promueve una coordinación desacoplada de los dispositivos. Las soluciones basadas en espacios de tuplas semánticos, que unen los dominios de la web semántica y los Tuple Spaces, combinan estos tres aspectos beneficiosos para UbiComp.

Muchos de estos espacios semánticos de tuplas consideran a los dispositivos móviles y embebidos como meros clientes de un espacio gestionado por dispositivos más potentes. Dicha delegación ayuda a reducir la carga de trabajo de los dispositivos con limitaciones

computacionales y energéticas. Sin embargo, al mismo tiempo distancia los datos de donde fueron generados. Esto crea un conflicto entre proveer datos actualizados y generar tráfico de red innecesario para información no usada. Además, esta delegación hace a los dispositivos limitados intrínsecamente dependientes de otros cuando no siempre es necesario. Esta tesis explora cómo esos dispositivos limitados pueden actuar como auténticos proveedores de conocimiento semántico para crear un espacio más descentralizado.

En conclusión, esta tesis describe una adaptación novedosa de los espacios semánticos de tuplas que considera el impacto computacional y energético en los dispositivos. Específicamente, esta tesis presenta las siguientes contribuciones:

- Un modelo de espacio que considera los principios que han hecho florecer a la web en las últimas décadas, junto con las propiedades de desacoplamiento de la computación basada en espacios.
- Un mecanismo de búsqueda energéticamente eficiente para dispositivos limitados autónomos.
- Un alineamiento entre dos formas de actuar en un entorno físico: actuación indirecta basada en espacios y actuación directa basada en la web.

Laburpena

Konputazio ubikuoak (UbiComp) gizakientzat era hautemanezinean lan egiten duten gailuen arteko interakzioa sustatzen duten inguruneak proposatzen ditu. Azken urteotan, UbiComp indartu egin da, Gauzen Interneten (IoT) eta konputazio mugikorraren gorakada dela eta. Hori dela eta, UbiComp inguruneetako eragile nagusiak ez dira ahalmen handiko ordenagailuak, baliabide gutxikoak baizik (adibidez, kapsulatutako gailuak eta mugikorrak). Hortaz, garrantzitsua da ingurune horietan beraien heterogeneotasuna, fidagarritasun eza eta ordezkagarritasuna kontutan hartzea.

Heterogeneotasunari aurre egiteko, web semantikoak hainbat estandar eta eredu proposatu ditu terminoei esanahi zehatza emateko eta jatorrizko aplikazioen mugetatik kanpo erabili ahal izateko. Bestalde, fidagarritasun faltari eta ordezkatzeko erraztasunari aurre egiteko, espazioetan oinarritutako konputazioak (edo Tuple Space delakoak) gailuen arteko koordinazio desakoplatua sustatzen du. Tupla espazio semantikoetan oinarritutako soluzioek, hau da, web semantikoaren eta Tuple Space delakoen domeinuak lotzen dituztenek, UbiComp-erako hiru ezaugarri onuragarri horiek bateratzen dituzte.

Kapsulatutako gailuak eta mugikorrak ahalmen handiko ordenagailuek kudeaturiko lekuen bezerotzat hartzen ditu Tupla espazio semantiko askok. Delegazio horrek konputazio ahalmen eta autonomia mugatuko gailuen lan karga murrizten laguntzen du. Zoritxarrez, datuak fisikoki sortzen diren lekutik aldentzen ditu delegazio horrek. Hori dela eta, datu eguneratuak eskaintzearen eta erabiltzen ez den informaziorako beharrezkoa ez den trafikoa sortzearen

arteko gatazka sortzen da. Gainera, delegazio horrek gailu mugatuak berez beste gailu batzuen menpe uzten ditu, beti beharrezkoa ez bada ere. Tesi honek ikertzen du nola erabili gailu mugatuak ezagutza semantikoaren hornitzaile gisa, espazio deszentralizatuagoa lortzeko.

Ondorioz, tesi honek tupla espazio semantikoen egokitzapen berri bat aurkezten du. Egokitzapen horrek gailuetako energia eta konputazio eragina kontuan hartzen ditu. Zehazki, tesiak honako ekarpenak proposatzen ditu:

- Azken hamarkadetan weba arrakastatsu bihurtu duten oinarriak kontuan hartzen dituen espazio eredua eta, era berean, espazioan oinarritutako konputazioaren propietate desakoplatuak mantentzen dituena.
- Gailu autonomo eta mugatuentzako energiaren aldetik eraginkorra den bilaketa mekanismoa.
- Ingurune fisikoa aldatzeko bi teknikaren errenkada: espazioetan oinarritutako zeharkako jarduera eta webean oinarritutako jarduera zuzena.

Acknowledgements

Si debo a alguien la posibilidad de haber concluido esta aventura es sin duda a aita y ama. Esta posibilidad va más allá del mundano y sin embargo tan necesario hecho de haberme proporcionado una cómoda vida en la que siempre pude elegir mi camino. Por vosotros soy como soy y soy quien soy. Podría haber salido un hijo un poco más hablador y menos pedante, pero no será porque no os esforzasteis. Echando la vista atrás no puedo sino estar orgullosísimo de la educación recibida. Por ello, vaya para vosotros mi primer agradecimiento.

Gracias Diego por haberme dado esta enorme oportunidad. Gracias por el tiempo y el espacio concedido. Gracias Íñigo por soportarme en momentos de máxima incertidumbre y pesimismo. Gracias por servirme de guía.

Gracias a todos mis compañeros y ex-compañeros. Gracias a mis compañeros de generación por compartir sufrimientos: Edu, Iván y Eguiluz. Gracias a esa infatigable e incansable máquina con piernas de la que tanto se aprende que es Pablo Orduña. Gracias a los ancianos del lugar: Aitor y Unai. Gracias a la gente del Garilab por aceptarme durante mi exilio. En definitiva, gracias a tanta gente que ha pasado por la vida *esmarlabense* estos últimos años: Don Jaime, Emaldi, Laiseca, el innombrable, Lázaro y un largo etc. Afortunadamente, ese "*largo etc*" se puede consultar en la página web del grupo haciendo uso de la consulta mostrada al final de esta sección o siguiendo el siguiente *QRCode*.

Thanks to all the inspiring people I've known during this journey. Thanks to Geoff Coulson, Barry Porter and the rest of the people

I met at Lancaster University. Thanks to all the ETH's *Distributed Systems group* members, specially to Simon Mayer, my cicerone in Zurich.

Gracias a todos mis compañeros de aventuras estos años. A los ocasionales y a los fieles. En viajes por el mundo, canchas de baloncesto, senderos, *eventos del esmarla*,... Cada segundo en que mantuve la cabeza fuera de la tesis fue un respiro enorme.

Gracias al resto de mi familia, la sanguínea y la *circunstancial*. Especialmente a mi hermano Mikel y a mi abuela Teo.

Eskerrik asko,

Aitor Gómez Goiri

April 2014

```
PREFIX swrcfe: <http://www.morelab.deusto.es/ontologies/swrcfe.owl#>

SELECT COUNT(DISTINCT ?faname) WHERE{
  ?job swrcfe:doneBy
     <http://www.morelab.deusto.es/labman/resource/people/aitor-gomez-goiri> .
  {
    ?job2 swrcfe:doneBy ?formerworkmate .
    ?formerworkmate foaf:familyName ?faname .
    ?formerworkmate foaf:firstName ?finame .
    ?job swrcfe:jobStartDate ?jobstartdate .

    OPTIONAL { ?job2 swrcfe:jobEndDate  ?jobenddate }
    FILTER(?jobstartdate < ?jobenddate)
  } UNION {
    ?job2 swrcfe:doneBy ?workmate .
    ?workmate foaf:familyName ?faname .
    ?workmate foaf:firstName ?finame .

    OPTIONAL { ?job2 swrcfe:jobEndDate ?jobenddate }
    FILTER(!bound(?jobenddate))
  }
  FILTER(?job != ?job2)
}
```

Contents

List of Figures

List of Tables

List of listings

Acronyms

AmI	Ambient Intelligence 1
API	application programming interface 12
CoAP	Constrained Application Protocol 15
DHT	Distributed hash table 36
DNS	Domain Name System 36
DNS-SD	DNS service discovery 95
EXI	Efficient XML Interchange 80
DPWS	Device Profile for Web Services 14
FOL	first-order logic 30
HTML	Hypertext Markup Language 13
HTTP	Hypertext Transfer Protocol 4
IoT	Internet of Things iii
JSON	JavaScript Object Notation 80
LD	Linked Data 19
LOD	Linked Open Data 67
mDNS	Multicast DNS 72
N3	Notation 3 107
NB	negative broadcasting 82
OWL	Web Ontology Language 31

P2P	peer-to-peer 36
RDF	Resource Description Framework 16
RDFS	RDF Schema 31
REST	REpresentational State Transfer 2
CS	Client-server 12
S	Stateless 12
$	Cache 12
U	Uniform interface 12
ID	Identification of resources 12
REP	Manipulation of resources through representations 12
DESC	Self-descriptive messages 12
HATEOAS	Hypermedia as the engine of application state 12
L	Layered system 13
COD	Code on Demand 13
SPARQL	SPARQL Protocol and RDF Query Language 77
SOAP	Simple Object Access Protocol 14
SW	Semantic Web 17
TS	Tuple Space 2
TSC	Triple Space Computing 2
SWS	Semantic Web Spaces 32
CSpaces	Conceptual Spaces 30
STuples	Semantic Tuple Spaces 32
TripCom	Triple Space Communication 34
UbiComp	Ubiquitous Computing iii
UDDI	Universal Description, Discovery and Integration 14
UPnP	Universal Plug and Play 72

Poets say science takes away from the beauty of the stars - mere globs of gas atoms. Nothing is "mere". I, too, can see the stars on a desert night, and feel them. But do I see less or more?

Richard P. Feynman

CHAPTER 1

Introduction

1.1 Motivation

Originally, the Internet was basically composed of a small number of computers that were physically connected to a wired network. Over the years, the popularity of the Internet grew and connecting computers became easier and cheaper. Thanks to wireless technologies, devices can connect to the Internet without having to be physically connected to a network.

These technologies have contributed to the emergence of end-user devices such as smartphones or tablets. Furthermore, everyday objects like cars or washing machines are now starting to be connected to the Internet to exchange information. This is what is currently known as the IoT [6]. The objects of the IoT, together with mobile devices, constitute the clearest sign of the importance of Ubiquitous Computing (UbiComp) in our lives today [15].

The UbiComp concept was envisioned by Weiser in the early nineties [113]. Weiser defended that devices should imperceptibly work on our behalf. This idea is particularly stressed by the concept of Ambient Intelligence (AmI) [97]. For instance, the washing machine could plan to finish washing the clothes two minutes before the car gets home.

Weiser remarked that the real power of UbiComp "*comes not from any one of these devices, it emerges from the interaction of all of them*". However, integrating devices to accomplish such task is not simple as they usually communicate using different protocols. To solve this problem, the Web of Things (WoT) initiative proposes using well-established web standards to ease their communication [32]. This, together with the already existing tools and libraries from the world of the web, simplifies its adoption by developers [35]. Furthermore, the WoT naturally integrates with widely used RESTful web applications.

Parallel to this initiative, Tuple Spaces (or *space-based computing*) [31] proposes a different integration style based on the *blackboard model*. In Tuple Space (TS) participants cooperate with each other by writing and reading information into a shared *space*. The main benefit of this paradigm is the higher-level of decoupling its indirect communication provides.

One common problem of both the WoT and Tuple Spaces is that the format of the data they exchange is multifarious and application domain dependent. This implies that data will not be meaningful in other domains unless a specialized system converts and reinterprets them. A way to solve this problem is annotating the data semantically as proposed by the World Wide Web (WWW) [9]. Specifically, the Triple Space Computing (TSC) paradigm [24] employs this semantic knowledge on shared spaces following the Tuple Spaces approach. We also argue that TSC can be designed to fulfil REpresentational State Transfer (REST)'s principles.

However, adding semantics may impose a burden on resource constrained devices. To reduce this overhead in such devices, part of this computation is usually delegated to intermediaries [54]. This approach reduces the overhead of semantically annotated data but brings other problems:

1. When devices rely on others to provide information, it is not guaranteed that the information accessed will accurately represent the last information available in the data providers (i.e., the sensors).

2. Once we rely on those intermediaries, they must be available at all times. Otherwise, the devices will not be able to talk to each other.

In this thesis, we propose to adapt TSC to the UbiComp environments, particularly involving limited devices in such tasks. To fulfil such an ambitious goal, we need to carefully consider the restrictions imposed by such environments. Two key aspects to take into account are the limited energy autonomy or computational capacity of many devices. Similarly, our solution should be flexible enough to support a wide range of scenarios and to reuse data from third-party applications.

This thesis addresses different key aspects of this adaptation:

- *TSC's compatibility with the REST style and the WoT vision.* By aligning the TSC design with REST one can inherit many of the benefits provided by this style and interoperate with other REST-based solutions. At the same time, the use of TSC can help developers to focus on higher-level problems.

- *Decentralized search on the semantic content provided by federated devices.* This search architecture enables direct communication between participants. Furthermore, it reduces the energy consumption of the resource-constrained devices.

- *Actuation on the physical environment.* We present common usage patterns for space-based computing and we analyse how they can be used in UbiComp. We also propose a space enhancement which seamlessly reuses external REST services. This enable us to build a bridge between our solution and other WoT and REST solutions.

1.2 Hypothesis

The hypothesis of this dissertation is as follows:

> The alignment of the TSC paradigm with the web's principles together with the consideration of its energy and computational impact, leads to UbiComp environments enabled by heterogeneous embedded and mobile devices that communicate autonomously in a decoupled and interoperable fashion.

Note that the consequence of the hypothesis involves the following aspects:

Heterogeneity : Fully-fledged computers and resource constrained devices (e.g., mobile and embedded devices) must coexist in these environments.

Autonomy : Devices must not depend on others to consume or provide data on their behalf. However, they might be aided by other devices to complete some related tasks (e.g., search the appropriate nodes to request).

Decoupling : The communication must be data-driven. From the user perspective devices do not directly refer to each other. Additionally, the provider and the consumer should not coexist in time. However, note that since this sub-aspect contradicts the autonomy principle, their selection might be left to the user.

Interoperability : Devices must be able to exchange information with other systems and to use that information.

1.3 Objectives

To validate the hypothesis, the main objective of this dissertation is to *design a middleware which follows the TSC paradigm according to web principles and considering the energy and computation aspects*.

This objective can be achieved through the following sub-objectives:

- Design of a dual space model where participants can coordinate throughout the space in an uncoupling manner and enrich it with their own managed knowledge.
- Design of a TSC interface for Hypertext Transfer Protocol (HTTP) which is compatible with most of the REST principles. This resource-oriented interface defines a minimal contract to access the semantic knowledge provided by the space.
- Creation of a search-aware architecture which promotes end-to-end communication between devices. This architecture tries to minimize the requests devices have to handle by enhancing their search mechanism.

The extra tasks introduced by this enhancement are performed by nodes chosen according to their capacities.

- Comparison of different mechanisms to act on a physical environment: the space-based actuation patterns and the direct REST service consumption.

- Alignment between space-based computing and direct REST service consumption to seamlessly reuse third-party application capabilities.

1.4 Research methodology

The author developed this thesis in the following phases: *(1)* awareness of the problem, *(2)* solution suggestion, *(3)* development and evaluation, and *(4)* conclusion. Figure 1.1 represents each of these phases together with their sub-phases.

The problem awareness phase requires an intensive review of the state of the art in the fields of space-based computing, Ubiquitous Computing and the Semantic Web. As a result, the limitations of the existing works and the areas where it is possible to make a scientific contribution are identified.

The second phase starts with the suggestion of a solution for the problem. The solution can be refined and adjusted thanks to the feedback given by experts in the field. These experts are the supervisors, colleagues and, partners from research projects where the author has worked or the researchers from external groups where the author has done a research stay.

The third phase comprises planning, developing, and evaluating the different parts which form our solution. The planning involves defining how to develop these parts and how to evaluate if the desired contributions are achieved through them. Then, the development and evaluation sub-phases are carefully scheduled. The development sub-phase does not only include development of the different modules, but also their design. The evaluation covers the preparation of the experimental environment, the development of the experiments, and their execution to precisely measure the indicators previously identified.

To conclude the third phase, we contrast the contributions with experts in the field through workshops, conferences and journals. Note that sometimes a thorough review of a rejected work can be as valuable as the face-to-face feedback obtained at the venue itself. As a result of the feedback obtained, the second phase can be iteratively resumed to refine the solution.

When the solution does not require further improvements and it is validated by the research community, the fourth phase starts. In this last phase, we analyse the results of our work to obtain conclusions and possible future research areas.

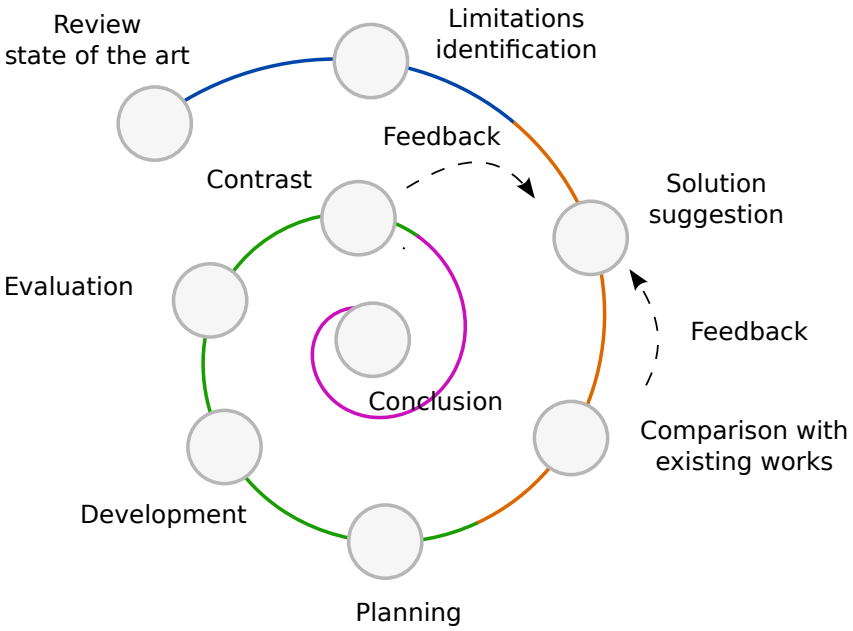


Figure 1.1: Methodology followed in this dissertation.

1.5 Thesis outline

The remainder of this dissertation is structured as follows: Chapter 2 contextualizes the thesis. Chapter 3 first presents and compares some relevant works from the field of Ubiquitous Computing. It then analyses the state of the art on existing semantic space-based solutions and scrutinizes their advantages and limitations.

Chapter 4 proposes and analyses a model to adapt TSC to the UbiComp. This model involves limited devices by enabling them to autonomously manage their own knowledge and later enrich the space with it. Simultaneously, it preserves most of the space-based computing benefits through a classical space model.

Chapter 5 describes the core of this thesis. It presents an energy-aware search architecture. This architecture dynamically adapts to the needs of resource-constrained devices. The goal of this architecture is two-fold: *a*) to promote direct and fully distributed communication between devices; and, at the same time, *b*) to reduce network communications between them, particularly the ones directed to the less powerful devices.

Chapter 6 explores two different mechanisms to change the physical environment using TSC. Ideally, these changes should be produced by writing new semantic information into the space. However, there is a range of actuators which already use REST services whose actuation capacities could be reused. To achieve this, we propose an alignment between both approaches.

Each chapter evaluates our proposals, discusses the related work, and states our intermediate conclusions.

Finally, Chapter 7 states the conclusions of this thesis. It remarks its advantages and limitations and discusses the achieved goals. We also explain the lessons learnt and future areas of research on this topic.

The beginning of wisdom is to desire it.

Solomon Ibn Gabirol

CHAPTER

2

Background

To clarify the foundations in which our work relies and to provide a starting point for understanding the rest of the thesis, this chapter introduces, categorizes, and describes the related research topics. These topics are classified attending to their relation with a critical aspect for UbiComp: interoperability.

The IEEE defines *interoperability* as "*the ability of two or more systems or components to exchange information and to use the information that has been exchanged*" [1]. The heterogeneity of technologies present in UbiComp environments makes this a key property to consider. The definition clearly distinguishes between two requirements: ***(1)* to exchange** information; and ***(2)* to use** that information.

Exchanging information in distributed systems covers the communication between two systems. For the lower communication levels, we rely on standard and widely accepted communication protocols (i.e., interoperability *ab-initio*). For higher-levels (i.e., application layer), this dissertation delves into the *space-based computing* and REST architectures. Section 2.1 categorizes both of them together with other integration approaches. Then, both integration styles are individually presented in sections 2.2 and 2.3.

Regarding the second goal, it can be analysed from two perspectives: syntactically and semantically. On the one hand, *syntactic interoperability* is associated with the format of the data (i.e., its syntax and encoding) [107]. On the other hand, *semantic interoperability* is concerned with ensuring that the exchanged information has a precise meaning. Its ultimate goal is to make the information "*understandable by any other application that was not initially developed for this purpose*" [2]. Section 2.4 describes a prominent movement which is focused on this goal: the Semantic Web.

2.1 Application integration

The integration of two applications is driven by how they communicate. To ease this communication the applications use middlewares. A middleware is a software layer which provides a higher level of abstraction and masks the underlying heterogeneity. Coulouris et al. [19] define two communication styles on the upper layer of a middleware: the remote invocation and the indirect communication (see Figure 2.1).

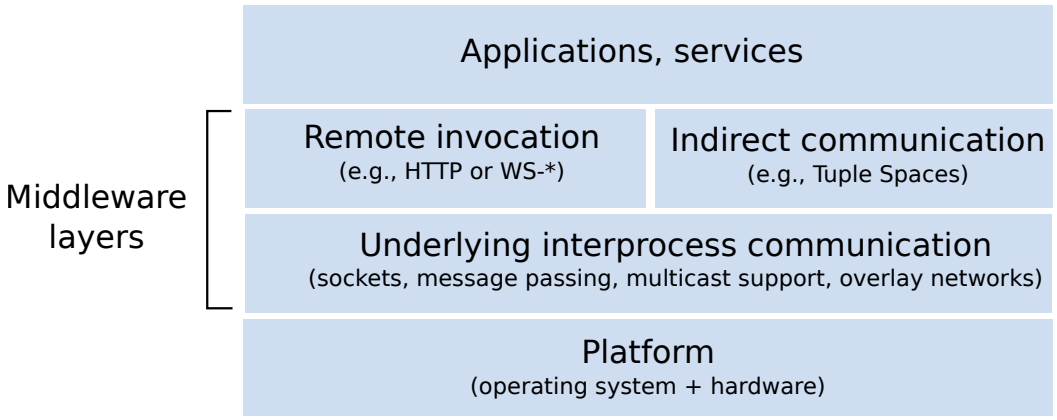


Figure 2.1: Middleware layers.
Middleware layers according to Coulouris et al. [19] classification.

The *remote invocation* involves the most common two-way exchange between senders and receivers in distributed systems. Among others, it covers request-

reply protocols, remote procedure calls and remote method invocation. Request-reply protocols are the simplest and most lightweight mechanisms for client-server computing. Within these protocols HTTP shines as one of the web pillars.

HTTP can be used as the baseline to design other *remote invocation* paradigms (e.g., the WS-* [3] standards). However, its design is intended to support the web, whose modern architecture follows the REST architectural style [26]. Consequently, one can see the pervasiveness of the web applications as a proof of REST's success.

The *indirect communication* style covers all the techniques with no direct coupling between the sender and the receiver. The group communication, publish-subscribe systems, message queues or shared memory approaches are examples of indirect communication. These paradigms are characterized by two key properties [31, 19] [1] :

- *Space uncoupling*, which is achieved when the sender does not need to know the receiver or receivers and vice versa.
- *Time uncoupling*, which happens when senders and receivers do not need to exist at the same time[2].

This dissertation delves into a particular shared memory approach: Tuple Space computing. However, as mentioned, REST architectures' properties have made them massively accepted to integrate applications. Consequently, we also take into consideration the latter mechanism in our solution conception.

[1] We use the terminology of the Tuple Spaces' seminal paper [31]. However, note that the *space uncoupling* property is referred as *reference autonomy* by some authors [24]. These same authors mention a third property confusingly called *space autonomy* (or *location autonomy*). According to Fensel [24] this autonomy is achieved because: "The processes can run in completely different computational environments as long as both can access the same space".

[2] Although some authors [24, 66] explain this property just in terms of communication asynchrony, Coulouris et al. [19] make a clear distinction between them. In their words, a communication is asynchronous when "*a sender sends a message and then continues without blocking*", whereas time uncoupling adds an extra dimension: "*the sender and the receiver can have independent existences*".

2.2 REpresentational State Transfer

REpresentational State Transfer (REST) is a network-based architectural style proposed by Fielding [26]. It aims to cover certain properties explained in Section 4.5.2. To achieve these properties, REST establishes the following constraints from other network-based architectural styles:

Client-server (CS). Providing an application programming interface (API) to the clients, they are isolated from back-end implementation details.

Stateless (S). The state is fully stored in the client and therefore each request has all the information needed to process it.

Cache ($). When added to the CS constraint, this style replicates content obtained from a server in the client.

Uniform interface (U). It is the key constraint which distinguishes REST from other architectural styles. This constraint is composed by the following ones:

Identification of resources (ID). Resources are the conceptual targets of hypertext references. Their identification offers a generic interface to access and change the values of a resource.

Manipulation of resources through representations (REP). Representations are composed by a sequence of bytes and the metadata to describe those bytes.

Self-descriptive messages (DESC). The client and server have to agree on standard methods and media types. Beyond that point, each request or response should contain all the needed data to process it [111]. Therefore, in Fielding's words, the type should be registered, the registry should point to a specification and the specification should explain how to process data according to its intent [27].

Hypermedia as the engine of application state (HATEOAS). This is a controversial constraint because most of the self-proclaimed "REST" APIs fail to follow it [77, 55]. It states that no out-of-the-band information should guide the interaction with an API. Instead, the hypertext should guide

it. In other words, the client must know just an initial URL and the application's media types. From that point, it should select the alternatives proposed by the server to change to the next application state [28].

Layered system (L). Each layer provides services to the top layer.

Code on Demand (COD). It is the only optional *constraint* in REST. It occurs when the client downloads from the server the *know-how* needed to process the set of resources it already has.

Richardson [98] came up with a maturity model to help people understand the REST principles in a less abstract manner. Instead of focusing on the presented constraints, the model identifies whether an API properly uses the web's three most important protocols: Uniform Resource Identifier (URI), HTTP and Hypertext Markup Language (HTML). Level zero encompasses those APIs which use a unique URI and HTTP method (i.e., use HTTP as a tunnelling mechanism). Level one is formed by the APIs which clearly identify several resources by providing an URIs to each of them. Level two uses HTTP as it was originally designed for, i.e., using its verbs, status codes, etc. appropriately. The correct use of HTTP forces a developer to comply with all the REST's constraints except for HATEOAS [77]. Level three comprises the APIs which return resources which describe their own capabilities and are interconnected using preferably standard representations (i.e., those which also address the Hypermedia as the engine of application state (HATEOAS) constraint).

According to Fielding [26], only those in the third level can be considered REST architectures. However, there is a generalized misconception of the REST term. Architectures and APIs at the first or second level of the *Richardson Maturity Model* are often informally (and incorrectly) considered REST [28, 77, 63].

To avoid confusions, we want to clarify the terminology used in this dissertation. To adhere to the precise meaning of REST, we consider orthodox architectures concrete instantiations of REST and heterodox ones REST-like architectures. In other words, in the scope of this dissertation, REST-like architectures are those described by second level of the *Richardson Maturity Model*. Recently, the *Hypermedia API* term has emerged to refer to HTTP APIs which

fully comply with all the REST principles [64, 77, 5]. We will indistinguishably refer to them as RESTful APIs or as *hypermedia APIs*.

2.2.1 REST vs WS-* services

WS-* [86, 3], also called "Big Web services", together with RESTful architectures are probably the most common remote invocation substyles currently used in the Internet. Universal Description, Discovery and Integration (UDDI) defines how to discover these *big web services* using registries. A registry returns information to point to the web services' interfaces. These interfaces are syntactically described using the Web Services Description Language (WSDL). Additionally, WSDL describes data and message types, interaction patterns and protocol mappings. Using this information, the service consumer and provider communicate through messages encapsulated using Simple Object Access Protocol (SOAP).

WS-* standards offer more features than the REST such as transactions, reliability or message-level security. In addition, they have been adapted to the needs of resource-constrained devices in the Device Profile for Web Services (DPWS) specification [148]. This specification defines a minimal set of implementation constraints. DPWS's most remarkable features are: decentralized multicast-based discovery, secure message transmission, subscription and event notifications [78].

Note that although the web implements REST's principles and WS-* can contradict them [1], WS-* stands for *web services*. The reason for this is that it employs web standards. However, WS-* uses some of them in ways that they were not designed for [66]. The most paradoxical case is how it uses HTTP as a transport layer instead of as an application layer protocol. This prevents the resulting architectures from achieving some of HTTP's desirable properties such as scalability or visibility. In other words, it sacrifices some of the web's properties on behalf of additional features.

[1] Interestingly, Moritz et al. [78] came to the conclusion that DPWS can be restricted to be fully compatible with the RESTful style and still cover some missing features (e.g., eventing and discovery).

One of the sacrificed properties is the simplicity. The complexity negatively affects to *(1)* the ability of limited computing platforms to adopt the WS-* stack; and *(2)* the developers. Specifically, developers need to take further architectural decisions on different layers of the WS-* stack. These decisions make people perceive it as more complex than the RESTful style [35]. This perception influences the adoption of REST on behalf of WS-* [23, 72].

In fact, apart from technical considerations [1], REST's acceptance is a powerful practical motivation for focusing on it. In the end, seamlessly integrating with a higher number of applications seems to be a logical choice if one of the middleware's goals is to reuse other applications' data. Although this acceptance level is subject to changes in the future, the life of the web backs REST as a long-term choice. Or in Fielding's words [28, comment 21], "*REST is intended for long-lived network-based applications that span multiple organizations*".

2.2.2 Suitable protocols for REST

Although an API does not adhere to the REST style just because it uses certain protocols, the correct use of some of them can help to achieve most of its principles [77]. Historically, HTTP has been considered a suitable protocol in that regard. HTTP is a simple protocol which has be adopted by a wide range of computing platforms [116, 51][143].

However, in the last few years the Constrained Application Protocol (CoAP) [156] has emerged as a specialized web transfer protocol for resource constrained devices. Some noteworthy features of CoAP are *(1)* the reduced message size, *(2)* the use of UDP as a transport layer (with the possibility of using *multicast* communication), *(3)* similarity with HTTP (both to reuse its properties and to ease cross-protocol proxying), and *(4)* a resource discovery mechanism.

One could argue that to implement a lightweight TSC solution, CoAP should be used as a baseline. However, we have chosen to work with HTTP for the following reasons:

[1] For an extensive analysis on the advantages and disadvantages of REST and WS-*, we refer the reader to [92] and [35].

- Direct interoperation with other web-solutions. Directly using HTTP we can avoid proxies. Proxies may introduce latency in the response time degrading network performance.
- Resource Description Framework (RDF)-based media-formats can be rather verbosed. This contrasts with CoAP's message size limitations. Dealing with these limitations was not one of the main goals of the dissertation. However, we have considered them at some points of the dissertation to avoid unrealistic assumptions.
- CoAP is an ongoing standard (i.e., its definition is currently subject to change). As a result of this immaturity, there are few libraries and tools to work with CoAP at the moment [110]. This limits the range of platforms which could adopt any proposed solution.

Nevertheless, due to HTTP's similarities with CoAP [65], the future adoption of the latter should be relatively straightforward.

2.3 Tuple Space

Tuple Space (TS) computing, also called space-based computing, offers an improvement over traditional distributed shared memory approaches. Whereas the latter ones work at byte-level and accessing to memory addresses, the Tuple Space works with semi-structured data which is accessed in an associative manner. In other words, in TS the participants read data specifying patterns of interest.

TS has its roots in the Linda parallel programming language [31]. In this communication model different processes read and write pieces of information so-called tuples into a common space. Tuples are composed by one or more typed data fields (e.g., $<$"$aitor$", 1984 $>$ or $<$ 3, 7, 21.0 $>$). The tuples are accessed associatively using a template. A template provides either a value or type for different fields (e.g., $< String, 1984 >$ or $< Integer, Integer, Float >$). The operations over the space are defined by different primitives. Although the primitives may change from implementation to implementation, the most common ones allow to:

- *Access the tuples non-destructively*, using a primitive which is usually called *read* or *rd*. This primitive returns a tuple from the space which matches the given template without changing the space.
- *Access the tuples destructively*, using a primitive which is usually called *take* or *in*. This primitive extracts a tuple which matches the given template from the space.
- *Write a new tuple into the space*. This primitive is usually called *write* or *out*.

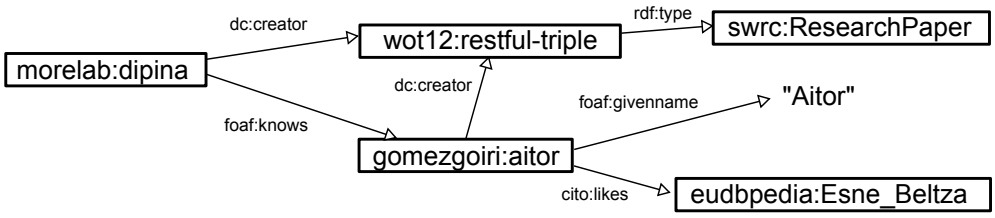
2.4 The Semantic Web

A problem of the initial view of the Web was that it was human-centred. Regardless of whether the contents were machine processable, a human needed to interpret them to give them a meaning. Then, Berners-Lee et al. [9] proposed a solution to this problem: the Semantic Web (SW). The *World Wide Web Consortium* defines the SW and its key features as follows [17]:

> The vision of the Semantic Web is to extend principles of the Web from documents to data. Data should be accessed using the general Web architecture using, e.g., URI-s; data should be *related to one another* just as documents (or portions of documents) are already. This also means creation of a common framework that allows data to be *shared and reused* across application, enterprise, and community boundaries, to be *processed automatically* by tools as well as manually, including revealing possible *new relationships* among pieces of data.

In the SW, triples are the most basic information units. A triple is composed by a subject, a predicate and an object like in a normal sentence (see Figure 2.2). As Berners-Lee et al. explained, using triples "*a document can assert that things (people, web pages or whatever) have properties (such as 'is a sister of', 'is the author of') with certain values (another person, another Web page)*". A key difference with a normal sentence is that each concept is unambiguously

defined by an URI. These URIs form links between different triples as shown in Figure 2.2.



subject	**predicate**	**object**
URI | bnode	URI	URI | bnode | literal
http://gomezgoiri.net/aitor	*cito:likes*	*eudbpedia:Esne_Beltza*
http://gomezgoiri.net/aitor	*foaf:givenname*	*"Aitor"^^xsd:string*
morelab:dipina	*foaf:knows*	*http://gomezgoiri.net/aitor*

Figure 2.2: Sample triples using different ontologies.
All the triples are represented graphically and some of them also textually. They describe a researcher's academic and personal details. The knowledge is expressed using four different ontologies: FOAF [125], DC [124], SWRC [140] and CiTO [121]. For the sake of clarity, some of the URIs in the figure are shortened using aliases or prefixes.

A problem with the information described so far is that two different databases may use different URIs to express the same concept. To overcome this the SW offers collections of information called ontologies. An ontology is a document which expresses the relations between terms commonly using a taxonomy and a set of rules to infer content. The taxonomy defines the classes of a given domain and how they relate with each other.

Of course, different content providers may provide similar data described according to different ontologies. To overcome this problem, ontologies can be easily mapped by providing equivalence relations within them. Similarly, an ontology can be extended to adapt it to different application domains. In any case, the reuse of the same models is beneficial to ease the interoperability. This reuse is promoted by the community through the standardization of vocabularies.

Remarkably, a significant part of the SW community has gathered around the Linked Data (LD) in the last years. The Linked Data [11] term refers to a series of principles on how to publish data. The goal of the LD is to publish linked terms using semantics and, in fact, it has been referred as the *"Semantic Web done right"* by Berners-Lee [8], coiner of both terms.

I was born not knowing and have had only a little time to change that here and there.

Richard P. Feynman

CHAPTER

3

State of the Art

After having presented the general research topics in which this work fits, we analyse the specific works closely related to ours. This thesis focuses on the intersection of the three topics represented in the Figure 3.1. It also stresses in which areas the following sections focus on.

Firstly, Section 3.1 explores relevant works on the UbiComp field. Rather than presenting an exhaustive (and exhausting) survey on middlewares for UbiComp, we analyse those which follow the REST architectural style. Then, we remark how we can solve the main limitations of these works by using a TS middleware.

Secondly, Section 3.2 scrutinizes relevant TS proposals for the UbiComp field. Specifically, we study their space model and, for the distributed proposals, how they distribute the tuples. Then, we present and justify our model's similarities and divergences with theirs.

Thirdly, Section 3.3 presents some semantic works for UbiComp. From these works we derive two main approaches to semantically annotate content provided by limited devices: *a*) do it in more powerful intermediaries; and *b*) directly involve all the devices in the management of their content. This

work is interested in the second approach. Regarding related research following this approach, we inspect their complementary proposals but we distinguish the unaddressed needs covered by Chapter 5.

Finally, Section 3.4 analyses and compares in detail other semantic TS middlewares independently of their application domain.

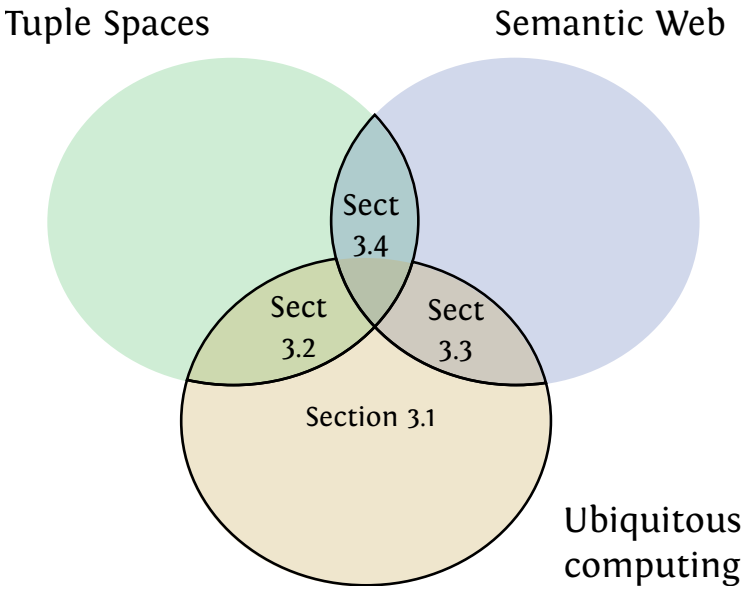


Figure 3.1: Chapter 3 outline.
This dissertation focuses on the confluence of the background areas represented in this figure.

3.1 REST on UbiComp

REST prioritizes the scalability and simplicity over other characteristics [26]. These two characteristics are particularly beneficial for UbiComp. On the one hand, UbiComp environments demand scalability as they are usually populated by a panoply of heterogeneous and limited devices. On the other hand, the simplicity helps these devices to overcome such limitations easing the adoption of the REST style. A representative example of how REST has influenced UbiComp is the Web of Things (WoT).

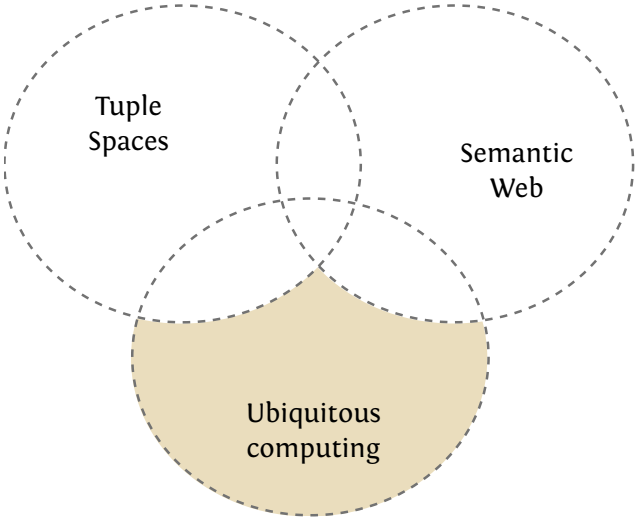


Figure 3.2: Non-semantic works for UbiComp apart from Tuple Spaces.
This subsection focuses on the solutions using REST architectures.

The Web of Things initiative encourages the use of REST-based solutions embedding web servers in daily objects [36, 32]. In this way, the things can seamlessly integrate with the WWW as first-class citizens [38]. This integration brings the following benefits:

- The smart-things can be linked to enable its discovery by browsing. This involves using the tool most users are familiar with: the browser.
- They can be bookmarked or shared through social networks [34].
- They can be integrated with other web applications through mash-ups [33, 90, 95, 12, 104].
- Existing mechanisms such as searching, caching, load-balancing and indexing can be used over the objects to achieve the scalability of the web [37].

3.1.1 Discussion

In the last decade, probably because of its perceived simplicity [35], REST and REST-like APIs have gained adopters in the IoT. Particularly, the WoT has

emerged as a simple mechanism to integrate smart objects with each other, but also with existing web applications.

However, as any direct communication style REST introduces coupling between senders and receivers. These complicates the application changes both in the short-term and in the long-term [59]. In the short-term because nodes constantly join and leave the environment due to mobility or to failures. In the long-term because the space is used to solve new problems and the obsolete devices are replaced with new technology.

This dissertation proposes a middleware to reduce this coupling. This middleware ensures that, from the developer perspective, the communication is always driven by the data. In other words, in our middleware the devices are unaware of others' presence (i.e., they are *space uncoupled*).

3.2 Tuple Space on UbiComp

So far, Tuple Space has been adapted to Ubiquitous Computing by different authors. This section presents the most relevant ones.

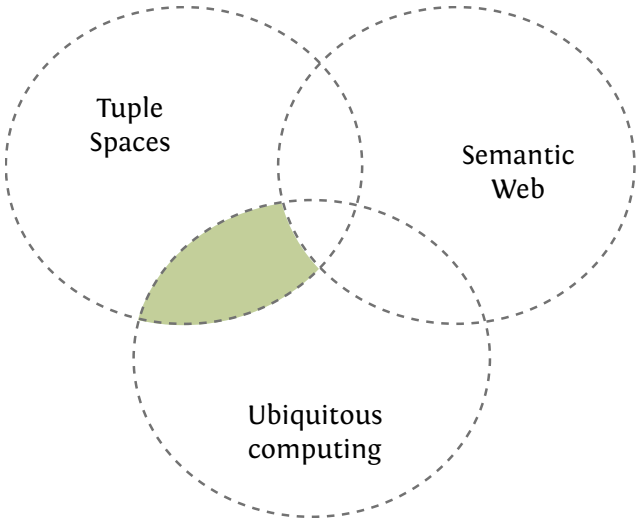


Figure 3.3: Tuple Spaces solutions for UbiComp.

Scope of section 3.2

The *event heap* [59] is a system used for a specific UbiComp sub-domain: interactive workspaces. In this scenario, there are rooms with different devices

deployed and where mobile devices can enter. Each room has its own space where the devices exchange tuples to cooperate. This work merely identifies these environments' requirements and the properties to cover them. Then, it discusses how these properties can be satisfied using TS or some extensions. Finally they compare their implementation both with other TSs and other coordination infrastructures.

L^2imbo [22, 30] replicates the tuples to avoid a single point of failure. Each node joined to a space uses an IP multicast address to exchange messages with other nodes in that space. Writing into a space involves sending a multicast message to inform to the rest of the nodes of the tuple written. Reading operation can usually be satisfied locally. Destructive reading of a tuple is more complex as it requires a global withdrawal. In *L^2imbo* only the owner of a tuple can remove it from the space. The ownership of a tuple initially belongs to its creator, but can be transferred.

LIME (Linda in a Mobile Environment) [94] is a TS solution for mobile systems. In LIME each mobile device has its own space where it generally writes its tuples. This space is shared with other devices creating federated spaces, i.e., the aggregation of different shared spaces. In this way, each mobile can access tuples in other mobiles whenever they become available. They also proposed a new writing primitive to insert tuples into remote spaces. *LIME* has subsequently been adapted to limited platforms [80] with TinyLIME [20] and TeenyLIME [18].

However, Coulouris et al. [19] complain about the unrealistic assumptions LIME's authors make to simplify the problem. These assumptions are the uniform multicast connectivity between devices whose tuple spaces are aggregated and the serialized and ordered connections and disconnections.

In the *TOTA* (Tuples On The Air) Project [73] tuples are disseminated to different devices. To that end, each tuple has 3 fields: *(1)* the content of the tuple; *(2)* a rule which defines how it should be propagated; and *(3)* a rule to define its maintenance.

For instance, they consider a museum where a visitor writes a query tuple describing a piece of art he wants to see. The propagation rule defines that it should be propagated to all nodes in the vicinity, increasing the distance by

one each time. The tuple is configured to be deleted after a time-to-live period using its maintenance rule. When it reaches the room where the piece of art is located, this art work writes a response tuple. This response tuple jumps from a device to another until it reaches the device which queried for it.

3.2.1 Discussion

The space can be used to *(1)* coordinate with other devices by writing and extracting content and *(2)* check what happens in the environment by reading. The solutions presented integrate both uses in the same space, which can be distributed or not. However, we argue that these uses face different needs and, therefore, should be treated separately (i.e., using two types of spaces).

The first usage demands consistency to avoid the unexpected consequences of two devices extracting the same tuple. For this usage, we propose a typical coordination space which will be accessed through a *resource oriented* HTTP API. Whether the space itself is distributed and how, is out of the scope of this dissertation and left up to the implementer.

For the second usage type, we aim to integrate as many external data sources as possible. This integration demands to acknowledge their independence, and therefore limit the collaboration requirements imposed to them. Consequently, for the second usage we propose a distributed space which resembles more to *LIME*'s *federated spaces* than to *TOTA* or *L*^{2}*imbo*.

3.3 The Semantic Web on UbiComp

This section analyses some notable works for UbiComp environments which use the Semantic Web (SW) to represent the contextual information. Rather than presenting an in-depth analysis of different architectures or their applications to concrete UbiComp scenarios, we focus on the resource constrained devices' perspective. We scrutinize the role of the mobile and embedded devices in the systems which use the SW.

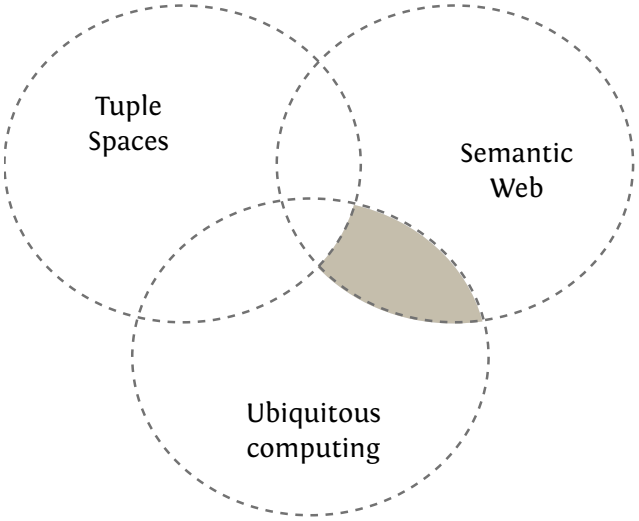


Figure 3.4: The Semantic Web for Ubiquitous Computing.
Scope of Section 3.3.

3.3.1 The Semantic Web using intermediaries

Adding semantics works well for devices with high computational capacity but may add too much overhead for limited devices. To reduce this overhead, part of this computation is usually delegated to an intermediary. Some noteworthy example is the one proposed by Bröring et al. [14].

These intermediaries or *Semantic Gateways* are in charge of managing the semantic annotation. The devices send raw data (which can be compressed) to the intermediaries and the gateways annotate the content semantically. Thus, the devices do not have to care about any semantic aspect and just collect the data as they did before.

These *Semantic Gateways* reduce the load to embedded devices with limited resources by decreasing the number of requests they have to provide. In addition, a centralized intermediary can gather all the information and thus, reduce the complexity of managing a distributed environment.

However, using intermediaries to store the semantic data of resource constrained devices also has some drawbacks. On the one hand, centralization does not faithfully represent mobility situations were individuals carry their own semantic information in their personal devices. In addition, the data obtained

from an intermediary will always be less fresh than the one obtained where it is actually being generated (i.e., sensors). On the other hand, the servers are critical in centralized systems and therefore, their availability determines the operation of these solutions. They also impose a burden on the maintenance which may be worthless in some simple scenarios.

3.3.2 The Semantic Web applied to data providers

Lately, the computing capabilities of some mobile and embedded devices have improved enough to afford semantic processing. As a consequence, some solutions have arisen to semantically annotate data where it is generated.

D'Aquin et al. [21] presents an architecture to deploy an SPARQL [162] endpoint in Android devices [118]. In their solution the mobile devices store and manage their own semantic knowledge by means of an embedded semantic repository. On top of these endpoints, D'Aquin et al. envision a query federation mechanism. This dissertation shares this view of independent data sources sharing their own managed semantic content.

In the Web of Things, multiple solutions have considered using semantics to enrich the data definition in a machine processable manner. Some solutions embed the metadata in HTML using microdata [164], microformats [130] or RDFa [165]. These contents are returned by the Internet connected objects and are used to enhance the search-ability of the data by existing third-party search engines. However, these search engines do not consider the mobile nature of the data providers in the WoT. To solve this problem Trifa et al. [106] propose a hierarchical discovery and lookup infrastructure. This infrastructure extracts the semantic annotations from different representations and converts them to an internal data model. This model is based on existing microformats, but limited to certain concepts. Furthermore, the particularities of the semantic annotations are hidden to searchers in their custom search API.

A more general way to represent semantic data is using RDF [161] based representations (i.e., full semantics). The SPITFIRE European project [139] represents the most remarkable effort on gathering full semantics and the WoT. It focuses on fully integrating sensor data with the Linked Data [11].

SPITFIRE shares with our solution the vision of a world populated by devices acting as semantic data providers no matter how small they are [52]. Therefore, many of their efforts are complimentary to this work. To search within these providers, Pfisterer et al. [93] propose a model which predicts the current state of things by computing their periodic patterns in past states. Again, the goal of this method is to adapt search engines to the new fashion of data provided on the *Semantic Web of Things*.

3.3.3 **Discussion**

Resource constrained devices have been extensively used to provide information to represent the physical environments. When this information has been semantically annotated, traditional solutions delegate on intermediaries the semantization of the raw data. However, recent advances in mobile and embedded devices' capabilities have enabled to increase their responsibilities. Some of these devices are now able to personally manage and share their own semantic content. This completely distributed approach tries to simplify the maintenance burden and promote the access to the most updated data.

This dissertation aims to explore this path. Specifically, the search architecture presented in Chapter 5, tries to enhance the searching capability of web-connected devices. It leads to environments where Internet-connected objects, mobile devices or regular computers directly interact and collaborate with each other. This contrast with the searching mechanisms presented by equivalent solutions in Section 3.3.2. They assume that these limited devices need more powerful machines to search, which is not always true.

3.4 **Semantic Tuple Spaces**

Semantic Tuple Spaces aim to join Tuple Spaces with the Semantic Web to propose a more uncoupled solution. Particularly, it benefits from the autonomies introduced by both TS and the SW:

- Space uncoupling.

- Time uncoupling.
- Data-schema uncoupling.

This section provides a state-of-the-art review of the existing Semantic Tuple Spaces solutions [87]. Instead of describing each of these solutions individually and then include a comparison, the review is divided in sections which describe the main aspects of a Semantic TS.

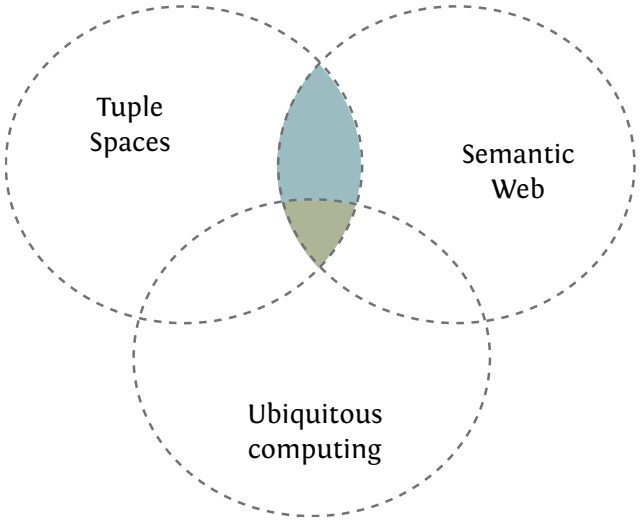


Figure 3.5: Semantic Tuple Spaces.
Scope of Section 3.4.

3.4.1 Use of semantics

The works analysed present two strategies to semantically annotate data: the use of SW standards or the definition of their own language. Conceptual Spaces (CSpaces) [74] and Nardini et al. [85] define their own language-independent up to first-order logic (FOL) notation. However, these works face an inconvenience: most of the libraries and tools available are based on the SW standards. Therefore, they have to internally translate between both worlds in a not-straightforward and resource consuming manner.

On the contrary, the majority of the semantic TSs use the metadata data model in which the SW relies: RDF. We also opt for using the widely accepted

RDF to ensure interoperability *ab-initio*. On top of RDF, the SW defines new layers to increase the data expressiveness. First, RDF Schema (RDFS) [159] describes a series of classes and properties to define vocabularies in RDF. On top of RDFS, Web Ontology Language (OWL)[158] offers additional vocabulary for describing properties and classes. These properties and classes allow to reason over the knowledge both to validate the data inserted or to infer implicit (unstated) knowledge.

Although reasoning may be desirable, its use is tangential to the model proposed in this thesis. In any case, it is by no means mandatory in our solution because:

- The devices which may provide information to our *federated space* are autonomous. They share a minimal contract with our middleware where we cannot define how they manage/provide their information.

- It is limited by the availability of reasoners, and nowadays there is no light enough reasoner to run in a reasonable time span in small devices.

Table 3.1: Use of semantics in the analysed solutions.

	Use of SW standards
TSC [25]	✓
SWS [105]	✓
STuples [60]	✓ [1]
CSpaces [74]	×
tsc++ [69, 13]	✓
TripCom [102]	✓
Smart-M3 [54]	✓
Nardini et al. [85]	×

[1] It uses DAML+OIL, a OWL precursor.

3.4.2 Tuple model

On the one hand, several works embed semantic content in one of the fields of a common tuple. CSpaces [74] defines seven-field tuples. Semantic Tuple Spaces (STuples) [60] extend the *JavaSpace* [29] middleware adding a field with semantic content associated to the tuple object. This content follows the DAML+OIL language (the OWL precursor). Nardini et al. [85] use the concept of *semantic tuples* which are expressed in logic terms.

On the other hand, the most common tuple model is the three-field tuple which corresponds with a RDF triple. That is, a tuple with a field for the *subject*, another for the *predicate* and a third one for the *object*. However, a triple by itself cannot express much information [67]. To solve this limitation, semantic TSs have usually adopted the concept of RDF graphs too. A RDF graph is a set of RDF triples identified by an URI. Although they can be accessed by their URI, more interestingly, these middlewares also guarantee associative access.

Table 3.2: Information units used by the different semantic TS middlewares.

	Tuples with semantic field	RDF triple-like tuples	RDF Graphs
TSC [25]		✓	✓
SWS [105]		✓	✓[1]
STuples [60]	✓		
CSpaces [74]	✓		
tsc++ [69, 13]		✓	✓
TripCom [102]		✓	✓
Smart-M3 [54]		✓	✓
Nardini et al. [85]	✓		

[1] Semantic Web Spaces (SWS)'s subspaces are conceptually equivalent to RDF Graphs: an abstraction to work with a set of RDF triples.

3.4.3 Query model

In TS, originally tuples were selected using special tuples where wildcard values were allowed in the fields. All the studied works which use a RDF triple as a tuple follow this approach. In addition, they provide access to the RDF graphs by their URIs.

Most of these works also offer advanced query languages (e.g., SPARQL [162]) as a more expressive way to match graphs. These languages can be decomposed in plain triple patterns. However, this requires a parser which may not be available for resource constrained platforms. This dissertation focuses on queries based on graph patterns and leaves the adoption of more complex querying languages as a future enhancement.

CSpaces, STuples and Nardini et al. use less standard querying approaches. CSpaces [74] and Nardini et al. [85] offer a formal language to select appropriate tuples. STuples[60] extends JavaSpace's template by adding assertional axioms that can be used to match semantic tuples.

Table 3.3: Querying units for semantic TSs.

	Graph patterns	Advanced query languages	Other
TSC [25]	✓		
SWS [105]	✓		
STuples [60]			✓
CSpaces [74]			✓
tsc++ [69, 13]	✓	✓[1]	
TripCom [102]	✓	✓	
Smart-M3 [54]	✓	✓	
Nardini et al. [85]			✓

[1] It completely depends on the underlying data store selected.

3.4.4 **Space model**

The *flat* model offers independent disjoint spaces. STuples, Nardini et al. [85], TSC, tsc++ and Smart-M3 use this model. Within them, Nardini et al. [85] present the most singular model. It extends TuCSoN [89], which presents an evolution of the TS called *tuple centre*. A *tuple centre* can be adapted to the application needs through reactions to communication operations. These reactions allow to trigger behaviours in response to any primitives or to define new ones.

More sophisticated models allow to create hierarchies of spaces. Three examples are SWS, Triple Space Communication (TripCom) [102] and CSpaces. SWS [105] proposes two ways to partition the spaces: sub-spaces and contexts. Sub-spaces are disjoint partitions of the main space. Contexts enable to virtually divide the space into overlapping partitions. These partitions are used to enable particular clients' views of the space.

However, probably SWS's most distinctive feature is that it virtually divides the spaces into two views. The *data view* stores syntactically valid RDF and it is accessed using Linda-like primitives. The *information view* stores consistent and satisfiable data which are managed using new primitives. The latter view takes into account the knowledge defined by ontologies to perform semantic matching over inferred triples.

TripCom shares some similarities with SWS's model. It uses subspaces to form nested multiple spaces. Doing so it restricts the communication to a part of the whole space leading to scalability and completeness. In addition, it offers a mechanism to overlap spaces called *scopes*. Using *scopes* a client can create a temporary copy of some tuples. However, any insertion and deletion would not apply to the whole Triple Space.

CSpaces [74] proposes two types of spaces: individual and shared. An *individual space* belongs to a single process. Two participants can agree on how to represent the knowledge to share their individual spaces forming a *shared space*. Shared spaces can join to others forming a tree structure. In a shared space the updates are versioned and can be revoked by any member. However,

neither the registration process needed for the agreement or the revocation process are detailed. Furthermore, to the best of our knowledge, this conceptual exercise never went beyond a rather limited prototype.

Table 3.4: Space model used by the different works.

	Flat	Nested Disjoint	Overlapping views
TSC [25]	✓		
SWS [105]		✓	
STuples [60]	✓		
CSpaces [74]		✓	✓
tsc++ [69, 13]	✓		
TripCom [102]		✓	✓
Smart-M3 [54]	✓		
Nardini et al. [85]	✓		

3.4.5 **Distribution**

Participant nodes usually access semantic spaces on client/server basis. Tsc++[69, 13] proposes an exception to the client/server access to the space. It relies on the Jxta P2P framework [128] to propagate queries using different strategies. In tsc++, spaces correspond to groups of nodes which locally manage their data.

In client/server spaces, the back-end of the server can be distributed or centralized in a single machine. Centralized Tuple Spaces are much simpler and easier to implement. Therefore, they usually offer more features than the distributed ones. However, they also impose a single-point-of-failure.

Within the distributed approaches we can distinguish those works which replicate data and those which do not. TSC belongs to the first group, and replicates all the triples in each deployed kernel. In TripCom each kernel stores one or more subspaces and can contact other kernels responsible for different spaces. To do that, if the space's Uniform Resource Locator (URL) is provided,

it simply resolves this URL using Domain Name System (DNS) and contacts the other kernel. Otherwise, the kernel uses three additional strategies:

- Triple Provider. It uses shortcuts to know who answered a query in the past.

- Recommender. It uses shortcuts to know which kernel successfully routed a query in the past.

- Indexing - Distributed hash table (DHT). It creates indexes using a hash function over the subject, predicate, object and space URL. Then, it stores these indexes in a distributed database which relies in a structured peer-to-peer (P2P) system.

CSpaces uses a similar but vaguely described super-peer network [75].

As discussed in Section 3.2, we also deal with distribution of data in one of the spaces of our hybrid model. As detailed in the following chapters, we locally manage the content and we distribute the queries. This resembles to the *tsc++*'s strategy. However, instead of using a P2P framework to access the contents, we individually access them using several HTTP requests. In other words, each client may access various servers to obtain a result for a given primitive.

Honkola et al. [54] defend that Smart-M3's space can be distributed using the *distributed deductive closure protocol*. However, to the best of our knowledge this idea has never been implemented or evaluated, making Smart-M3's space de facto centralized. For the communication between the clients and the space, Smart-M3 defines a stateful protocol called *Smart Access Protocol (SSAP)*. The authors defend that this protocol is communication agnostic because it can be implemented on top of different communication mechanisms (e.g., WS-* web services, XMPP [151], Bluetooth [120] or TCP/IP). Kiljander et al. [61] propose an enhanced stateless access protocol designed to fit the needs of low capacity devices.

Table 3.5: Distribution of the spaces.

	C/S access	Distributed space	Distribution strategy
TSC [25]	✓	✓	Replication
SWS [105]	✓	×	-
STuples [60]	✓	×	-
CSpaces [74]	✓	✓	Not detailed
tsc++ [69, 13]	×	✓	Local writing, different query strategies
TripCom [102]	✓	✓	Structured network, different strategies
Smart-M3 [54]	✓	✓	Theoretical
Nardini et al. [85]	✓	×	-

3.4.6 Discussion

The use of standard semantic protocols and RDF triple-like tuples characterizes Triple Space Computing. TSC was born to realign web services (WS-*) with the web. To ensure this alignment, it was based on REST architectural style's principles [24, 53]. However, TSC has never been true to all these principles. Furthermore, the more features are added to the TSC design (e.g., subscriptions or transactions), the more difficult it is to reconciliate both worlds.

Regardless of their incompatibility with other features or practical technical difficulties, we defend that, *per se*, TSC does not contradict in any sense the REST principles described in Chapter 2:

Client-server (CS). Accessing to a space through a server in a CS fashion is completely feasible. Indeed, this does not prevent to use a distributed solution in the *back end* (e.g., a distributed semantic repository).

Stateless (S). The primitives to access the space imply simple reads and writes which do not store any state in the server.

Cache ($). Despite of the difficulties detected by Fensel et al. [25], nothing prevents the semantic content stored in the space to be cached. However,

the dynamism of the knowledge can make effective caching challenging to achieve.

Identification of resources (ID). In TSC, there can be up to three type of resources which can be identified by an URI : spaces, RDF Graphs and certain elements of the RDF Triples. The space can be seen as a coarse-grained view of the underlying graphs. The graphs have sets of triples which are usually related and describe a unit of knowledge. Self-identified triple's subjects, predicates or objects are the source of concept linking in the SW.

Manipulation of resources through representations (REP). The RDF graphs and triples mentioned above can be represented using different standard serializations.

Self-descriptive messages (DESC). The messages derived from the primitives are self-describing since they are expressed on standard RDF-based languages. Therefore the server and clients know how to process the content according to its language and certain vocabularies. These vocabularies specifications (i.e., ontologies) are referenced in the content.

Hypermedia as the engine of application state (HATEOAS). The lack of native hypermedia support in RDF-based representations makes this the most challenging property to achieve [4, 91]. However, some recent works propose means to fulfil this property. Kjernsmo [62] proposes a vocabulary for hypermedia RDF. Steiner and Algermissen [103] and Verborgh et al. [108] propose to enrich the HTTP header with hypermedia information. While the first changes representations, the latter is a more general way to provide hypermedia.

Layered system (L). Encapsulation of functionalities can be achieved through a layered system. For example, to balance the load to a space replicated in two machines.

Code on Demand (COD). There are several cases where the *know how* can be downloaded from the server in TSC:

1. Modelling scripts using appropriate ontologies.
2. A semantic reasoner can be considered an interpreter for the content downloaded from the server. It is used to extract unstated information from the content received from the server. Therefore, the client downloads know-how to process the resource in the following situations:
   - Through the taxonomies defined in ontology files. These taxonomies are modelled using standard semantic languages.
   - Through semantically expressed rules [10].

Table 3.6 shows the conflicts that the analysed works present with these principles. None of them achieve the HATEOAS principle. In fact, its use in the SW is subject of current research. Most of them guarantee the access to the space in the CS basis. The only exception is tsc++ [69, 13], where each participant of the space is a peer in a P2P network. However, when the subscription/notification systems are offered as part of TSC, the CS property is certainly affected. Similarly, the statefulness of these middlewares conflicts with the transactionality feature. While the transactionality need in TSC can be argued, subscriptions are useful to ensure certain level of asynchrony in the system (see Section 6.1.2).

This dissertation proposes a return to the origins to recover the simplicity loss in the previous works. The rationale behind this decision is that resource constrained devices will benefit from this simplicity. In the proposed middleware they will be able not only to write and read knowledge into an external

Table 3.6: TSC's middlewares potential conflicts with REST principles.

The asterisk denotes a middleware which is not firmly compliant with the CS principle because it provides asynchronous notifications to the *clients*.

	CS	S	HATEOAS
TSC [25]	×*	×	×
SWS [105]			×
tsc++ [69, 13]	×		×
TripCom [102]	×*	×	×
Smart-M3 [54]	×*		×

space, but also to enrich it providing their own managed data. Table 3.7 summarizes the features of this middleware.

Table 3.7: Features of the solution presented in this dissertation.

Use of SW standards	✓
Tuple model	RDF triple-like tuples & RDF Graphs
Query model	Graph patterns
Space model	Flat
C/S access	✓
Distributed space	✓
Distribution strategy	Local management, distributed query

The original idea of the web was that it should be a collaborative space where you can communicate through sharing information.

Tim Berners-Lee

CHAPTER

4

Triple Spaces for constrained devices

In recent years, the Internet of Things (IoT) has become a reality due to the increasing number of everyday objects equipped with computing and networking capabilities. The use of these objects together with the rising of mobile computing greatly contributes to the UbiComp vision. This vision can benefit from the Triple Space Computing (TSC) paradigm's decoupling properties. However, other properties are derived from how TSC is materialized.

This chapter focuses on this materialization, i.e., the design of our own space model. To that end, Section 4.1 discusses the principles which guide our proposal. Section 4.2 proposes a mix model which encompasses *(a)* an independent space which can be used for coordination purposes, and *(b)* an enriched view of this space which also shows the information provided by autonomous devices. Section 4.3 describes a REST-like API to access the first space. Section 4.4 details the additional extensions needed to support the enriched view. Finally, Section 4.5 evaluates the properties achieved with this model.

4.1 Guiding principles

In Section 3.4.6 we showed how TSC can comply with all the constraints of the REST style. One of these constraints is that the access to a space is done in a client-server basis. The easiest way to achieve this constraint is by centralizing all the content of each space in the server itself.

However, data in a UbiComp environment is not generated at a single point. In fact, the plethora of sensors where data are generated can be enormous. In addition, each of these sensors may generate data continuously. This creates a trade off between efficiency and freshness difficult to overcome:

- The more frequently each sensor sends contents to the server, the more inefficient is the solution in terms of network usage.
- The less frequent this communication is, the less updated is the data in the server. This leads to spaces which misrepresent the environments.

Consequently, a sensible solution is to let these sensors manage their own information. This strategy is not only useful to ensure the access to up-to-date data, but also for the following causes:

- The devices directly connected to the sensors and actuators will know how to represent these contents better than others. For example, the unit of a temperature measure.
- They know when to create, update or delete data. Furthermore, they can opt for creating data on demand.
- Data may be reused by other spaces or even other applications. These applications would not be required to use a space-based approach. Therefore, they will not depend on the correct functioning of our whole system.
- Carrying the information can be useful in certain mobility situations. For example, let us imagine a person which stores her user profile in her personal smartphone. She could share it with different spaces or applications as it moves along the city.

Still, most of the IoT devices or mobile phones are unreliable: they can join and leave at any moment. As a result, distributing the shared space among unreliable nodes comes with a number of drawbacks:

- Devices rely on the data written and read from the space to coordinate themselves. Therefore, the access to these pieces of information must be guaranteed regardless of dynamic devices' availability.
- A blocking mechanism is important in space-based computing. For example, a worker node may block until some tasks become available in the space. A way to implement it in a distributed space is by means of a notification system. However, the efficient implementation of this system using unreliable devices is challenging.

Overcoming the previous difficulties, usually implies a high network traffic. This traffic negatively influences the energy consumption of nodes whose energy autonomy might be limited.

In conclusion, our design must face two apparently contradictory principles:

- To consider data from independent and limited data providers.
- To rely on the providers of the data which enables the coordination of the space participants.

4.2 A hybrid solution

We can distinguish two different usage patterns of a space: *(1)* coordinating with other devices by writing and extracting content; and *(2)* checking what happens in the environment by reading. So far, other semantic space-based works have always tried to integrate both strategies within the same space. However, the needs of both usages are different. The first one demands availability of the data regardless of its generator availability. The second does not. Therefore, we propose a novel approach: to treat them independently.

As explained in the previous section, limited devices are not reliable enough to manage data for the first usage. Nevertheless, they can contribute to an

enriched view of the space (i.e., the second usage pattern). Accordingly, we propose a dual model where: *(1)* the content needed to coordinate devices will be managed by independent machines; and *(2)* readings in the space will consider not only the previous content, but also the knowledge provided by autonomous providers. Summarizing, we propose to enrich the TS view with autonomous federated subspaces.

Figure 4.1 presents the key elements of this new model. The *coordination space* is where the graphs can be written, read and taken by any participant. The *coordination space* is held by one or many devices called *coordinators*. The *outer space* is composed by the graphs managed by independent and probably limited devices. We call these devices *asteroids* and their graphs *self-managed graphs*.

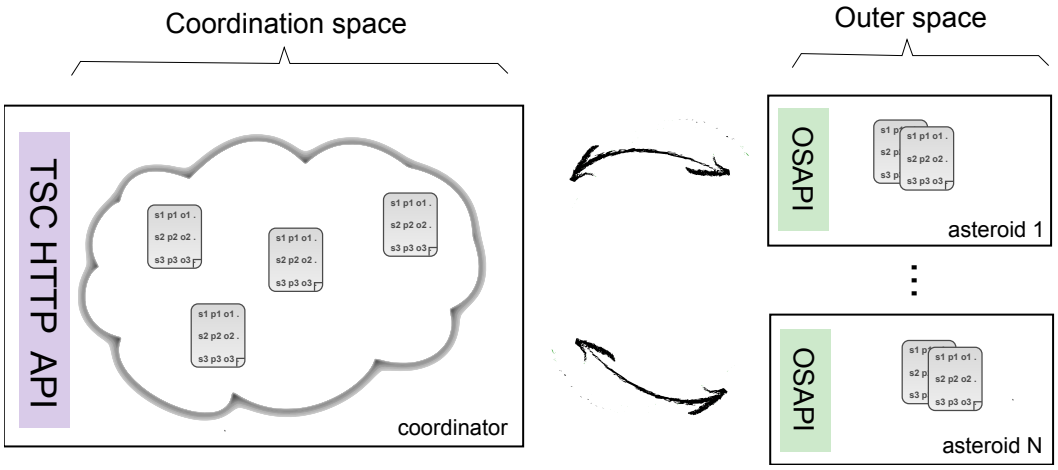


Figure 4.1: Key concepts of the new TSC model presented.

At networking level, we define a minimal contract between the *asteroids* and the *coordinators* through two APIs. Both are REST-like APIs. In our view, this choice's appropriateness for resource constrained devices is endorsed by WoT initiative.

For the *coordination space*, we propose a uniform access to the space through the HTTP API described in Section 4.3. This API can provide data from a distributed space like some approaches in Section 3.4.5 do. However, this dissertation

does not cope with this problem. Hence, the reader can assume that each space is managed by a unique server for the sake of clarity.

The *coordination space* will be enriched with data from the *outer space*. This data is exposed with the *OSAPI*, which must be implemented by any node willing to share *self-managed graphs*. The *outer space* demands an extension of the classical TSC model presented in Section 4.3. Section 4.4 presents this extension's new concepts, primitives and behaviour.

4.3 Aligning TSC with HTTP

This section presents our materialization of a TSC API over HTTP. This materialization gives a practical overview of TSC's RESTfulness.

4.3.1 TSC resources

Three key concepts are important to understand the resources in our proposal: agents share information in a common **space**. A space is identified by an URI. Therefore, all the operations in TSC are performed against a particular space. Within a space, the information is stored in sets of **triples** called **graphs**. Each graph can also be identified by an URI. Each triple is composed by a subject (which is a URI or a *blank node*), a predicate (a URI) and an object (which can be a URI, a *blank node* or a literal), as shown in the Figure 2.2.

As detailed later, TSC's primitives add or remove graphs. To perform these operations, which enable the selection of a subset of the semantic content hold in a given space, a **template** is required. The wildcard templates used are special triples with optional wildcard subject, predicate and/or object. For example, the template `?s foaf:knows gomezgoiri:aitor` could be employed to select instances which represent people who know Aitor (see Figure 4.2).

Note that this thesis does not consider using more sophisticated query languages like SPARQL [162]. The rationale behind this decision is that we wanted to avoid the complexity introduced by these languages in our API. While the advanced query languages need to be interpreted by parsers not available in

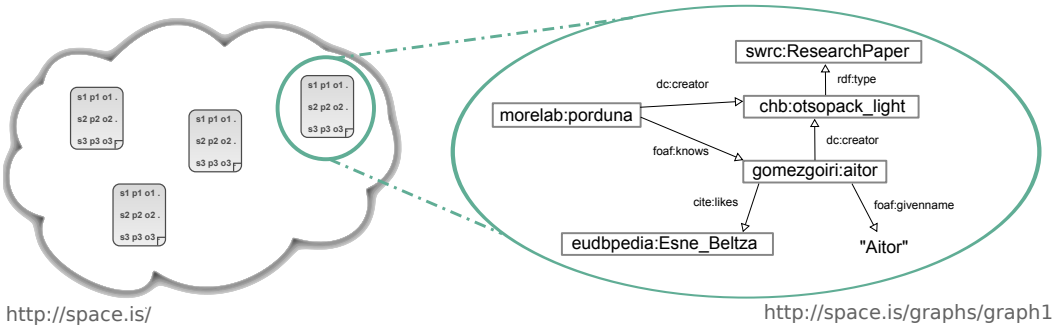


Figure 4.2: Representation of a space and the elements it is composed of.
The figure shows a space with four graphs and sample triples for one of these graphs. Note that the figure uses prefixes, i.e., aliases for the beginning of the URIs, for the sake of clarity.

many embedded platforms, wildcard templates are straightforward to implement and to process. This simplicity eases the adoption of the API by as many platforms as possible. However, since the wildcard templates are the base for the advanced query languages, our API could be extended to allow their use. In any case, this extension is left as a future work.

4.3.2 Adopted TSC primitives

TSC derives some primitives originally defined in the Linda language [31] to access the semantic information hold in each graph.

- The **write** primitive allows writing a graph into a given space (identified by its URI). The set of triples received by this primitive will be stored together in the same graph, returning the URI which identifies that graph. The graph URI can be used to access directly to that graph later on, or to create new triples and relate contents.

  `write(space_URI, triples): URI` [1]

- The **read** returns a graph belonging to a given space which contains at least a triple matching the given template or has the given URI as its identifier. If more than one graph fulfils one of these conditions, just one

of them is returned (nondeterministically). It should be remarked that it has been designed as a non blocking operation.

```
read(space_URI, graph_URI): triples          [2]
read(space_URI, template): triples           [3]
```

- The **take** primitive behaves like a destructive **read**, deleting the graph returned from the space.

```
take(space_URI, graph_URI): triples          [4]
take(space_URI, template): triples           [5]
```

4.3.3 HTTP API for TSC

As both TSC and HTTP are REST compliant, their similarities are evident. In the same way TSC has the already explained primitives, HTTP has verbs to get, create, update or remove resources (GET, PUT, POST, DELETE). Consequently, the translation between these two worlds is straightforward.

According to the REST principles the interaction with an API must be hypertext-driven [28]. To ease its usage by developers, we provide a human-oriented HTML representation of the API which is completely *browseable*. Regarding the machine-oriented representation of the API, Verborgh et al. [108] and Kjernsmo [62] have proposed solutions to enable hypermedia driven semantic APIs. Independently of the mechanism adopted, in this section we propose a optional API which stresses TSC's compliance with HTTP.

The list of spaces a node is joined to are available under */spaces*. Each space is identified by an URI (e.g., `http://space1`). If this URI is also a URL, */spaces/*{*space_uri*} (summarized by *sp* from now on) can simply redirect to it. All the resources of that space, both real (i.e., graphs) or virtual (i.e., query) are listed under {*sp*}/. Each graph is available on {*sp*}/*graph*/{*graph_uri*}. If we make an HTTP DELETE to that resource, under TSC's perspective, we take that graph from the space. The rest of the mappings are shown in the Table 4.1.

Table 4.1: HTTP mapping for the primitives detailed in the Section 4.3.2.

sp is a space URI, *g* is a graph URI, *s*, *p* and *o-uri* are subject, predicate and object URIs or wildcards (represented with an as *). When the template's object is a literal, it can be expressed specifying its value (*o-val*) and its type (*o-type*).

HTTP request	URL	Returns
POST	{sp}/graphs/	[1]
GET	{sp}/graphs/{g}	[2]
GET	{sp}/graphs/wildcards/{s}/{p}/{o-uri}	[3]
	{sp}/graphs/wildcards/{s}/{p}/{o-type}/{o-val}	
DELETE	{sp}/graphs/{g}	[4]
DELETE	{sp}/graphs/wildcards/{s}/{p}/{o-uri}	[5]
	{sp}/graphs/wildcards/{s}/{p}/{o-type}/{o-val}	

4.3.3.1 **Status codes**

The API should be compliant with the standardized HTTP status codes [150]. These codes are sent back in the response as part of the header. For instance, the TSC middleware returns the *404 error* when no significant result can be found for a primitive. This adoption, apart from enhancing the compatibility with other web applications, can enable the modular adoption of the API. For example, if a space does not offer a wildcard based *read*, it can simply return a *501 Not Implemented*. The participants would then be aware of the problem and use an alternative primitive to obtain the data needed. This modularity becomes crucial to ease the partial adoption on new platforms.

4.3.3.2 **Content negotiation**

Another key aspect of the HTTP protocol that our API should take advantage of is the *content negotiation*. This mechanism allows to specify the desired representation for a content on the client side and to express what representation is sent as a response from the data provider side. For that purpose, the client adds an *Accept* field to the HTTP header with a weighted list of media types it knows how to interpret. Then, the server will answer with the best possible format it understands, specifying the *Content-type* in the response.

The benefits of using this mechanism in the TSC middleware presented are two-fold. Firstly, it enhances the browsability of the primitives with human understandable HTTP responses. Secondly, it allows different semantic representations (e.g., RDF/XML [160], N-Triples [152] or N3 [145]). The latter characteristic becomes crucial since not all the nodes may understand all the formats (e.g., a mobile phone may not have a RDF/XML parser). In these cases, the compatibility of both sides can be ensured through a conversion carried out in the server side. Furthermore, expressing the preference for a semantic format can be useful too in other cases. For example, to obtain the less verbose answer.

4.4 Federated space

The *OSAPI* extends the API previously presented with the primitives and concepts explained in this section.

4.4.1 Self-managed graphs

These graphs are shared with other participants, but can only be managed by the devices called *asteroids*. In other words, *self-managed graphs* enrich the space but cannot be externally written or removed. Therefore, they are *second-class graphs* which provide information about the environment but cannot be used for coordination purposes.

Each *asteroid* makes these graphs accessible to others through HTTP. The final goal is to potentially allow to reuse the data provided by any existing RESTful service. Therefore, the API should be or tend to be RESTful. However, we leave the *hypermedia* API as a future work. Instead, we require a mandatory *OSAPI* to be implemented in each *asteroid* to guarantee access to the *self-managed graphs*.

4.4.2 New primitives

To make the most of the information in a space, we propose a new primitive to query all the semantic information stored in the space. The RESTfulness

Table 4.2: HTTP mapping for the *query* primitive.

sp is a space URI, *s*, *p* and *o-uri* are subject, predicate and object URIs or wildcards (represented with an as *). When the template's object is a literal, it can be expressed specifying its value (*o-val*) and its type (*o-type*).

HTTP request	URL	Returns
GET	{sp}/query/wildcards/{s}/{p}/{o-uri}	[6]
	{sp}/query/wildcards/{s}/{p}/{o-type}/{o-val}	

of this primitive could be argued since it does not operate at resource level (i.e., returning graphs), but mixing several resources (i.e., triples from different graphs). However, we believe that it is useful to have an endpoint for the queries which involve many graphs. Kjernsmo [62] discusses this topic in depth.

This new primitive is defined as follows:

- The **query** primitive aims to see the space as a whole, returning all the triples matching the given template.

  `query(space_URI, template): triples` [6]

The Table 4.2 extends Table 4.1 to include this new primitive.

A key point of the API is that the *asteroids* might not even follow the TSC paradigm. For instance, the *OSAPI* can encapsulate data provided by a third middleware. However, a primitive to ease that management can be a convenience for the developers which do not need a more customized behaviour. With that in mind, we propose another writing primitive. This primitive only has local effects and therefore has no HTTP equivalent:

- The **write_self** primitive writes a *self-managed graph* and returns an URI which identifies it.

  `write_self(space_URI, triples): URI`

- The **read_self** and **take_self** primitives only affect to *self-managed graphs*.

```
read_self(space_URI, graph_URI): triples
read_self(space_URI, template): triples
take_self(space_URI, graph_URI): triples
take_self(space_URI, template): triples
```

4.4.3 New behaviours

This section tries to clarify how the different behaviours coexist:

Writing. The most basic writing primitive allows a client to remotely write a graph into the *coordination space*. However, we also presented the *write_self* primitive. *Write_self* locally writes into an *asteroid* an externally untakeable graph (i.e., *self-managed graphs*).

Reading. *Query* performs a traversal query over the aggregated view of all the graphs in a space (including the *self-managed graphs*). *Read* and *take* work at resource level. *Read_self* and *take_self* are their equivalents for *self-managed graphs*. Finally, the *read* primitive's results will be enriched with *self-managed graphs* from the *outer space*.

4.5 Evaluation

With the model presented in this chapter, we aimed to retain the desirable properties of both REST and TSC. However, some questions arise from this integration: *(1)* does it in fact retain all the properties of REST and TSC separately? Otherwise, which properties are affected? *(2)* which other benefits does it offer compared with the usage of separate TSC and REST middleware?

To answer these questions, we analyse the presented solution from different points-of-view: *(1)* its coordination properties (Section 4.5.1), *(2)* its network–level properties (Section 4.5.2), *(3)* how the latter work together to contribute to the challenges of an UbiComp environment (Section 4.5.3).

4.5.1 Coordination properties

As has already been discussed in Section 2.1, indirect communication middleware can have two key properties:

- The sender does not need to know the receiver or receivers and vice versa (i.e., *space uncoupling*).
- Senders and receivers do not need to exist at the same time to communicate with each other (i.e., *time uncoupling*).

The primitives used in our space-based computing implementation force it to be *space uncoupled*. On the contrary, it is not always *time uncoupled*. If two nodes write and read from the *coordination space*, they are *time uncoupled*; but if a node accesses to others' content (i.e., *self-managed graphs*), they are not. In other words, *time uncoupling* is not achieved by the extension of the model presented in Section 4.4 (i.e., in the *outer space*). As a future work, this could be alleviated by a caching mechanism implemented in the *coordinator* or *coordinators*.

Table 4.3: Uncoupling levels achieved by the different parts of the middleware presented.

	Space uncoupling	Time uncoupling
Coordination space	✓	✓
Outer space	✓	×

4.5.2 Networking properties

Section 3.4.6 described how TSC does not intrinsically contradict any of the REST principles. However, the adaptation presented in this thesis does not completely adhere to the REST style. This section presents how these divergences affect to REST's properties.

In Fielding's words, the relevant properties which describe a network-based system are the following ones [1]:

Performance is divided into network performance, user-perceived performance and efficiency.

- **Network performance** is affected by the number of interactions and the granularity of data elements.
- **User-perceived performance** refers to the impact perceived by a user in front of an application.
- **Efficiency** is achieved by minimizing the use of the network.

Scalability measures how an architecture supports a big amount of components and interactions between them.

Simplicity is achieved through the separation of concerns for the components and the generality of architectural elements.

Modifiability indicates how easily gradual changes can be introduced in the system. These changes contribute to form different implementations which should coexist.

- **Evolvability** refers to the degree in which a component can be implemented without negatively impacting on others.
- **Extensibility** measures the ability to add functionality to the system.
- **Customization** is the ability to adjust the behaviour of an architectural element temporarily.
- **Configurability** is related with extensibility and reusability.
- **Reusability** is the ability to reuse components, connectors or data elements without modifying other applications.

Visibility is the ability of a component of monitoring or mediating in the interaction between other two components.

[1] We refer to the reader to Fielding [26] for the complete thorough analysis.

Portability is the ability of working in different environments.

Reliability is the degree in which an architecture depends on the failures of the system or components, connectors or partially incorrect data.

Table 4.4 summarizes how different architectural styles achieve these properties. Particularly, it shows the styles from which REST derives (see Section 2.2). The ultimate goal of the solution explained in Section 4.2 is twofold:

1. Provide a REST access to a semantic space. This would ease its integration with the rest of the web.

2. Enrich that space with the knowledge provided by other REST APIs.

Therefore, ideally, its networking properties would be the same as the REST style.

4.5.2.1 REST or REST-like API?

Defining a 100% REST compliant API may be challenging. Indeed, as explained in Section 2.2, most of the self-proclaimed RESTful APIs are not [55]. Most of them fail to achieve the HATEOAS constraint.

Regarding the SW, some recent efforts have tried to move closer to the *hypermedia* constraint [103, 62]. However, these worlds remain quite isolated. In fact, SW APIs usually present another main divergence with the REST style: the use of query endpoints. These endpoints intend to solve some inefficiency issues which REST shows when working with a big amount of data. To that end, they allow expressive query languages and offer results where the boundaries of the different resources often blurs [115].

In the UbiComp both the efficiency on the communications and the reusability of the API are important:

[1] *S* represents the difference between *CSS* and *CS* in [26].

[2] Corresponds to the *C$SS* style in [26].

[3] Although it is not explicitly included in the original table, *U* has been derived from Fielding's description.

[4] Derived from the addition of *U* to *LCODC$SS*.

Table 4.4: Properties of different architectural styles for network-based applications.

This table is an adaptation of the one originally conceived by Fielding [26]. These adaptations are remarked inside the table.
Each plus symbol (+) represents a positive influence and each minus symbol (-) a negative one. Plus-minus (±) denotes that it depends on some aspect of the problem domain.
The leftmost column contains each of the network substyles REST derives from: Client-server (CS), Stateless (S), Cache ($), Uniform interface (U), Layered system (L) and Code on Demand (COD).

The horizontal line indicates that the immediate row below is composed by all the rows from the upper level.

Style	Net Perform	UP Perform	Efficiency	Scalability	Simplicity	Evolvability	Extensibility	Customiz.	Configur.	Reusability	Visibility	Portability	Reliability
CS				+	+	+							
S[1]	−			+							+		+
$		+	+	+	+								
Early web[2]	−	+	+	++	+	+					+		+
L		−		+		+				+		+	
COD		+	+	+	±		+		+		−		
LCODC$SS	−	++	++	+4+	+±+	++	+		+	+	±	+	+
U[3]					+					+	+		
REST[4]	−	++	++	+4+	+±++	++	+		+	++	+±	+	+

- *Efficiency*: mobile and embedded devices have restricted energy autonomy. This autonomy is severely affected by network communications. Particularly, the access to the data they provide by means of hypertext may result in many HTTP requests.

- *Reusability*: due to the heterogeneity of the devices, assuming they all share a common and unevolvable API may not be realistic. Normally, an environment will be populated by different APIs from many WoT solutions. In that situation, allowing a client to autonomously learn how to use them would be ideal.

We opt for promoting efficiency at the expense of reusability. In other words, we use two REST-like APIs whose use is known out-of-band by the clients (i.e., they are not guided by the *hypermedia*). However, we provide a human-oriented version of our API to ease its use by developers which does comply with HATEOAS.The evolution of the APIs to fully RESTful machine-oriented ones is left as future work.

In conclusion, the properties achieved by the network communication style selected corresponds with *LCODC$SS* plus simplicity and visibility (see Table 4.4).

4.5.3 **Properties for UbiComp**

A middleware is a software layer which provides a higher level of abstraction and masks the underlying heterogeneity. The middleware presented in this dissertation is oriented to UbiComp environments and the devices which populate them (particularly mobile and embedded devices). Whereas the devices part of the IoT are a subset of the ones present in UbiComp, we observed that the challenges they have to cope with are the same ones. For instance, smartphones are not part of the IoT, but cope with similar energy and computational limitations as the embedded devices.

Therefore, we will consider the challenges identified by the *Internet-of-Things Architecture* European project [112] to analyse our solution. Walewski et al. state that the IoT must overcome the following challenges: interoperability, scalability, manageability, mobility, security and privacy and reliability.

Furthermore, we also consider energy-related and computational constraints to be addressed by devices in UbiComp.

Table 4.5, shows how the properties explained in the previous sections directly affect these challenges. The following sections clarify this table.

Table 4.5: Direct relations between the properties analysed in previous sections and the challenges a lightweight middleware faces.

	Network prop.							Coord. prop.		
	Performance	Scalability	Simplicity	Modifiability	Visibility	Portability	Reliability	Space uncoupl.	Time uncoupl.	Semantic Web
Interoperability				×		×				×
Scalability		×								
Manageability										
Mobility								×	×	
Security										
Reliability							×		×	
Limited comp.										×
Limited energy	×									×

4.5.3.1 **Interoperability**

Interoperability is the key to deal with a wide range of heterogeneous technologies. The use of both the **network style** and the **semantic data** contribute to this challenge.

4.5.3.1.1 Semantics

Using semantics we promote the reuse of data describing them in a more rich and abstract way. The semantic web is composed by a series of standardized

technologies which allow to formally describe the models (i.e., the concepts and how they relate to each other). Two key mechanisms of the Semantic Web in this aspect are the inference of new data and the mapping of equivalent models.

4.5.3.1.2 Networking

For the communication between nodes, we rely on the widely supported HTTP protocol. Communication with devices using other protocols must be done by means of specialized gateways. Therefore, adhering to a unique protocol does not help dealing with various communication technologies.

However, a wide range of current applications use HTTP to define their network accessible API's. This makes it a *de facto* requirement to interoperate with these applications. In fact, this success has also its counterpart on resource constrained platforms. The clearer sign for this tendency is the upcoming WoT initiative.

At a finer grained level, there are several properties which help different HTTP implementations to interoperate:

Modifiability is the ease for introducing gradual changes in the system. Particularly dynamic modifiability avoids restarting the entire system when introducing a change. This property allows different implementations to coexist reducing issues on the communication between two nodes.

Portability: HTTP's portability is backed by the plethora of HTTP libraries and frameworks available for most of the computing platforms. Consequently, a huge number of platforms can interoperate through HTTP.

In addition, in our HTTP API the following issues remain unsolved:

- *Using third-party applications' data*:

    1. Section 4.5.2 details how a pure REST API contributes to a higher decoupling between clients and servers. In REST architectures clients can autonomously adapt to use different APIs or to different

API versions. Of course, this comes at the cost of more complex clients. These clients have the additional responsibility of discerning the next state transition from the representations obtained.

Our middleware is not hypermedia-driven. As a consequence, it requires third-party applications to implement a uniform API (i.e., *OS-API*) on top of them to allow our middleware to consume their data. However, the API is resource-oriented and simple to implement.

2. Another obstacle for achieving a real application-level interoperability in our middleware is the use of distinct isolated spaces. In other words, two applications using two different spaces would not share their content. Although this could be avoided by defining a default space, as explained in Section 3.4.4, their use is justified in terms of scalability.

- *Making our data reusable by third-party applications*:

  We encourage sharing data in resource constrained devices through an HTTP API. Using this API any other application able to use HTTP and to manage semantics can reuse data from our middleware. However, since this API is not hypermedia-driven either, third-party applications must know how to use it beforehand.

4.5.3.2 **Scalability**

The architecture of the web is scalable by design. Indeed, the large number of components and interactions between them that coexist in the web are a good proof this. These components share an even larger amount of data which leads to scalability issues on the search process [68]. In fact, considering that in UbiComp environments the data is constantly generated or updated, this challenge can be harder. However, these challenges and other considerations related to the search process are analysed in the next chapter.

4.5.3.3 **Manageability**

Manageability helps to cope with a big number of devices allowing them to have an autonomous behaviour. It encompasses self-management, self-configuration, self-healing, self-optimization, and self-protection. This dissertation proposes a self-managed architecture which eases searching self-managed graphs.

4.5.3.4 **Mobility**

The ability to cope with mobility situations can be improved through the decoupling brought by space-based computing. Firstly, mobility situations may impede two nodes to coexist at the same time. In this case, *time uncoupling* enables their communication.

Secondly, the provider of a piece of information may vary over the time. The primitives used care about the data and not about the specific provider of these data (i.e., the nodes are *space uncoupled*). This avoids any reconfiguration of the data consumer when the provider is replaced. In fact, a content can be also delivered by more than a provider.

4.5.3.5 **Security and privacy**

This property is considered out-of-the-scope of this dissertation. However, we are currently working on a lightweight security solution [81]. For our future work, we are considering to integrate it with the presented middleware.

4.5.3.6 **Reliability**

Reliability in UbiComp is crucial to handle connectivity losses in various *ad hoc*-like ways. Particularly, HTTP handles it through caching.

4.5.3.7 **Limited computation**

Mobile and embedded devices impose computation restrictions. These restrictions result in unacceptable response times from the application perspective.

Due to HTTP's simplicity, computation complexity is found in semantic processing. For example, some really limited platforms may find difficult parsing long files. However, we made the following assumptions:

- Data providers will be able to process wildcard-based templates and generate appropriate responses from their self-managed graphs.
- Data consumers are able to process the responses they receive at least in one semantic format, e.g., N3 [145] or RDF/XML [160].

Beyond that point, our middleware will not impose anything. This means that a developer using the middleware cannot assume:

- The inference process on data providers. Reasoning over the content to provide unstated implicit knowledge must be desirable. Still, most of the mobile and embedded devices are not able to process big amount of data. Furthermore, reasoners are not available or optimized for many limited platforms.
- The inference process on data consumers. Another alternative or complement for inferring data on the providers is to do it in the consumers after receiving the responses. Again, it is a desirable property, but our middleware does not force it.
- General use of more complex query languages. Their expressibility benefits the network performance since they can restrict the results returned. However, parsing them is not trivial and consequently parsing libraries are unavailable for most limited platforms. Therefore, a provider cannot assume that providers know how to process something beyond the wildcard based templates. As an alternative, providers can decompose these queries in simpler wildcard based templates and locally process the results afterwards using the complex queries as filters.

The searching mechanism explained in the Chapter 5 takes into account the computing limitations of embedded and mobile devices.

4.5.3.8 **Limited energy**

Both data computation and networking operations directly affect energy consumption. As an example, Figure 4.3 shows the energy consumption on an embedded platform in an inactive period, whilst doing networking operations and when computing semantic data. There are some crucial characteristics which negatively affect energy autonomy of resource constrained devices:

- Too complex computation tasks. As explained in the previous section, semantic processing is the clearest example of that. We have limited its impact on resource constrained platforms through the searching mechanism.

- Regarding HTTP, its statelessness contributes to a bad **network performance**. In addition, the verboseness of most of the semantic formats also contributes negatively to that aspect.

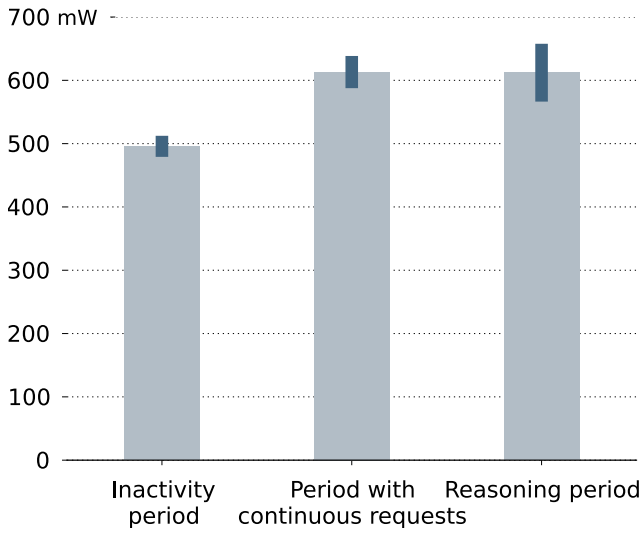


Figure 4.3: Energy consumption for an embedded platform.
Energy consumption, measured in miliwatts, for a FoxG20 [126] during different activity periods.

On the other hand, the network usage reduction contributes to energy savings (i.e., improves *network efficiency*):

- HTTP's network *efficiency* can be improved with *code on demand* and *caching*.
- The searching mechanism improves *recall* and *precision* (see Chapter 5).

4.6 Summary

The previous sections have reviewed the strengths and weaknesses of the proposed model. With regards to other semantic HTTP-based solutions, the solution presented provides a better handling of mobility situations. The uncoupling properties provided come at the cost of simplicity to implement the middleware. The main strength of our TSC model is that it enriches the knowledge shared in a classical space model with content provided by autonomous embedded and mobile devices. This opens de door to interoperate with other HTTP-based applications.

Within its weaknesses we found performance problems and a coupling between applications due to a non hypermedia-driven API. The performance problems may result of handling semantic data and are particularly important for resource constrained devices. The coupling due to a mandatory HTTP API complicates reusing third-party applications' data.

The benefits of bringing together TSC and HTTP is an unified and simple API for the developer. Due to the primitives' simplicity and to the standard and well accepted technologies used (i.e., HTTP and SW standards), this API can be ported to a range of platforms.

I woke up one morning thinking about wolves and realized that wolf packs function as families. Everyone has a role, and if you act within the parameters of your role, the whole pack succeeds, and when that falls apart, so does the pack.

Jodi Picoult

CHAPTER

5

Searching in a distributed space

In Chapter 4 we presented a TSC model for UbiComp environments which is composed of two spaces: the *coordination space* and the *outer space*. The *outer space* is formed by the semantic data provided (*self-managed graphs*) by any participating device. Searching for a graph in that *outer space* is not trivial because it is composed by dynamic and unreliable devices.

This chapter presents an architecture to enable searching for semantic content in a decentralized energy efficient manner. This searching mechanism can be generalized to other WoT solutions and scenarios. To empathize this, during the rest of the chapter we avoid explicitly mentioning the TSC middleware or related concepts (e.g., *outer space*).

This chapter is organized as follows: Section 5.1 gives an overview of the problem addressed. Section 5.2 discusses the related work. Section 5.3 presents in detail our energy-aware architecture. Section 5.4 describes the information that devices exchange to maintain this architecture. Section 5.5 presents our experimental environment and Section 5.6 evaluates our solution. Finally, Section 5.7 states the conclusions of this chapter.

5.1 Introduction

Integrating mobile or embedded devices is not trivial as they usually communicate using different protocols. To solve this problem, the Web of Things initiative proposes to use well-established web standards to ease their communication. However, the format of the data they exchange is also multifarious and application domain dependent. This implies that data will not be meaningful in other domains unless a specialized system converts and reinterprets them. A way to solve this problem is annotating the data semantically as proposed by the WWW.

Adding semantics to the IoT works well for devices with high computational capacity but it adds too much overhead for most of the devices composing the IoT. To reduce this overhead in such devices, part of this computation is usually delegated to intermediaries [54]. This approach reduces the overhead of semantically annotated data but brings other problems:

1. When devices rely on others to provide information, it is not guaranteed that the information accessed will accurately represent the last information available in the data providers (e.g., the sensors).

2. Once they rely on intermediaries, these intermediaries must be available at all time. Otherwise, the devices would not be able to talk to each other.

We propose a solution where intermediaries are used to release some workload from the less powerful devices, but at the same time, the direct communication between the devices is promoted. In particular, our system uses intermediaries to search where the data is located in it and then queries the final devices directly.

In our solution, the devices can become or stop being intermediaries dynamically. To decide which device will be an intermediary, we evaluate the state of a device (i.e., energy and computation capacity). Using this dynamic architecture, the absence of a particular intermediary does not collapse the system. In addition, our system is flexible enough to support a wide range of scenarios.

To enable the search for information in this system, intermediaries need to know the information available in it. To do that, they aggregate summaries

sent by each device. We propose alternatives to summarize and aggregate this information taking into account the payload of the shared information and the accuracy of the search.

Our approach is compared against other common approaches. In particular, we focus on the energy aspect, demonstrating that our approach helps energy constrained devices to cope with fewer unnecessary requests. We also evaluate other important metrics like the number of messages that a device has to exchange to perform a request, and the accuracy of the search results provided by the intermediaries. We perform this evaluation under different scenarios to prove the flexibility of our proposal.

In summary, we make the following contributions:

- Present a new architecture for managing semantics on the WoT which reduces the load for resource-constrained devices.
- Propose and evaluate different approaches to aggregate and summarize the semantic information available in the devices and enable semantic search in this architecture.
- Demonstrate the advantages of our approach compared to other typical searching approaches under different scenarios.

5.2 Background

Querying over the semantic content provided by independent sources transparently is a problem which often appears in the Linked Open Data (LOD). Görlitz and Staab [50] classify the possible LOD infrastructures according to *(1)* how they store data, *(2)* whether the index used to search is distributed, and *(3)* whether data sources cooperate. Table 5.1 summarizes the resulting infrastructure types: central repository, federation and P2P data management.

The solutions which distribute the semantic content are federation and P2P data management. While the latter distributes the index to look for content in multiple machines, federation does not. Our solution proposes a federation infrastructure with some particularities:

Table 5.1: Infrastructure paradigms according to their characteristics.

Note: This table was created by Görlitz and Staab [50].

<table>
<tr><td></td><td colspan="2">Central Data Storage</td><td colspan="2">Distributed Data Storage</td></tr>
<tr><td>Independent Data Sources</td><td>n/a</td><td rowspan="2">Central Repository</td><td rowspan="2">Federation</td><td>n/a</td></tr>
<tr><td>Cooperative Data Sources</td><td>n/a</td><td>P2P Data Management</td></tr>
<tr><td></td><td>Distr. Index</td><td colspan="2">Central Index</td><td>Distr. Index</td></tr>
</table>

- There is a unique node in charge of managing the main version of the metadata needed to create indexes. However, it is dynamically chosen among all the participants and can change over the time. Consumers hold copies of the manager's metadata.

- Each consumer builds its own index from its metadata replica. This allows consumers not to critically depend on the availability of any other node to search.

FedX [100, 101], DARQ [96] and SemWIQ [71] are the most relevant solutions to perform federated queries over independent semantic data sources. DARQ and SemWIQ require locally preprocessed metadata about data sources, while Fedex uses on-demand queries together with a caching mechanism. In that aspect, our solution resembles DARQ and SemWIQ since it also requires this metadata (called *clues* in our solution). In fact, we propose two types of *clues* which are equivalent to the metadata used by DARQ and SemWIQ. DARQ maintains predicate indexes to find relevant data sources. SemWIQ maintains a local catalog with the type information of RDF entities provided by data sources. In contrast with our solution, DARQ and SemWIQ also use statistics about data sources to optimize their queries.

A key difference with the mentioned systems is the context where we have applied our solution. It is intended to serve in UbiComp environments. This means that there will be many more data sources and with less information to process in each of them. Furthermore, their nature also differs from the usual

LOD endpoints' one: the mobile and embedded devices which may provide data are less reliable, more dynamic and more restricted in computation and energy. Therefore, we designed and evaluated our solution considering this nature and restrictions. To the best of our knowledge, this is the first approximation to the problem within the explained context.

Considering the constrained architecture of the data sources, we acknowledge that some of them will not be able to process SPARQL [162]. Consequently, we require the lowest common denominator: basic triple patterns. This causes the main weakness of our solution compared to FedX, DARQ and SemWIQ: a less sophisticated querying mechanism. For our future work, we are considering using SPARQL just with the endpoints able to manage it as a halfway solution. This will allow us to introduce some of the optimization techniques presented by other solutions.

In sections 3.3.1 and 3.3.2 we have analyzed other solutions for UbiComp which use semantics. However, none of them have addressed federated distributed search in resource constrained devices.

5.3 Energy-aware super-peer architecture

In a semantic WoT, nodes are part of a network where they share semantically described information. These nodes gather their information in *Spaces* [1].

For example, a hotel can create its own semantic *Space* with the semantically described information of its services and the data provided by its devices (e.g., sensors) spread around the hotel. Thus, clients can use their own personal devices (e.g., smartphones) to interact with the devices of the hotel and search useful information. Using this approach, a client, for example, can check the current swimming pool occupancy rate using his personal mobile phone.

[1] Note that in this chapter, the term *Space* refers to a group of devices usually co-located which form an intelligent environment. It has an obvious correspondence with our solution's *outer space*. However, note that despite of this correspondence, the solution presented in this chapter transcends space-based computing.

However, not all devices can afford the costs of fully managing the additional overhead of semantics. This section presents our architecture to manage such environments and deal with the limitations of resource-constrained devices.

5.3.1 Basic roles

Devices can be part of one or multiple *Spaces*. A device can provide data (*Providers*), consume data (*Consumers*) or both at the same time.

Providers: This is the simplest role which any device in our system can carry out, even the smallest sensor. *Providers* must manage their own semantic information. In particular, they must organize triples in RDF graphs.

Consumers: This is the role a device must take to get information from the system. These devices need to be able to use semantics to find the data they require.

Note that a device could hold just one or both roles at the same time or switch between them. For example, a *Provider* could become a *Consumer* to get data from the system or a *Consumer* could stop asking for data.

5.3.2 Use of intermediaries: White Page

Tasks involving the use of semantics can be expensive for some devices. To reduce the load on such resource-constrained devices, we need to use intermediaries to carry out some of these tasks. However, we do not want devices to completely rely on intermediaries for three reasons:

1. Data consumers need to get fresh data and we can only get this by direct access to the actual *Provider*.

2. We must support mobile scenarios where personal devices carry their own data.

3. We need to reduce the maintenance complexity (nodes may join or leave at any time).

For these reasons, we cannot rely on intermediaries to host and provide *all* the semantic information. However, we use intermediaries as searching enablers in the semantic *Space*. This kind of intermediary is what we call White Page.

White Pages (WPs) manage metadata or *clues* about the information shared by others. The *Consumers* rely on WPs to enhance the search process and reduce the number of requests generated. Consequently, the *Providers* process less semantic data and reduce their overall overhead. Thus, our architecture enables an intermediary-aided energy-aware information search.

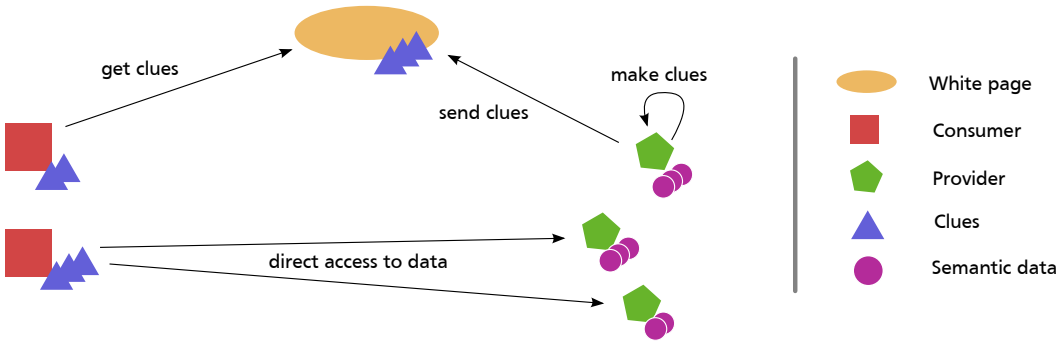


Figure 5.1: Role of White Pages in our proposal.

Figure 5.1 shows the role of WPs in our proposal. In particular, WPs manage *clues* which summarize the semantic information shared by the *Providers*. These *clues*, which are described in detail in Section 5.4, are pieces of information useful to determine which nodes can answer to a certain query. *Consumers* use these *clues* to directly access the semantic information on the *Providers*. Summarizing, the main tasks of a WP are:

- Manage the *clues* sent by all the nodes in the system.
- Aggregate clues sent by *Providers* in a unique response.
- Reply to requests from *Consumers* with aggregated clues or a list of nodes (details in Section 5.3.5).

WPs only store *clues* for a period of time, expiring afterwards. Using this approach, we avoid storing information from no longer available nodes.

5.3.3 **Versioning clues**

As previously explained, the WP aggregates clues sent by *Providers*. This aggregation is versioned using a *version number* and a *generation number*. The *version number* increases each time the WP receives a new clue update. The *generation number* (g_{id}) is a timestamp that represents the moment a new WP was chosen. This requires the clocks to be synchronized in all the nodes to avoid problems. In any case, the WP must guarantee that the g_{id} is higher than the one used by a previous WP.

The WP maintains two version numbers: *(1)* the one used during the setup to do the first load (*setup version*) and *(2)* the version of the last aggregated clue. The first one is shared through the discovery mechanism. The second one is shared with both *Providers* and *Consumers* in HTTP responses. The versioning is used to improve the WP selection process explained in Section 5.3.6.

5.3.4 **Discovering a White Page**

In our architecture, when a *Consumer* joins the *Space* it relies on a WP to find information. Hence, the first thing a node needs to do afterwards is discovering who is the WP in that *Space*.

To run our proposal, we require a discovery system able to *(1)* get the *Spaces* that a particular node belongs to, *(2)* identify the WP in the system and its *setup version*, and *(3)* provide additional information about nodes to decide which one can be the next WP. To implement our architecture, the information about the nodes must include: battery level, available memory, storage space and the approximate time passed since the node joined the *Space* for the last time (to estimate its reliability).

How to discover the nodes in the *Space* is transversal to this dissertation. We could either extend approaches like Universal Plug and Play (UPnP) [142], Multicast DNS (mDNS) [147] or lmDNS [58] or use HTTP, CoAP [156] and

DNS together as proposed by Ishaq et al. [56; 57]. In particular, in Section 5.6.5 we use mDNS to evaluate our solution.

Note that we assume that any node in the *Space* can reach to the other nodes. In particular, we assume IP addressability (no matter whether the node is wired or wireless). As a consequence, to use devices from Wireless Sensor Networks (e.g., Zigbee or 6LoWPAN) as *Providers*, one should rely on gateways.

5.3.5 Interacting with a White Page

When a node needs to access a piece of information, it asks the WP for the information about the other devices in the *Space*. Using this information, the node can find the owner of the required data and ask it for this information.

In our architecture, *Providers* and *Consumers* have additional duties.

Providers. They manage their own semantic information and generate *clues* about the information they host. Then, they send these *clues* to the *Space* WP. As a response, they receive the last version of the aggregated clue they have contributed to. *Providers* will send their *clues* to the WP *(1)* every time a *clue* is updated, *(2)* before the lifetime of a *clue* expires, or *(3)* whenever there is a new WP in the *Space* with a lower *setup version* than the one in the *Provider*. Note that *clues* do not change frequently since they represent the type of information the nodes host rather than the specific data which is constantly generated.

Consumers. When a *Consumer* needs to get information from the system, it first needs to find the WP in the *Space* and use this WP to obtain an aggregated clue of all the nodes. Then, it processes this aggregated clue to decide which are the nodes to query. Finally, it sends the query through HTTP requests, one per provider identified in the previous step.

They perform this process synchronously for the first query and periodically in an asynchronous manner for the following ones. This period should have an upper limit to ensure a fresh view of the *Space* and a lower limit to guarantee that the WP is not flooded. To adjust the update frequency within these two limits, we evaluate the frequency of the last 10 requests to that node. Thus, the view of the *Space* will be fairly up to date when the *Consumer* processes the next query.

Optimizations for resource constrained devices. Note that *Providers* will update the WP with new information and *Consumers* will periodically check the WP. Using this approach, the WP reduces the network load and in particular, it decreases the load for resource constrained devices.

In addition, *aggregated clues* can be too long for some really limited devices. As an optimization for such devices, the WP is able to answer specifically the node to address a query. Thus, these nodes will only maintain a list of the nodes they should ask for several predefined queries.

5.3.6 Selecting a White Page

Depending on the setup, having a dedicated WP (e.g., a high-end server) can be too expensive. For example, in a domestic environment, it is not worth dedicating a full server to be a WP. However, when we have a large setup (e.g., a hotel), it is convenient to dedicate a few servers to decrease the load in small devices.

We have to be able to manage the complexity of a system composed by heterogeneous devices. For example, making a small device a WP may be inappropriate in highly populated environments. On the contrary, having multiple dedicated servers implies a high management overhead which is unnecessary in simple environments. Our architecture is flexible and any node can be a *Provider*, a *Consumer*, a WP or any of them at the same time.

When does the WP selection start? The selection process can start *(1)* when no WP is available or *(2)* when the current WP gets a worse score than other nodes. In the first case, the first node to realize there is no WP starts the selection process. In the second case, the current WP periodically checks if there is another node with a better score. If this actually happens, it starts the process. We limit the frequency for this checking to avoid the overhead associated with this change.

How to select a WP? The node in charge of selecting a WP retrieves information of available nodes using the discovery mechanism. It ranks them, selects the top one, and informs it that it should become a WP. If a selected node rejects the new role, the selector chooses the next one on the ranking.

The node in charge of the selection sends its aggregated clue to the new WP. If this aggregated clue is fresher than the one in the WP, the WP will use this one. Otherwise, the WP will use its own. In case there is no previous aggregated clue, the new WP will start from scratch (version to -1). Once the node becomes the new WP, it notifies the version used to initialize the discovery system. Doing so, *Providers* will realize if they must send their *clue* again because it is not included in the aggregation provided by the WP.

Note that the discovery mechanism must provide the following information about each node in the *Space* to the selection algorithm: *(1)* memory of the device, *(2)* storage capacity of the device, *(3)* time since it joined the *Space*, and *(4)* its battery charge. We use the algorithm detailed in Listing 1 to rank the nodes.

1: $nodes \leftarrow filter_{threshold}(nodes, \text{“}memory\text{”}, threshold_{memory})$
2: $nodes \leftarrow filter_{threshold}(nodes, \text{“}storage\text{”}, total_{nodes} \times storage\_needed_{avg})$
3: **if** $anyWith(nodes, battery_{infinite})$ **then**
4: $\quad nodes \leftarrow filter(nodes, \text{“}battery\text{”}, battery_{infinite})$
5: $\quad nodes \leftarrow orderBy(nodes, \text{“}battery\text{”})$
6: **else**
7: $\quad$ **if** $anyGreaterThan(nodes, \text{“}joined\_since\text{”}, joined_{threashold})$ **then**
8: $\quad\quad nodes \leftarrow filter_{score}(nodes, \text{“}joined\_since\text{”})$
9: $\quad$ **end if**
10: $\quad nodes \leftarrow filter_{score}(candidates, \text{“}battery\text{”})$
11: $\quad nodes \leftarrow orderBy(nodes, \text{“}memory\text{”})$
12: **end if**
13: **return** $nodes$

Listing 1: White Page selection algorithm.

Lines 1 and 2 of the algorithm apply filters based on thresholds. In particular, the second threshold is variable and depends on the number of nodes (more nodes will generate more *clues* to store). After that, we check the power availability of the devices. We first select the nodes connected to the power grid (represented by $Battery_{infinite}$). If there are not devices connected to the electrical grid, we select devices which are steady enough to apply a filter by

joined_since field. Filters in lines 8 and 10 choose nodes with z-scores higher than one for the specified fields. If no node fits that filter, it returns the nodes with values higher or equal to the mean.

Summarizing, the algorithm prioritizes energy autonomous nodes and within the nodes with battery limitations, it prefers steady ones. In both cases, it finally selects a node with enough storage and memory (including mobile devices).

Due to out-of-date information, two nodes may become WPs at the same time. Our solution would eventually correct this situation thanks to a conflict resolution algorithm. When a *WP A* detects another *WP B* in the *Space*, it has to check which one has a better score according to Listing 1. If *B* has a better rank, *A* simply resigns as WP and notifies it to the discovery system. Otherwise, it forces *B* to resign through an HTTP invocation. Other nodes will be aware of these changes through the discovery mechanism.

5.4 Shared clues

As we introduced before, WPs host *clues* about the information in the *Space*. Using a *clue*, a *Consumer* can find which node (or nodes) is the *Provider* of a piece of information. *Providers* generate these *clues* by digesting the semantic information they store.

Thanks to these *clues*, resource constrained devices do not have to process unnecessary requests. We also limit the length of the *clues* to reduce the bandwidth, the memory, and the storage overhead on these devices.

This section describes in detail these *clues*, the information they contain and their format.

5.4.1 Querying basics

A *Consumer* directly queries the information to the selected *Providers*. This selection is done using the *clues* described in the rest of this Section. However, in this section we explain how *Consumers* directly query for information to the selected *Providers*.

We assume that *Providers* offer an HTTP API which accepts queries. Whether this API redirects the requests to the RDF graph which describes a specific resource, it creates a response with RDF triples belonging to different resources or it offers hyperlinks to the relevant graphs is transversal to our solution. There is only one requirement regarding this API: all the *Consumers* and *Providers* must agree on it. In the context of this thesis, this API corresponds with the one defined in Chapter 4.

To perform the queries, which enable the selection of a subset of the semantic content hold in a given *Space*, we require a **template**. We assume the use of wildcard templates, which are special triples with optional wildcard subject, predicate and/or object (see Figure 5.2). We could use more sophisticated query languages (e.g., SPARQL Protocol and RDF Query Language (SPARQL) [162]), but processing them may be too demanding for devices with constrained capabilities. On the other hand, simple wildcard templates can be managed by any node able to manage RDF triples. In any case, complex queries can be decomposed into wildcard templates and our solution would still apply.

5.4.2 Content of a clue

To find out which is the most appropriate solution for UbiComp scenarios, we have to consider scenarios populated by mobile devices and sensors. Mobile devices usually share data which rarely changes and is described using a few ontologies (e.g., the user profile and his preferences). On the other hand, sensors are constantly generating new instances of the same ontology (also called individuals). In both cases, the data shared by each node is described according to one or few vocabularies or taxonomies.

At this point, it is important to define the *TBox* and *ABox* concepts following the definition of Nardi and Brachman [84]. *TBox* contains knowledge describing general properties of concepts or terminology and *ABox* contains knowledge specific to the individuals of the domain of discourse. An example of *TBox* information is the device type or the elements which is made of, while *ABox* can describe the mobile phone brand or the temperature sensed by a thermometer.

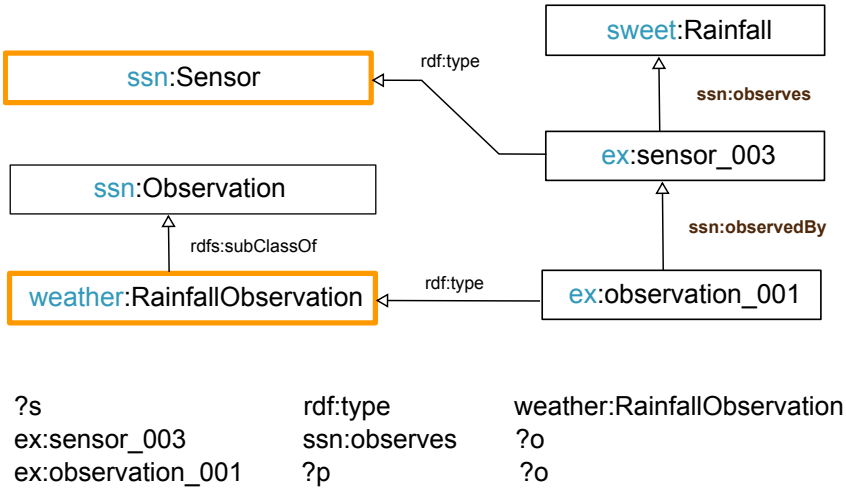


Figure 5.2: Sample triples and query templates .
The top of the figure shows some sample triples. It highlights the different parts used for each type of *clue*: the information used in class-based *clues* in orange, the prefix-based *clues* in blue, and the predicate-based *clues* in brown. The bottom shows query template examples for these triples.

Each mobile device or sensor usually generates *ABox* according to the same *TBox*. Using this information, we propose avoiding the use of URIs which represent *ABox* information in general terms. Specifically, we propose and evaluate three possible type of *clues* to share.

Prefix-based clue. The individuals (*ABox*) are described according to different ontologies (*TBox*). Each ontology usually employs a common *namespace*. This means that the URIs of the concepts in these ontologies share a unique *prefix*. Furthermore, each *Provider* uses a few *prefixes* to generate the URIs of its individuals. In this type of clue, we propose a coarse-grained method to filter nodes which do not have individuals with the prefixes used by the query template. To create prefix-based clues, the *Providers* have to extract the prefixes used in their graphs.

Following the example in Figure 5.2, a *Provider P* may have that graph. Thanks to the *clues*, the *Consumer* knows that *P* has URIs starting with *sweet*, *ssn*, *ex* and *weather*. Consequently, it will send to *P* a query which uses any of the templates shown in Figure 5.2. On the contrary, if the template is *?s*

pref2:sth ?o, it will avoid sending the template to *P*.

Predicate-based clue. The predicates relate subjects with other subjects or literals. These predicates are defined in the *TBox* (e.g., to state they relate concept *A* with concept B) and used in each triple of the *ABox*. In this approach, we propose extracting the set of predicates used in the graphs stored by each node. Using this information, they can be simply matched with the predicate defined in the query template.

Class-based clue. In the third approach, we propose sharing the classes of concepts (*rdf:type*) provided by the nodes. Using this information and the *TBox*, each node can check if the information matching certain query template is susceptible to be stored in other nodes. For this kind of clues, we assume that each node should have (or be able to obtain) the *TBox* related to the template used for querying. This is a reasonable assumption since the ontologies which mainly describe the *TBox* are usually accessible on the URL described by its prefix.

To understand how it works, consider that, according to the *TBox*, a predicate *p* relates the concept *A* with another concept. We assume that the nodes storing instances of the class *A* are more likely to use this predicate in their graphs. Therefore, if a query template uses that predicate, we will send it to all nodes storing instances of the class *A*.

5.4.3 Reasoning to expand clues

Through a reasoning process one can know unstated knowledge. Using this knowledge, we can detect more relevant nodes. For example, let us consider the previous example where the class *A* defines the predicate *p*. Providing that *C* is a subclass of *A*, a node which has many instances of the class *C*, may also use the predicate *p*. Thanks to the reasoning, we can discover that the node has knowledge of the type *A* and therefore, it may be relevant. The drawback, is that reasoning consumes a lot of resources. This limitation will be further analysed in Section 5.6.1.

5.4.4 Use of ABox in clues

As stated before, in general terms, we want to avoid the use of *ABox* URIs (individuals) in our clues. Thus, we fulfil two goals: *(1)* generate smaller *clues* and *(2)* the *clues* will not change too frequently and therefore, less communication to update clues will be required. However, in some cases, the use of *ABox* content in the *clues* may be beneficial. For instance, assume a URI that refers to the specific location *L*. If we want to search for devices in location *L*, we cannot deduce anything about it using the proposed *clues*.

For this reason, we need to consider sharing the *N* most queried individuals in our clues. To do that, the WP needs to store a list with the statistics about the information collected by each *Consumer*. *Consumers* can send this information together with the request to update a clue. *Providers* can obtain a list of the current most popular URIs before sending their updated *clues* to the WP. Using this list, *Providers* can know if they have these URIs and include them in the *clue* to be sent to the WP. Note that this process would imply an extra request per *Provider* before each update.

This simple approach implies sending not only *TBox* but also *ABox*. The amount of extra information added to each *clue* will depend on the size of this list (*N*). The effectiveness of this method will depend on the number of queries using one of the *N* URIs in their subjects or objects.

5.4.5 Format

Many formats to represent the content of a clue can be used. One option is the ongoing Efficient XML Interchange (EXI) format [157]. EXI is designed to efficiently interchange XML data and therefore, we could obtain better compression rates than with JavaScript Object Notation (JSON). However, we have chosen JSON for its simplicity and its wide adoption in the WWW.

The prefix-based *clue* is the easiest one to represent in JSON since it is formed by a set of URIs. We can also represent the predicate-based and the class-based *clues* using a set of URIs. However, the prefixes of those URIs are usually repeated and for this reason, we do not use plain URIs transmission for

these clues. We show an example in Listing 2. It first defines the prefixes used and gives them a name and then, it specifies the URI endings for each prefix.

```
{
  "s": [
    [ "so", "http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#" ]
  ],
  "p": {
    "so": ["result", "procedure", "observedProperty", "samplingTime"]
  }
}
```

Listing 2: Representation of a predicate-based *clue* in JSON. The node sending the *clue* has RDF triples which use the predicates *so:result*, *so:procedure*, *so:observedProperty* and *so:samplingTime*.

These are isolated *clues* sent from a *Provider* to the WP. However, the WP gathers all these *clues* in an aggregated clue which is sent to the *Consumer*. We show an example of an aggregated clue in Listing 3. As one can see, the aggregated clue defines the type of *clues* wrapped through the numeric field *i*. *G* and *v* form the version of this aggregated clue. The field *s* defines the prefixes. Finally, for each node, each prefix is related with the URI endings.

5.5 Experimental environment

We use simulation to study the performance of our solution and compare it against a flooding-based one. Using a simulation, we can evaluate multiple scenarios and repeat experiments for different approaches under the same conditions. The source code for the evaluation has been made publicly available[1].

5.5.1 Methodology

Table 5.2 shows the main parameters of our simulator. We vary these parameters to simulate a wide range of scenarios.

[1] http://gomezgoiri.net/files/code/gomezgoiri2014energy.html

Table 5.2: Configuration parameters.

Name	Description
Network size	The number of nodes in a network. In the simulations conducted, all the nodes are *Providers*.
Number of writes	Amount of writes performed during the simulation period.
Number of queries	Amount of queries performed during the simulation period.
Number of *Consumers*	Amount of nodes querying to other nodes in the *Space*.
Distribution strategy	Our solution or negative broadcasting (NB). In NB nodes write locally and spread the queries to the rest of the nodes in the *Space*.
Drop interval	At the beginning of this interval a node abruptly leaves the network. At the end, the node joins the network and a new one is chosen to leave it.

```
1  {
2    "i": 1,
3    "g": 2435467,
4    "v": 556,
5    "s": [
6      ["dc", "http://purl.org/dc/elements/1.1/"],
7      ["dul","http://www.loa.istc.cnr.it/ontologies/DUL.owl#"],
8      ["ssn", "http://purl.oclc.org/NET/ssnx/ssn#"] ],
9    "p": {
10     "node1": {
11       "ssn": ["observedBy", "observationResult"],
12       "dul": ["isClassifiedBy"]
13     },
14     "node0": {
15       "ssn": ["observes"],
16       "dc": ["description"]
17     }
18   }
19 }
```

Listing 3: Representation of an aggregated clue in JSON. Line 2 defines that it embeds predicate *clues* (i.e., type 1). Lines 3 and 4, contain the version of the aggregated clue. The remaining lines express the predicates used by two nodes. For example, *Node1* has at least a RDF triple which uses the predicate *ssn:observedBy*.

As we simulate HTTP, we assume point to point communication between devices which exchange RDF Triples. We discuss how the discovery process affects the solution in Section 5.6.4. In the remaining sections, the node discovery process is omitted as it represents the same overhead for all strategies.

To represent the data managed by each node, we use data from diverse sensor network environments. These data follow the *Semantic Sensor Network Ontology* (SSN) [163]. SSN has been used in many projects and scenarios to describe semantically the data provided by heterogeneous sensors. Specifically, we use data from the following datasets: AEMET metereological dataset [117], University of Luebeck Wisebed Sensor Readings [141], *Kno.e.sis* Linked Sensor Data [129] and Bizkaisense [119]. These datasets contain descriptions

Table 5.3: Technical characteristics of the assessed devices.

Device	Processor	RAM	Reference
XBee	-	8 MB	[122]
FoxG20	400Mhz Atmel ARM9	64 MB	[126]
Samsung Galaxy Tab	1 GHz Cortex-A8	512 MB	[135]
Regular computer	2.26 GHz Intel Core 2 Duo	4 GB	-

about the sensing stations and the data sensed by them during certain periods. The analogy between stations which have different sensors and the IoT devices is reasonable. The datasets are adapted to provide just one measure of each sensor at each moment (to emulate the storage restrictions from embedded devices) and to use as many stations as nodes has the network (depending on the network size).

However, not only sensors but also personal devices (e.g., mobile phones) usually populate UbiComp environments. To represent such circumstance, we add semantic data of people to represent their profiles [136].

We use SimPy [138] to simulate each scenario. SimPy is a process-based discrete-event simulation language for Python. To accurately simulate the time needed by each node to provide a response, we consider measures taken from real embedded web servers [46]. These servers run on a *ConnectPort X2* IP gateway [122] for *Digi*'s *XBee sensors* [123] (*XBee* from now on), on a FoxG20 [126] and on a Samsung Galaxy Tab [135]. We also provide measures taken from a regular computer. Table 5.3 shows the technical specifications of these devices.

These devices serve semantic content through HTTP. Table 5.4 shows the platforms and libraries used in each device to implement such servers.

Finally, Table 5.5 details the time needed by each device to answer a request depending on the number of concurrent requests. As mentioned, the next section's simulations are parametrized according to these measures.

Table 5.4: Core libraries used in the semantic HTTP server implementations.

Device	Platform version	REST libraries	Semantic libraries
XBee	Python 2.4 [131]	Python Std Lib	None
FoxG20	Python 2.5	Python Std Lib	RDFLib [133]
Samsung Galaxy Tab	Android 2.2 [118]	Restlet [134]	Sesame [137]
Regular computer	Java SE 6.0 [127]	Restlet	Rdf2Go (Sesame) [132]

Table 5.5: Response times for web servers running in different devices and providing semantic content.

Each provider replies from 1 to 15 concurrent requests. The response times' means and the standard deviations (in parenthesis) are measured in milliseconds.

	Devices			
Concurrent requests	XBee	FoxG20	Samsung Galaxy tab	Regular computer
1	77 (1)	17 (0)	223 (349)	5 (1)
5	392 (8)	97 (16)	256 (76)	8 (4)
10	775 (8)	174 (28)	372 (171)	13 (8)
15	-	282 (43)	497 (191)	18 (13)

5.5.2 Performance metrics

To evaluate the fundamental properties of the strategies, we use the following metrics:

- *Precision*: the fraction of nodes which answered relevant results (those responses which were not *not found* responses). It measures the exactness of the results.
- *Recall*: the fraction of relevant answers that are returned. It measures the completeness of the results.
- *Size*: the size of each type of clue.
- *Total requests*: the number of HTTP requests performed during a simulation.
- *Response time*: the average time needed to obtain an HTTP response.
- *Active time*: the total time spent by each node either querying other nodes or handling a query.

5.6 Evaluation

5.6.1 Types of clues shared

As presented in Section 5.4, the type of clue used will affect *(1)* the *precision* and *recall* to find the nodes with the appropriate information; and *(2)* the amount of information to transfer over the network (both requests and responses) and nodes have to process. Increasing *precision* reduces the number of unsuccessful requests to handle and thus, it reduces the energy consumption. Similarly, sending more information over the network implies more processing time and more energy consumption.

Precision and recall. We evaluate the *precision* and the *recall* of the proposed algorithm in a network of 470 nodes issuing the query templates shown in Table 5.6. In average, the nodes manage instances belonging to 6.34 different

classes (standard deviation, $SD = 1.31$) among a total of 17 distinct classes in the *Space*. The distinct predicates managed by each node in average are 16.01 ($SD = 1.53$) out of 68 different predicates in the *Space*.

Table 5.6: Templates used in the evaluation.

Name	Template
T1	`?s rdf:type ssn-weather:RainfallObservation`
T2	`?s wsg84:long ?o`
T3	`?s ssn:observedProperty ?o`
T4	`bizkaisense:ABANTO ?p ?o`
T5	`?s dc:identifier ?o`

In Figures 5.3 and 5.4, the class-based clue shows a good *precision* and *recall* for *T1* and *T2*. *T1* asks exactly for the information this type of clues define (i.e., nodes having instances of a certain class). *T2* evaluates which nodes have instances in the domain of the *long* predicate (*SpatialThing*). Note that this works thanks to the RDFS inference because some nodes in the *Space* only write *Point* instances (a subclass of *SpatialThing*). The domain of *T3* and *T5*'s predicates could not be inferred just using RDFS inference. Even solving this limitation, we would expect a bad *precision* since both predicates relate very general concepts. In addition, when a class-based clue has no enough information to predict the nodes, it simply floods the query. This is why the *recall* of *T4* is high.

We can see a bad prediction for *T1* and *T4* for predicate-based clues. *T1* defines a very common predicate and therefore, it cannot discriminate any node. *T4* suffers the same problem explained for the class-based clues. We proposed a possible solution for this problem in Section 5.4.4.

Finally, prefix-based clue shows a slightly better *precision* for *T4*, since it can discriminate some nodes not using the *bizkaisense* prefix. On the other hand, it obtains marginally worse *precision* than predicate-based clues for *T3* and *T5*. This worsening could be greater if few nodes using the prefixes *ssn* and *dc* used the predicates defined in both templates.

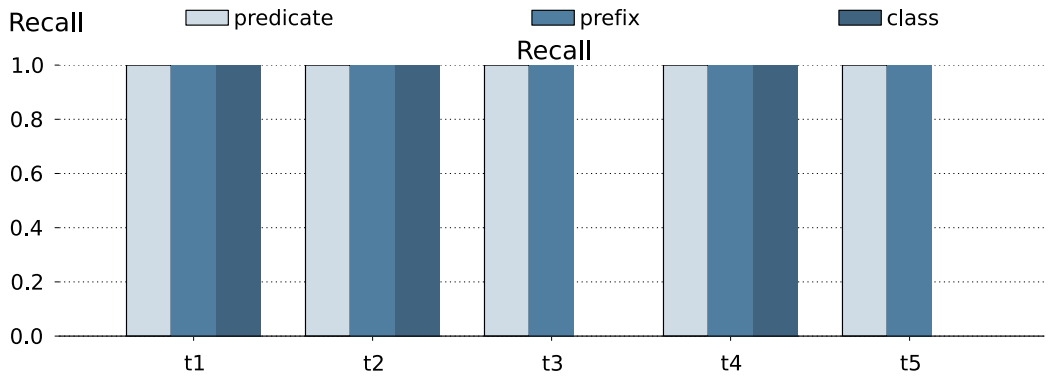


Figure 5.3: *Recall* for each type of clue.

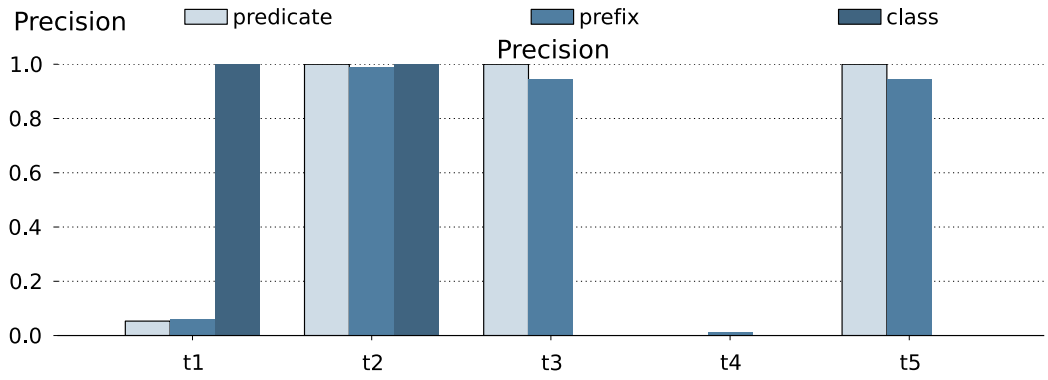


Figure 5.4: *Precision* for each type of clue.

Verbosity. The clues verbosity is also a critical aspect for resource constrained devices. Figure 5.5 shows a higher variance for prefix-based clues' length and lower verbosity of class based clues. This is because the nodes virtually have a different number of sensors. In addition, the links to concepts of other ontologies vary within the datasets used in the parametrization. In any case, the diagram shows a similar verbosity for all the clues for the semantic content considered in this evaluation.

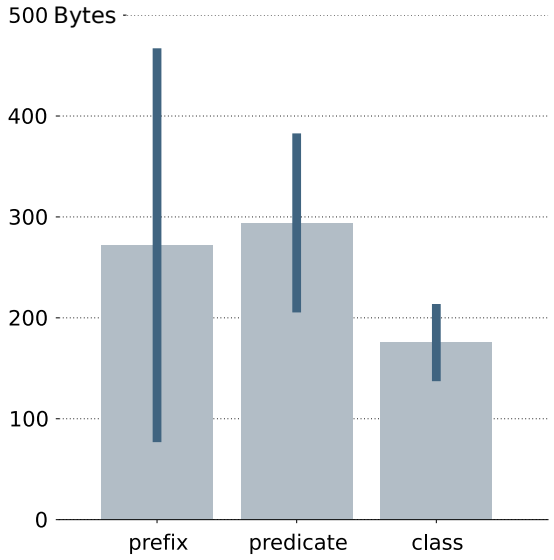


Figure 5.5: Length for types of clues.

Summary. Class-based clues are useful for templates asking for a specific type of content. However, they still require inference to obtain a good *precision*. In Gómez-Goiri et al. [46], we tested the inference process on the devices and data used in this simulation. We could not run any reasoner in the ConnectPort X2 Gateway. Actually, we could only run RDFS reasoners in more powerful embedded and mobile devices such as the FoxG20 and the Samsung Galaxy Tab. In the FoxG20, it took 48.9 seconds the first load of all the ontologies used and 1.4 seconds to reason over each measurement written. In a Samsung Galaxy Tab, it took 17.3 seconds and 0.2 the following measurement writings. Considering these results, we can conclude that there is a clear need for efficient

embedded reasoners. Therefore, the class-based approach is promising but it is impossible to adopt in current embedded and mobile devices.

Between the predicate-based and prefix-based clues, we propose to use the predicate-based clues since they subsume much of the information provided by the prefix-based clues. The rest of the prefixes are referred in the subjects or the objects. They could be easily added to predicate-based clues on the prefixes field. In addition to the use of predicate-based clues, we could implement the solution for the specific individual search proposed in Section 5.4.4.

5.6.2 Network usage

We conduct a simulation study to evaluate the benefits of our solution against a flooding-based approach (i.e., NB). In addition, to give a more exhaustive comparison, we implement and test query caching on top of NB. We simulate multiple nodes that join the same *Space* as *Providers* and periodically write new information to the *Space*. During one hour, 1 or 100 *Consumers* perform 1000 queries in total using the templates described in Table 5.6.

As expected, our solution scales much better than the one with NB (Figure 5.6). However, adding caching to NB works slightly better than our solution with just one node querying the *Space*. This is due to the limited amount of different query templates used in the simulation. When we increase the number of *Consumers* in the *Space*, the caching strategy behaves closer to the NB. In the same situation, our solution handles better an increase on the number of *Consumers* in the *Space*.

In Figure 5.7, we take a closer look to the origin of the traffic of our approach in a *Space* with 100 *Consumers*. The communication between the *Providers* and the WP is much more infrequent than the other communication types. The reason is that writing into a node only results in a clue update when the structure of the managed information changes. The first time the metadata about the node (sensor) is written, the second time the first measure and following writings, just add or replace a measure. Therefore, the clue does not change after the second step. This matches with the assumption made to share *TBox* information in our clues.

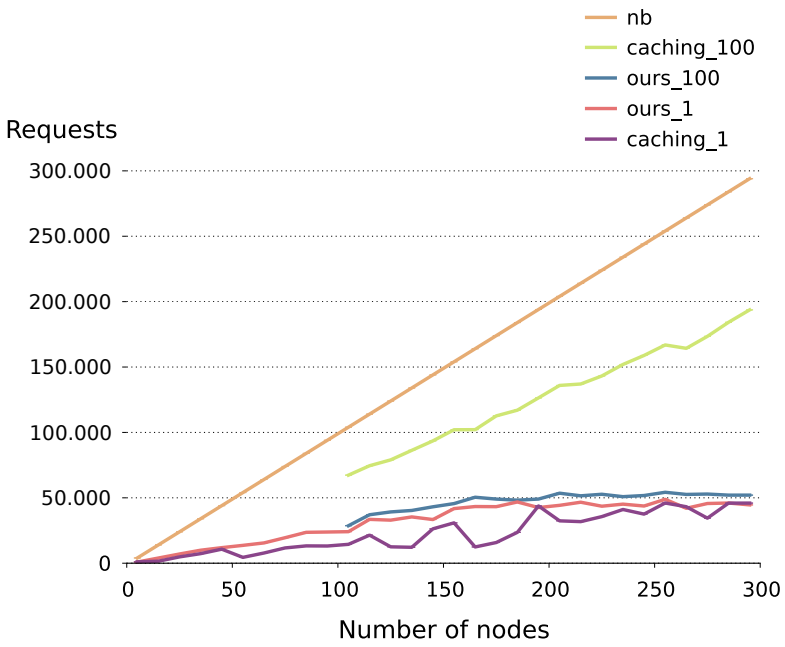


Figure 5.6: Required requests for different search strategies.

Required requests for negative broadcasting (NB), NB with caching with 1 and 100 *Consumers* and our solution with 1 and 100 *Consumers*.

The communication between *Consumers* and WP is in between the other two communication patterns. It is greater than the one from *Providers* to WP because *Consumers* need to maintain an updated view of the *Space*. Recall that the update time depends on the query frequency of each *Consumer*. The maximum and minimum updating frequency were set to 10 and 1 minute(s) respectively.

The communications between *Consumers* and *Providers* assumes most of the total communications. This shows that the overhead added by the use of WP on our solution is not significant and it is justified by the reduction of the total number of communications shown in Figure 5.6.

5.6.3 Energy consumption

Our solution tries to save energy by making *Providers* handle fewer requests from *Consumers*. These savings contrast to the overhead added by the communication with the WP. However, our results demonstrate that this overhead is

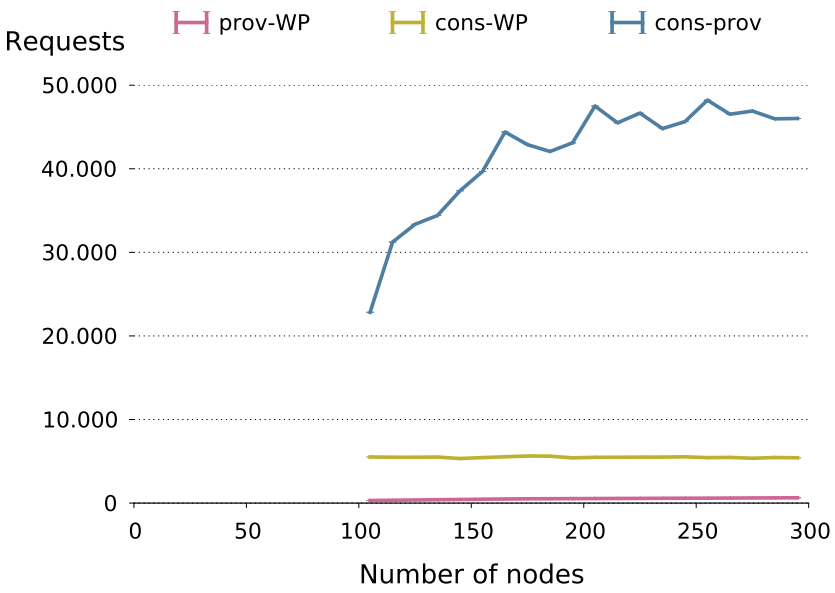


Figure 5.7: Requests between roles in our solution.

The *Space* evaluated has 100 *Consumers* and from 105 to 295 nodes in total (including the *Consumers*). "*Prov-WP*" denotes the communication between *Providers* and the WP, "*Cons-WP*" the communication between *Consumers* and the WP, and "*Cons-Prov*" *Consumers*' requests to *Providers*.

small in comparison to the total number of communications.

The energy consumption in mobile and embedded devices increases each time a device needs to process something or communicate with another node (see Figure 5.8). To analyse how communications impact their energy autonomy, we have to consider not only the number of communications but also their time length (see Table 5.5). For example, a mobile phone will consume less energy asking clues to a server than asking them to an embedded device as it has to wait less for the response.

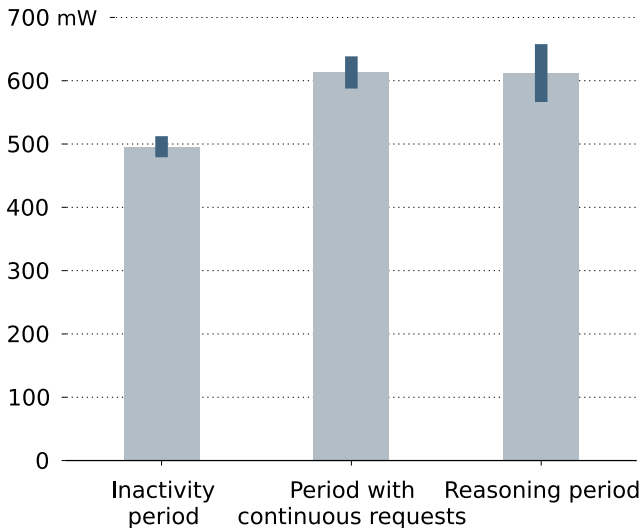


Figure 5.8: Average power consumption for FoxG20 during different activity periods.

The experiment consists of 300 nodes joined to a *Space* running on 1 server, 30 galaxy tabs, 75 FoxG20 and 194 XBees. We increase the number of devices as their price and capacity decrease. Using this approach, we mimic a typical *Space* where cheap devices are more common.

As shown in Figure 5.9, our solution reduces the activity of each device by more than 5 times compared to negative broadcasting. The diagram on the right details the average activity for each type of device.

In our solution, we can check how the load moves from the embedded devices (XBee and FoxG20) to the server (which is indeed chosen as a WP).

The exceptional activity registered by the Galaxy Tabs is caused by their extremely high response time. However, we plan to reduce this response time changing the HTTP library used in our Android implementation.

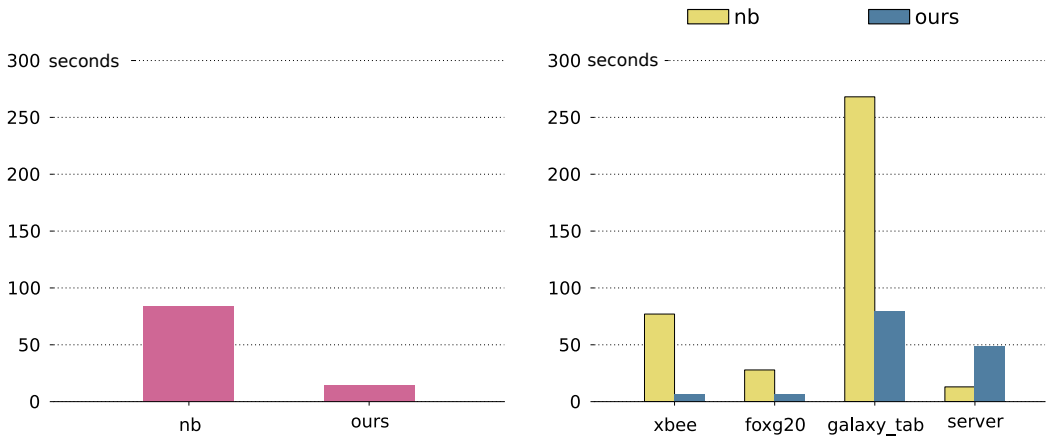


Figure 5.9: Activity time for each strategy.
The first part shows the average active time a node spends on each strategy. The second one shows the active time classified by the type of device each node has run on.

5.6.4 Performance in dynamic environments

We evaluate the network usage of our solution in ordinary situations in sections 5.6.2 and 5.6.3. Nevertheless, we do not evaluate scenarios where the nodes frequently join and leave the *Space*. In such situation, the communication needed to manage the clues might be a burden.

To assess the effect of dynamic networks on the performance of our solution, we used the scenario presented in Section 5.6.3, Then, we simulate nodes joining and leaving the *Space* at different intervals: 30 seconds, 1 minute, 5 minutes, 10 minutes, 20 minutes, 30 minutes, 45 minutes. Particularly, for our solution, we tested the most harmful situation: the node leaving the *Space* abruptly is always the WP. We also added an scenario with no drops as a baseline. Note that we represent this scenario by configuring the drop-interval with a greater value than the simulation time.

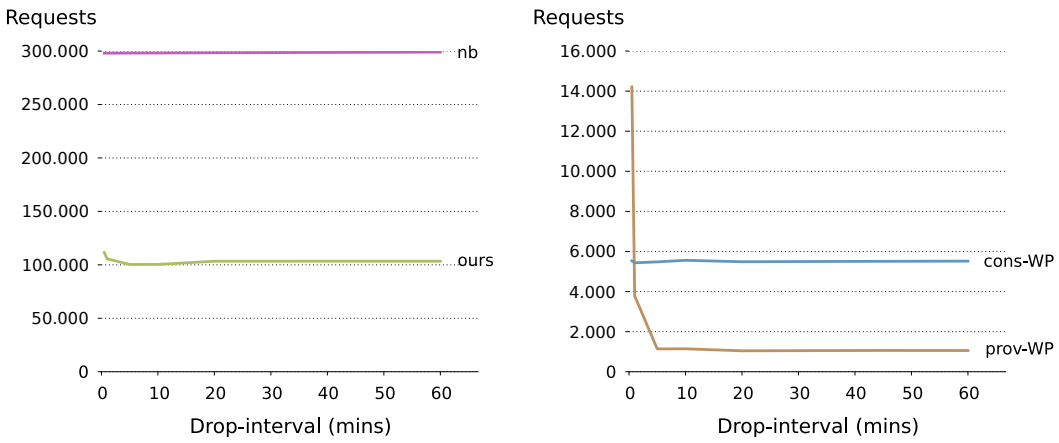


Figure 5.10: Effects of dynamic scenarios in our solution.

Note that the last interval in the x-axis represents a simulation with no drops.

In Figure 5.10, we see the results of these simulations. These results show that even in such dynamic situations, our solution requires fewer communications than negative broadcasting. In our solution, most of the communications are between *Consumers* and *Providers*. To evaluate the overhead added by our solution, the graphic on the right hand side shows the communications involving WPs.

We can appreciate that the updates on *Consumers* are independent of the number of times the WP changes. We can also see a minimal change between the scenario where the WP is always available and the one with 5 minutes drop-interval. In this case, when the WP drops, many *Consumers* have the latest version of the aggregated clue. This situation increases the chances of getting an updated version for the initialization of the new WP (see Section 5.3.6). Thus, it reduces the number of messages from *Providers* to the new WP.

5.6.5 Effects on discovery mechanisms

We want to prove the feasibility of our solution using a common discovery mechanism. To that end, we simulate the behaviour of the mDNS and DNS service discovery (DNS-SD) [146] protocols. Both protocols are based on the

well known and widely accepted DNS. DNS associates different pieces of information (i.e., records) with domain names in a distributed manner.

On the one hand, DNS-SD proposes a new use for DNS's TXT records. The TXT record was originally intended to associate an arbitrary human-readable text with a domain name. DNS-SD proposes to use this type of record, to share key-value pairs in its data field. We use these key-value pairs to share the information needed by the selection algorithm among the nodes.

On the other hand, mDNS defines how this and other records are shared through UDP multicast (or unicast in certain situations). We ignore the cost of browsing the nodes to discover new nodes because it is the same for both strategies. However, note that as explained above, our strategy does differ from negative broadcasting in the use of TXT records.

The nodes announce records during the start up or whenever they have a resource record with new data. Therefore, each time a record is updated, we send a multicast message that increases the network traffic. In our solution the TXT record may change *(1)* when a new WP is selected or *(2)* when we update the time elapsed since it joined the *Space* and its battery charge level. The last two parameters need to be updated to select an appropriate WP but they do not need to change too frequently.

In the most static scenario the TXT record is written only once. The more dynamic scenario from the previous section, on the contrary, updates that record 126 times after writing it for the first time. This demonstrates that the overhead generated on the discovery system by our solution is minimal even in the worst-case scenario.

5.7 Summary

This chapter presented a dynamic architecture to enhance the search of semantic contents in the Web of Things. In particular, this architecture chooses an intermediary according to its capabilities to support resource constrained devices. Intermediaries can use different types of clues to summarize the information in the semantic *Space*. Thanks to this support, the devices can directly

interrogate others to obtain fresh information while reducing their semantic overhead.

The main characteristic of our solution is the ability of the devices to share semantic data directly, no matter how simple they are. Under this assumption, a flooding-based strategy would obtain a high recall. However, our evaluation shows that our solution requires fewer messages between devices than a flooding-based strategy (i.e., is far more precise). Even if we use caching strategies to alleviate these effects, our solution performs better for scenarios with more data consumers and query types. In addition, our approach reduces the workload of mobile and embedded devices which indirectly results into energy savings.

For our future work, we will consider allowing more expressive query languages such as SPARQL. This would force *Consumers* to discriminate between the *Providers* able to process it and the rest. On top of these queries, we could use query optimization techniques in the *Consumers* [101]. Using these techniques, we could transform the queries before sending them to *(a)* obtain better results or *(b)* use the network more efficiently.

If you live among wolves you have to act like a wolf.

Nikita Khrushchev

CHAPTER

6

Remote actuation

In the previous chapter, we presented an energy-aware technique to search on the completely distributed semantic space. However, acting over an UbiComp environment is as important as observing what happens on it. Although this dissertation describes this problem less thoroughly, we consider interesting to discuss it to provide the complete story.

This chapter presents and compares two techniques to change the physical environment. The first technique is based on common TS usage patterns while the second one relies on semantically described REST services to create a plan to fulfil a given goal. The rest of this chapter refers to them as *Space-based actuation* and *REST actuation* respectively.

Note that the second actuation technique is out the scope of the space-based communication covered in this dissertation. However, the seamless integration of these REST services opens the door to reuse the capabilities of many existing devices. Therefore, we propose reusing the HTTP APIs of already existing semantically described providers.

This chapter explores this reuse helped by a simple real-world scenario. The goal of the scenario is to remotely change the light of a lamp.

This chapter is organized as follows. Sections 6.1 and 6.2 present the two techniques and their corresponding implementations of the baseline scenario[1]. Section 6.3 compares these techniques and explores how they can interoperate. This exploration is done by means of a new implementation[1] which reuses much of the implementations from sections 6.1 and 6.2. After that, Section 6.4 discusses the advantages and limitations of the previous implementation presenting further implementation alternatives. Finally, Section 6.5 concludes the chapter.

6.1 Space-based actuation

This section presents the first technique to change the physical environment. That technique is space-based, i.e., participants coordinate by reading and writing into a shared *Space*. This encourages an uncoupled communication between applications using the same *Space*.

Section 6.1.1 briefly describes Tuple Space's most common application patterns and discusses which ones are suitable for UbiComp environments. These patterns constitute the base of what we called *Space-based actuation*. Section 6.1.2 presents the *Space-based actuation*'s core requirement: a subscription mechanism. Finally, Section 6.1.3 explains how to implement the baseline scenario according to the space usage patterns.

6.1.1 Background

According to Freeman et al. [29], there are four main application patterns which can be used in TS:

Replicated-worker pattern. In this pattern, there is a master process and many worker processes able to compute the same task. The master takes a problem, divides it into smaller tasks, and writes these tasks into the space. Any available worker takes a task, processes it, and writes the result back

[1] The implementations of the baseline scenario have been made public at `https://github.com/gomezgoiri/reusingWebActuatorsFromSemanticSpace`.

into the space. When all the workers have written their results, the master takes these results and combines them into a meaningful merged solution. This pattern is scalable and naturally balances the load on the space.

Command pattern. This pattern encapsulates the behaviour into the tuples shared in the space. Therefore, it requires *(1)* to share behaviour through the space, and *(2)* any generic worker to be able to compute the behaviour.

Marketplace pattern. In this pattern, producers (or sellers) and consumers (or buyers) of resources interact to find the best deal.

Specialist patterns. In contrast to the replicated-worker pattern, in this pattern, each worker is specialized. Therefore, each worker performs a particular task. Freeman et al. enumerate three subtypes:

Blackboard pattern. It associates the concept of the space to a *blackboard*. Following this analogy, the master is associated with a teacher, tasks with *problems* and the specialized workers with *students*. The blackboard pattern starts when the *teacher* writes a *problem* in the *blackboard*. The *students* observe the space and write their intention to contribute to solve the problem (i.e., *raise their hands*). The *teacher* selects an expert which will make a modification. After the modification, the *teacher* decides if it has found the solution or another *student* should contribute.

Trellis pattern. In this pattern, the master arranges the problem into low-level, mid-level, and high-level pieces. The workers of each level benefit from the refined data provided by the immediate level below.

Collaborative patterns. It encompasses all the patterns which allow nodes to collaborate to complete a greater task (i.e., by creating a workflow).

To summarize, the *replicated-worker pattern* is centred in optimizing the computation by parallelising tasks. The *command pattern* can be seen as an abstraction of the latter where the behaviour is shipped in the tuples. The *marketplace pattern* allows negotiation of two entities through the space. Finally, the *specialist pattern* allows nodes with distinct capabilities to cooperate towards a common goal.

Regarding the application of these patterns to UbiComp, we observed that:

- Devices in UbiComp often serve to very specific and local needs. For example, let us imagine a mobile phone showing a message or an embedded device turning on the air conditioning of a room.
- These tasks are usually lightweight. They intent to achieve concrete and simple goals which usually imply more I/O operations than processing ones.

Therefore, UbiComp generally faces a problem of collaboration between nodes with different capabilities. In this problem, computation or negotiation aspects are secondary. Consequently, we believe that the *specialist patterns* fit UbiComp needs best.

6.1.2 **Notification mechanism**

In the patterns described in the previous section, some writings in the space trigger other node's action. For example, a device might show a message whenever a new warning is written into the space. To be aware of these writings, the node can either poll the space or rely on a notification mechanism. Obviously, the later leads to a more efficient use of the network.

As a consequence, a notification mechanism is highly advisable to fulfil TS's application patterns in a distributed environment. Although the implementation of this notification mechanism is beyond the scope of this thesis, it should comply with the following aspects:

- Do not substitute the pull-based search. The middleware must provide additional *read* and *take* primitives.
- Since the notification mechanism intends to allow coordination patterns, it must consider the knowledge from the *coordination space*. In other words, knowledge from the *outer space* will not trigger notifications.
- The evaluation of the subscriptions must not interfere with the writing process (e.g., introducing a delay). Therefore, it must run asynchronously.

- Ease its adoption by any type of computing platform. This can be achieved by reducing the requirements on the *clients*. For instance, a callback URL passed during the subscription can represent a minimal contract between the client and the server.

- Provide additional subscription removal mechanisms. UbiComp scenarios are composed by unreliable devices which may frequently join and leave the space. In this situation, the correct use of unsubscription primitives cannot be guaranteed. This may worsen the performance of the system with useless subscriptions from absent devices. Therefore, the device managing the subscriptions should adopt more proactive mechanisms. For example, it may let the subscriptions expire after a lifetime or remove them when it discovers the unavailability of a callback URL.

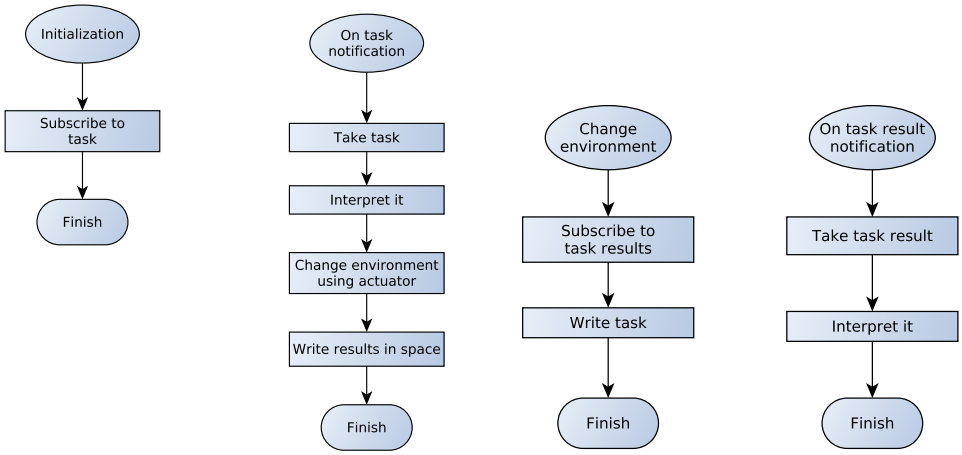
The main drawback of any subscription mechanism is that it breaks the Stateless property of the REST style. According to Section 4.5.2, this implies that network performance will improve at the cost of scalability, simplicity and reliability.

6.1.3 **Baseline scenario: Implementation 1**

This section describes how to implement the baseline scenario using the *Space-based actuation*. The implementation presents the following nodes:

(A) A node which reacts to the tasks written into a shared semantic space. The node running this implementation is aware of the tasks written into the space and changes the light's value accordingly. Figure 6.1(a) shows the initialisation of this node and the process performed when a graph is detected.

(B) A node which writes tasks into a shared semantic space describing its desire to change the light's value. Figure 6.1(b) shows the actions taken by this node. As it can be seen, the process is divided in two temporarily independent processes. The first writes the task and the second processes the result whenever it is written into the space.



(a) *Node A* (b) *Node B*

Figure 6.1: Flow charts for *Node A* and *Node B*

Node A first subscribes to changes in the space (see Listing 4). Then, *Node B* writes a task to be performed (see Listing 5) and subscribes to its result. As a consequence of the writing, *Node A* reacts by taking the task, interpreting it, changing the environment through its actuator and writing the result. Finally, *Node B* gets a notification of the written result, takes it and processes it.

```
select ?value where{
  ?observation a frap:Preference ;
        a ssn:ObservationValue ;
        dul:isClassifiedBy  ucum:lux ;
        dul:hasDataValue ?value .
}
```

Listing 4: Subscription to light preferences written into the space.

To simplify the implementation of the scenario, we consider a purely centralized space with a subscription mechanism. This mechanism uses SPARQL [162] and defines two different primitives. The first one takes into account the knowledge from the last graph written to trigger the notifications. The second primitive considers the knowledge from the whole space and hence is more computationally costly.

```
:obsv a ssn:ObservationValue, frap:Preference ;
      dul:isClassifiedBy  ucum:lux ;
      dul:hasDataValue 21 .
```

Listing 5: Task to *set* the light level in a space. Note that the preference is conceptually equivalent to a task.

6.1.3.1 **Actuation modelling**

The most abstract way to actuate on the environment would be to invoke a change referring to the sensed values. For example, stating "set *Room A*'s light-level to 19 luxes". This would require the coordination of all the actuators which directly or indirectly affect this value. Following the example, the lamps in *Room A* and the curtains of *Room A*'s windows should coordinate to give the exact desired value.

Even in this case, many other physical aspects would affect the value. For instance, the daylight at each time of the day or the location of the light sensor. As a consequence, we opt for clearly distinguishing between actuators and sensors. In our implementations, the data measured by the sensors can only be indirectly affected by the actuators.

These indirect effects on the sensed values can be modelled for each actuator. For example, to state that switching on a lamp increases the light measured by a sensor. However, precisely anticipating the exact value of the new measure is difficult if not impossible due to the mentioned external physical conditions. Considering this difficulty to predict their relationship and its tangential importance for the baseline scenario's implementations, we simply ignore these relationships. We assume that the consumer previously determined which actuators it should change to fulfil its initial abstract goal.

6.1.3.2 **Ontologies used**

The *glue* between the two actuation worlds presented in this chapter is their data. Therefore, the implementations presented in this chapter represent the environment according to the same ontologies. This section provides the rationale behind the selection of the ontologies used.

To represent the value of a lamp actuator, we use the SSN ontology [163]. For example, we express that "the lamp actuator has a light value of *N* luxes". The SSN ontology is intended to describe measures for sensors, so its use for actuators can sound contradictory. Note how the previous statement differs from "the light sensed by a sensor located near the lamp has a value of *N* luxes". However, to the best of our knowledge, there is no more widely-accepted ontology to specifically describe actuators.

The preferences ontology is another core ontology used in our implementations. It is used to clearly distinguish between the value of an actuator and a consumer's intention or desire to change it. A preference can turn into an actuator's new state, but it is not always necessarily true. The actuator can discard a preference due to an usage policy, a malfunction or any other reason.

6.2 REST actuation

The actuation technique presented in the previous section uses the *Space* (i.e., it is indirect). This technique implies that the actuator must be aware of the *Space*'s content. For instance, a heater must check the space to find if a new desired temperature was written.

In contrast, this section presents a second technique which directly actuates in the environment without using any *Space*. A consumer following this technique directly uses REST APIs. These APIs expose the devices' capabilities to make physical changes in the environment.

However, this apparently simple mechanism hides a difficulty: the consumer needs to discern how to use these APIs to make such changes. According to the REST principles, a client should navigate through the resources of an API with no prior knowledge of it. The client should *(1)* interpret the representations provided by the server and then *(2)* choose the appropriate state transition from the hypertext according to its intention. To do this, this section advocates for a semantic description of these APIs' resources.

Section 6.2.1 briefly compares different alternatives to build semantic REST APIs. Section 6.2.2 presents the selected alternative and Section 6.2.3 implements the baseline scenario using it.

6.2.1 Background

As Section 4.5.2 explained, semantic representations do not include a native way to express the hypertext. To solve this, three solutions can be adopted:

1. To use an ontology to represent the hypertext [62].

2. To embed the hypertext independently to the representations on the HTTP headers [88].

3. To provide a description of the state changes each HTTP request triggers [108, 109].

The latter two enable to discover resources and state transitions without adding metadata to the representations. This allows not only to describe semantic representations, but any type of formats.

Nottingham's [88] approach is extended by Wilde [114] to define how to embed additional semantics to process a resource representation. Wilde [114] calls these additional semantics *profiles* and identifies them using URIs.

Verborgh et al. [109] present a more expressive solution which goes beyond simply describing a resource type. It also allows to semantically describe the knowledge needed to use a concrete HTTP verb on a resource, and the content this request returns. The materialization of this proposal is called *RESTdesc* [108]. Mayer and Basler [76] use *RESTdesc* in an environment populated by web-powered devices. This environment is analogous to the ones envisioned by this dissertation.

We consider *RESTdesc* the best current solution which helps to achieve truly RESTful APIs. Therefore, this chapter assumes that the HTTP APIs whose capabilities we want to reuse in our space model describe their APIs with *RESTdesc*.

6.2.2 *RESTdesc*

RESTdesc describes HTTP methods using rules expressed in the Notation 3 (N3) language [145]. A rule's *premise* expresses the requirements to invoke a REST

service. A rule's *conclusion* expresses both the REST call that needs to be made and the description of that invocation result.

Verborgh et al. [108] suggest three complementary alternatives to discover descriptions. The first and second alternatives use HTTP mechanisms: content negotiation and the HTTP OPTIONS verb. Using content negotiation a client can obtain a dereferenceable link's representation which corresponds to the description. Requesting a resource using the HTTP OPTIONS verb a client can obtain the description in the response body. Finally, the third alternative is not HTTP-centric and it depends on repositories to store and retrieve the descriptions.

Verborgh et al. [109] propose a service composition mechanism for Web APIs using *RESTdesc*. This mechanism uses as inputs: *(1)* an initial state, *(2)* a goal state, *(3)* Web API descriptions using *RESTdesc*, and *(4)* optional background knowledge. Each of these inputs are semantically expressed and therefore, they can be processed by standard N3 reasoners. These reasoners generate proofs about how to achieve the goal starting from the initial state and using the rest of the inputs. These proofs can be seen as steps that need to be made to reach a desired state.

Additionally, Verborgh et al. distinguish between pre-proof and post-proof. The first, are those which assume that the execution of all API calls will behave as expected. The latter, can be seen as a *revision* of the pre-proof. It executes the Web API of the pre-proofs and uses actual execution results to generate a new proof.

6.2.3 **Baseline scenario: Implementation 2**

The proposed implementation for the baseline scenario using *RESTdesc* presents the following nodes:

(C) A node which exposes the lamp and its actuators through a REST API. This API is described using *RESTdesc*. To physically change the light value, any client must send an HTTP request to the resource which represents the light actuator.

(D) A node which directly communicates with the desired provider. To discern which provider's resources has to call and how to do it, this implementation reasons to obtain a plan. This plan determines how to fulfil the node's initial goal invoking the needed HTTP APIs. Figure 6.2 shows the actions performed by this node in detail.

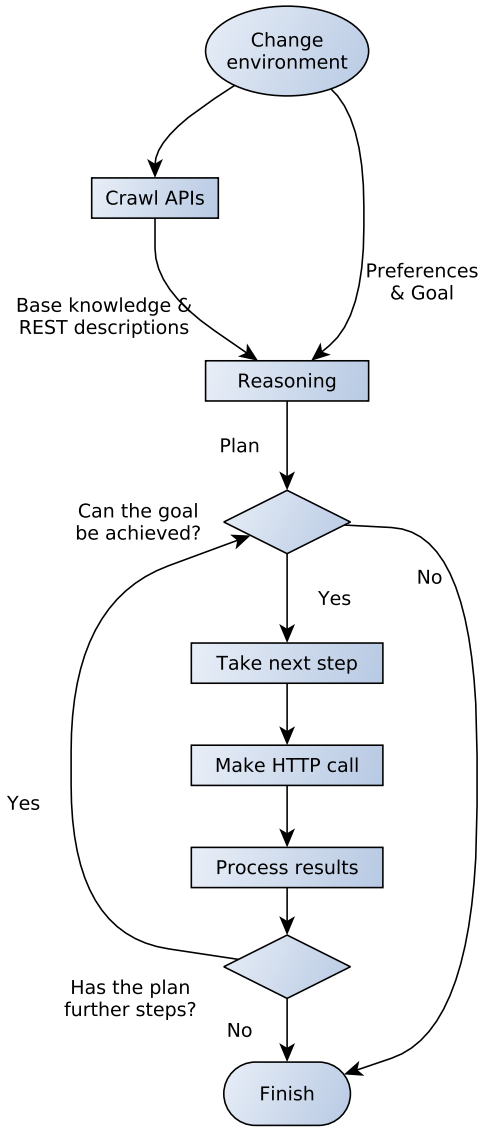


Figure 6.2: Flow chart for *Node D*.

The HTTP API provided by the *Node C* is modelled using the following resources:

- */lamp*: It provides basic information about the lamp.
- */lamp/actuators*: It enumerates the actuators which compose the *smart lamp*.
- */lamp/actuators/light*: It represents the unique actuator which composes the lamp in our simple example (i.e., the lamp's light).
- */lamp/actuators/light/01*: It represents a concrete preference to change the light.

To instruct consumers on how to use the services provided, they are annotated using *RESTdesc*. The HTTP OPTIONS returns listings 7 and 6 for */lamp/actuators/light*. Thanks to these descriptions and to the dereferenceable URIs [99], starting from */lamp* any client can crawl the API to autonomously learn how to use it.

In addition to the crawled content, the *Node D* provides two extra pieces of information to the reasoner: a preference and a goal (see listings 8 and 9). The preference allows the consumer to express the interest in changing a resource, which may not always be feasible. The goal drives the reasoning process, which tries to extract a plan to achieve it.

With this plan, the consumer just needs to call to the different HTTP resources that it defines. If more than a resource needs to be called, the plan also indicates how to use the information obtained from previous steps, i.e., prior calls, in the current one.

6.3 Hybrid actuation

The actuation technique presented in Section 6.1 is the natural way of acting using a *Space*. However, REST and REST-like services are well-accepted mechanisms to expose limited devices' actuation capabilities [36]. Therefore, their

```
{
  actuators:light ssn:madeObservation ?light_obs .
} => {
  _:request http:methodName "GET" ;
            http:requestURI ?light_obs ;
            #http:body actuators:light ;
            http:resp [ http:body ?light_obs ].

  ?light_obs a  ssn:Observation ;
         ssn:observedProperty  sweet:Light ;
         ssn:observedBy actuators:light ;
         ssn:observationResult ?so .

  ?so ssn:hasValue ?ov .

  ?ov a ssn:ObservationValue ;
      dul:isClassifiedBy  ucum:lux ;
      dul:hasDataValue _:val .
}.
```

Listing 6: Rule which expresses that having a light sensor observation, one can obtain details about the observation through an HTTP GET.

seamless integration (i.e., interoperation) opens the door to reuse the capabilities of many existing devices. This section aims to integrate semantically described REST services in our space-based model. That is, mix *Space-based actuation* with *REST actuation*.

Section 6.3.1 compares *Space-based actuation* and *REST actuation* and argues the need of this integration. Then, Section 6.3.2 presents a new implementation of the baseline scenario. This implementation shows how the *Space* can transparently invoke REST services on behalf of any consumer following the *Space-based actuation*. To this end, it reuses nodes from the implementations presented in sections 6.1.3 and 6.2.3.

6.3.1 Comparison

The actuation technique explained in Section 6.1 requires a subscription mechanism, but in exchange, it provides space and time decoupling. Obviously, this decoupling comes at the price of dependency on the *Space*. That is, two nodes will not be able to communicate with each other without the *Space*.

The second approach presented in Section 6.2 offers independence of the *Space*: any node can directly invoke a service to act on the environment. This approach allows reusing WoT applications' actuation capabilities, providing they are properly described. This reuse is enabled by a rule-based reasoning engine.

Table 6.1: Characteristics of the discussed actuation mechanisms.

Actuation	Communication style	Benefits	Required features
Space-based	Indirect	Decoupled communication	Subscriptions
REST-based	Direct	Reuse of third-party WoT applications	Rule-based reasoning

The effect of both mechanisms in resource-constrained devices is affected by their computing and networking activities. In order to analyse these activities, we measure the time needed to make a change and the total amount of requests. In addition, as the number of additional actuators in the physical environment affects these measurements, we consider four additional variations of the scenarios previously explained. Apart from the scenario with 1 actuator, we contemplate scenarios with 200, 400, 600, 800 and 1000 actuators.

In order to measure the networking activity, we calculate the total number of requests performed in each scenario. We assume that each new actuator will behave exactly as each of the actuators described in *Implementation 1* and in *Implementation 2*. Consequently, in *Space-based actuation* each new actuator writes 2 graphs into the space and in the *REST actuation* each API is crawled once. The crawler checks the 5 different calls clients can make to each API

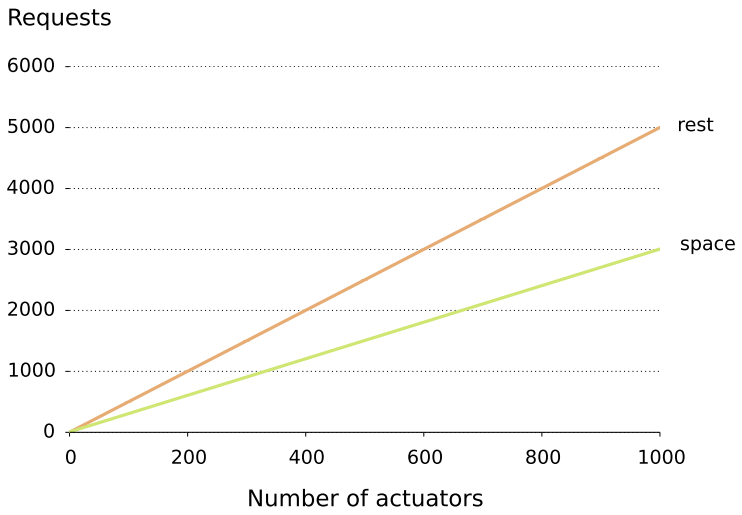


Figure 6.3: Total amount of requests per technique.

(1 HTTP OPTIONS request and 4 HTTP GET requests). Figure 6.3 shows the estimations for these scenarios.

The number of requests which need to be made per new provider depends on the parameters previously detailed (i.e. graphs written, times each API is crawled and calls needed to discover a whole API). As these parameters are design-dependent, the slopes shown by the Figure 6.3 might vary from one implementation to another. In any case, the figure shows that none of the techniques behave in a scalable manner.

In *Space-based actuation*, all the participants must be aware of what is written into the space to react (i.e., they are proactive). Both consumers and providers read and write from the space, subscribe to specific changes and receive notifications. Thanks to this specificity, they are only affected by the contents they are interested in. On the contrary, the *Space* will be involved in any networking activity. Therefore, although it depends on the number of providers and consumers' interactions, the networking activity will presumably be high for the node hosting the *Space*.

On the other hand, *REST actuation* requires consumers to have prior knowledge about the environment to reason over it. Since this knowledge must be acquired from remote nodes before reasoning, this approach demands an

extra network usage that the first does not. Therefore, depending on the network size, this technique might not be appropriate for resource-constrained consumers.

In order to analyse the computation activity in the scenarios' implementations, we measure the time needed to run both implementations of the scenario in a *Raspberry Pi* (Model B[1]). In this case, we also contemplate scenarios with 1, 200, 400, 600, 800 and 1000 actuators. The additional actuators represent smart-heaters and mimic *Node A* and *Node C* in each implementation. That is, in *Space-based actuation*'s executions each actuator writes 2 graphs (5 triples in total) and subscribes to temperature preference changes. In the *REST actuation*'s executions each actuator has an API equivalent to *Node C*'s API (i.e., 2 rules and 14 triples are exposed). For example, in *REST actuation*'s executions with 1000 actuators, *Node D* reasons over 2000 additional rules and 14000 additional triples. Figure 6.4 shows the results of these executions (each case is executed N=50 times). Note that the time measured ***(1)*** only includes the time *Node B* and *Node D* need to make a change in the physical environment (i.e., it excludes actuators' previous writings and subscriptions), and ***(2)*** ignores the delays added by the communication between nodes.

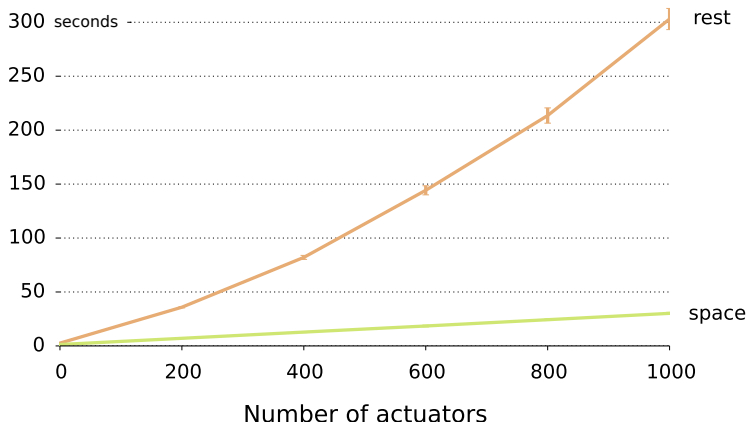


Figure 6.4: Time needed to make a change in the environment per technique.

[1] RAM Memory: 512MB.
CPU: 700 MHz Low Power ARM1176JZ-F Applications Processor.

Figure 6.4 shows that *Space-based actuation* needs much less time than *REST actuation* to make a change. Furthermore, consumers and providers in *Space-based actuation* only perform trivial computing tasks: interpreting results. The space, which corresponds to our solution's *coordination space*, will be in charge of querying and notification mechanisms. The computation activity in the space varies depending on the complexity of these mechanisms' implementation, which is beyond the scope of this dissertation. However, note that due to the prototype nature of the subscription mechanism implemented, the *Space-based actuation* has still room for a great performance improvement in this metric.

Finally, regarding the time needed to complete *REST actuation*'s executions, the consumer spends most of it reasoning. Consequently, from the data consumer perspective, this approach will generally be more energy demanding than the first one. In contrast, this mechanism demands few things to the provider: to serve HTTP resources and to provide their descriptions.

Table 6.2: Foreseeable networking and computing impact on the nodes involved in the actuation mechanism.

Actuation	Perspective	Activity	
		Networking	Computation
Space-based	Provider	Proactive, limited activity	Limited: Results Interpretation
	Consumer	Proactive, limited activity	Limited: Results Interpretation
	Space	Reactive, high activity	Varies with the implementation
REST-based	Provider	Reactive, limited activity	Limited: Handling requests
	Consumer	Proactive, high activity	Demanding: Reasoning

Table 6.2 summarizes the previous discussion. As can be appreciated, the providers in the second actuation mechanism are more lightweight. They just attend to the request received using HTTP. Probably as a consequence of these few requirements, exposing the actuation capabilities of the limited devices

with HTTP is a consolidated trend. This tendency is backed by the WoT initiative. To make these web-enabled actuators automatically reusable by consumers, the second mechanism only requires them to describe their resources semantically. This can be done before deploying them and does not affect to their usual operation.

From the perspective of our space-based computing solution, reusing the existing HTTP providers will potentially open a new world of actuation possibilities. Preferably, we should do it keeping *Space-based actuation* consumers' simplicity.

6.3.2 **Baseline scenario: Implementation 3**

This implementation aims to prove that our space-based middleware can easily reuse third-party applications' providers capabilities. It presents the space and the consumer (i.e., *Node B*) from the *Implementation 1*, but it replaces the provider with the one from the *Implementation 2* (i.e., *Node C*). Both nodes have a good foundation for the interoperation because they use the same vocabularies. However, the *Node C* and the *Node B* rely on different communication mechanisms: direct communication and indirect communication.

To close the gap between these two worlds, we avoid changing the participant node's implementations. Instead, we create an agent which acts on the consumers' behalf. This agent resides in the same machine as the *Space*, but its existence must not interfere with the *Space* one. Therefore, it should run on an independent process.

The agent takes care of the tasks that *Node D* does in the *Implementation 2*: *(1)* crawls the discovered APIs[1], *(2)* reasons about their data to get a plan, and *(3)* follows the resulting plan performing HTTP requests.

To trigger the reasoning, the agent awaits for new tasks written into the space. Listing 10 shows the subscription template used by the agent. Providing that after reasoning the agent does not find a plan to achieve a task, it will write it into the space again. This way, another node which may know how to process it may take the task.

[1]The discovery process is out of the scope of this implementation.

Demanding new data from the developer would impede the transparent reuse of the nodes from *Implementation 1* and *Implementation 2*. Therefore, the agent reuses all the information pieces that it needs:

- It uses the *Node B*'s subscription to the task result as a goal for the reasoning. In our implementation, this correspondence needs a minimal mapping between N3QL [144] and SPARQL [162]. The reason why we use both languages are the underlying frameworks: EYE [149] and RDFLib [133].

- The agent uses all the content written into the space as *additional knowledge* for the reasoning process. This is feasible because the agent resides in the same machine as the space. Otherwise, acquiring this knowledge through the network would be too consuming both in bandwidth and in time.

6.4 Discussion

The following points scrutinize the strengths and weaknesses of the previous implementation and discuss other alternative designs.

6.4.1 Obtaining resource descriptions

The core of the *REST actuation* are the descriptions of the resources. They must be read by the nodes willing to actuate prior to reasoning. This action is performed by the consumer in *Implementation 2* and by the agent in *Implementation 3*. Both nodes crawl a given API starting from an URL to obtain the descriptions. The discovery of the initial URLs is out of the scope of this chapter.

Another alternative to discover these descriptions is to make them part of the *clues* presented in Chapter 5. This option ensures that they will be available in any *Consumer*. Furthermore, the static nature of these descriptions does not break the *clues* stability assumption in which our architecture is founded.

In any case, this chapter focuses on the interoperability problem, not on how to obtain the descriptions. Consequently, for the sake of clarity, we opted for

the more intuitive and common alternative. That is, we assume that a process crawls the descriptions in the background.

6.4.2 Obtaining background knowledge

In addition to resource descriptions, Verborgh et al.'s actuation mechanism also requires an initial state and background knowledge as inputs (see Section 6.2.2). In *Implementation 2*, the consumer obtains this knowledge crawling all the possible APIs. In *Implementation 3*, we also add all the knowledge from the space. This is feasible because it is located in the same machine as the agent which needs it. So, it does not demand any network usage.

However, as explained in Chapter 5, the data in UbiComp changes too frequently to simply crawl it from time to time. Crawling all the APIs each time a change occurs is also highly inefficient. Hence, the approach used to obtain knowledge is a simplification.

A possible optimization would be to benefit from the search architecture presented in Chapter 5. To reduce readings on the space (i.e., network usage), we propose a procedure composed by two reasoning steps. In the first one, we only use local incomplete knowledge derived from the *clues*. Then, we read from the *Space* just the knowledge needed to confirm the pre-proofs obtained in the first reasoning. The second reasoning uses this knowledge to get real pre-proofs.

Therefore, a node which wants to actuate will need to obtain the *clues* from the WP. Let us assume that these *clues* are composed by the predicates used by the nodes which provide content. The existence of a predicate used in a premise does not necessarily imply that this rule can be used. Nevertheless, its absence does imply that it will not be used (see Figure 6.5). Therefore, we can create temporary *activation rules* from *clues* which activate those potential rules.

An *activation rule* for a rule R contains a *true* in the premise. The conclusion is made by R's premise substituting the variables with fictitious URIs with a common prefix (see Figure 6.5). These fictitious URIs are used to distinguish when a triple should be replaced by actual knowledge from the space.

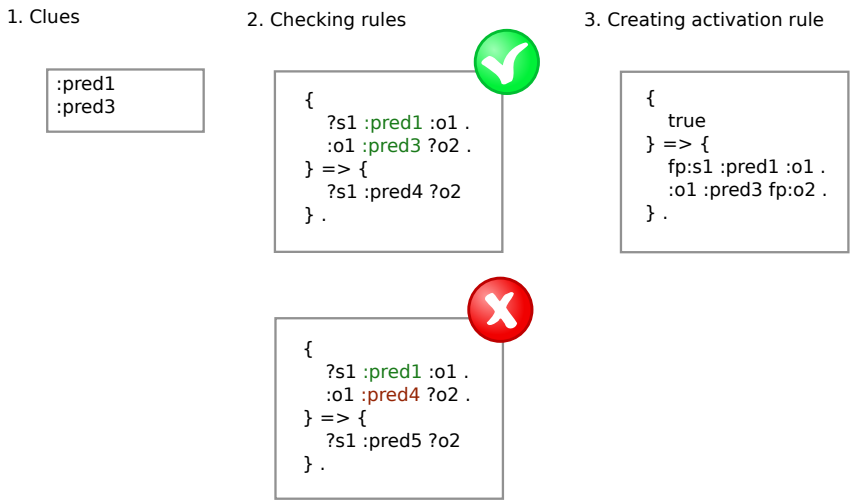


Figure 6.5: Sample clues, rules and the activation rule.
According to the shown clues, the second rule will never be invoked. Consequently, only an activation rule for the first one is created.

6.4.3 Responsibility for triggering REST actuation

Section 6.2.2 describes two coarse-grained steps to learn how to use web APIs which use *RESTdesc*:

1. Reasoning over the descriptions, background knowledge, an initial state and a goal state. The result of the reasoning process if a pre-proof, which can be seen as a tentative *execution plan* to achieve the goal.

2. Check the execution plan by following it.

Implementation 3 opts for triggering the reasoning process when an agent receives a notification. Previously, it subscribes to any type of task written into the space.

The reasoning can be performed in any node apart from the one which holds the *Space*:

- Any *Consumer* interested in changing the environment can trigger the process.

  If these *Consumers* use the search mechanism presented in Chapter 5, they will have background knowledge about other nodes.

This reduces the dependency on the node providing the *coordination space*. However, it requires them to perform tasks such as reasoning and checking the pre-proofs.

Unfortunately, the first task increases the computation and the second the network usage. As we already mentioned in previous chapters, these tasks severely affect to the energy consumption. Furthermore, some constrained platforms will not even be able to reason.

- To mitigate this problem, we can delegate this task only on the nodes able to perform such tasks. In fact, these nodes can follow the *replicated-worker pattern*. That is, they can read from the space goals to trigger the process (i.e., *reasoning tasks*). Apart from balancing the load between all the worker nodes, any node can stop being worker at any time by not taking more *tasks* (e.g., if it has low energy). These nodes must be *Consumers* to use the *clues* from the search mechanism as background knowledge.

Although both alternatives avoid the dependency on the *Space*, the space-based actuation mechanism intrinsically depends on the *Space*. Therefore, it makes sense that the unavailability of the space will cause the unavailability of actuating on the space. On the contrary, it simplifies the consumers' responsibilities, which just need to worry about writing a task into the space.

6.4.4 Interoperation weakness

The previous sections presented various alternative designs and their likely impacts on the actuation performance. However, none of them addresses the interoperability flaws of *Implementation 3*. In this regard, the simple mapping between a consumer's subscription and a goal is probably its most evident interoperability flaw.

In *Implementation 3*, the consumer subscription to a result and the goal for the reasoning must match. However, there is no guarantee that the consumer will always use a subscription which matches with a goal. For instance, the consumer could use a more general subscription and then filter the particular tasks

it is interested in. Even worse, there is no guarantee that the consumer will subscribe to any result. Thus, the universality of the proposed alignment can be easily affected. To avoid this undesired effect, any developer of a consumer node should bear in mind some good practices.

A more universal approach would be to deduce the goal from the task. For instance, from a task of *regulate temperature to 6° C* the *Space* could deduce the goal state of *temperature of 6° C*. In this case, the mapping should be either *(1)* provided by the consumer or *(2)* pre-set in the space. The first choice demands to feed the *Space* with additional information. The second choice assumes that a predefined ontology must be used or extended by the user to represent tasks. Therefore, it would limit the freedom of choosing any vocabulary to define a task.

Since this chapter simply wants to remark the potential interoperability of the presented approaches, we opted for selecting the automatic translation from a subscription to a goal. The implementation of the rest of the approaches is left as future work.

6.4.5 **Advanced challenges**

The scenario used as a guiding example is very basic. Consequently, the shown interoperation example requires further work to check its feasibility in more advanced scenarios. We anticipate the following challenges:

- If there are two or more paths to reach a goal, how can we discern which one to follow? This problem is specific to the *REST actuation*.
- How does the middleware deal with the coexistence of both mechanisms? When both methods can be applied, which one is triggered? Will one of them prevail over the second?

6.5 **Summary**

This chapter presented two techniques to actuate on the physical environment. The first technique is the usual way to operate through spaces and provides a

higher degree of decoupling. However, it requires actuator nodes to use our middleware's primitives.

The second actuation mechanism directly consumes RESTful HTTP APIs. This mechanism relies in the semantic description of the services, additional background knowledge and a goal state. With that information, it is able to generate executions plans towards a goal. These plans detail which resources to change and how to reach the desired goal.

We implemented the same scenario using these two actuation mechanisms. In addition, since interoperability is one of our middleware's guiding principles, we sketched a third implementation which reuses these RESTful HTTP APIs in our *Space* model. This hybrid actuation technique avoids any alteration on the space-based consumer or the HTTP provider. Instead, it improves the *Space* implementation with an agent in charge of generating execution plans. This agent reuses the information from the space-based actuation not to require any additional information to the developer.

This implementation alignment between our space-based actuation and the direct web APIs consumption-based one presents some limitations. For some of these limitations, we discussed other design alternatives and compared them with the chosen one. The rest of the limitations only appear in more complex scenarios which are beyond the scope of this dissertation. For our future work, we will implement these complex scenarios where advanced conflicts between the REST and space-based computing worlds can arise.

```
{
  # it is not 'just a measure'
  ?obsv a ssn:ObservationValue ;
      # it is also a preference
      a frap:Preference ;
      dul:isClassifiedBy  ucum:lux ;
      dul:hasDataValue ?desired_value .
} => {
  _:request http:methodName "POST";
            http:requestURI actuators:light ;
            http:body ?desired_value ;
            http:resp [ http:body ?lightObs ].

  actuators:light ssn:madeObservation ?lightObs .

  ?lightObs a ssn:Observation ;
         ssn:observedProperty sweet:Light ;
         ssn:observedBy actuators:light ;
         ssn:observationResult ?so .

  ?so ssn:hasValue ?ov .

  ?ov a ssn:ObservationValue ;
      dul:isClassifiedBy ucum:lux ;
      dul:hasDataValue ?desired_value .
}.
```

Listing 7: Rule which expresses that having a preference which is measured in luxes, one can create a light observation using the HTTP POST.

```
@prefix frap:  <http://purl.org/frap/> .
@prefix dul:  <http://www.loa.istc.cnr.it/ontologies/DUL.owl#> .
@prefix ssn:  <http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#> .
@prefix ucum:  <http://purl.oclc.org/NET/muo/ucum/> .
@prefix : <http://example.org/lamp/>.


# Description of the preference
:obsv a ssn:ObservationValue, frap:Preference ;
      dul:isClassifiedBy  ucum:lux ;
      dul:hasDataValue 19 .
```

Listing 8: A preference which expresses the interest in modifying the sensed value of a light.

```
{
  # More things could be specified.
  # E.g. location.

  actuators:light ssn:madeObservation ?light .

  ?light ssn:observedProperty  sweet:Light ;
         ssn:observationResult ?so .

  ?so ssn:hasValue ?ov .

  ?ov a ssn:ObservationValue ;
      dul:isClassifiedBy  ucum:lux ;
      dul:hasDataValue 19 .
} => {
  ?ov  dul:hasDataValue  ?val .
}.
```

Listing 9: A goal which expresses the interest in modifying the value for a light.

```
prefix frap: <http://purl.org/frap/>

select ?pref where{
        ?pref a frap:Preference .
}
```

Listing 10: Subscription to any task written into the space.

If we build for a better future the immigrants will stay there.

Man in Big Gay Pile

South Park. Season 8, Episode 6.

CHAPTER

7

Conclusions

This dissertation explores the design of a distributed TSC middleware suitable for UbiComp environments. The middleware's most remarkable characteristic is its alignment with the web world. This alignment pursues the interoperability of the middleware with other semantic web applications and semantic WoT solutions. Firstly, it reuses their data requiring them to implement a minimal REST-like API. Secondly, it enables these third-party web applications to reuse the data written into the space by means of an analogous API.

To preserve as many beneficial properties as possible from the web and TS worlds, the middleware presents a conceptual division of each *Space* (Chapter 4). Any *Space* is composed by the *coordination space* and the *outer space*. The *coordination space* enables a completely indirect communication, time and space decoupled, between the nodes which write and read semantic content into it. The *outer space* acknowledges the distributed nature of UbiComp environments by letting the nodes manage their own semantic content. However, this content is shared with the rest of nodes forming a *virtual* space. Developers access the content of the space in a associative manner keeping the space decoupling property. Both spaces are accessed through a resource-oriented HTTP API to promote their data reuse by third-party REST or REST-like applications.

The distribution of the information avoids participants' dependency on other devices, but arises other challenges. This dissertation tackles two main challenges: how to search in a decentralized way (Chapter 5) and how to act on the physical environment through the space (Chapter 6).

To enhance the search process between devices limiting their dependencies on others, this dissertation proposes an architecture. This architecture dynamically chooses an intermediary according to its capabilities to support the most limited devices. The final result is that fewer requests are required in overall, specifically reducing the load on resource constrained devices. As a result, their energy consumption is also reduced.

To actuate on the physical environment, the dissertation explains how to apply TS application patterns to the UbiComp vision. Although these actuation mechanisms promote indirect communication between nodes, they require them to use the coordination space. This avoids the seamless integration of actuators from the WoT world. To overcome this limitation, i.e., to promote the reuse of third-party WoT applications' actuation capabilities, we describe how to integrate a REST-based actuation mechanism with the regular TSC usage.

7.1 Contributions

This section summarizes the contributions of this dissertation.

- Chapter 3 presented an in-depth state-of-the-art review.
  - It portrayed this dissertation regarding the relevant works from related research fields. Particularly, it intensified the analysis on the semantic space-based computing middleware. This analysis showed that other works do not address the problems tackled in this dissertation.
- Chapter 4 depicted a two-space model which *(1)* preserves TSC's decoupling properties and *(2)* considers UbiComp's distributed nature.

  - It described a dual space model. The first space is a common place where all the participants can write and read. The second space is a *read-only* virtual *Space* formed by contents managed by the different participants. It is accessed by any node as a whole using the same primitives as the first one.
  - It designed the middleware by presenting the adopted primitives and how the spaces can be accessed through an HTTP API.
  - It analysed the properties of the designed middleware from different perspectives. Particularly, it scrutinized which beneficial properties from TSC or REST are retained. It also remarked the middleware's suitability for resource constrained devices.

- Chapter 5 presented a search architecture for semantic WoT solutions. This architecture is compatible with, but it is not only applicable to, the space model proposed.
  - It presented a search architecture that:
    * Balances the load that the architecture management generates. This balance considers devices' computing and energy capacities.
    * Structures the nodes in different dynamic roles with different responsibilities.
    * Promotes end-to-end HTTP requests between constrained devices. They do not need an intermediary to search (i.e., to select the appropriate providers to request).
  - It assessed different types of information summaries the nodes can use to improve their search.
  - It evaluated the proposal with a simulated but yet realistic environment.

- Chapter 6 analysed how to actuate on the physical environment using a space. Particularly, it showed how to interoperate with existing REST-based actuators from space-based computing.

  - It explained how to use TSC to coordinate actuator nodes and nodes willing to actuate.
  - It depicted the need of a subscription system and its key requirements.
  - It compared the space-based actuation technique with a direct actuation one based on the REST architectural style.
  - It contributed with a hybrid approach which seamlessly reuses REST APIs from our space model.

- In addition, as a result of the theoretical work made on this dissertation, we have done the following technical contributions:
  - *Otsopack* [155]: a TSC middleware which works over HTTP and implements most of the ideas presented in the dissertation. This middleware is publicly available for different computing platforms and has been used in several research projects.
  - A fully parametrizable simulation framework to evaluate different communication strategies [154].
  - Three different implementations of the same simple UbiComp scenario which illustrate the ideas explained in Chapter 6. Much of these implementations are fully reusable for further future evaluations [153].

Finally, parallel to this dissertation, but deeply influenced by it, the author of this dissertation has collaborated with other academic institutions, research centres and companies to make further scientific and technical contributions.

- Lightweight user access control for resource constrained devices.
- Apply the *Otsopack* middleware in several scenarios:
  - Homes: automating them according to the user's preferences.
  - Hospitals and residences: tracking the evolution of patients with cognitive impairments.

- Supermarkets: aiding to buy and easing mobility using robots.
- Hotels: intelligently adapting them to the customers' needs.

7.2 Relevant publications

Most of the previous section's contributions have been presented to the scientific community in journals, conferences and workshops.

Particularly, the following publications have addressed how to search on a distributed space:

> Aitor Gómez-Goiri, Iñigo Goiri, and Diego López-de-Ipiña. Energy-aware architecture for information search in the semantic web of things. *International Journal of Web and Grid Services*, 10:192–217, 2014. To appear.
>
> Aitor Gómez-Goiri and Diego López-de-Ipiña. Assessing data dissemination strategies within triple spaces on the web of things. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 763 –769, July 2012. doi: 10.1109/IMIS.2012.120

We have considered the interoperation issues between different actuation techniques in:

> Aitor Gómez-Goiri, Iñigo Goiri, and Diego López-de-Ipiña. Reusing web-enabled actuators from a semantic space-based perspective. In *Third International Workshop on Extending Seamlessly to the Internet of Things (esIoT)*, July 2014. To appear.

Although, this dissertation includes some updates and improvements on the middleware design, early works on aligning TSC and REST/WoT include:

> Aitor Gómez-Goiri, Pablo Orduña, Javier Diego, and Diego López-de-Ipiña. Otsopack: Lightweight semantic framework for interoperable ambient intelligence applications. *Computers in Human Behavior*, 30: 460–467, January 2014. ISSN 0747-5632. doi: 10.1016/j.chb.2013.06.022. URL `http://www.sciencedirect.com/science/article/pii/S0747563213002148`

Aitor Gómez-Goiri, Pablo Orduña, and Diego López-de-Ipiña. RESTful triple spaces of things. In *Proceedings of the Third International Workshop on the Web of Things*, WOT '12, page 5:1–5:6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1603-3. doi: 10.1145/2379756.2379761. URL http://doi.acm.org/10.1145/2379756.2379761

Aitor Gómez-Goiri, Pablo Orduña, David Ausin, Mikel Emaldi, and Diego López-de-Ipiña. Collaboration of sensors and actuators through triple spaces. In *2011 IEEE Sensors*, pages 651–654, Limerick, Ireland, October 2011. IEEE. ISBN 978-1-4244-9290-9. doi: 10.1109/ICSENS.2011.6127316

Aitor Gómez-Goiri and Diego López-de-Ipiña. On the complementarity of triple spaces and the web of things. In *Proceedings of the Second International Workshop on Web of Things*, WoT '11, page 12:1–12:6, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0624-9. doi: 10.1145/1993966.1993983. URL http://doi.acm.org/10.1145/1993966.1993983

Prior to that work, we explored the idea of a distributed TSC middleware and how to use it in UbiComp in several scientific articles.

Aitor Gómez-Goiri and Diego López-de-Ipiña. A triple space-based semantic distributed middleware for internet of things. In Florian Daniel and Federico Facca, editors, *Current Trends in Web Engineering*, volume 6385 of *Lecture Notes in Computer Science*, page 447–458. Springer Berlin / Heidelberg, September 2010. URL http://dx.doi.org/10.1007/978-3-642-16985-4_43. 10.1007/978-3-642-16985-4_43

Aitor Gómez-Goiri, Mikel Emaldi, and Diego López-de-Ipiña. A semantic resource oriented middleware for pervasive environments. *UPGRADE journal*, 2011, Issue No. 1:5–16, February 2011. ISSN 1684-5285. URL http://www.cepis.org/upgrade/media/UPGRADE_1_2011_Full1.pdf

Aitor Gómez-Goiri, Mikel Emaldi, and Diego López-de-Ipiña. Middleware semántico orientado a recursos para entornos ubicuos. *Novatica journal*, (209):9–16, February 2011. ISSN 0211-2124

As previously stated, this TSC middleware has been used to solve problems from different domains. The following publications explain some of these application domains.

Eduardo Castillejo, Pablo Orduña, Xabier Laiseca, Aitor Gómez-Goiri, Diego López-de-Ipiña, and Fínez Sergio. Distributed semantic middleware for social robotic services. In *Robot 2011*, Seville, Spain, November 2011

Aitor Gómez-Goiri, Eduardo Castillejo, Pablo Orduña, Xabier Laiseca, Diego López-de-Ipiña, and Sergio Fínez. Easing the mobility of disabled people in supermarkets using a distributed solution. In José Bravo, Ramón Hervás, and Vladimir Villarreal, editors, *Ambient Assisted Living*, number 6693 in Lecture Notes in Computer Science, pages 41–48. Springer Berlin Heidelberg, January 2011. ISBN 978-3-642-21302-1, 978-3-642-21303-8. URL `http://link.springer.com/chapter/10.1007/978-3-642-21303-8_6`

Xabier Laiseca, Eduardo Castillejo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña, and Ester González Aguado. Distributed tracking system for patients with cognitive impairments. In José Bravo, Ramón Hervás, and Vladimir Villarreal, editors, *Ambient Assisted Living*, number 6693 in Lecture Notes in Computer Science, pages 49–56. Springer Berlin Heidelberg, January 2011. ISBN 978-3-642-21302-1, 978-3-642-21303-8. URL `http://link.springer.com/chapter/10.1007/978-3-642-21303-8_7`

Finally, we have also worked on securing the communication with resource constrained devices.

Juan Álvaro Muñoz Naranjo, Aitor Gómez-Goiri, Pablo Orduña, Diego López-de-Ipiña, and Leocadio González Casado. Extending a user access control proposal for wireless network services with hierarchical user credentials. In Álvaro Herrero, Bruno Baruque, Fanny Klett, Ajith Abraham, Václav Snášel, André C. P. L. F. de Carvalho, Pablo García Bringas, Ivan Zelinka, Héctor Quintián, and Emilio Corchado, editors, *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, number 239 in Advances in Intelligent Systems and Computing, pages 601–610. Springer International Publishing, January 2014. ISBN 978-3-319-01853-9, 978-3-319-01854-6. URL http://link.springer.com/chapter/10.1007/978-3-319-01854-6_61

Juan Álvaro Muñoz Naranjo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña, and Leocadio González Casado. Enabling user access control in energy-constrained wireless smart environments. *Journal of Universal Computer Science*, 19(17):2490–2505, November 2013. URL http://www.jucs.org/jucs_19_17/enabling_user_access_control

Juan Álvaro Muñoz Naranjo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña, and L. G. Casado. Lightweight user access control in energy-constrained wireless network services. In José Bravo, Diego López-de-Ipiña, and Francisco Moya, editors, *Ubiquitous Computing and Ambient Intelligence*, Lecture Notes in Computer Science, pages 33–41. Springer Berlin Heidelberg, January 2012. ISBN 978-3-642-35376-5, 978-3-642-35377-2. URL http://link.springer.com/chapter/10.1007/978-3-642-35377-2_5

7.3 Future work

Previous chapters of this dissertation have pointed out several limitations and open issues. These limitations and issues open the door to interesting future work which is detailed below.

7.3.1 Closing the gap with third-party WoT applications

One of the middleware's goals is to achieve interoperability with third-party web applications. That is, ideally, applications using our middleware should reuse these other applications' data transparently and vice-versa. This goal can be seen as a two-fold strategy: ease the access to data and make data reusable.

Accessing data. It must be done by means of an API. In this thesis we have defended the need of using HTTP for that purpose. An adaptation to the upcoming CoAP would ease its adoption by new limited platforms. However, the most challenging aspect is not to replicate it in other suitable protocols. The most challenging aspect is to automatize the consumption of heterogeneous third-party APIs. Instead, this dissertation proposes a common API that needs to be implemented in any device willing to share content.

According to the REST principles, the HATEOAS constraint can help to automate clients to use any API independently of its specific shape. However, making any of these clients understand other APIs from the same domain would require the standardization of data types and application states [28]. Therefore, any step in that direction would require the agreement of the community around the domain (e.g., home automation lighting). Furthermore, hypermedia-driven APIs may lead to more network usage, which is particularly harmful for devices with energy autonomy limitations. Consequently, this trade-off between the interoperability and the network efficiency should also be assessed.

Reusing data. The semantic knowledge contributes to that goal. However, as Barnaghi et al. [7] point out, the use of semantic descriptions alone does not ensure interoperability. Additionally, they suggest the need of: *(1)* standardizing ontologies and using them or referencing to standardized upper ontologies whenever custom ontologies are used; and *(2)* interpreting the annotations effectively. While the first one requires the consensus of the community, the second demands for efficient reasoners for mobile and embedded platforms [79]. This efficiency is particularly important since it directly influences their energy consumption.

7.3.2 Increasing search expressibility and efficiency

We did not consider using query languages such as SPARQL [162] on the distributed search because of the difficulties most constrained platforms may find in parsing them. With this decision, we design a uniform API for all types of devices. However, we could adopt SPARQL in an optional extended API. In this case, all the nodes should consider that some of them are only able to interpret triple pattern templates.

The use of SPARQL in some nodes would benefit the developers and improve the middleware performance. The users of the middleware would be able to made more expressive queries. The performance could be enhanced by introducing optimization techniques from the distributed databases field [101].

7.3.3 Coordination space's distribution, replication or migration

The current design demands the *coordination space* to be accessible through an HTTP API. For the sake of simplicity, at some points of the dissertation we have assumed that this space is centralized in a unique machine. However, the access through an HTTP API does not avoid the distribution of the *coordination space*. Some alternatives that can be worth exploring are:

- Replication of the space to increase the system performance (e.g., using load balancing).
- Distribution of the content over different machines to enhance its scalability.
- Migration from a machine to another to avoid a single point of failure. For example, this strategy would allow to select the space holder dynamically, easing the *coordination space* manageability and deployment. This would be analogous to the WP role of our search architecture.

In addition, by monitoring the middleware usage, one alternative or another could be dynamically chosen depending on the specific needs of each situation.

7.3.4 Integration of *REST actuation* in the space

Chapter 6 presented a hybrid actuation technique which combines consumers using the *Space-based actuation* and REST providers. The key for this integration is the direct translation of the *coordination space*'s subscriptions into goals needed by the *REST actuation*. This avoids requiring any new piece of information which does not conceptually belong to TSC to the developer. However, the consumer is not forced to perform any subscription. Therefore, the middleware should force the developer to use subscriptions for a task result or extract the goals from other pieces of information.

An option would be to translate the tasks written into the space to trigger an *actuator change* into goals. This way, no new information would be required to the consumer following the *Space-based actuation*. This translation requires further work to ensure that it is generalizable.

Another interesting issue is to discern which path to follow to achieve a goal when two or more paths are available. We believe that additional high-level rules or policies can be defined (e.g., to select the path which consumed less energy). Storing them into the space may enable reusing these *behaviours* by third-party applications' developers.

Finally, although we put effort in reusing third-party WoT applications' actuation capabilities in TSC, the opposite problem raises an unanswered question: How to reuse actuation mechanisms of nodes using TS patterns from WoT solutions?

7.3.5 Security for the middleware

Two key aspects for UbiComp deployments are *(1)* to authenticate the parts involved in the communication; and *(2)* to ensure the privacy of the message being transmitted. Although both problems have been deeply discussed in the literature, devices in UbiComp face severe computing and energy restrictions. To solve that, we have already proposed a solution appropriate for sensor networks and mobile devices.

The next step is to evaluate it in more demanding scenarios and the final goal is to integrate it in our TSC middleware.

7.3.6 **Measuring ease of usage**

A subtle goal of any middleware is to ease the development of applications by encapsulating complex tasks (i.e., searching on a distributed space). Some objective indicators can be extracted from the code by implementing the same application with and without the middleware (e.g., the total number of lines needed). However, from the human perspective these indicators do not necessarily express how easy, comfortable or pleasant is to use the middleware. Although the primitives are rather simple, the configuration or just the semantic annotation can complicate the learning process of any developer. Future work could measure the impact the middleware has on them. As a result, some improvements could be proposed (e.g., to create a version for a specific domain which masks all the semantic issues).

7.3.7 **Further evaluation on real deployments**

Rather than designing the best theoretical but yet unrealistic solution, in this dissertation we have tried to present a realizable solution. For that, we have implemented most of the ideas in the *Otsopack* middleware and we have promoted its use in different research projects. This has helped us to identify the key problems to solve.

In addition, to efficiently evaluate our solution under very different and extreme circumstances, in some occasions we have simulated the scenarios. In these simulations, we have tried to present always realistic assumptions. However, there is always a gap between what is perceived as realistic and the reality.

To bridge both worlds, the next natural step would be assessing some aspects of our middleware in a real world environment:

- The search architecture. For that, we should deploy an scenario with a considerable number of heterogeneous devices running a task during a long period. Then, we could measure their network usage and energy consumption to either tune up some parameters of the architecture or redesign some aspects.

- Interoperability. New unconsidered problems could arise trying to reuse data from existing WoT applications. Each manual intervention needed to make our middleware work together with these applications would be a sign of an interoperability problem.

7.4 Final Thoughts

This dissertation is the result of years of work towards a Triple Space Computing middleware for Ubiquitous Computing. It describes the latest shape of the idea which I began sketching even before I finally decided to start this thesis. But the evolution of this idea would not have been possible without other researchers who have directly or indirectly helped me. In the same way, I hope that this thesis will contribute or inspire at some degree to the scientific community.

If I have seen further it is by standing on the shoulders of giants.
Isaac Newton

(This figure is derived from the following images: "Standing upon the shoulders of giants" by Mushon Zer-Aviv and "Aerial view - mountains" by Gabriel Gilder)

Bibliography

[1] IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, 1990. doi: 10.1109/IEEESTD.1990.101064.

[2] European interoperability framework for pan-european eGovernment services. Technical Report 1, European Communities, 2004. URL `http://ec.europa.eu/idabc/servlets/Docd552.pdf?id=19529`.

[3] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer Publishing Company, Incorporated, 2010.

[4] Mike Amundsen. API for RDF? don't do it!, December 2010. URL `http://www.amundsen.com/blog/archives/1083`.

[5] Mike Amundsen. *Building Hypermedia APIs with HTML5 and Node*. O'Reilly, 2011. ISBN 978-1449306571.

[6] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010. ISSN 1389-1286. doi: 10.1016/j.comnet.2010.05.010. URL `http://www.sciencedirect.com/science/article/pii/S1389128610001568`.

[7] Payam Barnaghi, Wei Wang, Cory Henson, and Kerry Taylor. Semantics for the internet of things. *International Journal on Semantic Web and Information Systems*, 8(1):1–21, 2012. ISSN 1552-6283, 1552-6291.

doi: 10.4018/jswis.2012010101. URL http://www.igi-global.com/article/content/70584.

[8] Tim Berners-Lee. Linked open data, 2008. URL http://www.w3.org/2008/Talks/0617-lod-tbl/#(3).

[9] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[10] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3Logic: a logical framework for the world wide web. *Theory and Practice of Logic Programming*, 8(03):249–269, 2008. ISSN 1475-3081. doi: 10.1017/S1471068407003213. URL http://journals.cambridge.org/article_S1471068407003213.

[11] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009. ISSN 1552-6283, 1552-6291. doi: 10.4018/jswis.2009081901. URL http://dx.doi.org/10.4018/jswis.2009081901.

[12] Michael Blackstock and Rodger Lea. WoTKit: a lightweight toolkit for the web of things. In *Proceedings of the Third International Workshop on the Web of Things*, WOT '12, page 3:1–3:6, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1603-3. doi: 10.1145/2379756.2379759. URL http://doi.acm.org/10.1145/2379756.2379759.

[13] Daniel Blunder, Reto Krummenacher, and Dieter Fensel. A distributed triple space realization. Master's thesis, University of Innsbruck, Institute of Computer Science, Innsbruck, June 2009. URL http://www.sti-innsbruck.at/sites/default/files/thesis/danielblunder_DistrTripleSpace_2009.pdf.

[14] Arne Bröring, Krzysztof Janowicz, Christoph Stasch, and Werner Kuhn. Semantic challenges for sensor plug and play. In James D. Carswell,

A. Stewart Fotheringham, and Gavin McArdle, editors, *Web and Wireless Geographical Information Systems*, number 5886 in Lecture Notes in Computer Science, pages 72–86. Springer Berlin Heidelberg, January 2009. ISBN 978-3-642-10600-2, 978-3-642-10601-9. URL `http://link.springer.com/chapter/10.1007/978-3-642-10601-9_6`.

[15] Ramón Cáceres and Adrian Friday. Ubicomp systems at 20: Progress, opportunities, and challenges. *IEEE Pervasive Computing*, 11(1):14–21, March 2012. ISSN 1536-1268. doi: 10.1109/MPRV.2011.85.

[16] Eduardo Castillejo, Pablo Orduña, Xabier Laiseca, Aitor Gómez-Goiri, Diego López-de-Ipiña, and Fínez Sergio. Distributed semantic middleware for social robotic services. In *Robot 2011*, Seville, Spain, November 2011.

[17] World Wide Web Consortium. W3c semantic web faq, August 2011. URL `http://www.w3.org/2001/sw/SW-FAQ`.

[18] Paolo Costa, Luca Mottola, Amy Lynn Murphy, and Gian Pietro Picco. Programming wireless sensor networks with the TeenyLime middleware. In Renato Cerqueira and Roy H. Campbell, editors, *Middleware 2007*, number 4834 in Lecture Notes in Computer Science, pages 429–449. Springer Berlin Heidelberg, January 2007. ISBN 978-3-540-76777-0, 978-3-540-76778-7. URL `http://link.springer.com/chapter/10.1007/978-3-540-76778-7_22`.

[19] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison Wesley, 5 edition, 2012. ISBN 9780132143011.

[20] Carlo Curino, Matteo Giani, Marco Giorgetta, Alessandro Giusti, Amy Lynn Murphy, and Gian Pietro Picco. TinyLIME: bridging mobile and sensor networks through middleware. In *Third IEEE International Conference on Pervasive Computing and Communications, 2005. PerCom 2005*, pages 61–72, 2005. doi: 10.1109/PERCOM.2005.48.

[21] Mathieu D'Aquin, Andry Nikolov, and Enrico Motta. Enabling lightweight semantic sensor networks on android devices. In *Proceedings of the 4th International Workshop on Semantic Sensor Networks*, volume 839 of *CEUR Workshop Proceedings*, pages 89–94, 2011. URL http://ceur-ws.org/Vol-839/daquin.pdf.

[22] Nigel Davies, Adrian Friday, Stephen P. Wade, and Gordon S. Blair. L^{2}imbo: a distributed systems platform for mobile computing. *Mobile Networks and Applications*, 3(2):143–156, August 1998. ISSN 1383-469X. doi: 10.1023/A:1019116530113. URL http://dx.doi.org/10.1023/A:1019116530113.

[23] Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3): 319, September 1989. ISSN 02767783. doi: 10.2307/249008. URL http://www.jstor.org/discover/10.2307/249008?uid=20425920&uid=3737952&uid=2&uid=3&uid=16734728&uid=67&uid=62&sid=21102588550311.

[24] Dieter Fensel. Triple-space computing: Semantic web services based on persistent publication of information. In Finn Arve Aagesen, Chutiporn Anutariya, and Vilas Wuwongse, editors, *Intelligence in Communication Systems*, volume 3283 of *Lecture Notes in Computer Science*, pages 43–53. Springer Berlin / Heidelberg, 2004. URL http://dx.doi.org/10.1007/978-3-540-30179-0_4. 10.1007/978-3-540-30179-0_4.

[25] Dieter Fensel, Reto Krummenacher, Omair Shafiq, Eva Kühn, Johannes Riemer, Ying Ding, and Bernd Draxler. Tsc – triple space computing. *Elektrotechnik und Informationstechnik*, 124:31–38, 2007. ISSN 0932-383X. doi: 10.1007/s00502-006-0408-1. URL http://dx.doi.org/10.1007/s00502-006-0408-1.

[26] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine,

2000. URL http://www.industrex.com/dynamic/reading/ics.uci.edu.fielding.dissertation.1col.pdf.

[27] Roy Thomas Fielding. Seeking feedback on the blinksale API, October 2006. URL http://groups.yahoo.com/neo/groups/rest-discuss/conversations/topics/6594.

[28] Roy Thomas Fielding. REST APIs must be hypertext-driven, October 2008. URL http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven.

[29] Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces principles, patterns, and practice*. Addison-Wesley Professional, 1999.

[30] Adrian Friday, Nigel Davies, Jochen Seitz, Matt Storey, and Stephen P. Wade. Experiences of using generative communications to support adaptive mobile applications. *Distributed and Parallel Databases*, 7(3):319–342, 1999.

[31] David Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(1):80–112, 1985. ISSN 0164-0925.

[32] Dominique Guinard. *A Web of Things Application Architecture – Integrating the Real-World into the Web*. Ph.d., ETH Zurich, 2011. URL http://webofthings.org/dom/thesis.pdf.

[33] Dominique Guinard, Vlad Trifa, Thomas Pham, and Olivier Liechti. Towards physical mashups in the web of things. In *Networked Sensing Systems (INSS), 2009 Sixth International Conference on*, pages 1–4, June 2009. doi: 10.1109/INSS.2009.5409925.

[34] Dominique Guinard, Mathias Fischer, and Vlad Trifa. Sharing using social networks in a composable web of things. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 702–707. IEEE, April 2010. ISBN 978-1-4244-6605-4. doi: 10.1109/PERCOMW.2010.5470524.

[35] Dominique Guinard, Iulia Ion, and Simon Mayer. In search of an internet of things service architecture: REST or WS-*? a developers' perspective. In *Proceedings of Mobiquitous 2011 (8th International ICST Conference on Mobile and Ubiquitous Systems)*, Copenhagen, Denmark, December 2011.

[36] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource oriented architecture and best practices. In *Architecting the Internet of Things*. Springer, May 2011. ISBN 978-3-642-19156-5.

[37] Vipul Gupta, Poornaprajna Udupi, and Arshan Poursohi. Early lessons from building Sensor.Network: an open data exchange for the web of things. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 738–744. IEEE, March 2010. ISBN 978-1-4244-6605-4. doi: 10.1109/PERCOMW.2010.5470530.

[38] Vipul Gupta, Ron Goldman, and Poornaprajna Udupi. A network architecture for the web of things. In *Proceedings of the Second International Workshop on Web of Things - WoT '11*, page 1, San Francisco, California, 2011. doi: 10.1145/1993966.1993971. URL `http://dl.acm.org/citation.cfm?id=1993971&CFID=38852141&CFTOKEN=66966366`.

[39] Aitor Gómez-Goiri and Diego López-de-Ipiña. A triple space-based semantic distributed middleware for internet of things. In Florian Daniel and Federico Facca, editors, *Current Trends in Web Engineering*, volume 6385 of *Lecture Notes in Computer Science*, page 447–458. Springer Berlin / Heidelberg, September 2010. URL `http://dx.doi.org/10.1007/978-3-642-16985-4_43`. 10.1007/978-3-642-16985-4_43.

[40] Aitor Gómez-Goiri and Diego López-de-Ipiña. On the complementarity of triple spaces and the web of things. In *Proceedings of the Second International Workshop on Web of Things*, WoT '11, page 12:1–12:6, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0624-9. doi: 10.1145/1993966.1993983. URL `http://doi.acm.org/10.1145/1993966.1993983`.

[41] Aitor Gómez-Goiri and Diego López-de-Ipiña. Assessing data dissemination strategies within triple spaces on the web of things. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 763 –769, July 2012. doi: 10.1109/IMIS.2012.120.

[42] Aitor Gómez-Goiri, Eduardo Castillejo, Pablo Orduña, Xabier Laiseca, Diego López-de-Ipiña, and Sergio Fínez. Easing the mobility of disabled people in supermarkets using a distributed solution. In José Bravo, Ramón Hervás, and Vladimir Villarreal, editors, *Ambient Assisted Living*, number 6693 in Lecture Notes in Computer Science, pages 41–48. Springer Berlin Heidelberg, January 2011. ISBN 978-3-642-21302-1, 978-3-642-21303-8. URL `http://link.springer.com/chapter/10.1007/978-3-642-21303-8_6`.

[43] Aitor Gómez-Goiri, Mikel Emaldi, and Diego López-de-Ipiña. Middleware semántico orientado a recursos para entornos ubicuos. *Novatica journal*, (209):9–16, February 2011. ISSN 0211-2124.

[44] Aitor Gómez-Goiri, Mikel Emaldi, and Diego López-de-Ipiña. A semantic resource oriented middleware for pervasive environments. *UPGRADE journal*, 2011, Issue No. 1:5–16, February 2011. ISSN 1684-5285. URL `http://www.cepis.org/upgrade/media/UPGRADE_1_2011_Full1.pdf`.

[45] Aitor Gómez-Goiri, Pablo Orduña, David Ausin, Mikel Emaldi, and Diego López-de-Ipiña. Collaboration of sensors and actuators through triple spaces. In *2011 IEEE Sensors*, pages 651–654, Limerick, Ireland, October 2011. IEEE. ISBN 978-1-4244-9290-9. doi: 10.1109/ICSENS.2011.6127316.

[46] Aitor Gómez-Goiri, Pablo Orduña, and Diego López-de-Ipiña. RESTful triple spaces of things. In *Proceedings of the Third International Workshop on the Web of Things*, WOT '12, page 5:1–5:6, New York, NY, USA, 2012.

ACM. ISBN 978-1-4503-1603-3. doi: 10.1145/2379756.2379761. URL http://doi.acm.org/10.1145/2379756.2379761.

[47] Aitor Gómez-Goiri, Iñigo Goiri, and Diego López-de-Ipiña. Energy-aware architecture for information search in the semantic web of things. *International Journal of Web and Grid Services*, 10:192–217, 2014. To appear. .

[48] Aitor Gómez-Goiri, Iñigo Goiri, and Diego López-de-Ipiña. Reusing web-enabled actuators from a semantic space-based perspective. In *Third International Workshop on Extending Seamlessly to the Internet of Things (esIoT)*, July 2014. To appear. .

[49] Aitor Gómez-Goiri, Pablo Orduña, Javier Diego, and Diego López-de-Ipiña. Otsopack: Lightweight semantic framework for interoperable ambient intelligence applications. *Computers in Human Behavior*, 30: 460–467, January 2014. ISSN 0747-5632. doi: 10.1016/j.chb.2013.06.022. URL http://www.sciencedirect.com/science/article/pii/S0747563213002148.

[50] Olaf Görlitz and Steffen Staab. Federated data management and query optimization for linked open data. In Athena Vakali and Lakhmi C. Jain, editors, *New Directions in Web Data Management 1*, number 331 in Studies in Computational Intelligence, pages 109–137. Springer Berlin Heidelberg, January 2011. ISBN 978-3-642-17550-3, 978-3-642-17551-0. URL http://link.springer.com/chapter/10.1007/978-3-642-17551-0_5.

[51] Michael J Hammel. Mongoose: an embeddable web server in c. *Linux Journal*, 2010(192):2, 2010.

[52] Henning Hasemann, Alexander Kroller, and Max Pagel. RDF provisioning for the internet of things. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 143 –150, October 2012. doi: 10.1109/IOT.2012.6402316.

[53] Antonio Garrote Hernández and María N. Moreno García. A formal definition of RESTful semantic web services. In *Proceedings of the First International Workshop on RESTful Design*, WS-REST '10, page 39–45, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-959-6. doi: 10.1145/1798354.1798384. URL `http://doi.acm.org/10.1145/1798354.1798384`.

[54] Jukka Honkola, Hannu Laine, Ronald Brown, and Olli Tyrkko. Smart-m3 information sharing platform. In *2010 IEEE Symposium on Computers and Communications (ISCC)*, pages 1041–1046. IEEE, June 2010. ISBN 978-1-4244-7754-8. doi: 10.1109/ISCC.2010.5546642.

[55] Cory House. How RESTful is your API?, August 2012. URL `http://www.bitnative.com/2012/08/26/how-restful-is-your-api/`.

[56] Isam Ishaq, Jeroen Hoebeke, Jen Rossey, Eli De Poorter, Ingrid Moerman, and Piet Demeester. Facilitating sensor deployment, discovery and resource access using embedded web services. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 717–724, Palermo, Italy, July 2012. doi: 10.1109/IMIS.2012.48.

[57] Isam Ishaq, Jeroen Hoebeke, Jen Rossey, Eli De Poorter, Ingrid Moerman, and Piet Demeester. Enabling the web of things: Facilitating deployment, discovery and resource access to iot objects using embedded web services. *International Journal of Web and Grid Services*, 10, 2014. To appear.

[58] Antonio J. Jara, Pedro Martinez-Julia, and Antonio Skarmeta. Light-weight multicast DNS and DNS-SD (lmDNS-SD): IPv6-Based resource and service discovery for the web of things. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 731–738, Palermo, Italy, July 2012. doi: 10.1109/IMIS.2012.200.

[59] Brad Johanson and Armando Fox. Extending tuplespaces for coordination in interactive workspaces. *Journal of Systems and Software*, 69(3):243 – 266, 2004. ISSN 0164-1212. doi: 10.1016/

S0164-1212(03)00054-2. URL http://www.sciencedirect.com/science/article/pii/S0164121203000542. Ubiquitous Computing.

[60] Deepali Khushraj, Ora Lassila, and Tim Finin. sTuples: semantic tuple spaces. In *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004*, pages 268–277, August 2004. doi: 10.1109/MOBIQ.2004.1331733.

[61] Jussi Kiljander, Francesco Morandi, and Juha-Pekka Soininen. Knowledge sharing protocol for smart spaces. *International Journal of Advanced Computer Science and Applications*, 3(9), 2012.

[62] Kjetil Kjernsmo. The necessity of hypermedia RDF and an approach to achieve it. In *Proceedings of the First Linked APIs workshop at the Ninth Extended Semantic Web Conference*, May 2012. URL http://lapis2012.linkedservices.org/papers/1.pdf.

[63] Steve Klabnik. Nobody understands REST or HTTP, July 2011. URL http://blog.steveklabnik.com/posts/2011-07-03-nobody-understands-rest-or-http.

[64] Steve Klabnik. REST is OVER!, February 2012. URL http://blog.steveklabnik.com/posts/2012-02-23-rest-is-over.

[65] Matthias Kovatsch. CoAP for the web of things: From tiny resource-constrained devices to the web browser. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, page 1495–1504, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2215-7. doi: 10.1145/2494091.2497583. URL http://doi.acm.org/10.1145/2494091.2497583.

[66] Reto Krummenacher, Martin Hepp, Axel Polleres, Christoph Bussler, and Dieter Fensel. WWW or what is wrong with web services. In *Proceedings of the Third European Conference on Web Services*, ECOWS '05, page 235, Washington, DC, USA, November 2005. IEEE Computer Society. ISBN

0-7695-2484-2. doi: http://dx.doi.org/10.1109/ECOWS.2005.30. URL http://dx.doi.org/10.1109/ECOWS.2005.30.

[67] Reto Krummenacher, Edward Kilgarriff, Galway Brahmananda Sapkota, and Galway Omair Shafiq. Specification of mediation, discovery and data models for triple space computing, 2006.

[68] Reto Krummenacher, Elena Simperl, and Dieter Fensel. Scalability in semantic computing: Semantic middleware. In *2008 IEEE International Conference on Semantic Computing*, pages 538–545. IEEE, August 2008. ISBN 978-0-7695-3279-0. doi: 10.1109/ICSC.2008.39.

[69] Reto Krummenacher, Daniel Blunder, Elena Simperl, and Michael Fried. An open distributed middleware for the semantic web. *International Conference on Semantic Systems (I-SEMANTICS)*, 2009.

[70] Xabier Laiseca, Eduardo Castillejo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña, and Ester González Aguado. Distributed tracking system for patients with cognitive impairments. In José Bravo, Ramón Hervás, and Vladimir Villarreal, editors, *Ambient Assisted Living*, number 6693 in Lecture Notes in Computer Science, pages 49–56. Springer Berlin Heidelberg, January 2011. ISBN 978-3-642-21302-1, 978-3-642-21303-8. URL http://link.springer.com/chapter/10.1007/978-3-642-21303-8_7.

[71] Andreas Langegger, Wolfram Wöß, and Martin Blöchl. A semantic web middleware for virtual data integration on the web. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications*, number 5021 in Lecture Notes in Computer Science, pages 493–507. Springer Berlin Heidelberg, January 2008. ISBN 978-3-540-68233-2, 978-3-540-68234-9. URL http://link.springer.com/chapter/10.1007/978-3-540-68234-9_37.

[72] Paul Legris, John Ingham, and Pierre Collerette. Why do people use information technology? a critical review of the technology acceptance model. *Information & Management*, 40(3): 191–204, January 2003. ISSN 0378-7206. doi: 10.1016/S0378-7206(01)00143-4. URL http://www.sciencedirect.com/science/article/pii/S0378720601001434.

[73] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications: The TOTA approach. *ACM Trans. Softw. Eng. Methodol.*, 18(4):15:1–15:56, July 2009. ISSN 1049-331X. doi: 10.1145/1538942.1538945. URL http://doi.acm.org/10.1145/1538942.1538945.

[74] Francisco Martín-Recuerda. Towards cspaces: A new perspective for the semantic web. In Max Bramer and Vagan Terziyan, editors, *Industrial Applications of Semantic Web*, volume 188 of *IFIP International Federation for Information Processing*, pages 113–139. Springer Boston, 2005. ISBN 978-0-387-28568-9. URL http://www.springerlink.com/content/a8655013r477v151/abstract/.

[75] Francisco Martín-Recuerda. Application integration using conceptual spaces (CSpaces). In Riichiro Mizoguchi, Zhongzhi Shi, and Fausto Giunchiglia, editors, *The Semantic Web – ASWC 2006*, volume 4185 of *Lecture Notes in Computer Science*, pages 234–248. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-38329-1. URL http://www.springerlink.com/content/7863872281654711/abstract/.

[76] Simon Mayer and Gianin Basler. Semantic metadata to support device interaction in smart environments. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, UbiComp '13 Adjunct, page 1505–1514, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2215-7. doi: 10.1145/2494091.2497584. URL http://doi.acm.org/10.1145/2494091.2497584.

[77] Jon Moore. Hypermedia APIs, November 2010. URL http://vimeo.com/20781278.

[78] Guido Moritz, Elmar Zeeb, Steffen Prüter, Frank Golatowski, Dirk Timmermann, and Regina Stoll. Devices profile for web services and the REST. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, page 584–591, 2010.

[79] Boris Motik, Ian Horrocks, and Su Myeon Kim. Delta-reasoner: A semantic web reasoner for an intelligent mobile platform. In *Proceedings of the 21st International Conference Companion on World Wide Web*, WWW '12 Companion, page 63–72, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1230-1. doi: 10.1145/2187980.2187988. URL http://doi.acm.org/10.1145/2187980.2187988.

[80] Amy Lynn Murphy and Gian Pietro Picco. Transiently shared tuple spaces for sensor networks. In *Proc. of the Euro-American Workshop on Middleware for Sensor Networks*, 2006.

[81] Juan Álvaro Muñoz Naranjo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña, and L. G. Casado. Lightweight user access control in energy-constrained wireless network services. In José Bravo, Diego López-de-Ipiña, and Francisco Moya, editors, *Ubiquitous Computing and Ambient Intelligence*, Lecture Notes in Computer Science, pages 33–41. Springer Berlin Heidelberg, January 2012. ISBN 978-3-642-35376-5, 978-3-642-35377-2. URL http://link.springer.com/chapter/10.1007/978-3-642-35377-2_5.

[82] Juan Álvaro Muñoz Naranjo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña, and Leocadio González Casado. Enabling user access control in energy-constrained wireless smart environments. *Journal of Universal Computer Science*, 19(17):2490–2505, November 2013. URL http://www.jucs.org/jucs_19_17/enabling_user_access_control.

[83] Juan Álvaro Muñoz Naranjo, Aitor Gómez-Goiri, Pablo Orduña, Diego López-de-Ipiña, and Leocadio González Casado. Extending a user access control proposal for wireless network services with hierarchical user credentials. In Álvaro Herrero, Bruno Baruque, Fanny Klett, Ajith Abraham, Václav Snášel, André C. P. L. F. de Carvalho, Pablo García Bringas, Ivan Zelinka, Héctor Quintián, and Emilio Corchado, editors, *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13*, number 239 in Advances in Intelligent Systems and Computing, pages 601–610. Springer International Publishing, January 2014. ISBN 978-3-319-01853-9, 978-3-319-01854-6. URL `http://link.springer.com/chapter/10.1007/978-3-319-01854-6_61`.

[84] Daniele Nardi and Ronald J. Brachman. An introduction to description logics. *The description logic handbook: theory, implementation, and applications*, pages 1–40, 2003.

[85] Elena Nardini, Andrea Omicini, and Mirko Viroli. Semantic tuple centres. *Science of Computer Programming*, 78(5):569–582, May 2013. ISSN 0167-6423. doi: 10.1016/j.scico.2012.10.004. URL `http://www.sciencedirect.com/science/article/pii/S0167642312001876`.

[86] Eric Newcomer. *Understanding Web Services: XML, WQSDL, SOAP and UDDI*. Addison-Wesley Professional, 2002.

[87] Lyndon J. B. Nixon, Elena Simperl, Reto Krummenacher, and Francisco Martin-Recuerda. Tuplespace-based computing for the semantic web: a survey of the state-of-the-art. *The Knowledge Engineering Review*, 23(02): 181–212, 2008.

[88] Mark Nottingham. Web linking, October 2010. URL `http://tools.ietf.org/html/rfc5988`.

[89] Andrea Omicini and Franco Zambonelli. TuCSoN: a coordination model for mobile information agents. In *Proceedings of the 1st Workshop on Innovative Internet Information Systems*, volume 138, 1998.

[90] Benedikt Ostermaier, Fabian Schlup, and Kai Romer. WebPlug: a framework for the web of things. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, page 690–695. IEEE, 2010.

[91] Kevin R. Page, David C. De Roure, and Kirk Martinez. REST and linked data: a match made for domain driven development? In *Proceedings of the Second International Workshop on RESTful Design*, WS-REST '11, page 22–25, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0623-2. doi: 10.1145/1967428.1967435. URL `http://doi.acm.org/10.1145/1967428.1967435`.

[92] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceeding of the 17th international conference on World Wide Web*, page 805–814, 2008.

[93] Dennis Pfisterer, Kay Römer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth, Marcel Karnstedt, Myriam Leggieri, Alexandre Passant, and Ray Richardson. SPITFIRE: toward a semantic web of things. *IEEE Communications Magazine*, 49(11):40–48, November 2011. ISSN 0163-6804. doi: 10.1109/MCOM.2011.6069708.

[94] Gian Pietro Picco, Amy Lynn Murphy, and Gruia-Catalin Roman. LIME: linda meets mobility. In *Proceedings of the 21st international conference on Software engineering*, pages 368–377. ACM Press, 1999. ISBN 1581130740. doi: 10.1145/302405.302659. URL `http://dl.acm.org/citation.cfm?id=302659`.

[95] Antonio Pintus, Davide Carboni, and Andrea Piras. The anatomy of a large scale social web for internet enabled objects. In *Proceedings of the Second International Workshop on Web of Things - WoT '11*, page 1, San Francisco, California, 2011. doi: 10.1145/

1993966.1993975. URL http://dl.acm.org/citation.cfm?id=1993975&CFID=38852141&CFTOKEN=66966366.

[96] Bastian Quilitz and Ulf Leser. Querying distributed RDF data sources with SPARQL. In Sean Bechhofer, Manfred Hauswirth, Jörg Hoffmann, and Manolis Koubarakis, editors, *The Semantic Web: Research and Applications*, number 5021 in Lecture Notes in Computer Science, pages 524–538. Springer Berlin Heidelberg, January 2008. ISBN 978-3-540-68233-2, 978-3-540-68234-9. URL http://link.springer.com/chapter/10.1007/978-3-540-68234-9_39.

[97] Carlos Ramos, Juan Carlos Augusto, and Daniel Shapiro. Ambient intelligence - the next step for artificial intelligence. *IEEE Intelligent Systems*, 23 (2):15–18, 2008. ISSN 1541-1672. doi: http://doi.ieeecomputersociety.org/10.1109/MIS.2008.19.

[98] Leonard Richardson. Introducing real-world REST, November 2008. URL http://qconsf.com/sf2008/dl/qcon-sanfran-2008/slides_/LeonardRichardson.pdf.

[99] Leo Sauermann and Richard Cyganiak. Cool URIs for the semantic web, December 2008. URL http://www.w3.org/TR/cooluris/.

[100] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: a federation layer for distributed query processing on linked open data. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *The Semanic Web: Research and Applications*, number 6644 in Lecture Notes in Computer Science, pages 481–486. Springer Berlin Heidelberg, January 2011. ISBN 978-3-642-21063-1, 978-3-642-21064-8. URL http://link.springer.com/chapter/10.1007/978-3-642-21064-8_39.

[101] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: optimization techniques for federated query processing on linked data. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor,

Abraham Bernstein, Lalana Kagal, Natasha Noy, and Eva Blomqvist, editors, *The Semantic Web – ISWC 2011*, number 7031 in Lecture Notes in Computer Science, pages 601–616. Springer Berlin Heidelberg, January 2011. ISBN 978-3-642-25072-9, 978-3-642-25073-6. URL http://link.springer.com/chapter/10.1007/978-3-642-25073-6_38.

[102] Elena Simperl, Reto Krummenacher, and Lyndon Nixon. A coordination model for triplespace computing. In *Proceedings of the 9th international conference on Coordination models and languages*, COORDINATION'07, page 1–18, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72793-4. URL http://dl.acm.org/citation.cfm?id=1764606.1764608.

[103] Thomas Steiner and Jan Algermissen. Fulfilling the hypermedia constraint via HTTP OPTIONS, the HTTP vocabulary in RDF, and link headers. In *Proceedings of the Second International Workshop on RESTful Design*, WS-REST '11, page 11–14, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0623-2. doi: 10.1145/1967428.1967433. URL http://doi.acm.org/10.1145/1967428.1967433.

[104] Vlad Stirbu. Towards a RESTful plug and play experience in the web of things. In *2008 IEEE International Conference on Semantic Computing*, pages 512–517, 2008. doi: 10.1109/ICSC.2008.51.

[105] Robert Tolksdorf, Elena Paslaru Bontas, and Lyndon J. B. Nixon. A coordination model for the semantic web. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, page 419–423, New York, NY, USA, 2006. ACM. ISBN 1-59593-108-2. doi: 10.1145/1141277.1141375. URL http://doi.acm.org/10.1145/1141277.1141375.

[106] Vlad Trifa, Dominique Guinard, and Simon Mayer. Leveraging the web for a distributed location-aware infrastructure for the real world. In Erik Wilde and Cesare Pautasso, editors, *REST: From Research to Practice*, pages 381–400. Springer New York, January 2011. ISBN 978-1-4419-8302-2, 978-1-4419-8303-9. URL http://link.springer.com/chapter/10.1007/978-1-4419-8303-9_17.

[107] Hans Van der Veer and Anthony Wiles. Achieving technical interoperability - the ETSI approach, October 2006. URL `http://www.etsi.org/website/document/whitepapers/wp3_iop_final.pdf`.

[108] Ruben Verborgh, Thomas Steiner, Davy Van Deursen, Sam Coppens, Joaquim Gabarró Vallés, and Rik Van de Walle. Functional descriptions as the bridge between hypermedia APIs and the semantic web. In *Proceedings of the Third International Workshop on RESTful Design*, WS-REST '12, pages 33–40, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1190-8. doi: 10.1145/2307819.2307828. URL `http://doi.acm.org/10.1145/2307819.2307828`.

[109] Ruben Verborgh, Thomas Steiner, Sofie Van Hoecke, Jos De Roo, Sam Coppens, Erik Mannens, Rik Van de Walle, and Joaquim Gabarró Vallés. The pragmatic proof: A functionality-based approach to Web API composition. *International Journal of Communication Systems*, 2014. ISSN 1074-5351.

[110] Berta Carbadillo Villaverde, Dirk Pesch, Rodolfo De Paz Alberola, Szymon Fedor, and Menouer Boubekeur. Constrained application protocol for low power embedded networks: A survey. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 702–707, 2012. doi: 10.1109/IMIS.2012.93.

[111] Andrew Wahbe. Self-descriptive hypermedia, July 2010. URL `http://linkednotbound.net/2010/07/19/self-descriptive-hypermedia/`.

[112] Joachim W. Walewski, Martin Bauer, Nicola Bui, Pierpaolo Giacomin, Nils Gruschka, Stephan Haller, Edward Ho, Ralf Kernchen, Mario Lischka, Jourik De Loof, Carsten Magerkurth, Stefan Meissner, Sonja Meyer, Andreas Nettstrater, Francisco Oteiza Lacalle, Alexander Salinas Segura, Alexandru Serbanati, Martin Strohbach, and Vicent Toubiana. Project deliverable d1.2 – initial architectural reference model for IoT, June

2011. URL http://www.iot-a.eu/public/public-documents/documents-1/1/1/d1.2/at_download/file.

[113] Mark Weiser. The computer for the 21st century. *Scientific American*, 265 (3):94–104, 1991.

[114] Erik Wilde. The 'profile' link relation type, March 2013. URL http://tools.ietf.org/html/rfc6906.

[115] Erik Wilde and Michael Hausenblas. RESTful SPARQL? you name it! aligning SPARQL with REST and resource orientation. In *Proceedings of the 4th Workshop on Emerging Web Services Technology*, WEWST '09, page 39–43, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-776-9. doi: 10.1145/1645406.1645412. URL http://doi.acm.org/10.1145/1645406.1645412.

[116] Dogan Yazar and Adam Dunkels. Efficient application integration in IP-based sensor networks. In *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, page 4348, 2009.

Secondary Web Resources

[117] Aemet metereological dataset. http://datahub.io/dataset/aemet.

[118] Android mobile platform. http://www.android.com.

[119] Bizkaisense dataset. http://helheim.deusto.es/bizkaisense/.

[120] Bluetooth technology. http://www.bluetooth.com/.

[121] Cito ontology. http://purl.org/spar/cito.

[122] Connectport x2 ip gateway. http://tinyurl.com/connectportx2.

[123] Digi's *XBee sensors*. http://tinyurl.com/xbee-sensors.

[124] Dublin core ontology. http://dublincore.org/.

[125] Foaf ontology. http://www.foaf-project.org/.

[126] Foxg20. http://www.acmesystems.it/FOXG20.

[127] Java se 6 documentation. http://docs.oracle.com/javase/6/docs/.

[128] Jxta protocol. https://jxta.kenai.com.

[129] Kno.e.sis linked sensor data. http://wiki.knoesis.org/index.php/LinkedSensorData.

[130] Microformats.org. http://microformats.org.

[131] Python programming language. http://www.python.org.

[132] Rdf2gb. http://semanticweb.org/wiki/RDF2Go.

[133] Rdflib. https://github.com/RDFLib.

[134] Restlet. http://restlet.org/.

[135] Samsung galaxy tab. http://www.samsung.com/global/microsite/galaxytab/2010/.

[136] Semantic profiles of morelab members. http://www.morelab.deusto.es/joseki/articles.

[137] Sesame framework for processing rdf data. http://www.openrdf.org/.

[138] Simpy process-based discrete-event simulation framework. http://simpy.readthedocs.org.

[139] Spitfire european project. http://spitfire-project.eu.

[140] Swrc ontology. http://ontoware.org/swrc/.

[141] University of luebeck wisebed sensor readings. http://thedatahub.org/dataset/university-of-luebeck-wisebed-sensor-readings.

[142] Upnp forum. http://upnp.org.

[143] Webduino. https://github.com/sirleech/Webduino.

[144] Tim Berners-Lee. N3ql - rdf data query language. http://www.w3.org/DesignIssues/N3QL.html, July 2004.

[145] Tim Berners-Lee and Dan Connolly. Notation3 (n3): A readable rdf syntax. http://www.w3.org/TeamSubmission/n3/, March 2011.

[146] S. Cheshire and M. Krochmal. Dns-based service discovery. http://tools.ietf.org/html/rfc6763, February 2013.

[147] S. Cheshire and M. Krochmal. Multicast dns. http://tools.ietf.org/html/rfc6762, February 2013.

[148] OASIS consortium. Devices profile for web services (dpws). http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01, June 2009.

[149] Jos De Roo. Euler yet another proof engine. http://eulersharp.sourceforge.net.

[150] Roy Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Status code definitions, hypertext transfer protocol – http/1.1. http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html, June 1999.

[151] XMPP Standards Foundation. The extensible messaging and presence protocol (xmpp). http://xmpp.org.

[152] Jan Grant and Dave Beckett. Rdf test cases, n-triples. http://www.w3.org/TR/rdf-testcases/#ntriples, February 2004.

[153] Aitor Gómez-Goiri. Actuation using triple spaces. https://github.com/gomezgoiri/actuation-using-Triple-Spaces.

[154] Aitor Gómez-Goiri. Semantic wot environment simulation. https://github.com/gomezgoiri/Semantic-WoT-Environment-Simulation, https://bitbucket.org/gomezgoiri/semantic-wot-environment-simulation.

[155] Aitor Gómez-Goiri and Pablo Orduña. Otsopack middleware. https://github.com/gomezgoiri/otsopack.

[156] The Internet Engineering Task Force (IETF). Constrained application protocol (coap). https://datatracker.ietf.org/doc/draft-ietf-core-coap/, June 2013.

[157] John Schneider and Takuki Kamiya. Efficient xml interchange (exi) format 1.0. http://www.w3.org/TR/exi/, March 2011.

[158] World Wide Web Consortium (W3C). Owl web ontology language. http://www.w3.org/TR/owl-features/, February 2004.

[159] World Wide Web Consortium (W3C). Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/, February 2004.

[160] World Wide Web Consortium (W3C). Rdf/xml syntax specification. http://www.w3.org/TR/REC-rdf-syntax/, February 2004.

[161] World Wide Web Consortium (W3C). Resource description framework (rdf). http://www.w3.org/RDF/, February 2004.

[162] World Wide Web Consortium (W3C). Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/, January 2008.

[163] World Wide Web Consortium (W3C). Semantic sensor network ontology. http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/, June 2011.

[164] World Wide Web Consortium (W3C). Html microdata. http://www.w3.org/TR/microdata/, October 2012.

[165] World Wide Web Consortium (W3C). Rdfa 1.1 primer. http://www.w3.org/TR/rdfa-primer/, August 2013.

This dissertation was finished writing in Bilbao on Monday 7th April, 2014

This page is intentionally left blank