



UNIVERSIDAD DE DEUSTO

# **A REACTIVE BEHAVIOURAL MODEL FOR CONTEXT-AWARE SEMANTIC DEVICES**

Tesis doctoral presentada por D. JUAN IGNACIO VÁZQUEZ GÓMEZ  
dentro del Programa de Doctorado en CIENCIA DE LA COMPUTACIÓN  
E INTELIGENCIA ARTIFICIAL

Dirigida por el Dr. D. JOSÉ LUIS DEL VAL ROMÁN

El doctorando

El director

Bilbao, Enero de 2007



A mis padres, los verdaderos autores de todo mi trabajo.



# Abstract

Our environment is being populated by an increasing number of digital devices. Even traditional objects are being substituted by their electronic versions, demanding more skills from users.

The amount of intelligence in these devices does not match the pace at which they are being disseminated through our lives. This situation leads to interaction problems with the environment, since the user is the sole responsible for contextualising the information and solving cooperation problems between devices; that is, the user is the intelligence provider. One of the major consequences of this situation is that people are continuously disturbed and required to configure, operate and interact with these devices.

We deem that digital objects shall be more intelligent, autonomous, and able to share, interpret and reason upon exchanged information to release users from part of these activities.

In this thesis, we propose a model for knowledge sharing between devices and for representing their behaviour in a way that promotes autonomous context-aware reactivity. In our approach, we bring Semantic Web technologies to the Ubiquitous Computing world in order to provide the means for collaboratively transforming data into knowledge and enabling intelligent reasoning mechanisms.

The model leads to a decentralised architecture where devices spontaneously discover each other, share context information, perform reasoning and adapt their behaviour dynamically in order to create a feel for Ambient Intelligence in the environment. In order to validate our approach we have identified some evaluation criteria and developed a number of prototypes that were deployed in experimental scenarios.

The results of our research may contribute to pave the way for a new wave of *social* digital objects that we denominate *semantic devices*.



# Resumen

Nuestro entorno está siendo invadido por un número creciente de dispositivos digitales. Esto es así hasta el punto en que objetos tradicionales están siendo sustituidos por sus versiones electrónicas, que requieren más habilidades por parte de los usuarios para ser utilizadas.

La cantidad de inteligencia existente en estos dispositivos no se ajusta al ritmo en el que están siendo desplegados en nuestra vida cotidiana. Esta situación conlleva problemas de interacción con el entorno, ya que el usuario se convierte en el único responsable de contextualizar la información y resolver los problemas de cooperación entre dispositivos: el usuario es la entidad que proporciona la inteligencia. Una de las principales consecuencias de este hecho es que las personas son continuamente molestadas para configurar, manejar e interactuar con estos dispositivos.

Consideramos que los dispositivos digitales deben ser más inteligentes, autónomos y capaces de compartir, interpretar y razonar sobre la información que intercambian, con el propósito de liberar a los usuarios de parte de estas actividades.

Es esta tesis, proponemos un modelo para compartir conocimiento entre dispositivos y representar su conducta de un modo que fomente la reactividad autónoma sensible al contexto. En el enfoque que proponemos, las tecnologías de Web Semántica se integran con el mundo de la Computación Ubicua con el objetivo de facilitar medios que permitan transformar datos en conocimiento de manera colaborativa, y proporcionar mecanismos de razonamiento inteligente.

El modelo conduce al diseño de una arquitectura descentralizada, donde los dispositivos, de manera espontánea, se descubren unos a otros, comparten información de contexto, razonan y adaptan su conducta dinámicamente para implementar la visión de la Inteligencia Ambiental en el entorno. Con el propósito de validar nuestro enfoque, hemos identificado

unos criterios y desarrollado una serie de prototipos que se han desplegado en escenarios experimentales.

Consideramos que los resultados de esta investigación pueden contribuir a establecer el camino de una nueva generación de objetos digitales *sociales* que denominamos *dispositivos semánticos*.

# Acknowledgements

This work owes a lot to different people who have supported me during all these years.

I would like to thank my advisor, Dr. José Luis del Val, who tried all the time to push me further to finish my research, while at the same time trying to stop me from exploring new branches that would have made this process never-ending. I am particular grateful for the opportunity that he and Maria José Gil gave me to spend some months abroad, which undoubtedly helped me to finish the dissertation, as well as to the University of Deusto and the Basque Government that funded parts of this research.

I owe a huge debt of gratitude to Dr. Diego López de Ipiña for his invaluable support from the very initial steps of the research until the final phases reviewing the dissertation. Diego provided me with very fruitful discussions and inspiring moments working together in several research projects. It is easier to carry out a stormy process, as a Ph.D. dissertation that you have to make compatible with other duties during several years, when you can count with the help of talented people such as Diego.

I would like to thank Prof. Hans Gellersen from Lancaster University, who accepted our proposal to host me at their prestigious Embedded Interactive Systems Group from May to July 2006. It was a great and fruitful experience; I would have never finished my dissertation on time without being there.

I am also grateful to different people at the University of Deusto, especially Iñigo Sedano who developed some parts of the initial prototype of the smobject, and my friends and colleagues David Buján, Pablo Garcia, Ana Lago and Asier Perallos; they assumed some tasks that made easier to save time to work on the dissertation. I am also grateful to all my colleagues and research staff at Fundación Deusto.

*Quiero dar las gracias a mis padres, José y Puri, familia y amigos, por todo su apoyo durante estos años, siempre preguntándome cuando iba a acabar la tesis.*

*Finalmente, ya había iniciado el doctorado cuando conocí a Iratxe y ahora estamos casados. Quiero darle las gracias por apoyarme durante todo este tiempo, especialmente por su comprensión cuando me ofrecieron la oportunidad de pasar unos meses fuera trabajando en la tesis, y el no haber podido dedicar a nuestras cosas la atención que merecían durante estos últimos meses. Espero poder recompensarlo con creces.*

*Muchas gracias a todos*

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Problem description . . . . .	4
1.2	Semantic Web in Ubiquitous Computing scenarios . . . . .	6
1.3	Semantic devices . . . . .	10
1.3.1	Semantic discovery protocol . . . . .	10
1.3.2	Semantic devices as social devices . . . . .	12
1.4	Hypothesis and goals . . . . .	13
1.5	Evaluation scenarios . . . . .	14
1.5.1	Generalisation . . . . .	18
1.6	Research methodology . . . . .	18
1.7	Thesis outline . . . . .	20
<b>2</b>	<b>Related Work</b>	<b>23</b>
2.1	Evaluation criteria . . . . .	24
2.2	Universal Plug and Play . . . . .	29
2.2.1	UPnP Architecture . . . . .	30
2.2.2	UPnP Activities . . . . .	32
2.2.3	Conclusion . . . . .	36
2.3	Task Computing . . . . .	38
2.3.1	Task Computing architecture . . . . .	39
2.3.2	Semantic-ization and Service-ization . . . . .	42
2.3.3	Conclusion . . . . .	43
2.4	CoBrA and SOUPA . . . . .	45
2.4.1	CoBrA: Context Broker Architecture . . . . .	47
2.4.2	SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications . . . . .	51
2.4.3	Conclusion . . . . .	53
2.5	Gaia . . . . .	54
2.5.1	Gaia architecture . . . . .	55
2.5.2	Semantic Knowledge in Gaia . . . . .	58

2.5.3	Conclusion	60
2.6	Semantic Spaces, SOCAM and CONON	62
2.6.1	Conclusion	65
2.7	Other related work	66
2.7.1	Triple Spaces	66
2.7.2	The Context Toolkit	68
2.7.3	Oxygen	69
2.7.4	Other relevant activities	70
2.8	Comparative analysis	70
<b>3</b>	<b>mRDP: A Semantic Discovery Protocol</b>	<b>73</b>
3.1	Previous approaches	73
3.2	Introduction to mRDP – Multicast Resource Discovery Protocol	76
3.2.1	Operation	77
3.2.2	Resource identification	81
3.3	Plant: Pattern Language for N-Triples	82
3.3.1	The Plant Query Resolution Algorithm	84
3.3.2	mRDP SPARQL queries	88
3.4	mRDP message format	88
3.5	ReDEL: Resource Description Endpoints Language	91
3.6	Example of advanced uses of semantic queries	93
3.7	Performance evaluation of lexical and semantic discovery with mRDP	94
3.8	Comparative analysis	96
<b>4</b>	<b>A Theoretical Model for Context-Aware Reactivity</b>	<b>99</b>
4.1	Passively influencing the environment	99
4.2	A set-theory based approach for context-aware reactivity	103
4.2.1	Environment-oriented approach	104
4.2.2	Entity-oriented approach	106
4.2.3	Managed constraints	108
4.2.4	The context-awareness process	110
4.2.5	Behavioural profiles	112
4.3	Semantic Web mapping	116
4.3.1	Environment and Entity	116
4.3.2	Context information	117
4.3.3	Knowledge domain	119
4.3.4	Knowledge domain item	121
4.3.5	Knowledge domain item value	122
4.3.6	Perception capability	122
4.3.7	Operation capability	124
4.3.8	Constraint	125
4.3.9	Behavioural profile, precondition and postcondition	126
4.4	Serialisation	127

4.4.1	Capabilities . . . . .	128
4.4.2	Constraints . . . . .	131
4.4.3	Behavioural profiles . . . . .	132
4.5	Summary . . . . .	136
<b>5</b>	<b>SoaM Architecture</b>	<b>139</b>
5.1	Smobject . . . . .	140
5.1.1	Base components . . . . .	142
5.1.2	Base core components . . . . .	144
5.1.3	Platform interfaces . . . . .	145
5.1.4	Built-in platform components . . . . .	151
5.1.5	Awareness components . . . . .	151
5.1.6	The Profiles Resolution Algorithm . . . . .	154
5.1.7	Selective and comprehensive context information collection . . . . .	156
5.1.8	Optimising behavioural profiles resolution . . . . .	159
5.1.9	Intelligence and reasoning at the smobject . . . . .	159
5.1.10	Smobjects as context-aware entities . . . . .	163
5.1.11	An example scenario . . . . .	167
5.1.12	Advanced perceptors and effectors . . . . .	169
5.2	Orchestrator . . . . .	171
5.2.1	Example scenario with orchestrator . . . . .	175
5.2.2	Reasoning at the orchestrator . . . . .	176
5.3	Ontologies and domain rules discovery for Ubiquitous Computing . . . . .	176
5.4	Smobjects-only versus orchestrator-powered scenarios . . . . .	181
5.5	BPinjector . . . . .	183
5.5.1	The smobject as BPinjector . . . . .	185
5.6	Interactions . . . . .	187
5.7	SoaM Discovery . . . . .	191
5.7.1	SoaM discovery with UPnP integration and SSDP extensions . . . . .	192
5.7.2	SoaM discovery with mRDP and SoaMonto . . . . .	195
5.7.3	Comparison . . . . .	198
5.8	SoaM Entity Management API . . . . .	199
5.8.1	HTTP binding operations and messages . . . . .	201
5.9	SoaMonto: the SoaM support ontology . . . . .	207
5.9.1	SoaMonto classes . . . . .	208
5.9.2	SoaMonto properties . . . . .	209
5.9.3	SoaMonto instances . . . . .	210
5.9.4	Examples . . . . .	210
5.10	Comparative analysis . . . . .	212
<b>6</b>	<b>Prototypes and Evaluation</b>	<b>217</b>

6.1	Prototyping . . . . .	219
6.1.1	Smobject base components . . . . .	220
6.1.2	UPnP SSDP extensions . . . . .	223
6.1.3	Orchestrator . . . . .	223
6.1.4	BPinjector . . . . .	224
6.1.5	mRDP client and server . . . . .	225
6.1.6	Smobject awareness components . . . . .	227
6.1.7	Prototyping issues . . . . .	228
6.1.8	Second generation prototypes . . . . .	230
6.1.9	Integration with wireless sensor networks . . . . .	232
6.2	Evaluation . . . . .	232
6.2.1	Performance tests . . . . .	233
6.2.2	Scenarios tests . . . . .	241
6.3	Conclusions . . . . .	263
<b>7</b>	<b>Conclusion</b>	<b>265</b>
7.1	Contributions . . . . .	271
7.1.1	Discussion . . . . .	272
7.1.2	Publications . . . . .	275
7.2	Future research and challenges . . . . .	278
7.2.1	Exploring other forms of semantic devices . . . . .	278
7.2.2	Exploring and extending the SoaM model and architecture . . . . .	279
7.3	Final remarks . . . . .	280
	<b>Bibliography</b>	<b>283</b>
	<b>Appendices</b>	<b>303</b>
<b>A</b>	<b>Basic Semantic Web technologies</b>	<b>305</b>
A.1	Resource Description Framework . . . . .	305
A.2	Ontologies . . . . .	308
A.3	SPARQL Protocol And RDF Query Language . . . . .	309
<b>B</b>	<b>SoaM Numbers, Ports and Namespaces</b>	<b>311</b>
<b>C</b>	<b>ReDEL: Resource Description Endpoints Language</b>	<b>313</b>
C.1	ReDEL XML Schema . . . . .	313
C.2	ReDEL Web Service . . . . .	314
C.3	Simple RDF Web Service . . . . .	315
<b>D</b>	<b>SoaM XML Datatypes and Exchange Messages</b>	<b>317</b>
D.1	SoaM XML Datatypes . . . . .	317
D.2	SoaM XML Exchange Messages . . . . .	321

<b>E</b>	<b>SoaM Entity Management API: SOAP and HTTP bindings</b>	<b>323</b>
<b>F</b>	<b>SoaMonto specification</b>	<b>333</b>
<b>G</b>	<b>Example of smobject configuration file</b>	<b>337</b>



# List of Figures

1.1	The Pervasive Semantic Web. . . . .	9
1.2	Schematic view of the research process. . . . .	19
2.1	Criteria dependency matrix. . . . .	27
2.2	Criteria dependency map. . . . .	28
2.3	UPnP protocol stack. . . . .	30
2.4	UPnP device architecture and services. . . . .	31
2.5	Ordered sequence of UPnP phases. . . . .	32
2.6	Example of a <code>SEARCH</code> message and its response during UPnP Discovery. . . . .	33
2.7	UPnP Description Phase. . . . .	34
2.8	Task Computing architecture. . . . .	40
2.9	Semantic-ization and Service-ization in Task Computing. . . . .	43
2.10	CoBrA Architecture. . . . .	48
2.11	The context acquisition process in CoBrA. . . . .	49
2.12	CoBrA Ontology. . . . .	50
2.13	SOUPA Ontology. . . . .	52
2.14	Gaia architecture. . . . .	56
2.15	Gaia Context Infrastructure. . . . .	58
2.16	SOCAM architecture. . . . .	64
2.17	CONON ontology. . . . .	64
2.18	Triple Space example with publish/subscribe interaction. . . . .	68
3.1	Example of mRDP operation. . . . .	78
3.2	mRDP over the TCP/IP protocol stack. . . . .	80
3.3	mRDP UML sequence diagram with all the interactions. . . . .	81
3.4	Optimised mRDP UML sequence diagram. . . . .	82
3.5	Matchmaking performance during the first execution in mRDP discovery. . . . .	95
3.6	Stabilised matchmaking performance in mRDP discovery. . . . .	96
4.1	Active and passive influence. . . . .	100

4.2	A context-aware entity featuring static behavioural reactivity to context modification. . . . .	102
4.3	A context-aware entity featuring influenceable behavioural reactivity to context modification. . . . .	103
4.4	Constraints as subsets and elements. . . . .	109
4.5	Valid values for a postcondition and one constraint representing a subset of it. . . . .	114
4.6	An infinite number of constraints as subsets of the postcondition. . . . .	115
4.7	RDF graph representing the Listing 4.1. . . . .	120
5.1	Smobject base components communication interfaces. . . . .	143
5.2	Smobject internal structure with base components. . . . .	144
5.3	Example of internal perception and operation process in a smobject. . . . .	147
5.4	Complete smobject communication interfaces. . . . .	152
5.5	Complete Smobject internal structure. . . . .	154
5.6	The context awareness process for smobjects. . . . .	166
5.7	Example scenario with four smobjects. . . . .	169
5.8	Orchestrator communication interfaces. . . . .	172
5.9	Orchestrator internal structure. . . . .	174
5.10	Example scenario with one orchestrator coordinating four smobjects. . . . .	175
5.11	Number of connections for retrieving context information with and without orchestrator. . . . .	182
5.12	BPinjector internal structure. . . . .	183
5.13	Interactions in a smobjects-only scenario. . . . .	188
5.14	Interactions in a orchestrator-powered scenario. . . . .	190
5.15	Transformation of a UPnP device into a SoaM UPnP device. . . . .	193
5.16	Context information provided by a smobject, including SoaMonto data. . . . .	211
6.1	Incremental prototyping and testing process in SoaM. . . . .	218
6.2	Image of the ConnectCore 7U platform that hosted the smobject. . . . .	221
6.3	The mRDP Browser browsing through located smobjects in the network. . . . .	227
6.4	Image of the Gumxtix 400xm with Wi-Fi card and antenna that hosted the second generation of smobjects. . . . .	231
6.5	Performance of the different XML parsers parsing an XML document in the ConnectCore 7U platform. . . . .	234
6.6	Performance of the different XML parsers parsing an RDF/XML document in the ConnectCore 7U platform. . . . .	235
6.7	Performance of the different XML parsers in relation to its size. . . . .	236

6.8	Performance measures of the smobjects-powered CC9U topology. . . . .	237
6.9	Performance measures of the orchestrator-powered CC7U topology. . . . .	238
6.10	Comparison of performance measures for the smobjects-only CC9U and orchestrator-powered CC7U topologies. . . . .	239
6.11	Comparison of relative effort for the activities in the smobjects-only CC9U and orchestrator-powered CC7U topologies. . . . .	240
6.12	3-D view of relationships among relevant factors for smobjects-only networking. . . . .	241
6.13	Two dimensional projection of the relationships among relevant factors for smobjects-only networking. . . . .	242
6.14	3-D view of relationships among relevant factors for a single smobject. . . . .	243
6.15	Two dimensional projection of the relationships among relevant factors for a single smobject. . . . .	244
6.16	The complete smobject prototype with Wi-Fi, audio and battery.	244
6.17	The smobject prototype in the plant protected with a plastic case. . . . .	246
6.18	The smobject prototype in the umbrella. . . . .	254
6.19	The workwear jacket with two wireless accelerometers to detect body orientation, and the smobject prototype (covered by fabric) with the BPinjector software. . . . .	258
6.20	An <code>EffectorTrayAlertter</code> at a monitoring centre that generates an alarm if a worker collapsed in a dangerous environment. . . . .	262
6.21	The souvenir-aware Google Earth display (the RFID reader and a tagged souvenir). . . . .	264
7.1	Graphical comparison of SoaM and other architectures. . . . .	266
A.1	RDF graph representing the Example A.1. . . . .	307
A.2	The Semantic Web stack. . . . .	309



# List of Tables

2.1	Criteria's relative weights of importance. . . . .	29
2.2	Analysis of UPnP against the evaluation criteria. . . . .	38
2.3	Analysis of Task Computing against the evaluation criteria. . .	46
2.4	Analysis of CoBrA/SOUPA against the evaluation criteria. . . .	55
2.5	Analysis of Gaia against the evaluation criteria. . . . .	62
2.6	Analysis of SOCAM against the evaluation criteria. . . . .	66
2.7	Analysis of architectures against the evaluation criteria. . . . .	72
3.1	Comparison of current discovery systems and mRDP. . . . .	98
5.1	Comparison among possible context collection strategies. . . .	158
5.2	Comparison of SoaM SSDP extensions and mRDP as SoaM discovery mechanisms. . . . .	198
5.3	Analysis of SoaM and other technologies against the evaluation criteria. . . . .	215
6.1	Smobject components size. . . . .	228
6.2	Relation among computing power and platform size. . . . .	231
A.1	URIs assigned to identify the resources in Example A.1. . . . .	306



# List of Listings

3.1	Example of triple patterns based on N-Triples. . . . .	83
3.2	An example Plant query. . . . .	85
3.3	Augmented BNF grammar for mRDP messages. . . . .	88
3.4	An example mRDP resource identification message. . . . .	90
3.5	An example mRDP resource identification message with SPARQL. . . . .	90
3.6	An example mRDP resource description location message. . .	91
3.7	An example of HTTP callback conveying ReDEL payload. . . .	92
4.1	RDF/XML representation of context information from the Example 4.1. . . . .	117
4.2	Example of a capability using the <i>any</i> wildcard. . . . .	128
4.3	Fragment of a capability involving a concrete object. . . . .	128
4.4	Example capabilities in SoaM XML Datatypes. . . . .	128
4.5	Fragment of a capability involving all the predicates in an ontology. . . . .	130
4.6	An example of constraints usage. . . . .	131
4.7	A simple behavioural profile. . . . .	133
4.8	A more complex behavioural profile with variables. . . . .	134
4.9	Generated constraints from the behavioural profile of Listing 4.8. . . . .	136
5.1	Behavioural profile for the example scenario. . . . .	167
5.2	ABNF grammar for SoaM SSDP extensions. . . . .	194
5.3	UPnP Control Point request message. . . . .	194
5.4	UPnP Device response messages. . . . .	194
5.5	Example of a IDENTIFY mRDP message during SoaM mRDP Discovery. . . . .	195
5.6	Complex smobject discovery message in mRDP. . . . .	195
5.7	Example of a HTTP POST message during SoaM mRDP Discovery. . . . .	196
5.8	Example of smobject RDF description retrieval. . . . .	197
5.9	Example of constraints retrieval. . . . .	202

5.10	Example of constraints injection with add. . . . .	203
5.11	Example of constraint renewal using HTTP GET. . . . .	205
5.12	Example of constraint removal using HTTP POST. . . . .	206
6.1	Behavioural profile disseminated by the smart plant to adapt the environment. . . . .	247
6.2	Example of native behavioural profile for the smart plant. . .	248
6.3	Excerpt of integrated context information obtained from several sources by the smart plant. . . . .	250
6.4	Domain rules associating sensor nodes measures to their location. . . . .	252
6.5	Example of native behavioural profile for the aware umbrella.	255
6.6	Excerpt of integrated context information obtained from several sources by the aware umbrella. . . . .	256
6.7	Example of a native behavioural profile for an electrical tool. .	259
6.8	Example of a behavioural profile for sending an alarm whenever a worker collapsed in a dangerous environment. . .	260
6.9	Domain rules associating risk levels of task and materials to their location. . . . .	262
A.1	RDF/XML serialisation of Example A.1. . . . .	306

# Introduction

*“The most profound technologies are those that disappear.”*

*Mark Weiser*

*The Computer for the 21st Century*

**U**BIQUITOUS computing is a major field of research nowadays as surrounding environments are becoming increasingly populated with small embedded devices and appliances. The term “Ubiquitous Computing” was coined and popularised by Mark Weiser in his seminal article “The Computer for the 21st Century” [Wei99], but other terms such as pervasive computing, invisible computing, calm computing and *everyware* [Gre06] have been increasingly popular.

Ambient Intelligence (AmI) has become a widely overused term, mostly within Europe, as it has been one of the main ICT research focuses in the 6<sup>th</sup> Framework Program [Eur03a] [Eur03b].

A great deal of research effort during the last years has been devoted to making devices more context-aware, smarter and more reactive. There are several, sometimes overlapping terms to define the spaces created by the population of these objects: intelligent environments, smart environments, smart spaces, AmI Spaces, and so forth.

Weiser summarised the requirements for pervasive technologies in the four principles of Ubiquitous Computing [Wei96]:

- The purpose of a computer is to help you do something else.
- The best computer is a quiet, invisible servant.

- The more you can do by intuition the smarter you are; the computer should extend your unconscious.
- Technology should create calm.

Clearly, these principles are not being honoured in most of current computing systems which demand our attention continuously<sup>1</sup>.

In turn, Aarts defines Ambient Intelligence as the synergy of three attributes [Aar02]:

- Ubiquity: surrounded by a multitude of interconnected embedded systems.
- Transparency: the equipment is invisible and integrated into the background of the user's surroundings.
- Intelligence: the system is able to recognize the people that live in it, adapt itself to them, learn from their behaviour, and even show emotion.

Among the novelties introduced by Aarts is the extension to the Ubiquitous Computing concept towards a more humanistic perspective, promoting devices that “show emotions” such as the iCat [vBYM05].

The lemma of the project Aura [GSSS02], developed by Carnegie Mellon University, reads as “*towards distraction-free Ubiquitous Computing*”. It is remarkable the lucid interpretation the authors made about user-related issues:

*The most precious resource in a computer system is no longer its processor, memory, disk or network. Rather, it is a resource not subject to Moore's law: User Attention. Today's systems distract a user in many explicit and implicit ways, thereby reducing his effectiveness.*

The point is that there is a remarkable amount of interactions, both with the environment and with the objects contained within, people have to carry out everyday in order to achieve their goals.

The Ubiquitous Computing and Ambient Intelligence visions pursue to create reactive environments, populated by smart devices, that react intelligently and autonomously to user activities.

---

<sup>1</sup>Weiser predicted in 1996 that these challenges should be resolved by 2006.

In order to illustrate how these systems are expected to perform, we reproduce excerpts from probably the first scenario about Ubiquitous Computing, found in Weiser's original article "The Computer for the 21st Century" [Wei99]:

**Example 1.1.** At breakfast Sal reads the news. She still prefers the paper form, as do most people. She spots an interesting quote from a columnist in the business section. She wipes her pen over the newspaper's name, date, section and page number and then circles the quote. The pen sends a message to the paper, which transmits the quote to her office. [...] She had lost the instruction manual and asked them for help. They have sent her a new manual and also something unexpected – a way to find the old one. According to the note, she can press a code into the opener and the missing manual will find itself. In the garage, she tracks a beeping noise to where the oil-stained manual had fallen behind some boxes. [...] Once Sal arrives at work, the fore view helps her find a parking spot quickly. As she walks into the building, the machines in her office prepare to log her in but do not complete the sequence until she actually enters her office. [...]

And yet another example. The following are excerpts of one scenario depicted in "Scenarios for Ambient Intelligence in 2010" [Eur01] by the IST Advisory Group.

**Example 1.2.** After a tiring long haul flight Maria passes through the arrivals hall of an airport in a Far Eastern country. [...] Her computing system for this trip is reduced to one highly personalised communications device, her "P-Com" that she wears on her wrist. [...] Her visa for the trip was self-arranged and she is able to stroll through immigration without stopping because her P-Comm is dealing with the ID checks as she walks. [...] A rented car has been reserved for her and is waiting in an earmarked bay. The car opens as she approaches. It starts at the press of a button: she doesn't need a key. [...] Her room adopts her "personality" as she enters. The room temperature, default lighting and a range of video and music choices are displayed on the video wall. She needs to make some changes to her presentation – a sales pitch that will be used as the basis for a negotiation later in the day. Using voice commands she adjusts the light levels and commands a bath. [...]

In the long road to make these visions true, our environments are being invaded with digital objects, which embed more computing power than preceding generations. Everyday examples are automatic doors that open on the presence of people, taps that provide water as hands are placed underneath, lights that turn on autonomously when someone gets closer, digital cameras that contact nearby printers to generate photo printouts or mobile phones that perceive surrounding partners to play a game.

But often, using these devices becomes cumbersome and troublesome, as they require advanced knowledge to operate, thus disturbing user's activities.

As we mentioned before, people need to be released from the burden of interacting with the environment all the time and concentrate on their goal, not on eliminating the barriers. Surrounding objects should be able to perceive users' goals and existing barriers, performing the required operations to facilitate human activities. This is the final goal of Ambient Intelligence.

## 1.1 Problem description

After analysing the scenarios depicted in [Wei99] [Eur01] [Eur03a] and [Eur03b], we found several issues that drew our attention:

1. The scenarios described in those visionary papers are far from being reached in real life nowadays: existing devices and environments are not very intelligent. More research efforts must be targeted at porting AI techniques into Ubiquitous Computing.
2. Current experimental proposals do not feature a spontaneous collaboration model among devices. They are generally based on a unique central server and generally need heavy manual configuration during deployment. We claim that spontaneous discovery and interoperation [KF02] must be promoted.
3. In fact, it is difficult to find the proper balance between decentralisation and intelligence, as it seems there exists underlying opposing forces between both. An additional challenge is to integrate all these desired features into a small and efficient computing embodiment.
4. These proposals also tend to integrate different unrelated technologies to solve a problem, making the resulting system highly coupled, failure-prone and too large for embedding into small devices.

5. Our environments are getting more and more populated with electronic / automatic devices, sometimes featuring advanced complex interfaces to operate them. Since they are not context-aware they require manual operation, thus troubling users and unnecessarily taking up their time. Future environments and objects should be more autonomous, more perceptive, intelligent and reactive.

We can only expect a high degree of intelligence in the environment if such environment is populated by intelligent devices. In this type of scenario, devices are able to gather existing information, share it with others, analyse and reason upon the data, and determine the best reactive behaviour to perform.

Of course, such environment should not only be composed of individual autonomous devices acting by themselves, but a certain level of emerging collaboration is required to perform cooperative tasks in consistent ways. Some objects may feature specific types of perception capabilities while others feature complementary ones; some can perform concrete operations on the environment or on their own state, while others work and act over a different set of aspects.

Specialised appliances and heterogeneous objects are the common rule (heating systems, light bulbs, temperature control systems, iris identification mechanisms, camera-based surveillance systems), but although concrete features are different, coordination and collaboration should be achieved among them in order to truly realise the concept of Ambient Intelligence.

While individual behaviours are still required, rich collaboration among devices boosts the environment to a further degree of intelligence, greater than the addition of single capabilities. But collaboration requires some kind of common communication mechanism among entities.

This aspect is of crucial importance, since the expressiveness provided by the communication language may determine the *social* capabilities of devices, and thus, their ability to create a common knowledge space.

Therefore, communication and intelligence are considered two attributes at the core of the Ubiquitous Computing / Ambient Intelligence vision. Communication contributes to information sharing and coordination of activities, intelligence contributes to analysing, reasoning and decision taking, and both together contribute to device inter-collaboration for the user's sake.

It is of foremost importance for the creation of these environments to identify a technology in which communication and intelligence capabilities can coexist seamlessly.

## 1.2 Semantic Web in Ubiquitous Computing scenarios

We consider the web model to be the most suitable technology to provide both communication and intelligence capabilities to Ubiquitous Computing environments.

Regarding the communication area, it is clear that the web model based on HTTP [FGM<sup>+</sup>99] as a transport protocol, and IRIs [DS05] or URIs (including URL and URN) [BLFM98] for resource identification and location, has proved to be one of the most successful technologies of the last decade.

The application of the web communication model in the Ubiquitous Computing field has been explored thoroughly in the past (see section 2.7) and it is widely applied today in the UPnP (Universal Plug and Play) [UPn03] protocol stack (see section 2.2).

However, how the web model can provide intelligence to pervasive computing environments is not so obvious.

During the last years the whole web model is undergoing an evolution towards a new paradigm called “the Semantic Web” [BL99]. The basics of the Semantic Web were outlined in a Scientific American article by Berners-Lee et al. [BHL01]:

- To date, the World Wide Web has developed most rapidly as a medium of documents for people rather than of information that can be manipulated automatically. By augmenting Web pages with data targeted at computers and by adding documents solely for computers, we will transform the Web into the Semantic Web.
- Computers will find the meaning of semantic data by following hyperlinks to definitions of key terms and rules for reasoning about them logically. The resulting infrastructure will spur the development of automated Web services such as highly functional agents.
- Ordinary users will compose Semantic Web pages and add new definitions and rules using off-the-shelf software that will assist with semantic markup.

Basically the Semantic Web is a web of knowledge, where concepts and information are represented in a machine readable and understandable form and linked via URIs. Every concept (people, places, objects, time events, verbs, and so forth) can be identified via a unique URI, in such a way that a universe of concepts can be related to each other.

Appendix A provides a basic background on Semantic Web technologies.

As we mentioned earlier, the web model and Semantic Web technologies feature a series of characteristics that seem to fulfil the two major attributes of Ubiquitous Computing (communication and intelligence) identified at the end of section 1.1.

The joint application of the web model and the Semantic Web in pervasive computing scenarios results in a coherent architectural model, since core technologies such as URI or namespaces constitute their technological basis.

The web communication model, based on a network of resources linked via URIs and the HTTP communication protocol [FGM<sup>+</sup>99], has been widely employed in the past, even in pervasive computing scenarios [KB01] [IST<sup>+</sup>05], and its validity has been unquestionably proven.

Moreover, existing HTTP complementary mechanisms such as cookies [Lau98], Basic or Digest Authentication [FHBH<sup>+</sup>99], or HTTPS can be also reused in the pervasive computing arena to achieve session information persistence, authentication or secure communication channels, respectively. In this way, Ubiquitous Computing architectures can take advantage of existing HTTP-related technologies to fulfil a great amount of communication requirements.

The only issue not covered by HTTP is distributed discovery of devices or services. UPnP [UPn03] proposes SSDP, an HTTP-based alternative that is analysed in section 2.2. mDNS [CK06b], DNS-SD [CK06a]<sup>2</sup> and Bonjour (a variant of DNS-SD) [App05] are other candidate technologies (see section 3). But neither of them embraces the potential of semantic mark-up.

On the other hand, Semantic Web technologies are a suitable candidate both for context representation and reasoning. Domain specific RDF [Wor04f] or OWL [Wor04d] vocabularies can help defining the terms used in particular situations in order to represent the existing knowledge in a concrete moment of time.

For example, to represent the context information captured by a light sensor, a hypothetical vocabulary with verbs such as `hasLuminance` or `hasColor` could be used.

```
<rdf:Description rdf:about="urn:uuid:light1">
  <lit:hasLuminance
    rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
    30
```

---

<sup>2</sup>Both in draft as of December 2006.

```
</lit:hasLuminance>
<lit:hasColor
  rdf:resource="http://www.awareit.com/onto/2005/12/color#Yellow
  "/>
<rdf:type
  rdf:resource="http://www.awareit.com/onto/2005/12/light#Light"/>
</rdf:Description>
```

Not only an endless number of vocabularies can be created for representing the context information about multiple knowledge domains, but reuse of existing vocabularies must be promoted in order to share common concepts among applications and objects.

Devices with built-in sensors can retrieve raw data from the environment and annotate these data as illustrated above, applying concrete vocabularies they have been explicitly configured for when manufactured.

We deem feasible to build small “annotation processes” even in limited devices in order to characterise captured raw context data in this way and create a more expressive level of knowledge that can be shared and analysed with other devices or entities.

Moreover, the Semantic Web does not only provide a mechanism for context information representation but also for reasoning. OWL is an example of a Semantic Web technology that can be used to represent description logics formalisms in such a way that new information (new context information) can be automatically generated from existing one by applying OWL intrinsic reasoning mechanisms.

Emerging Semantic Web rules technologies such as SWRL (Semantic Web Rules Language) [HPSB<sup>+</sup>04] can be also applied to generate new context information based on Horn-like clauses instead of description logics.

Using SWRL it is possible to describe a rule such as “*if the color of `light1` is yellow and the ambient sound has a low volume then the environment is suitable for reading*”, which generates new information about the suitability of the environment for certain task based on previously existing knowledge. Thus, a concrete device can behave differently depending on whether the environment is suitable for reading (for example, an electronic ink book), taking advantage of the results obtained by the reasoning process.

Semantic Web is particularly interesting for Ubiquitous Computing because of its *future-proof* characteristics: it can provide a solution framework for “*problems and situations yet to be defined*” [Las06].

Semantic Web -enabled artifacts would be able to interpret and process concepts and relationships not defined at the moment they were designed;

they would be able to cope with new situations and exchange currently undefined information structures with other entities, featuring a higher degree of interoperability than previous technologies.

Our conclusion is that the application of the Web model as communication infrastructure and Semantic Web technologies for context representation and reasoning seems to provide a consistent framework for creating context-aware devices and environments. We coined the term *Pervasive Semantic Web* for designating this ubiquitous communication model [VLnS06].

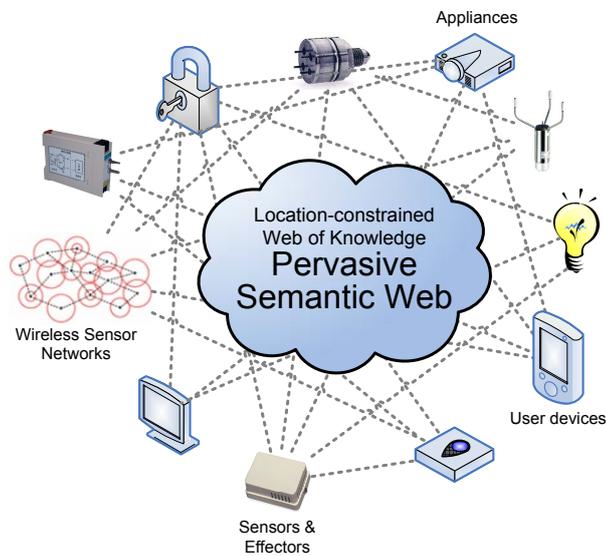


Figure 1.1: The Pervasive Semantic Web.

The vision of the Pervasive Semantic Web pursues to create a space of knowledge, where devices are interconnected, hosting information about environmental perceived conditions and using URIs to link resources inside and outside this space.

This vision determines the creation of a new type of logical environment in Ubiquitous Computing scenarios: a location-constrained Semantic Web with information flows back and forth among communicating devices, sharing their knowledge about the environment and coordinating their tasks via distributed reasoning procedures in order to provide an ambient intelligence experience (see Figure 1.1).

The evolution in the ratio of computing power per square centimeter, as well as improvements in battery-saving technologies support the feasibility of creating intelligent devices at low cost in the near future.

## 1.3 Semantic devices

Lassila and Adler [LA03] introduced the concept of *semantic gadget* to describe devices capable of performing “*discovery and utilization of services without human guidance or intervention, thus enabling formation of device coalitions*”. Again, the approach of taking humans out of the loop and smart objects working in the periphery of our attention appears here.

Although semantic discovery and service composition are identified as two major focuses of the semantic gadget concept, no advances have been achieved so far in creating such a kind of intelligent and cooperative objects.

However, some of the ideas presented in [LA03] contributed to clarify our vision for semantic-powered objects, and our definition for semantic device:

*A semantic device is a system that is spontaneously aware of surrounding context information, capable of reasoning and interpreting this information at a semantic level, and finally able to develop a reactive behaviour accordingly.*

*A semantic device should be able to spontaneously discover, exchange and share context information with other fellow semantic devices as well as augmenting this context information via reasoning in order to better understand the situation and perform the appropriate reactive response.*

Under this point of view, a fundamental issue concerning semantic devices is semantic discovery.

### 1.3.1 Semantic discovery protocol

Concerning discovery protocols we regard semantic technologies as the means to provide expressiveness for rich descriptions and search queries, not only for devices and services in the network, but at a much broader extent for any existing resource.

If every device in the network could create an RDF graph of the information it deals with, including its device type, ID, manufacturer, owner, device status, stored documents and any other available data, this RDF graph could be queried by other objects to perceive its overall state.

Moreover, reasoning could be applied to augment device-related information via inference, thus increasing the intelligence during the discovery process.

For instance, some examples of queries that cannot be accomplished by traditional Ubiquitous Computing discovery protocols, but could be carried out by a semantic-powered one could be:

- “Find all the devices that store a popular document”
- “Find all the devices that store an image authored by a friend of A”
- “Find all the services in the network managed by a computer located near O”
- “Find the location of all the users of mobile phones”
- “Find all the devices whose user is a woman”

Some of these queries are much more powerful than required for usual Ubiquitous Computing applications, but they illustrate the almost unlimited expressive capabilities of semantic querying. Queries are not limited to devices and services, but any kind of resource can be searched for<sup>3</sup>: devices, users, documents, emails, radio stations, towns, colours, and so forth, anything that can be defined using RDF/OWL or other semantic mechanism.

But the real power of semantics is achieved by using ontologies and reasoning: queries could be resolved to produce results that were not explicitly stated. For instance, let us consider the following scenario:

- Fact 1: The mobile phone `mphone1` has a built-in microphone
- Fact 2: `mphone1` is located in `room21`
- Fact 3: `room21` is located in `buildingJ`
- Ontology declaration: Location is a transitive relation
- Domain rule: Every device with a built-in microphone is an input device
- A client issues the query “find all the input devices in `buildingJ`”

If reasoning is performed over the data, before actually executing the query, two new facts would be generated:

---

<sup>3</sup>There is a collateral effect about this assumption: since RDF predicates are resources themselves, such a discovery protocol could be even used to find “predicates” or “properties”, such as in the query “find all the properties relating the user U with the device D”.

- Fact 4 (from facts 2, 3 and the ontology declaration): The mobile phone `mphone1` is located in `buildingJ`
- Fact 5 (from fact 1 and the domain rule): The mobile phone `mphone1` is an input device

Therefore, the previous query would obtain `mphone1` as a valid result, which could not be accomplished without generating facts 4 and 5 through reasoning and, of course, would have never been obtained via any of the traditional discovery protocols.

Semantic discovery for Ubiquitous Computing seems very promising and there are not any references in this field, since most of the work about semantic matchmaking is oriented to enterprise information systems (“*although RDF has been proposed as the service description format for interoperability between service discovery systems [Rey01], so far there is no service description standard yet*” [ZS05]).

### 1.3.2 Semantic devices as social devices

We consider semantic devices to be highly *social*: they are natively collaborative in the sense that they share all the information they can.

The Web has transitioned from a basically “one publisher – many readers” model to a more collaborative “many publishers – many readers” model, in an approach that was called Web 2.0 [O’R05]. The major representatives of this culture are weblogs, social bookmarking, wikis, RSS feeds and so forth.

We consider that this model can be also applied to semantic devices, featuring a collaborative nature, sharing information, and creating a community of intelligent objects in the environment in order to better serve their users.

Semantic devices behave in a social way because a higher and more useful knowledge can be obtained from the generous contributions of individual entities, rather than from selfishly not sharing information (of course, taking into account existing privacy concerns).

Semantic devices must be *communicative* in order to cooperate for the purpose of helping people in their everyday activities.

## 1.4 Hypothesis and goals

Based on the definition of semantic device, we enunciate the hypothesis of our research as:

*To prove that devices based on semantic technologies provide the level of context-awareness, intelligence and adaptability required in smart environments.*

In order to validate the hypothesis a complete model and communication architecture for this kind of devices must be designed. Therefore, the general goal of our research is:

*To design an architectural model for the collaboration of semantic devices.*

Three specific goals originate from the general goal:

**SG1.** *To design a semantic discovery mechanism for environments populated by semantic devices.*

**SG2.** *To define a theoretical model for representing context-aware reactivity involving semantic devices.*

**SG3.** *To design a decentralised communication architecture in which semantic devices can spontaneously share context information and feature an adaptive behaviour, while allowing the implementation in resource constrained platforms.*

The first specific goal addresses the need for an intelligent discovery mechanism as already mentioned. The second specific goal emerges from the convenience of creating some theoretical basis to better understand the context-awareness process in semantic devices. This basis will serve as a guide for designing the architecture as covered by the third specific goal.

Since the synergistic integration of the web architecture and the Semantic Web provide both communication and intelligence capabilities, we will focus on the use of these technologies as the main constituent for semantic devices <sup>4</sup>.

We will accomplish the specific goals, and thus the general goal, by means of a number of operational goals:

---

<sup>4</sup>This approach is also coherent with the vision of semantic devices as ubiquitous representatives of the Web 2.0 concept.

- OG1. To establish a set of criteria for analysing Ubiquitous Computing architectures that embrace Semantic Web technologies.*
- OG2. To design and implement a discovery protocol based on the web architecture and the Semantic Web.*
- OG3. To identify the constituent concepts of the context-awareness process and their relationships.*
- OG4. To design mappings between context-awareness theory and concrete web-based technologies.*
- OG5. To identify the constituent entities of the architecture, their activities and their relationships.*
- OG6. To design and implement the software modules representing the entities in the architecture, in such a way that they are light enough for deployment in embedded platforms.*
- OG7. To design and implement the communication protocols and languages for the architecture.*
- OG8. To build, deploy and evaluate several prototypes in experimental scenarios recreating the Ambient Intelligence vision.*

These operational goals depict the concrete activities we will carry out during the research process.

In order to validate the hypothesis we describe a number of scenarios, with similar characteristics to those that appear in the Ubiquitous Computing and Ambient Intelligence literature. These scenarios represent real-life situations our model should be able to cope with.

## **1.5 Evaluation scenarios**

All of these scenarios share a common distinguishing mark: they are populated by semantic devices collaborating spontaneously. They share their knowledge with others, take decisions based on existing information, and are autonomous and reactive.

### **Scenario 1. Autonomous objects: plants that influence their environment**

It is sometimes difficult to remember when plants need watering, more light or some fresh air. As a result their lifespan can be severely reduced

or they can die. A semantic device could be installed on the plant flowerpot, periodically checking plant's life signs and alerting the user via a buzzer or a flashing light, when some action must be accomplished.

Even more, the plant could manage to take care of itself and influence their environment to adjust the conditions depending on the plant requirements. For example, if the plant needs some more light, it could instruct the room curtains to move apart; or if some fresh air is needed it could instruct the room window to open for some minutes; powered by a small motor, these plants could autonomously move around to find a better place in the room, away from undesirable conditions.

Automatically watering the plant as needed can be difficult except in special premises or greenhouses, but plants could balance the lack of water, maybe with a reduction of temperature or adjusting the air humidity. A small self-regulated water tank attached to the flowerpot could also accomplish this task.

Of course, users' preferences about environmental conditions have priority over those from plants, but they do not always happen to be disjoint.

These plants would become active elements in our home, "users" themselves of the environment, worth not only for decorative purposes but also because of their initiative and self-protective behaviour, requiring less effort from owners to bloom and become healthier.

## **Scenario 2. Augmented objects: an umbrella that knows about the weather forecast**

Traditional passive objects can be activated in new and amazing ways. We generally need to be aware of the weather when leaving home for some hours, and remember to take an umbrella if necessary.

There are several ways of empowering the umbrella with context awareness capabilities to assist the user in these situations. At an initial level, this "aware umbrella" could discover rain sensors in the vicinity and retrieve information from them periodically. It could also discover the home access control system, the main entrance, and display a visual cue or a sound whenever the user is leaving home without the umbrella while it's raining.

At a second, more challenging level, the umbrella could retrieve the weather forecast from the Internet (or from any surrounding device acting as a weather information provider) and issue an alert if chances of raining are probable during the next hours.

This kind of device would feature both local and global communication capabilities to poll surrounding devices (rain sensors, main door) and

remote information sources (weather forecast site). Its behaviour might be fully configurable: which conditions trigger the warning (user leaving, rain sensors, others), as well as the desired reactivity (visual clue, acoustic warning), depending on built-in actuators.

While the first level (local communication) is moderately useful, since users would notice immediately the rain after leaving home, the second level (global communication with weather forecast providers) provides a remarkable added value, making the umbrella to seem more perceptive than humans are.

### **Scenario 3. Protective working environments: enforcing user care in adaptive spaces**

Health and user care is also an important concern, even more in the case of impaired users, when reactive environment behaviour can provide better living conditions.

If environments were populated with different semantic sensors providing information about temperature or humidity levels, for instance, the user could be notified if current conditions were harmful, according to his profile.

An intelligent room or workplace could be configured with a particular behaviour to address these situations and modify the environmental conditions depending on the users' preferences or current activities. Alerts could be notified to very sensitive users whose health requirements recommend to abandon the place immediately.

This kind of scenario acquires a particular importance in the case of workplaces where workers spend intensively 8 hours a day, 5 days a week. Certifying safe conditions at work in terms of parameters as those referred earlier, as well as others such as background noise, air conditioning operation or non disturbing elements, are of utmost importance.

### **Scenario 4. Collective awareness among home devices**

Home environments are more heterogeneous than office environments, with different appliances and electronic devices. Could users benefit from a common exchange of information among them?

Digital TV offers a plethora of new possibilities for interaction. But the digital TV can also provide users with additional information and reactive behaviour in their home environment.

For example, if the user leaves the room, the TV could be aware of it, as the information is provided by the indoor location system, and turn itself off completely, or just turn off the screen.

While several users are watching a concrete TV show, the device, aware of their preferences, could display suggestions unintrusively in a small area about other shows that best match collective predilections. This feature implies that users should share their profiles with the TV set – maybe through a PDA or a wearable computer – if they actually want to influence the suggestions.

The same mechanism could be used to configure the mapping of remote control buttons to concrete TV channels in places such as hotel or hospital rooms, so that the user is not required to learn different mappings, but the device is aware of his / her preferred remote control configuration.

The TV set could also act as notifier of several circumstances of interest the user is not aware of while engaged in watching TV, such as oven timeout while cooking or phone ringing at the other end of the house.

The TV interaction while the user is making a phone call is also interesting. The TV set could reduce the volume to facilitate conversation, restoring the original level after the call is over. The phone could provide state information, so that surrounding devices can adapt their behaviour accordingly, as it is the case.

### **Scenario 5. Driver-aware vehicles**

Both rental cars and shared cars by several members of a family, pose certain problems every time there is a driver switch: driving settings need to be readjusted and some hazardous situations may happen if this is not carried out properly. This situation happens typically due to forgetting to customise the settings (e.g. the rear mirror), or doing it while driving.

Used to drive one's car, rental car driving is even more difficult, specially till the driver gets used to the brake and accelerator pressures, the steering wheel calibration, and so forth.

A driver-aware car, with the users providing their driving profiles, could be able to adjust all these settings automatically, those related to the driving position as well as those related to controls' calibration and pressure. Although the user would still be able to readjust some of them manually, this would reduce the "start-up" time of driving as well as enforce security aspects.

### 1.5.1 Generalisation

All these problems have to do with context-aware devices collecting information from their built-in sensors, sharing this information with others, reasoning over their collective knowledge and trying to react accordingly to honour some behaviour. It is also about actors, devices or users, who influence surrounding objects to change their behaviour and achieve environmental personalisation.

The problem is not designing and implementing these individual devices separately and specifically for the described purposes. The scientific challenge is designing a universal context-awareness model where information can be discovered and shared among all participating entities, whatever the nature of this information.

A model in which devices' behaviour can be dynamically modified depending on existing demands and conditions, thus adapting their context-aware reactivity and taking advantage of rich information flows to interpret new situations. That is, a self-describing semantic information model.

Our challenge is to create an open model where the aware umbrella can use the TV to inform the user about raining chances, or the smart plants can make a phone call asking for being taken to the terrace if raining.

Although the above scenarios illustrate specific situations, our purpose is not to design a model for developing a "phone call"-aware TV but a context-aware TV that can be "profiled" for being "phone call"-aware; not just a rain-aware umbrella, but a context-aware umbrella whose behaviour can be configured as desired; not just self-protective plants, but plants that can collect and provide information to surrounding devices, reacting appropriately, maybe to create a balanced living ambient.

We pursue to create a model for designing fully versatile and flexible context-aware devices, able to discover, share and process semantic information from available sources and develop any kind of required reactive behaviour.

## 1.6 Research methodology

Similar research goals may be faced in completely different ways depending on the research context: availability of infrastructures, accessibility and proximity of experts, synergies with ongoing research projects, and so forth.

Based on our context, we designed a research strategy based on the following activities:

1. Update our knowledge by reviewing recent and state-of-the-art publications, and attending congresses.
2. Design and development of the different parts of the model and architecture, augmenting the scope gradually in an iterative process.
3. Experimentation and evaluation of the incremental prototypes.
4. Attending congresses and workshops to present partial results and to check existing state-of-the-art progresses.
5. Networking with experts in congresses, meetings, via email, and visiting other research centres<sup>5</sup>.
6. Redesign with the feedback obtained from all the above means.
7. Development and deployment of final prototypes for embedded platforms in real world -like scenarios to gather results. Integration with ongoing projects.
8. Dissemination of the obtained knowledge and experiences to the research community.

Figure 1.2 illustrates graphically this research process, with the major activities as well as the inputs and outputs that contribute to the final results.

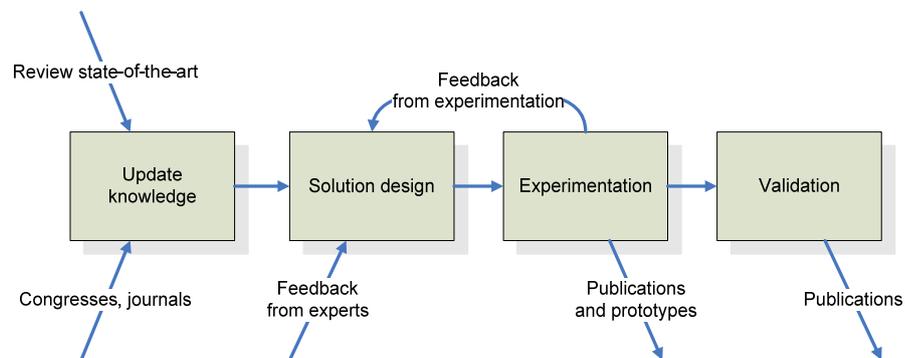


Figure 1.2: Schematic view of the research process.

<sup>5</sup>The author was a visiting Ph.D. student during three months at the Embedded Interactive Systems Group of Lancaster University, where similar initiatives, such as the “Cooperative artifacts” project, were under development.

Underlying this research process is the action-research methodology composed of five different phases:

- Diagnosing: identifying the problem.
- Action planning: considering the possible courses of action.
- Taking action: selecting the course of action.
- Evaluating: analysing the consequences of the action.
- Specifying learning: identifying general findings.

These phases will be applied throughout all the mentioned activities in our research with the aim of providing rigor, reflexive critique and continuous challenges.

## 1.7 Thesis outline

The thesis is structured in seven chapters and a number of appendices.

Chapter 1 (this chapter) outlines the motivation, problem description, the hypothesis and goals of our research. It also presents the concept of *semantic device*, a number of evaluation scenarios and an overview of the research methodology we applied.

In the analysis of related work, chapter 2, we identify the evaluation criteria to apply and describe different initiatives linked to our research and, more importantly, we analyse them under the light of the evaluation criteria. A final comparative table is provided at the end of the chapter.

Chapter 3 is devoted to mRDP, the semantic discovery protocol designed for the model. Different parts of the protocol are described and compared to other discovery protocols at the end of the chapter.

A theoretical model defining the major concepts underlying our context-aware reactivity mechanism is introduced in chapter 4, along with Semantic Web mappings and XML serialisation mechanisms for these concepts.

Chapter 5 contains the description of the architectural elements, along with their behaviour and interactions. The SoaM (Smart Objects Awareness and Adaptation Model) architecture honours the theoretical model described in the previous chapter, providing a concrete materialisation of the concepts. The SoaM Entity Management API over HTTP, and SoaMonto, the ontology for the model, are thoroughly described along with possible implications. A final section updates the table provided at the end of chapter 2, now comparing the SoaM architecture with the previous initiatives.

Chapter 6 provides a description about the hardware and software prototypes developed in order to evaluate SoaM and the results themselves. The evaluation is performed from two complementary perspectives: testing the performance of different entities in the architecture and deploying some of the evaluation scenarios described in chapter 1.

Finally, chapter 7 provides the conclusions of our research. The hypothesis and goals are revisited, and the major contributions, along with some discussion about several issues, are provided. Open research lines and challenges are also included for future reference, and the chapter ends with some final remarks.

The thesis also includes some appendices to contextualise and obtain more in-depth knowledge about several aspects of the research, such as Semantic Web technologies or formal specifications developed during the design phase.



## Related Work

*“Research is what I’m doing when I don’t know what I’m doing.”*

*Wernher Von Braun*

*NASA Center Director 1960–1970*

**S**EVERAL initiatives have experimented with the integration of Semantic Web technologies into Ubiquitous Computing architectures in recent years. This chapter is devoted to the analysis of these experiences, identifying their constituent parts, the extent to which they have applied semantic technologies, their contributions, benefits and drawbacks.

In order to evaluate their suitability for our goals, we identified a number of evaluation criteria representing different complementary aspects to examine, and ranked the architectures according to these criteria. In this way, it will be also possible to rank our proposal and compare it to previous work in the field.

Analysis of the following architectures are included:

- UPnP as a web-based infrastructure architecture
- CoBrA (Context Broker Architecture) and SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications)
- Task Computing
- Gaia
- Semantic Space, SOCAM (Service Oriented Context-Aware Middleware) and CONON (CONtext ONtology)

- Other related work
  - Triple Spaces
  - The Context Toolkit
  - Oxygen
  - Other relevant activities

The list starts with UPnP (Universal Plug and Play). Even though not directly using Semantic Web technologies, UPnP is an example of Ubiquitous Computing architecture based on web standards that has been the inspiration and communication infrastructure of other technologies (including some of those analysed below).

Afterwards, four initiatives (CoBrA, Task Computing, Gaia and SOCAM) are thoroughly examined and evaluated: these are the major experiences in which we focused our investigation at this stage. We applied the evaluation criteria identified at the beginning of the process to generate a profile of these systems.

Finally, other related work, a comparative table ranking UPnP and the four major initiatives using the evaluation criteria and the final conclusions of the analysis are provided.

## 2.1 Evaluation criteria

In order to analyse and evaluate the state-of-the-art architectures as well as to determine how our proposal ranks, we need to define a set of evaluation criteria.

Most of the selected criteria can be found, implicitly or explicitly, throughout all the literature concerning ubiquitous and pervasive computing architectures: they represent core concepts and hot topics. Examples of these criteria are decentralisation, context-awareness, autonomy or standards adherence. Other criteria are more specific to our research goals such as reasonability or device implementation cost. However, the latter principles can also be easily found in similar related research.

We have organised the criteria into four different categories depending on their nature:

### Architectural

**Decentralisation:** at the heart of any Ubiquitous Computing system, decentralisation promotes an spontaneous and serendipitous nature, non-critical components in the architecture, dynamic reconfiguration according to every situation, natural deployment of elements and scalability, among others. However, the design and planning of decentralised systems is more difficult, as well as resulting in a higher load of network traffic due to synchronisation and coordination messages.

**Lightness:** architectural elements and software components in particular should be designed in such a way that they can be easily embeddable in resource-constrained platforms and devices, as well as promote operational simplicity if possible.

### Intelligence

**Reasonability:** is the ability of the system to acquire and apply knowledge via reasoning processes. In order to create the intelligence-enabling component of any smart device or environment, artificial intelligence techniques must be applied to a certain degree.

**Context-awareness:** is the ability of the system to perceive and identify relevant information and perform reactive behaviour to provide the desired response. Intelligent context-awareness is the ultimate goal of Ubiquitous Computing and Ambient Intelligence.

**Autonomy:** is the ability of the system to operate and perform without user intervention. Autonomic computing is one of the major trends in the Ubiquitous Computing arena, in order to design self-\* systems: self-configuring, self-healing, self-optimised and self-protected [KC03] [MK07].

### Technological

**Technological consistency:** represents the degree of coherence among the technologies used in a system. If possible, complementary technologies must be applied in order to obtain synergistic performance and future reusability. For example, transporting XML messages over HTTP is more natural, standardised and desirable than doing it over CORBA.

**Standards adherence:** represents the degree to which the designed system reuses and applies widely accepted standards, thus taking advantage of previous works and promoting skill reuse within the industry and academia. It is a major intention of our research to create new original work while assuring the highest possible degree of standards adherence.

### Economical

**Device implementation cost:** is the cost, both in terms of money and effort, required for the desired system to be implemented in actual devices and appliances. In some architectures, due to their centralised nature, device implementation is neither required nor feasible.

**Scenario deployment cost:** is the cost, both in terms of money and effort, required for the desired system to be deployed in a particular scenario. Hard-to-configure systems use to exhibit the higher degree of cost at this item, while seamlessly connected decentralised systems can be deployed naturally as connections and information flows emerge spontaneously. Our intention is to achieve some kind of plug-and-play architecture with minimum cost during deployment.

These criteria are not isolated. Certain dependencies exist among some of them in such a way that the degree of fulfilment in one concrete criterion can affect other in a positive or negative manner. These dependencies are illustrated in Figure 2.1, where rows represent influencing criteria while columns represent influenced criteria.

First, a high degree of decentralisation favours the design of distributed lightweight components, instead of a bulky and heavy central controller. Decentralisation also promotes a higher level of autonomy in architectural elements while reducing the scenario deployment cost, since environments are created by the emergent coordination of distributed elements.

Lightweight components make feasible the actual implementation and promote a lower device implementation cost, but they also reduce the degree and quality of the embedded reasoning processes which normally demand some amount of software complexity.

Reasonability affects lightness negatively in the same way, since the more reasoning power the device is provided with, the heavier the component becomes. In a similar way, hosting reasoning processes in constrained devices increases their costs and the efforts in terms of specialised workforce to implement these features. However, reasonability promotes both

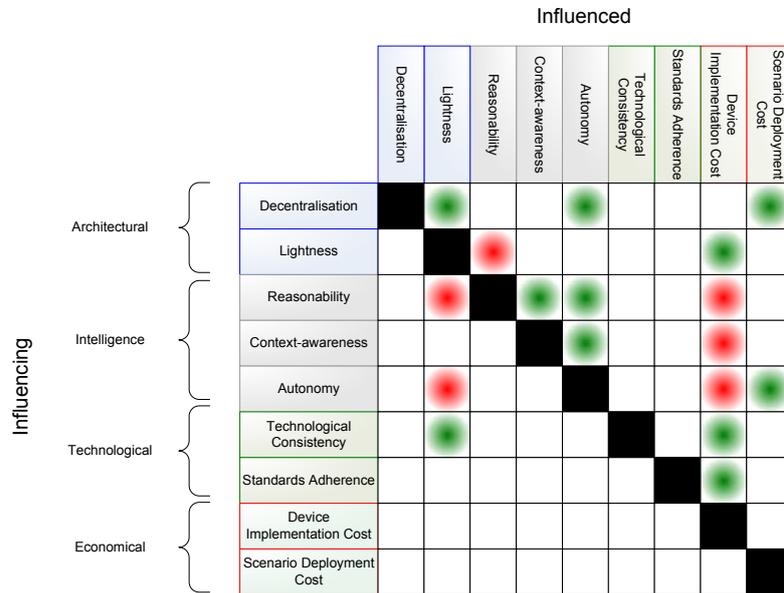


Figure 2.1: Criteria dependency matrix.

context-awareness and autonomy, which can take advantage of the built-in intelligence to better determine the behaviour to perform.

Context-awareness promotes autonomy, since architectural components can self-regulate their behaviour depending on context information provided by surrounding entities. The penalty for achieving a greater degree of awareness is, as usual, a higher device implementation cost.

Again, device implementation cost and component lightness can be affected by the degree of individual autonomy in the resulting architecture, since this autonomy also demands provision of intelligent mechanisms. On the other hand, scenario deployment cost is reduced significantly due to the self-management, self-healing and the other self-\* properties of the autonomic nature, which eases the deployment process in any environment with minimal human intervention.

Technological consistency can significantly reinforce APIs reusability during the implementation, thus promoting more lightweight components that would be negatively affected by mixing up non-complementary technologies. At the same time, we consider that technological consistency simplifies overall design by taking advantage of existing mechanisms, procedures and interfaces, thus reducing somehow the device implementation costs.

Finally, standards adherence promotes reduction of time and device implementation costs, since existing designs and APIs can be reused or slightly adapted, as well as trained workforce can be more easily found.

Figure 2.2 illustrates in a graph-like way the existing influences among the different criteria. As expected, device implementation cost is the more sensitive item, being affected by several others, but possibly counterbalanced by lightness, technological consistency and standards adherence.

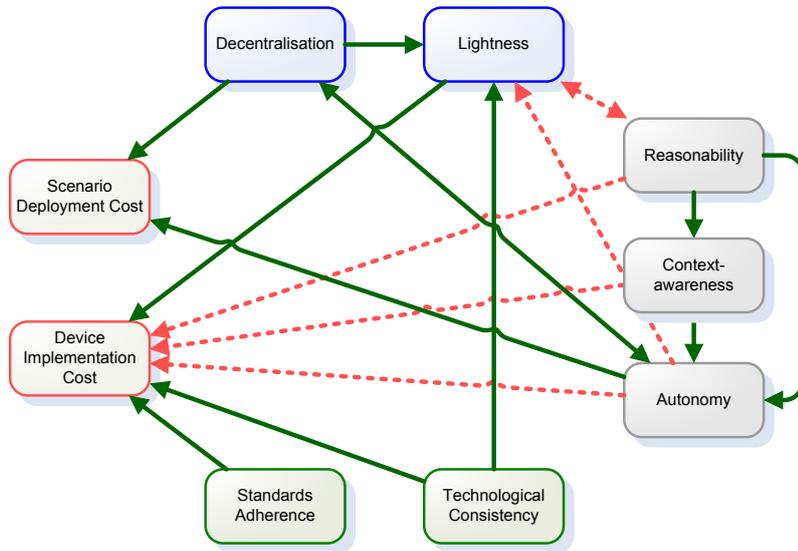


Figure 2.2: Criteria dependency map.

On the other hand, decentralisation and autonomy do not only contribute to enhance other criteria, but also create a somewhat synergistic positive flow, as well as reasonability enhance the other intelligence-related factors.

Not all these criteria are equally important, but depending on the desired strengths of the resulting architecture some of them must be promoted. We will focus primarily on maximising decentralisation, reasonability, context-awareness and, in a lesser extent, technological consistency. We think that the three first items retain the core functionality of pervasive computing, while technological consistency contributes to a neat and academically elegant solution.

Standards adherence, device implementation cost and scenario deployment cost form our second target group, while lightness and autonomy, despite important, constitute value-added factors.

Table 2.1 enumerates the criteria with relative weights depending on their importance to our goals.

<i>Criterion</i>	<i>Weight</i>
<i>Decentralisation</i>	4
<i>Reasonability</i>	4
<i>Context-awareness</i>	4
<i>Technological Consistency</i>	3
<i>Standards Adherence</i>	2
<i>Device Implementation Cost</i>	2
<i>Environment Deployment Cost</i>	2
<i>Lightness</i>	1
<i>Autonomy</i>	1

Table 2.1: Criteria's relative weights of importance.

Jointly, all these criteria also support the physical integration and spontaneous interoperation characteristics of Ubiquitous Computing [KF02], and the resulting architecture should meet the main intention and hypothesis of our research.

## 2.2 Universal Plug and Play

Universal Plug and Play (UPnP) is a standard architecture for pervasive peer-to-peer connectivity of computers, devices and appliances, mainly aimed at home environments. UPnP is strongly based on TCP/IP and specially on web technologies (HTTP and XML), with open specifications distributed by the UPnP Forum, a consortium backed up by more than 700 firms, covering all the range from software solutions companies such as Microsoft or Hewlett-Packard, semiconductors manufacturers such as Intel or Siemens, to appliances manufacturers such as Philips or Sony.

The UPnP architectural concept is based on the traditional Personal Computers' Plug and Play model that eases the process of installing and configuring a peripheral device. The goal in UPnP is to extend this model in order to allow users to have automatically installed and configured their own network of interconnected heterogeneous devices (computers, printers, routers, TV, video & audio appliances, surveillance cameras, lighting controls, and so forth) without all the burden of manual work: this is called *zero-configuration networking*. No drivers are required in UPnP; standard interfaces over communication protocols, mainly XML over HTTP, enable devices for universally interacting in dynamic ad-hoc networks.

The UPnP architecture [UPn03] is capable of discovering new devices and disconnections, retrieving devices' characteristics, invoking functions and sending notifications about detected events, all these features with device and platform independence, vendor interoperability and extensibility. Device and platform independence is achieved through standard communication protocols that can be implemented in any language or operating system; in fact, there are UPnP stacks for Microsoft Windows, PocketPC, Linux, and several embedded OS among others. In order to achieve interoperability, vendors collaborate standardising device capabilities on a device-type basis in documents called DCPs (Device Control Protocol) that constitute declarative contracts about device's characteristics and capabilities. Finally, extensibility is achieved by augmenting DCPs with vendor's own proprietary extensions.

### 2.2.1 UPnP Architecture

The UPnP protocol stack illustrates clearly how the architectural requirements are met (see Figure 2.3). Upper layers negotiate UPnP specific information structures: from the generality of the device architecture to the DCPs standardized by the UPnP Forum, and finally vendor provided extensions at the topmost layer. Lower layers are populated by communication protocols, some of them specially created and designed for UPnP and not yet accepted as *de jure* standards, such as SSDP, GENA and unicast/multicast HTTP over UDP; and others widely accepted as standards: IP, UDP, TCP, HTTP and SOAP.

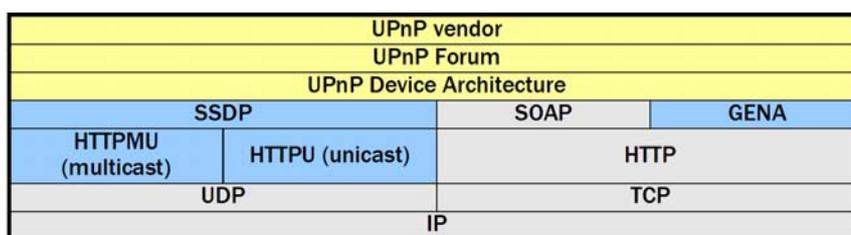


Figure 2.3: UPnP protocol stack.

There are two types of devices in UPnP: control points and controlled devices (referred simply as “devices”). Control points act as clients requesting services from devices in a master/slave-like model. Devices act as servers exhibiting services to surrounding control points and fulfilling their requests. Control points are generally complex appliances like computers, PDA or TVs, with some kind of user interface (graphical or vocal), so that the user can interact with them and perform control actions on devices, which

are simple appliances featuring communication capabilities with control points.

Controlled devices can be composed logically by other “embedded devices” in a hierarchical way. For example, a DVD player/recorder is a unique “root device” that is composed logically of two embedded devices, a DVD player and a DVD recorder, which are announced to the environment and managed individually. This features allow a separation of roles in UPnP: despite having a unique physical device, there are different logical devices.

Root and embedded devices are populated by services, which are the real target of control points in order to invoke specific actions on devices and perform the desired behaviour. For example, the DVD player can offer services such as “play”, “pause” and “stop playing”, while the DVD recorder can provide “record” and “stop recording” services.

The diagram of Figure 2.4 illustrates the hierarchical model of UPnP device architecture and services.

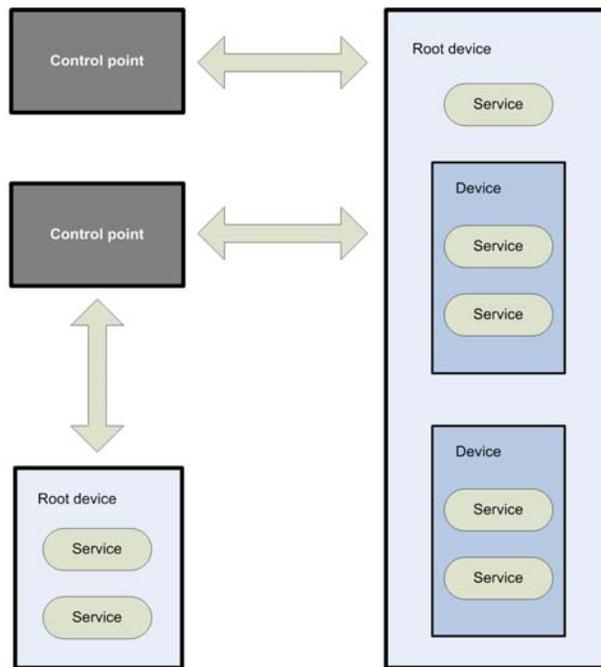


Figure 2.4: UPnP device architecture and services.

### 2.2.2 UPnP Activities

UPnP activities are divided into six steps that can be partially or fully performed by control points and devices: addressing, discovery, description, control, eventing and presentation [JW03].

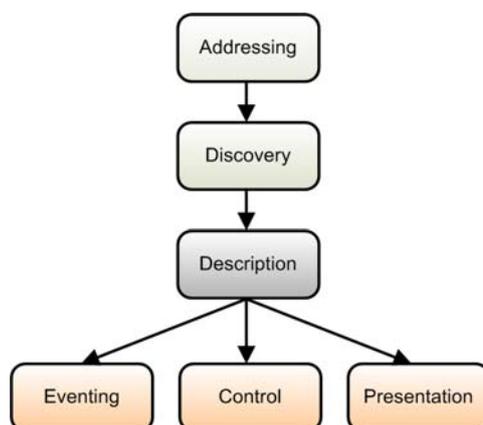


Figure 2.5: Ordered sequence of UPnP phases.

#### Addressing

Addressing is one of the foundations for UPnP networking, defining how a device can obtain a valid IP address in the setting-up process. If a DHCP server is available in the network, the address is obtained by leasing from the DHCP server. Otherwise, the device must implement the Auto IP mechanism: selecting randomly a link-local IP address in the range 169.254/16, checking whether the address is already taken by any other device or not via ARP probe and gratuitous ARP, and selecting another if so.

#### Discovery

Discovery is the second step in UPnP, enabling control points to find surrounding devices via either polling mechanisms or device advertisements. When a new control point joins the network it multicasts a discovery message, requesting existing devices to show up and respond. Similarly, when a new device joins the UPnP network it multicasts advertisements messages about itself, its embedded devices and services. The combination of these two mechanisms allow control points to have a continuously updated list of existing devices and services in the network. A device can disseminate a revocation of its advertisements when abandoning the network to inform that it is no longer available, or when its IP

address has changed. The multicast endpoint for the Discovery process is 239.255.255.250:1900.

SSDP (Simple Service Discovery Protocol) [GC<sup>+</sup>99] is the protocol used during Discovery in UPnP. SSDP is an extension on HTTPMU (HTTP multicast over UDP) [GS00], similar to the traditional HTTP, but conveniently adapted for UDP limitations. SSDP includes `NOTIFY` and `SEARCH` messages for advertisements and polling. Advertisements and search responses originated by a device include several important headers that are used later on during the UPnP Description phase:

- `Location`: contains the URL where description of the root device can be downloaded to retrieve its full set of characteristics .
- `USN (Unique Service Name)`: identifies the advertised device or service via its unique UUID.

An example of interaction with a `SEARCH` message and one response is shown in Figure 2.6.

UPnP Control Point	UPnP Device
Request →	← Response
<pre>M-SEARCH * HTTP/1.1 HOST: 239.255.255.250:1900 MAN: "ssdp:discover" MX: 3 ST: upnp:rootdevice</pre>	<pre>HTTP/1.1 200 OK Cache-Control:max-age=60 EXT: Location:http://192.168.2.1/igd.xml Server:DSL router/1.02 UPnP/1.0 UPnP-Device-Host/1.0 ST:upnp:rootdevice USN:uuid:00000000-0000-0001-0000-000d54a55be6::upnp:rootdevice</pre>

Figure 2.6: Example of a `SEARCH` message and its response during UPnP Discovery.

### Description

After Discovery, the UPnP control point still knows very little about the device. The `Location` header of the response message contains a URI where

the control point can retrieve the full description of the device during the Description phase. Description is partitioned into two complementary sections: device description and services description.

The device description is an XML document that provides information about the manufacturer, device type, name, model, serial number and other specific characteristics depending on the device type. The UPnP Forum has standardized some common device descriptions, called Device Control Protocols (DCP), but vendors are free to extend them to add extra information about the device following the standard template. Of course, a vendor can create a device description from scratch for a particular device for which no DCP exists. Device description also includes a list of available services the device is able to execute under request (including control and eventing information about those services), embedded devices and the presentation URL.

Service description is achieved by accompanying every entry at the list of available services by a URL, contained in the XML element `<SCPDURL>`, where extended information about the concrete service can be retrieved in an XML-based language called Service Control Protocol Definition (SCPD). SCPD contains descriptions about service arguments, return values, state variables and its ability to send events when variables change. Service descriptions are also standardized by the UPnP Forum for common devices, but vendors can extend them to incorporate new services.

Both device and service descriptions are retrieved via a simple HTTP GET operation on the URI of the `Location` header for device descriptions and `<SCPDURL>` for service descriptions as shown in Figure 2.7.

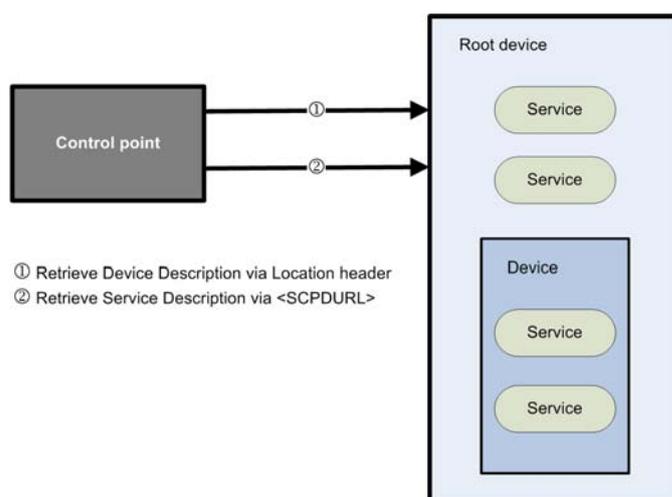


Figure 2.7: UPnP Description Phase.

## Control

After Description, UPnP control points are aware of the characteristics of existing devices and available services. During the Control phase, control points take advantage of this information in order to invoke services on devices and poll them for values. Invocation takes the form of remote invocations using SOAP [Wor03] over HTTP, although the language for describing the invocation interface is not the traditional and standardized WSDL [Wor01a] but SCPD, being the service description available at `<SCPDURL>` as explained above.

The response to a UPnP SOAP invocation can convey the appropriate return values honouring the service description, or UPnP defined error codes.

UPnP devices are generally modelled through state variables that represent state information about the device. As previously explained, service descriptions in SCPD do not only provide information about in/out arguments for invocation, but also related state variables that can be altered, so the service operation interface is completely provided.

## Eventing

Eventing is one possible step in UPnP after Description, so being sibling of Control, and allowing control points to subscribe to event notifications on devices when state variables change their value. Not every state variable in a UPnP device is evented, so it is up to the manufacturer to decide which notifications are allowed.

UPnP Eventing uses GENA (General Event Notification Architecture) [CAG00] as underlying protocol. GENA is an HTTP-like protocol that defines new kinds of methods and headers for event notification management.

In order to subscribe to some state variable changes, the subscriber sends a subscription message (`SUBSCRIBE`) that is confirmed by the device, acting during Eventing as a notification publisher. The request message contains a callback URL for receiving notifications, and the response message contains the subscription duration. Subscription works following a leasing mechanism, so that subscription renewal is mandatory for obtaining longer subscription periods. Of course, unsubscription messages (`UNSUBSCRIBE`) exist for subscription cancellation.

After subscription, the publisher sends an event message (`NOTIFY`) notifying the actual values for all the evented variables in the device. From that point on, every time a value changes a notification message is delivered to the specified callback URL at the subscriber.

## Presentation

The UPnP Presentation phase is the unique user-oriented step in the architecture. During the Description phase a presentation URL is provided, where an HTML page is available for direct user interaction with the device. This feature allows a user to directly access and visualize a friendly representation of the UPnP device, including functions such as querying state variables and invoking actions directly over this user-interface, acting as a kind of remote control.

The web page can be hosted in an embedded server at the device or at an external server acting as UPnP gateway.

### 2.2.3 Conclusion

Among the main advantages of UPnP are its simplicity to create a distributed inter-device communication system based on web technologies, and the evidence that it is probably the most popular architecture in the market in terms of manufactured devices and commercial success, specially Internet gateways. Among the disadvantages, the most important are the lack of security mechanisms and the limitations in terms of scalability as the number of devices increases [FDW<sup>+</sup>04, MAA<sup>+</sup>04].

UPnP is a suitable candidate for a pervasive computing architecture based on simplicity and performance in constrained devices, and backed up by web technologies. Particularly, SSDP performs well in small environments with a limited number of devices and its HTTP-like format seems suitable for extension with new headers to support smarter negotiation processes during discovery, including some form of semantic information exchange.

However, UPnP does not feature any built-in intelligent mechanism. Intelligence must be provided from other external sources, either the user or other reasoning infrastructure.

An analysis of UPnP against the evaluation criteria leads to the following results:

**Decentralisation:** UPnP features a somewhat centralised nature, since control points constitute a sort of central coordination entity. UPnP devices do not act autonomously and communicate themselves directly, but control points command their operation. However, since control points can be easily embeddable in any device, it is possible to design appliances with both built-in control points and devices in order to achieve a more decentralised architecture. Medium.

**Reasonability:** UPnP does not provide any intelligence at the device side: either the user or an external process must provide the required intelligence for governing the devices. There are no native elements in the UPnP architecture hosting any kind of reasoning processes or capabilities. UPnP devices and control points are unable to understand the information they use and, of course, no form of semantic discovery or processing is performed [PB04]. None.

**Context-awareness:** UPnP does not include any kind of autonomous reactivity mechanism at devices or control points. Reactive behaviour must be programmed ad hoc using the platform facilities, although UPnP messages can be used both for retrieving information or invoking remote operation. None.

**Technological Consistency:** UPnP applies web technologies throughout almost all the levels in the protocol stack, except for discovery and eventing purposes, where HTTP-like protocols are provided. High.

**Standards Adherence:** UPnP is based on standard Internet communication mechanisms, TCP/IP and HTTP, as well as non-standard ones such as HTTP over UDP, SSDP and GENA. These last protocol specifications have not been completed yet, and exhibit some inconsistencies not resolved during the last years. Despite these drawbacks SSDP has been used as a lightweight discovery protocol suitable for home and office environments [FDW<sup>+</sup>04], being a well-positioned candidate for this task in pervasive computing systems [LH02]. Medium.

**Device Implementation Cost:**

UPnP requires augmenting existing appliances with TCP/IP enabled stacks, XML processing and internal web servers. Suitable platforms are not very expensive and widely available. Low.

**Environment Deployment Cost:** from a networking point of view UPnP deployment is not costly at all. But in order to provide environmental intelligence, further work must be performed since UPnP does not provide any mechanism to inject intelligence into the devices populating the environment: ad hoc device and control points programming must be performed for every particular scenario. Low.

**Lightness:** UPnP does not demand much CPU and memory resources at the host machine, keeping its architecture simple and easy to deploy. The unique requirements needed for UPnP devices are TCP/IP capabilities and a limited form of XML processing, which can be performed without remarkable CPU load. High.

**Autonomy:** since UPnP devices act as slaves of control points, they cannot perform any autonomous high-level action within the UPnP model

without being commanded by the latter. Moreover, users have a very active role in UPnP environments, as the existence of the presentation layer proves, making the whole model very user intrusive. Low.

<b>Criterion</b>	<b>Value</b>
<i>Decentralisation</i>	<i>Medium</i>
<i>Reasonability</i>	<i>None</i>
<i>Context-awareness</i>	<i>None</i>
<i>Technological Consistency</i>	<i>High</i>
<i>Standards Adherence</i>	<i>Medium</i>
<i>Device Implementation Cost</i>	<i>Low</i>
<i>Environment Deployment Cost</i>	<i>Low</i>
<i>Lightness</i>	<i>High</i>
<i>Autonomy</i>	<i>Low</i>

Table 2.2: Analysis of UPnP against the evaluation criteria.

## 2.3 Task Computing

Task Computing is an ongoing joint effort by Fujitsu Laboratories of America and the MINDSWAP group, devoted to Semantic Web research, at the University of Maryland Institute for Advanced Computer Studies.

The goal of Task Computing is to “*fill the gap between the tasks that users want to perform and the services that constitute available actionable functionality*” [MLPS03]. Task Computing presumes initially that users do not know how to achieve their goals when using computing facilities due to increased complexity at computing environments and tasks, and tries to ease the process by providing the user with an intelligent aid that hides the complexity of coordinating existing devices and services.

Task Computing provides dynamic service discovery, service publishing and management, task creation and execution on the fly [SHP03]. It even assists users in discovering what their goals are by suggesting possible tasks that can be performed with available facilities.

All these features try to solve the frustration of users in application-rich environments, where they have to orchestrate a variety of devices and applications. Using Task Computing they can focus on their final goal and accomplish it with a reduced number of simple interactions.

Service composition can be seen as the “*process of creating customized services from existing services by a process of dynamic discovery, integration*”

*and execution of those services in a planned order to satisfy a request from a client” [CPJ<sup>+</sup>02].*

Some examples of documented scenarios [MPL03] that can be accomplished using Task Computing technology are: exchanging business cards, showing and sharing a presentation, scheduling a future presentation or checking and printing directions to the airport. All of them are accomplished by sharing services on different devices and orchestrating those services to create a workflow in order to carry out the desired task.

A prototype of Task Computing environment has been implemented experimentally for Smart Conference Rooms and Home Multimedia Environments [SMAL04] and the first public results date back to 2003, where the group of researchers led by Dr. Ryusuke Masuoka at Fujitsu Laboratories and Dr. James Hendler at the University of Maryland published several papers [MPL03, MLPS03, MMS<sup>+</sup>05] explaining the basics of their approach. It consisted in applying Semantic Web technologies to Pervasive Computing scenarios for semi-automatic composition of tasks, based on their previous research [SHP03].

The eBiquity research group has taken similar approaches, but trying a more decentralised model [CPJ<sup>+</sup>02] to cope better with mobility issues. Indeed, in wired systems, a centralised composition entity is assumed, which is no longer valid for pervasive systems. Their approach was based on a distributed broker that could be executed in any node of the environment, using Bluetooth as communication mechanism and DAML-S [DAM02] (precursor of OWL-S) for describing services.

### 2.3.1 Task Computing architecture

The Task Computing architecture is composed of four different layers, performing complementary activities:

- Realization layer: it is the bottommost layer, directly representing available facilities. There are three different types of entities at this layer: devices, applications and e-services over the Web.
- Service layer: available facilities from the Realization layer are embodied into the form of service at this layer and services interfaces are constructed. Semantic Service Descriptions (SSD) comprising knowledge about these services are also created in order to disseminate information.
- Middleware layer: this layer is in charge of service discovery, service composition and execution, and other management activities such as

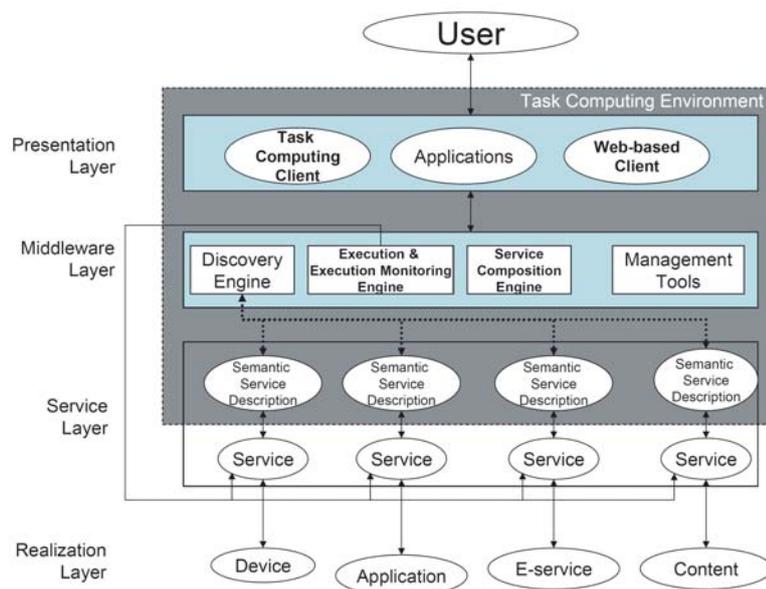


Figure 2.8: Task Computing architecture. Source: [SLM04].

service publishing. In some way, it glues services created at the Service layer with available underlying technologies to support transport and management functions over them.

- **Presentation layer:** it is considered the most important layer in the architecture. It provides the user with an abstraction of available tasks that can be performed in the environment, hiding underneath complexity, and allowing the user to dynamically assemble components to perform the desired task. A client implementing the Presentation layer is usually referred to as Task Computing Client (TCC) and makes use of well-defined interfaces to the Middleware layer.

Task Computing is implemented in concrete TCEs (Task Computing Environments), which are computational systems able to perform Task Computing functionality and composed of Task Computing Clients, Semantic Service Descriptions, Semantic Service Discovery Mechanisms and Service Controls.

### Task Computing Client (TCC)

Task Computing Clients are utilised by users to accomplish desired tasks. STEER (Semantic Task Execution Editor) is a prototype TCC developed

for this goal. Depending on available discovered services, STEER creates possible task compositions and suggestions for the user about the most likely desired tasks. The user, generally via a graphical interface, creates connections, orchestrating and composing the services to perform the task.

### **Semantically Described Services**

Semantically Described Services are orchestrated to perform the desired task in a coordinated manner. They are described through Semantic Service Descriptions (SSD), a document type encoded using OWL-S [Wor04b], the W3C standardized language for semantically annotating web services. However, service invocation documentation is represented via WSDL, so there is a need to bound semantic descriptions to WSDL invocation parameters. WSDL is adequate for programmers, but OWL-S generated descriptions are better for end users who do not know about the artifacts to perform the actual invocation, but do understand the semantics of the desired service.

SSDs are constituted of three parts that implement the separation between semantic descriptions and actual implementation: profile (high level descriptive information), process (semantic description of the process and glue between profile and grounding) and grounding (service implementation and mappings).

### **Semantic Service Discovery Mechanisms (SSDM)**

Semantic Service Discovery Mechanisms are provided in a Task Computing Environment to explore and search for new services to make them available to the user. Task Computing typically uses UDDI, UPnP SSDP or Jini for this purpose, but other discovery mechanisms could be applied as well.

Whatever the mechanism employed, Task Computing defines four distinct discovery ranges:

- Empty: services located in the empty range cannot be discovered by anyone. A user can decide whether to assign this range for a service to make it unavailable for everyone, including him/herself.
- Private: services located in the private range are only available to the owner
- Group by subnet: services in this range can be discovered by other entities located within the same computing environment. This range is typically associated with Ubiquitous Computing environments, where

services are available to nearest clients. SSDP (Simple Service Discovery Protocol) [GC<sup>+</sup>99] and Jini [Sun99] can be used as discovery mechanisms here (and also for service advertisement, which is complementary to discovery).

- **Group by interest:** services in this range can be located by other people with similar interests or group membership as the owner. Currently, there is no discovery mechanism specified for this range by the authors.
- **Public:** services in this range can be discovered by anyone, whatever the interest and the location. UDDI [OAS04] is an example of discovery mechanism that can provide Semantic Service Descriptions of public services.

### Service Control (SC)

Service Controls allow management of services by dynamically customizing some of their properties, such as discovery range, expiration time, limiting the number of possible invocations and even the service name or description. Service Controls are also able to hold temporarily, remove or change how the services are provided.

### 2.3.2 Semantic-ization and Service-ization

Task Computing defines these two concepts in order to refer different steps in the process of transforming existing services to shape them to Task Computing Environments requirements.

Semantic-ization is the process of creating a semantic object from any other application or OS object. Currently, Task Computing supports a dozen of object types such as PIM contact, schedule entry, file from the OS, and so forth.

White Hole is the component of the architecture in charge of creating a OWL semantic description based on the object type.

Service-ization is the process of creating a service, with an associated endpoint, based on the OWL semantic description generated in the previous step, and prepare it for publication using the appropriate mechanism to allow discovery (e.g.: UDDI or SSDP).

PIPE is the component in Task Computing in charge of service-ization. PIPE creates a web service, a WSDL document describing the service invocation mechanism, and an SSD document describing the service

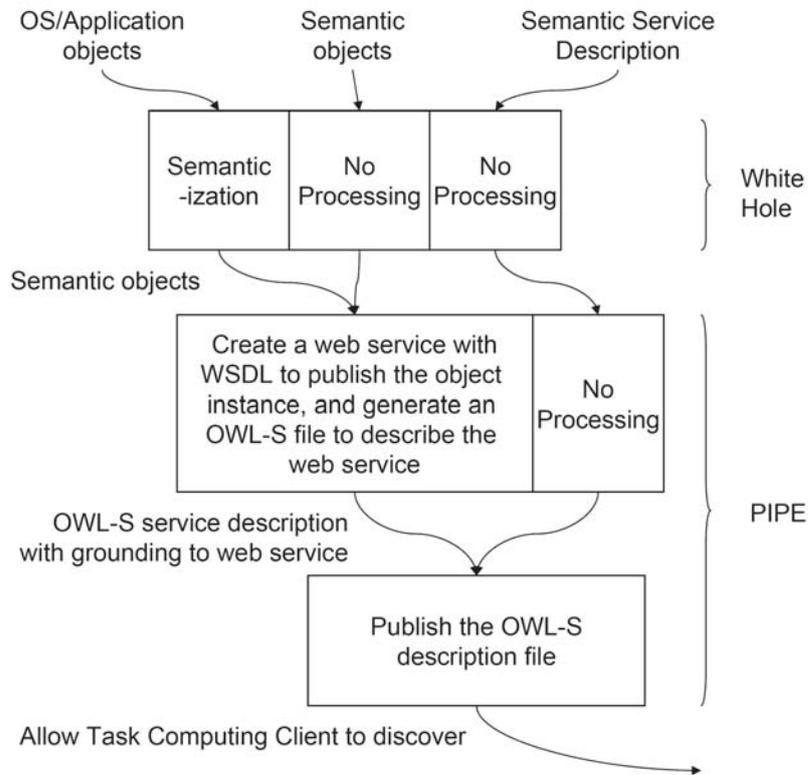


Figure 2.9: Semantic-ization and Service-ization in Task Computing. Source: [SLM04].

semantically. PIPE also acts as a Service Control allowing management of service provision.

Semantic-ization and Service-ization are not carried out if the source object is already represented in a semantic way or if it is shaped as a web service accompanied by all the required information. In that case, PIPE only publishes the service without performing any transformation.

### 2.3.3 Conclusion

Task Computing is primarily a framework for services orchestration, composition, and execution. All the mechanisms it features are aimed at this goal: service publishing and discovery, semantic descriptions, service-ization of resources to make them available to any requester, and so forth. These features can be implemented in multiple environments, not being specially addressed for Ubiquitous Computing scenarios. In fact, Task Computing

applies well-known Internet-wide technologies such as UDDI for discovery or traditional Web Services at external servers as endpoints.

In order to assume a more context-aware nature, Task Computing has embraced some pervasive computing technologies to complement existing ones, such as SSDP for discovery, so services can be found both at a global level via UDDI and at a local level via the *group by subnet* discovery range. This approach allows surrounding services discovery in a physical environment and the possibility of making local resources such as documents available as services to nearest parties, via semantic-ization and service-ization.

As reported by the authors, after Task Computing evaluation and despite automatic service composition, users feel more confident taking decisions about suggestions based on natural language labels that accompany service descriptions (*“it is the human-centered descriptions that give the user confidence to make the final selection”* [MLPS03]). This aspect leads to the conclusion that natural language annotation of services is more appropriate when presented to users for selection and operation than semantic descriptions, that can be processed automatically without user intervention.

On the other hand, Task Computing seems to be more oriented to solve usability problems when non-technical users have to select and orchestrate devices and services, than to address Ubiquitous Computing problems. This opinion is backed up by the fact that authors stress several times that the most important layer or components in Task Computing are user interface related ones (*“the most important aspect of Task Computing is the presentation layer”* [SLM04], *“what is unique about our Task Computing implementation is [...] driven by end-users, rather than by developers, via a user-interface”* [MPL03]).

An analysis of Task Computing against the evaluation criteria leads to the following results:

**Decentralisation:** Task Computing is not aimed at creating a network of interconnected devices. The TCEs are intended to run in desktop computers or servers that can be connected to UPnP or Jini networks if device control is required. Low.

**Reasonability:** Task Computing uses semantic information to annotate service descriptions and perform service composition, but neither reasoning nor domain ontologies are provided to understand context information. TC only is aimed at matching required and offered services. Low.

**Context-awareness:** except for service discovery mechanisms (which strictly is a technical issue, not related to context-awareness), no other form of capturing context, sensing environmental conditions, and so forth, are provided directly by Task Computing. No form of automatic responsive behaviour is provided in the model, but user intervention is required. Low.

**Technological Consistency:** Task Computing applies web services and Semantic Web technologies in a coherent and synergistic manner. High.

**Standards Adherence:** in general terms, Task Computing honours standards and recommendations trying to reuse existing technologies. Standards such as OWL-S, WSDL, HTTP, UDDI, and even industry *de facto* standards such as SSDP are widely used. High.

**Device Implementation Cost:** the Task Computing Environment is not expected to be implemented on devices due to the software size and complexity as well as to requirements about user interaction that cannot be accomplished by limited devices. High.

**Environment Deployment Cost:** Task Computing requires deployment of at least one TCE in the environment as well as configuring the discovery mechanisms and available services to provide intelligent service composition. Medium.

**Lightness:** processes such as Semantic-ization and Service-ization as well as the need of the Task Computing Client, result in complex and heavy software components. Devices need an amount of computing resources (processing power and screen size, among others) not presently available in every embedded platform. Low.

**Autonomy:** Task Computing requires users to actively participate in the process of adapting the environment for the desired task: the user is continuously required to be in the loop. Services are discovered and presented to the user which has the final decision over the process. Low.

## 2.4 CoBrA and SOUPA

CoBrA (Context Broker Architecture) and SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) are efforts from the eBiquity group

<b>Criterion</b>	<b>Value</b>
<i>Decentralisation</i>	<i>Low</i>
<i>Reasonability</i>	<i>Low</i>
<i>Context-awareness</i>	<i>Low</i>
<i>Technological Consistency</i>	<i>High</i>
<i>Standards Adherence</i>	<i>High</i>
<i>Device Implementation Cost</i>	<i>High</i>
<i>Environment Deployment Cost</i>	<i>Medium</i>
<i>Lightness</i>	<i>Low</i>
<i>Autonomy</i>	<i>Low</i>

Table 2.3: Analysis of Task Computing against the evaluation criteria.

at the University of Maryland, Baltimore County, USA, to create a context-aware pervasive system applying Semantic Web technologies. These projects were primarily led by Dr. Harry Chen [Che04] during the years 2003-2005.

The eBiquity research group had already worked on some approaches using ontologies for context-awareness in pervasive applications [CTS<sup>+</sup>01], assuming Dey’s definition of context as the driving concept [Dey01]. Their goal then was creating context-aware teamwork cooperative environments. Within the scope of this work, they noticed how humans share some attributes that enable them to understand situations, and these same attributes should be migrated to machines in order to obtain similar results (“*sharing ontology, sensing context and reasoning are crucial to the realization of context-aware software applications*”) [CTS<sup>+</sup>01].

At the same time, they also noticed that much effort was put on modelling generic context information as part of domain-specific information. Thus, basic data such as time, location, identity, and so forth, were once and again modelled for every ubiquitous application or service, without reusing any previously developed structures. Representation of these concepts using ontologies would allow reuse of information structures and relationships, so the eBiquity research group adopted Semantic Web technologies.

The first experiences were applied in the CoolAgent Recommendation System using RDF and  $P_{fc}$  (Prolog Forward Chaining) at Hewlett-Packard Laboratories. CoolAgent RS is a multi-agent system that can automatically recommend different types of tailored information to users by reasoning about the context without any explicit manual input (“*Enabling context-awareness is one step closer to the realisation of computing systems that can act in advance and anticipate user’s needs*”) [CTS<sup>+</sup>01].

The prototype for CoolAgent RS Document Recommendation Service is configured to gather and reason upon some concrete context information

such as presence information, user's profile, meeting's subject or schedule and organisational information among participants, in order to select the documents they need to attend the meeting. They also developed other versions for different domains, such as the CoolAgent RS Food Recommendation Service for finding the nearest restaurant that best matches the user's profile.

Again, through these experiences, it became clear the need for sharing a common ontology to represent concepts such as places or people, and Dr. Chen started modelling them using RDF and RDF Schema.

At the same time, while creating pervasive applications there were more concerned about "*Ubiquitous Computing systems based on the cooperation of autonomous, self-describing, highly interactive and adaptive components that are located in the vicinity of one another*" [CFJ01].

All these experiences led to the conclusion that agents teamwork and collaboration was a hot issue, specially interesting in Ubiquitous Computing environments [CF02], since agents can act on behalf of devices and cooperate for a common goal.

Some previous background existed about this: the Wooldridge-Jennings CPS (Cooperative Problem Solving) Model [WJ99] structured the collaboration process into several steps:

1. Recognition: in which an agent identifies the potential for cooperation
2. Team formation: in which the agent solicits assistance
3. Plan formation: in which the newly formed collective attempts to construct an agreed joint plan
4. Execution: in which members of the collective play out the roles they have negotiated

The Wooldridge-Jennings CPS model seemed valid for coordinating agents in Ubiquitous Computing environments. In fact, it defined some sort of choreographic model, before the Choreography Markup Language [Wor05a] was designed.

For the resulting cooperative agent-based architecture they selected a central broker-based approach [CFJ03b], and named it CoBrA (Context Broker Architecture).

### 2.4.1 CoBrA: Context Broker Architecture

Figure 2.10 illustrates the CoBrA architecture. The central broker is in charge of several responsibilities [CFJ03b, CFJ03c]:

1. It provides a centralised model of context that can be shared by all devices, services, and agents in the space
2. It acquires contextual information from heterogeneous information sources, unreachable from resource-limited devices
3. It reasons about contextual information that cannot be directly acquired from the sensors (e.g., intentions, roles, temporal and spatial relations)
4. It detects and resolves inconsistent knowledge that is stored in the shared model of context
5. It protects the privacy of users by enforcing user defined policies to control the sharing and the use of their information

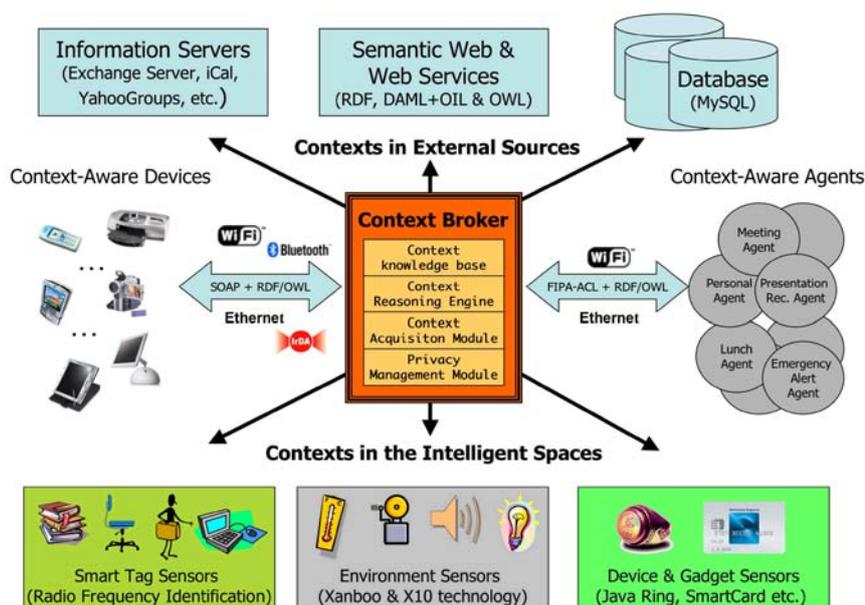


Figure 2.10: CoBra Architecture. Source: [CFJ03b].

The broker communicates with the information sources using an agent-based mechanism with JADE API (Java Agent DEvelopment Framework) [BPR99], which in turn can apply different communication protocols such as Java RMI, HTTP or IIOP. The payload is represented in FIPA ACL (Agent Communication Language) [fIPA02] (see Figure 2.11).

When message size limitations apply, such in the case of Bluetooth transport, the solution is to include a URL that points to the complete resource description to download from some server. However, this approach

shifts the problem to how and when must the information be published and updated at that server.

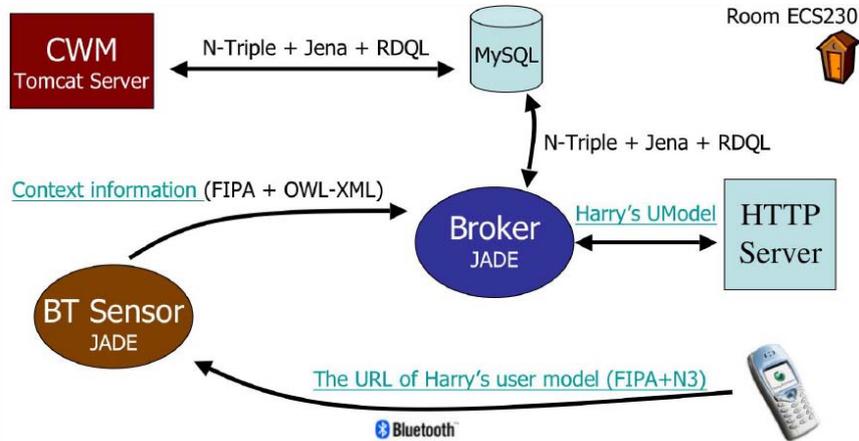


Figure 2.11: The context acquisition process in CoBrA. Source: [CFJ04c].

CoBrA does not apply any pervasive discovery mechanism, so elements in the environment must have a previous knowledge about the existence of the context broker, or make use of JADE API provided mechanisms for agent discovery.

The broker is composed of different elements that are combined to fulfil the above responsibilities [CFJ04a, CFJ05]:

- CoBrA Ontology (COBRA-ONT): it is required to describe contextual information. It declares classes and relationships for the specific domain CoBrA is aimed at: smart meeting rooms (EasyMeeting project). The CoBrA ontology defines people, agents, places and presentation events for supporting an intelligent meeting room system on a university campus, as well as its properties and relationships as shown in Figure 2.12.
- Context-Knowledge Base: it is an RDF triples storage facility that acts as a central repository of all the context information gathered from different sources. Implemented through an SQL database, this module offloads devices with the burden of maintaining the context knowledge, while providing access to it.
- Context Acquisition Module: it is in charge of acquiring context information from all the available sources. This is usually performed by means of agents that can implement specific communication mechanisms to obtain desired data from the original source (sensors, devices, appliances, and so on).

- Context Reasoning Engine: it is a logic inference engine for reasoning with ontologies and detecting inconsistencies. It is formed by two different tiers: the first tier uses ontology-based reasoning, that is, description logics, while the second tier uses domain-specific heuristics in the form of rules. For the first tier CoBrA can use any of the available ontology inference engines, such as TRIPLE [SD02], FACT [HST99] or RACER [HM01].
- Policy Management Module: it checks the users' policies before sharing their information. The privacy language in CoBrA is based upon the Rei policy language [KFJ03a], an ontology for modelling “rights, prohibitions, obligations and dispensations (deferred obligations)”.

CoBrA Ontology Classes		CoBrA Ontology Properties	
“Place” Related	Agents’ Location Context	“Place” Related	Agent’s Location Context
Place AtomicPlace CompoundPlace Campus Building AtomicPlaceInBuilding AtomicPlaceNotInBuilding Room Hallway Stairway OtherPlaceInBuilding Restroom Gender LadiesRoom MensRoom ParkingLot	ThingInBuilding SoftwareAgentInBuilding PersonInBuilding ThingNotInBuilding SoftwareAgentNotInBuilding PersonNotInBuilding	latitude longitude hasPrettyName isSpatiallySubsumedBy spatiallySubsumes accessRestricted- ToGender lotNumber	locatedIn locatedInAtomicPlace locatedInRoom locatedInRestroom locatedInParkingLot locatedInCompoundPlace locatedInBuilding locatedInCampus
	<b>Agent’s Activity Context</b>	<b>“Agent” Related</b>	<b>Agent’s Activity Context</b>
	PresentationSchedule Event EventHappeningNow PresentationHappeningNow RoomHasPresentationHappeningNow ParticipantOfPresentation- HappeningNow SpeakerOfPresentationHappeningNow AudienceOfPresentationHappeningNow  PersonFillsRoleInPresentation PersonFillsSpeakerRole PersonFillsAudienceRole	hasContactInformation hasFullName hasEmail hasHomePage hasAgentAddress  fillsRole isFilledBy intendsToPerform desiresSomeone- ToAchieve	participatesIn  startTime endTime Location hasEvent hasEventHappeningNow invitedSpeaker expectedAudience presentationTitle presentationAbstract presentation eventDescription eventSchedule
<b>“Agent” Related</b>			
Agent Person SoftwareAgent Role SpeakerRole AudienceRole IntentionalAction ActionFoundInPresentation			

Figure 2.12: CoBrA Ontology. Source: [CFJ03b].

CoBrA brokers do not need to be completely isolated. They can form federations, in which an individual manages part of the context information space (e.g. a room) while exchanging context information with others periodically [CFJ03b].

Probably the main innovation of CoBrA is the application of Semantic Web technologies, such as RDF and OWL for ontology modelling in pervasive spaces [CFJ03a], instead of programming languages classes and objects as previous systems did (Context Toolkit, Cooltown or Intelligent Room).

The CoBrA ontology, COBRA-ONT, embraced knowledge from different subdomains, some of them very specific to the problem to solve (smart meeting rooms), but others being clear candidates for reusability in other scenarios, such as those related to people, time, space, events and so forth.

This situation prompted the recommendation of segregating common concepts that could be applied in multiple scenarios, into one “standard” ontology: SOUPA.

### **2.4.2 SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications**

The application of ontologies in Ubiquitous Computing environments are important due to diverse factors [CFJ03b] [CFJ03c]:

1. Ontologies expressed in Semantic Web languages provide a means for independently developed context aware systems to share context knowledge, minimizing the cost of and redundancy in sensing.
2. RDF and OWL are knowledge representation languages with rich expressive power that are adequate for modelling various types of contextual information, e.g. information associated with people, events, devices, places, time, and space.
3. Because context ontologies provide an explicit representation of semantics, they can be used by available logic inference engines. Systems with the ability to reason about context can detect and resolve inconsistent context knowledge that often result from imperfect sensing.
4. The Semantic Web languages can be used as metalanguages to define other special purpose languages such as communication languages for knowledge sharing or policy languages for privacy and security. A key advantage of this approach is improved interoperability: tools for languages that share a common root of concepts can interoperate better than tools for languages that have diverse roots of concepts.

The conclusion is that a set of common ontologies enable knowledge sharing in an open and dynamic distributed system, provide a means for intelligent agents to reason about contextual information, and allow devices and agents to interoperate.

From this point of view, defining and reusing a common ontology for pervasive applications could bring all these advantages to future projects.

Taking COBRA-ONT as a starting point, Dr. Chen performed a selection and grouping of reusable ontologies and created SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) embracing what he thought were the fundamental ontologies for any pervasive application [CFJ03d] [CFJ04b] [CFJ04c] [CPC<sup>+</sup>04].

Moreover, OWL was used for ontology representation, which enabled SOUPA to stay ahead with the latest standard of Semantic Web technologies.

SOUPA adopted a modular approach, reusing some previously created ontologies [CPFJ04] such as FOAF [DZfJ05], DAML-Time (evolving later into OWL-Time [PH05] [Wor06c]), spatial ontologies of OpenCyc, Regional Connection Calculus (RCC), COBRA-ONT, MoGATU BDI [PJC05] and Rei policy ontology [KFJ03b].

SOUPA was structured into two levels: SOUPA Core and SOUPA Extension. SOUPA Core defines generic concepts for Ubiquitous Computing applications and constitutes the primary source of reusability, while SOUPA Extension defines additional concepts for supporting specific applications and domains (meeting and scheduling, digital documents, and so forth), and thus, it constitutes the basis for extensibility (see Figure 2.13).

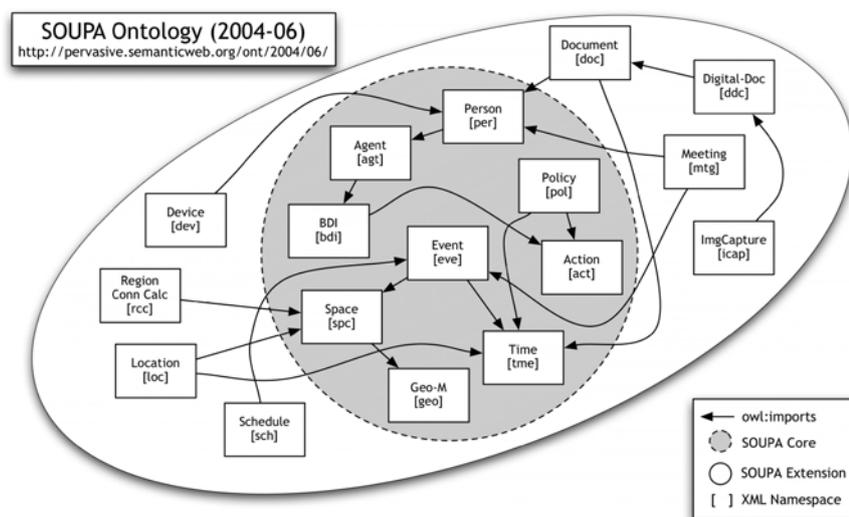


Figure 2.13: SOUPA Ontology. Source: [CPFJ04].

SOUPA does not define an architecture or behavior, it is just a set of ontologies, that can be used in any kind of pervasive application to support knowledge sharing.

After the definition of SOUPA, COBRA-ONT was restructured to honour SOUPA, becoming a SOUPA extension for smart meeting rooms [CFJ<sup>+</sup>04d].

### 2.4.3 Conclusion

There are several drawbacks in the CoBrA model for the goals of our research. Its centralised architecture unburden limited-devices from the inconveniences of managing knowledge and information sources, but relies in a core element, the broker, that is practically the sole responsible of implementing the overall functionality.

It is necessary to place a broker in every scenario that needs to be activated with CoBrA, increasing the cost, complexity and maintenance of the deployment process. Moreover, no mechanism is proposed for devices to discover the broker, which is at the core of any Ubiquitous Computing architecture, but in turn, this task is delegated to the JADE API, and so, only suitable for agent (not device) discovery.

CoBrA does not provide any implicit reactivity mechanism or architecture; ad hoc agents must be created and connected to the broker to perform reactive behaviour in every case, checking existing context information and reacting as programmed. Thus, CoBrA is not self-descriptive from this point of view and further programming efforts and adaptations are required to have it configured and ready to run in a particular scenario.

A remarkable feature in CoBrA is its support for security and user's privacy via a policy language such as Rei. Rei was not initially based on OWL, which constituted an inconsistency within the CoBrA model and the current trends in ontologies, but OWL Lite support was added shortly after. However, the policy documents are cumbersome, since they need to be edited manually.

On the other hand, SOUPA is a brilliant effort to create a modular set of common ontologies to be applied in every pervasive computing environment. The word "standard" is probably too ambitious, but the reusability of other existing ontologies pushes SOUPA ahead towards this goal.

An analysis of CoBrA/SOUPA against the evaluation criteria leads to the following results:

**Decentralisation:** CoBrA's core element is the Context Broker which determines the centralised nature of the whole architecture. Low.

**Reasonability:** Semantic Web, agents, user policies and the reasoning module make CoBrA a highly intelligent system, with enough potential to deal with complex situations. High.

**Context-awareness:** agents and adapters provide and share context information via the broker, but devices still remain outside this model;

agents are always required to represent them. CoBrA performs automatic adaptation of the environment via programming of specific agents or adapters. Medium.

**Technological Consistency:** CoBrA mixes up the Semantic Web model with smart agent platforms, thus creating interactions between unrelated technologies such as RDF or OWL and FIPA ACL. HTTP could be used instead of FIPA ACL resulting in a better integration with XML, RDF and OWL. Low.

**Standards Adherence:** standards such as RDF, OWL, FIPA ACL and others are used in CoBrA, as well as SOUPA reuses several existing ontologies. High.

**Device Implementation Cost:** CoBrA defines a sole element, the broker, that embraces all the functionality and responsibilities for the whole architecture. The authors have not reported any experience involving limited devices managing all the complexity attached to it. Moreover, the CoBrA model bases its foundations on representing devices as agents that can reside in other parts of the network, thus not being intended for direct device development. High.

**Environment Deployment Cost:** the need to deploy a central broker at every scenario, performing heavy agent programming, and adding device discovery mechanisms before having it ready for use, makes CoBrA a prohibitive solution for emergent pervasive deployment everywhere. It is only recommended in concrete cases where high deployment costs can be assumed in a predefined scenario, such as a particular meeting room or a smart home. Every existing information source must be adapted (via agents) to connect and feed the broker. The broker has to be deployed and needs previous configuration to discover existing agents. High.

**Lightness:** CoBrA is not intended to be implemented in limited devices. Low.

**Autonomy:** once configured, CoBrA could operate without much user intervention, being the agents responsible of operations execution. High.

## 2.5 Gaia

Since 2001, the research group at the Department of Computer Science of the University of Illinois at Urbana-Champaign, led by Dr. Roy H. Campbell

<b>Criterion</b>	<b>Value</b>
<i>Decentralisation</i>	<i>Low</i>
<i>Reasonability</i>	<i>High</i>
<i>Context-awareness</i>	<i>Medium</i>
<i>Technological Consistency</i>	<i>Low</i>
<i>Standards Adherence</i>	<i>High</i>
<i>Device Implementation Cost</i>	<i>High</i>
<i>Environment Deployment Cost</i>	<i>High</i>
<i>Lightness</i>	<i>Low</i>
<i>Autonomy</i>	<i>High</i>

Table 2.4: Analysis of CoBrA/SOUPA against the evaluation criteria.

has been working on the design of an infrastructure to support intelligent Ubiquitous Computing environments.

The Gaia project is the result of these efforts, constituting a software infrastructure to support Active Spaces. An active space is a “*model that maps the abstract perception of a physical space as a computing system, into a first class software entity*” [RC00].

Thus, the active space acts as a mapping between the real and virtual space, connecting both in such a way that real world actions affect virtual world objects and vice versa. The active space hides the complexity of the real world elements into one unique entity that provides functions for manipulating the space, discovering and locating internal entities, storing and retrieving information from the space and so forth.

In this way, Gaia relieves the burden of manipulating highly object-populated environments by providing a single consistent interface instead of a bulk of multiple ones [RHC<sup>+</sup>02a], simplifying application programming and user interaction.

The name Gaia was adopted from the Gaia theory by James Lovelock that advocated for the earth as a self-regulated superorganism, who in turn borrowed it from the Greek Earth Goddess. The Gaia project tried to replicate the same global awareness and self-regulated behaviour for smart environments and their constituent elements.

### 2.5.1 Gaia architecture

The Gaia Operating System (Gaia OS) is the core element of the whole architecture. It is defined as a meta-operating system, running at the top of others, providing a distributed communication model for coordinating active spaces [RHC<sup>+</sup>02b].

Gaia is composed of three main components as shown in Figure 2.14:

- **Gaia kernel:** provides basic services such as component life-cycle management or remote component execution and management. Gaia relies on CORBA as underlying distributed component model, and extends some CORBA services to provide the so-called Gaia services, such as Event service, Context service, Presence Service, Space repository and Context file system.
- **Application framework:** consists of a distributed component-based infrastructure following the MVC model, including new functionality to manipulate component bindings, a mapping mechanism and policies/rules for application customisation.
- **Active Space Applications:** implement the desired functional behaviour in the active space, such as the Presentation manager application, that lets users present slides in multiple displays simultaneously, move slides from one display to another, as well as move the input device functionality.

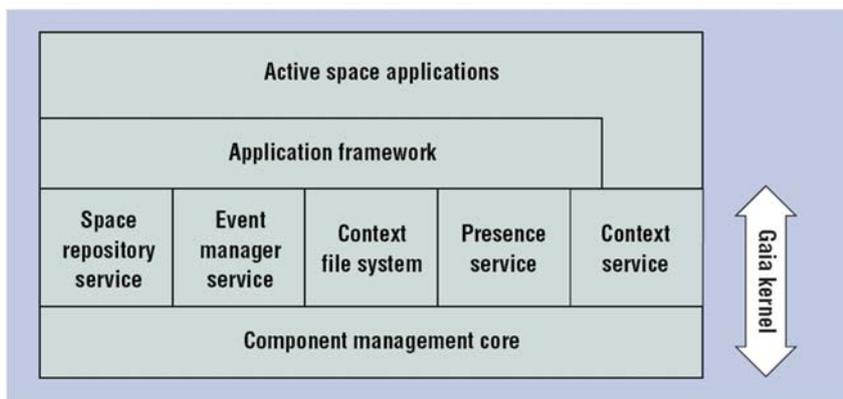


Figure 2.14: Gaia architecture. Source: [RHC<sup>+</sup>02b].

The mapping mechanism of the Application framework offers the possibility of describing requirements to find the suitable real device to assign a functional behaviour (for example, audio output) so that matching devices can be found within the active space to perform that function during a task.

Specially interesting is how Gaia represented context in the form of a 4-components structure: `Context(<ContextType>, <Subject>, <Relater>, <Object>)` that in many ways resembles that of the Semantic Web, for example: `Context(temperature, roomlab21, is, 24 C)`.

Later, this model evolved into a predicate-based representation of context information:

- `Location(inaki, entering, roomlab21)`
- `Temperature(roomlab21, '=', 24 C)`
- `TVStatus(smallTV, is, off)`

During 2003, Gaia was extended with a semantic middleware layer for context awareness endorsing existing Semantic Web technologies in order to model and annotate context information, perform reasoning and carry out reactive behaviour in response to context changes [RC03b]. DAML+OIL (later OWL) was selected to represent the context information following a predicate model [MRCM03b] [MRMC03].

In order to map the predicates onto the ontology, an ontology class is created for each predicate structure [RC03a]. So, the `Location` predicate becomes a `Location` ontology class with three possible relationships to denote the information that was previously enclosed in the predicate parameters [MRCM03b].

Representing the context in this way, operations such as search, querying, fusion and so forth, become possible.

There are several different entities involved in Gaia's context information infrastructure depicted in Figure 2.15:

- **Context Providers:** they are sources of context information, probably obtained by sensors.
- **Context Synthesisers:** they retrieve context information from different providers and perform some form of reasoning to elicit new information making it available to other agents. Both static rules and machine learning techniques (such as Naïve Bayes) can be applied to obtain new information.
- **Context Consumers:** they gather context information from providers and synthesisers, reason about it and perform reactive behaviour accordingly.
- **Context Provider Lookup Service:** one per environment, it is used by context providers to publish the kind of context information they provide in order to be found by context consumers.
- **Context History Service:** one per environment, it contains database records of past context information to make them available to requesting parties.

- **Ontology Server:** one per environment, stores ontologies for the different information types.

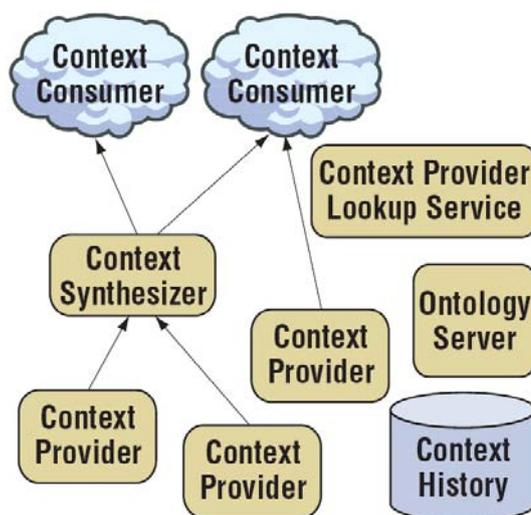


Figure 2.15: Gaia Context Infrastructure. Source: [RAMC04].

It is noteworthy how this architecture requires three elements to be provided by the environment such as the Context Provider Lookup Service, the Context History Service and the Ontology Server. These requirements make harder and more cumbersome the deployment process.

### 2.5.2 Semantic Knowledge in Gaia

Basically, Gaia uses Semantic Web technologies to address three different issues [MRM03] [MRM03b]:

- **Discovery and Matchmaking:** the process of discovering suitable entities that provide the requested service in the environment meeting some previously defined requirements. When applying Semantic Web technologies during the discovery phase the authors use the terms “semantic discovery” and “semantic queries” [MRM03a].
- **Inter-operability between different entities:** the problem of creating a shared model of concepts, interfaces, commands, relationships, and so forth, in order to achieve a common interaction framework by the environmental entities.
- **Context-awareness:** the need for entities in a Ubiquitous Computing environment to share a common knowledge model of the environment

in order to characterise the situations and react rapidly. The reactive behaviour can be programmed by means of rules on existing context information.

Basically, ontologies are used in Gaia to create a taxonomy of the different entity types available in the system, so that they can be managed in a formal way.

Although the issue is not further explored, the term “bridge concept” defines the use of dynamic relationships to characterise new device types that appear in the environment, in relation with concepts and definitions already present.

For instance, a “fingerprint recogniser” could be identified as a subclass of “authentication device” via a bridge concept, without providing the relationship explicitly. However, this possibility was not implemented in Gaia as the problem of dynamically creating bridge concepts was not addressed.

Gaia divides the types of context into several groups:

- Physical contexts, such as location and time.
- Environmental contexts, such as weather, light or sound level.
- Informational contexts, such as stock quotes or sports score.
- Personal contexts, such as health, mood, schedule or activity.
- Social contexts, such as group activity or social relationships (the FOAF ontology [DZPJ05] can be categorised here).
- Application contexts, such as email or web bookmarks.
- System contexts, such as network traffic or printer status.

The role of this classification in the architecture is not clear, and it seems to be used purely for reader-understanding purposes.

A later introduction in the Gaia model was attaching confidence values to context information in order to perform reasoning with a certain degree of uncertainty [RAMC04], which is required in many situations.

The confidence values are expressed using the same predicate constructions applied in the model, for example the following expression `prob ( location ( inaki, in, lab21 ) ) = 0.5` means that the probability that Inaki is in Lab21 is 0.5. Probabilistic logic, fuzzy logic and Bayesian networks can be applied in this way to generate confidence values.

Context producers and synthesisers are in charge of associating probabilities to the generated context information, using any or a combination of the above mechanisms.

The model has proven validity at determining the location of people wearing RFID badges based on disperse probabilistic measures, which has a lot to do with the concept of sensor fusion.

Another interesting application is the detection of system failures with a certain confidence value. For instance, if a Context Provider does not send any message over one hour, maybe is due to a communications failure.

Other additions such as the creation of a specific Ubiquitous Computing oriented programming language called Olympus [RCAM<sup>+</sup>05] and the shift towards autonomic computing [Ran05] were finally added to the Gaia architecture.

### 2.5.3 Conclusion

Gaia fulfils many of the requirements established for a smart Ubiquitous Computing architecture, mainly those related to intelligence support. Gaia makes use of context predicates for representing context information and OWL ontologies for taxonomical purposes.

A rule-based reactivity model is provided and the latest efforts on adding probabilistic measures and confidence values to context predicates boost up the built-in intelligent capabilities of the system to even higher degrees.

However, Gaia has two important drawbacks and several inconsistencies.

The first main drawback is that Gaia requires three different elements in the architecture to be previously deployed and properly configured in the environment: the Context Provider Lookup Service, the Context History Service and the Ontology Server. This constraint prevents Gaia from creating spontaneous emergent pervasive computing environments anywhere, since a deployment phase must be performed beforehand, enforcing an undesirable centralisation.

The second main disadvantage arises from the fact that core elements in the architecture, such as those three mentioned above, seem to be suitable for installation in desktop computers or servers, but not in embedded computers. Since true Ambient Intelligence invisibility can only be achieved by embedding computers everywhere, it is difficult that Gaia could be considered appropriate to fulfil these requirements.

The main inconsistencies with Gaia are originated from the initial selection of technologies and the subsequent integration of newer ones, that result in a strange mixture.

For instance, representation of context information through predicates was present at the very initial stages. When OWL ontologies were integrated in Gaia, context predicates remained as knowledge representation mechanisms instead of shifting to RDF, which would have sound more sensible.

Another strange mixture is related to the communication model. Gaia uses CORBA as distributed computing architecture, instead of the web model. This would not have constituted a problem per se, if no web technologies were used at all. However, OWL is applied and DQL (DAML Query Language) [FHH03] is reported to be used in the future [MRCM03a] (moreover, since DQL works on RDF graphs that are not used in Gaia, its application becomes controversial). A web-based communication model would enforce integration between these technologies.

An analysis of Gaia against the evaluation criteria leads to the following results:

**Decentralisation:** the requirement about a previous three-elements deployment in the environment contributes to a centralised architecture in Gaia. Low.

**Reasonability:** the application of ontologies, rules, probabilistic logic, fuzzy logic and Bayesian networks in order to perform all sorts of reasoning about context information, promotes a very high degree of intelligence in Gaia. Very High.

**Context-awareness:** the integration of rules and confidence values contribute to situation identification and subsequent adaptation. High.

**Technological Consistency:** technological evolution during the Gaia project, resulting in changes as new technologies were adopted, has created a strange mixture in the final outcome. OWL is combined with predicate logics, instead of shifting to RDF. CORBA is applied as distributed computing architecture instead of newer more lightweight web-based communication models that would make the Semantic Web fit better. While the result undoubtedly works, lots of different APIs and different technologies are used. Low.

**Standards Adherence:** standards and recommendations such as CORBA and OWL are honoured. High.

**Device Implementation Cost:** Gaia is not intended for being implemented on resource-limited devices, but for integrating exiting appliances by means of adapters. While Context Providers could be implemented

in resource-limited devices, nothing is reported concerning Context Synthesisers or Consumers. High.

**Environment Deployment Cost:** the need to deploy the above mentioned three elements in every scenario makes Gaia a prohibitive solution for pervasive deployment everywhere. It is only recommended in concrete cases where high deployment costs can be assumed in a concrete scenario. There is a need for a previous development phase for integrating context providers and consumers with the real devices, sensors and effectors. Moreover, the other elements in the architecture need also to be properly configured for the environment. High.

**Lightness:** some elements in the architecture are difficult to migrate to resource-constrained devices, such as the Context History Service or the Ontology Server. The Gaia architecture does not seem to be intended for deployment in embedded devices, but in full computers. Low.

**Autonomy:** once configured, Gaia's reasoning and context-awareness mechanisms can contribute to a user-free operation, taking decisions autonomously. High.

<i>Criterion</i>	<i>Value</i>
<i>Decentralisation</i>	<i>Low</i>
<i>Reasonability</i>	<i>Very High</i>
<i>Context-awareness</i>	<i>High</i>
<i>Technological Consistency</i>	<i>Low</i>
<i>Standards Adherence</i>	<i>High</i>
<i>Device Implementation Cost</i>	<i>High</i>
<i>Environment Deployment Cost</i>	<i>High</i>
<i>Lightness</i>	<i>Low</i>
<i>Autonomy</i>	<i>High</i>

Table 2.5: Analysis of Gaia against the evaluation criteria.

## 2.6 Semantic Spaces, SOCAM and CONON

The initiative Semantic Space, SOCAM (Service Oriented Context-Aware Middleware) and CONON (CONtext ONtology) are all efforts to create a smart environment infrastructure using Semantic Web technologies, developed by the National University of Singapore and the Institute for Infocomm Research, Singapore.

These projects share many similarities both in goals and architectural styles with CoBrA and SOUPA. In fact, SOCAM and CoBrA are very similar, as well as CONON and SOUPA are both alternative ontologies for pervasive environments.

The Semantic Space [WDC<sup>+</sup>04] [TZWC05] seems to be the precursor of SOCAM, since certain details are not deeply analysed and only two papers have been published. It introduces a context architecture, very similar to CoBrA, where UPnP is used for device interoperability and an ontology called ULCO (Upper Level Context Ontology) is designed in order to represent smart spaces context information.

The SOCAM architecture seems to be a fork of the Semantic Space but is more thoroughly described. It is divided into five different components [GPZ04b] [GPZ04c] [GPZ05], as illustrated in Figure 2.16:

- Context Providers: provide context information from different sources.
- Context Interpreter: consists on a reasoning engine and a knowledge base.
- Context Database: stores context ontologies and past contexts for a concrete scenario.
- Context-aware Applications/Services: adapt the way they behave depending on the current context.
- Service Locating Service: provides a mechanism for the Context Providers and Interpreter advertise their presence. The locating architecture is formed by a hierarchical tree of servers.

In SOCAM, context is categorised into direct or indirect context, depending whether was directly obtained from context providers by sensors or was obtained from a reasoning process. These characterisation of context enables SOCAM to assign confidence levels to context information [GWPZ04]. An extension was proposed to deal with uncertainty using Bayesian networks, and assigning probability values [GPZ04a], but the mechanism proposed, creating two new ontology classes, does not allow to naturally annotate already existent context information in the form of RDF triples, requiring explicit generation of probabilistic structures. The mechanism was not further explored in subsequent papers.

In SOCAM, Semantic Space's ULCO seems to have evolved into CONON (CONtext ONtology), which defines the abstract ontology class `ContextEntity` from which all others, such as `Location`, `Person` or `Activity`

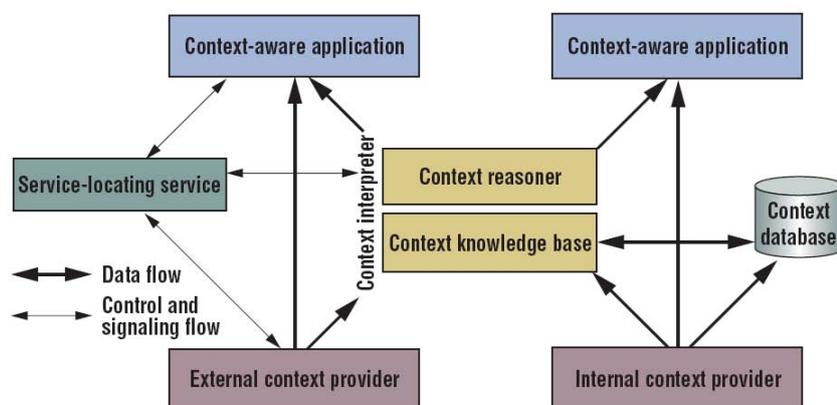


Figure 2.16: SOCAM architecture. Source: [GPZ04c].

derive. The use of such wide concepts can probably enter into conflict with other with existing ontologies (such as SOUPA or FOAF). Moreover, in CONON, these concepts cannot be used directly, but they need to be parametrised and extended for every concrete environment, thus leading to the creation of specific ontologies [WZGP04] as illustrated in Figure 2.17.

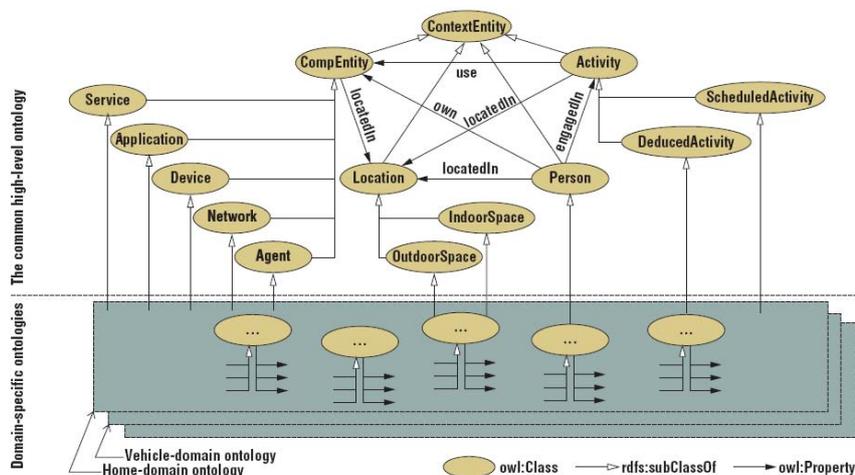


Figure 2.17: CONON ontology. Source: [GPZ04c].

As a differentiation factor, SOCAM uses OSGi and not UPnP directly as distributed computing architecture for devices [GPZ04c]. This decision seems appropriate for dealing with platform interoperability issues. SOCAM software is implemented in Java and uses JavaRMI for communication among components, including the transport of RDF context information, which seems somehow inappropriate having other suitable web technologies such as HTTP or SOAP.

Currently, the authors' interest has shifted to the possibility of using P2P overlay networks for searching context information [GTPZ05a] [GTPZ05b].

### 2.6.1 Conclusion

Again, Semantic Space and SOCAM are centralised Ubiquitous Computing architectures accompanied with an upper level ontology for description of abstract concepts, that needs to be extended and completed for every specific scenario. This latter requirement imposes an additional design time during the environment preparation process.

Their centralised nature is clearly illustrated by the deployment requirements of the constituent components: the Context Interpreter, the Context Database and the Service Location Service need to be deployed previously for any particular scenario.

SOCAM exhibits a clear service-oriented reactivity regarding its reactive behaviour, and it is based on JavaRMI, which is not an appropriate choice for transporting RDF information.

An analysis of SOCAM/CONON against the evaluation criteria leads to the following results:

**Decentralisation:** SOCAM features a centralised architecture derived from the requirement about the Context Interpreter, Context Database and Service Location Service. Low.

**Reasonability:** the application of ontologies and rules leverages SOCAM to an average degree of reasonability. High.

**Context-awareness:** SOCAM features a high degree of context-awareness promoting reactivity in the form of JavaRMI invocations to Context-aware Applications/Services. High.

**Technological Consistency:** the transport of RDF over JavaRMI packets instead of applying HTTP-based communication is not very natural and does not provide any advantage over the latter. It is probably reminiscent of a former JavaRMI oriented design decision. Integration with OSGi is an advantage. Medium.

**Standards Adherence:** although communication standards are widely used, ontology reusing is not promoted. Medium.

**Device Implementation Cost:** centralised components are not intended to be implemented on limited devices. Context Providers and Context-aware Applications/Services could be hosted in appliances

but JavaRMI support must be provided, which narrows the possible platform choices. High.

**Environment Deployment Cost:** the need to deploy the centralised elements in every scenario makes SOCAM a prohibitive solution for pervasive deployment everywhere. As other alternatives, it is only recommended in concrete cases where high deployment costs can be assumed. Specific scenario ontologies must be created and OSGi integration must be carried out. High.

**Lightness:** the central elements in the architecture seem difficult to migrate to resource-constrained devices, such as the Context Database or the Service Location Service. Only Context Providers seem suitable and enough lightweight for embedding purposes. Low.

**Autonomy:** once properly configured for the particular scenario, SOCAM could operate without much user intervention. High.

<i>Criterion</i>	<i>Value</i>
<i>Decentralisation</i>	<i>Low</i>
<i>Reasonability</i>	<i>High</i>
<i>Context-awareness</i>	<i>High</i>
<i>Technological Consistency</i>	<i>Medium</i>
<i>Standards Adherence</i>	<i>Medium</i>
<i>Device Implementation Cost</i>	<i>High</i>
<i>Environment Deployment Cost</i>	<i>High</i>
<i>Lightness</i>	<i>Low</i>
<i>Autonomy</i>	<i>High</i>

Table 2.6: Analysis of SOCAM against the evaluation criteria.

## 2.7 Other related work

### 2.7.1 Triple Spaces

DERI (Digital Enterprise Research Institute) is one of the leading organizations researching in the Semantic Web field. Their WSMO (Web Service Modeling Ontology) [FRP<sup>+</sup>05] architecture is an alternative to OWL-S for Semantic Web Services [SKB06]. The application of DERI's Semantic Web background into the Ubiquitous Computing arena has led to the concept of Triple Spaces [Fen04].

Triple-space computing is based on the previous concept of tuple-space computing: a shared information repository where processes can read or

write data upon in the form of tuples (ordered collection of typed fields with values).

The authors argue that applying RDF triples instead of tuples would create a shared semantic information space referenced by a URI, suitable for web services decoupled communication. Exchanged messages could be referenced via URIs, becoming full resources naturally, which is not currently supported by existing web services technologies such as SOAP. This mechanism would bring the web services closer to the web model [Fen04] [BKK<sup>+</sup>05]. Other authors have also explored the convergence of Semantic Web technologies and tuple spaces in initiatives such as sTuples - Semantic Tuple Spaces [KLF04]

Triple based computing de-couples different orthogonal dimensions involved in information exchange:

- Reference: objects involved in the communication process do not need to know each other, since they exchange messages via the tuple space.
- Time: communication among participants can be asynchronous, since the tuple space acts as a persistent data repository.
- Space: processes can run anywhere and exchange information as long as they have access to the tuple space.
- Flow: an asynchronous publish/subscribe communication model can be applied [BKK<sup>+</sup>05], so that a participant does not need to block waiting for a response, but it is notified through a callback when the response is available.
- Vocabulary: the use of RDF and ontologies promotes mapping between vocabularies as well as reuse and common understanding of concepts [KKS05].

The publish/subscribe model closely resembles the way the Web works in the sense that publications can be later accessed by interested parties, again achieving decoupling [Bus05].

The possible approach for the implementation of the Triple Spaces architecture consists in a number of clients and servers, the latter acting as information repositories, and using the TSTP (Triple Space Transport Protocol) for communication.

Triple Space Computing is a very recent initiative and no much work has been carried out. There are no designs or further information about how discovery should be performed or how the TSTP looks like.

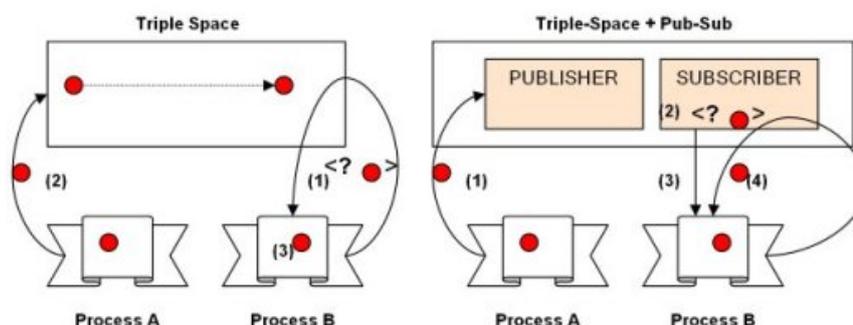


Figure 2.18: Triple Space example with publish/subscribe interaction. Source: [KMRMR06].

DERI envisions Ubiquitous Computing as one of the major application fields for Triple Space Computing [KS05] to create a shared information space populated with RDF triples that can be accessed by existing devices. Future evolution is intended to spread over the fields of Semantic Grid [STK<sup>+</sup>06] and semantic P2P networks [KSF06].

### 2.7.2 The Context Toolkit

One of the main contributions that the authors of the Context Toolkit, Anind K. Dey, Gregory D. Abowd and Daniel Salber, provided to the world of Ubiquitous Computing and context-awareness was precisely a uniform definition of context and context-awareness that has been widely used since then [ADB<sup>+</sup>99] [Dey01].

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

*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task.*

The Context Toolkit [DAS99] [Dey00] is a software package that provides mediation capabilities between the application and its operating environment by using “context widgets” in a very similar way as GUI widgets are used. The widget abstraction features several advantages when dealing with context such as hiding complexity and specifics of the low level sensing mechanisms to the application; they abstract context information in such

a way that only relevant knowledge is passed back to the application; and they provide reusable and customizable building blocks for context sensing.

The Context Toolkit architecture was later complemented with several other components, such as Interpreters, Aggregators, Services and Discoverers [DAS01], some of which were imported by other projects (e.g. Gaia or SOCAM).

The Context Toolkit was designed with a distributed nature in mind, using coherently HTTP and XML for transport and information representation purposes respectively, in order to facilitate communication between the context widgets and the applications [SDA99].

The discovery mechanism provided by the Context Toolkit is a centralised one, which is obviously perceived as a disadvantage by the authors [DAS01], but justified for simplicity purposes.

The Context Toolkit is not intended to and does not provide any form of built-in reasoning mechanism, but a uniform way of communication and distributed architecture to easily deploy context-aware applications.

### 2.7.3 Oxygen

Oxygen [Den02] was probably the largest multi-technological effort about creating smart spaces. Started by Dr. Michael L. Dertouzos in 1999 at the Computer Science and Artificial Intelligence Laboratory (CSAIL) at MIT, its lemma “*doing more by doing less*” [Der99] represents the core of their approach: to enhance machine and software intelligence in order to improve user interaction and goals achievement.

Oxygen led to the design of various interrelated technologies, among them:

- Handy 21 (H21): a handheld computer that connects the user with his environment.
- Enviro 21 (E21): embedded engine suitable to be plugged to everyday devices, such as cameras, heating systems, refrigerators, and so forth, in order to perform intelligent control.
- Net 21 (N21): an innovative network design to create a collaborative space between H21s and E21s.
- Cricket: is a low cost indoor location system that uses beacons and listeners to determine user’s position [PCB00] [PMBT01].

- Haystack: a context-aware Semantic Web browser aimed at organising and presenting user's most important information in every moment [QK04] [KQ04].

All these activities and a remarkable number of others related to Oxygen are still pushing Ambient Intelligence technologies forward in different fields.

#### 2.7.4 Other relevant activities

There are some other efforts and technologies related to our research to a much lesser extent, whose contribution has been also analysed. Special mention deserve the GAS ontology [CGK05] [CGZK04] [CK05] [CK04] developed within the eGadgets European IST Project, the work of Strang et al. on context modelling [SLP04] and the CoOl ontology [SLPF03b] [SLPF03a] [SLPF03c], Carnegie Mellon University's Aura [GSSS02] [SG02] [SG03], the work of Kindberg et al. in HP's Cooltown [KB01] [KBM<sup>+</sup>02] [BK01], the work of Issarny et al. in WSAMI [IST<sup>+</sup>05], Microsoft's EasyLiving initiative [SBM00] [BMK<sup>+</sup>00] and the already mentioned visions of Lassila et al. [LA03] [Las06].

## 2.8 Comparative analysis

The Table 2.7 shows the comparative values obtained by the different analysed technologies. The weighted final value for each criteria has been obtained by multiplying *criterion\_value*  $\times$  *criterion\_weight*.

The criterion value is assigned in the following way: None = 0, Low = 1, Medium = 2, High = 3, Very High = 4. Note that the two economical criteria are interpreted as negative figures, thus promoting low values.

In general terms, all the architectures feature a correlation among intelligence and centralisation. That is, the higher level of intelligence in the system, the more centralised it is.

A single element must be previously deployed in the environment, sometimes concentrating the majority or all of the reasoning processes, sometimes providing supporting services: the Context Broker in CoBrA; the Context Provider Lookup Service, the Context History Service and the Ontology Server in Gaia; and, the Context Interpreter, the Context Database and the Service Location Service in SOCAM. UPnP sacrifices intelligence for decentralisation, while Task Computing is not suitable for embedded computing.

In terms of intelligence, Gaia ranks better than any other alternative due to the provision of complementary reasoning technologies such as ontologies and fuzzy logic.

It is surprising and remarkable how UPnP ranks better than Task Computing. This can be explained through the fact that the decentralised nature of UPnP supports Ubiquitous Computing better than the intrinsic intelligence of Task Computing, according to our criteria.

In fact, this is the main drawback of existing experiences: with the aim of providing more intelligence, the basic nature of Ubiquitous Computing has been abandoned.

Thus, the experimental initiatives lack of the spontaneous and serendipitous collaboration required in Ambient Intelligence scenarios. Moreover, they are not even suitable for embedded computing platforms, and they also lack of an appropriate advanced discovery mechanism that can provide an appropriate means for creating a dynamic network of cooperative devices.

Guided by the criteria, we faced the challenge of creating a system that maximised the values for each one: a system featuring an appropriate balance between decentralisation, reasonability and feasibility of implementation in embedded platforms, while being powered by an intelligent discovery mechanism.

<b>Criterion</b>	<b>UPnP</b>	<b>Task Computing</b>	<b>CoBrA SOUPA</b>	<b>Gata</b>	<b>Semantic Spaces SOCAM</b>
<b>Decentralisation</b>	Medium(8)	Low(4)	Low(4)	Low(4)	Low(4)
<b>Reasonability</b>	None(0)	Low(4)	High(12)	Very High(16)	High(12)
<b>Context-awareness</b>	None(0)	Low(4)	Medium(8)	High(12)	High(12)
<b>Technological Consistency</b>	High(9)	High(9)	Low(3)	Low(3)	Medium(6)
<b>Standards Adherence</b>	Medium(4)	High(6)	High(6)	High(6)	Medium(4)
<b>Device Implementation Cost</b>	Low(-2)	High(-6)	High(-6)	High(-6)	High(-6)
<b>Environment Deployment Cost</b>	Low(-2)	Medium(-4)	High(-6)	High(-6)	High(-6)
<b>Lightness</b>	High(3)	Low(1)	Low(1)	Low(1)	Low(1)
<b>Autonomy</b>	Low(1)	Low(1)	High(3)	High(3)	High(3)
<b>TOTAL</b>	21	19	25	33	30

Table 2.7: Analysis of architectures against the evaluation criteria.

Chapter  
**3**

# mRDP: A Semantic Discovery Protocol

*“The only real voyage of discovery consists not in seeing new landscapes, but in having new eyes.”*

*Marcel Proust  
French novelist*

**D**ISCOVERY is one of the most important activities in ubiquitous and distributed computing; we have already mentioned some discovery mechanisms that previous initiatives apply. However, we early noticed that we needed a different approach, a lightweight protocol able to cope with semantic requests in a spontaneous network of devices.

In this chapter, we explain the basics for a novel semantic discovery mechanism called mRDP (Multicast Resource Discovery Protocol) and compare it to other existing approaches.

## 3.1 Previous approaches

Discovery has always been a hot topic in Ubiquitous Computing and it will probably keep on being such in the future. Although several different discovery protocols have been proposed and used over the last years in concrete architectures, mainly for device and service discovery, there is no common agreement about a unified discovery protocol.

Edwards [Edw06] defines discovery in Ubiquitous Computing systems as “a mechanism for dynamically referencing a resource on the network”. Since devices and resources in Ubiquitous Computing networks come and go in a highly dynamical basis, spontaneity is one of the major features to be supported during the discovery process.

Continuing this line of thought, McGrath [McG00] describe the most important features of a discovery protocol as:

- “Spontaneous” discovery and configuration of network devices and services.
- Selection of specific types of service.
- Low (preferably no) human administrative requirements.
- Automatically adaptation to mobile and sporadic availability.
- Interoperability across manufacturers and platforms.

The architectures analysed in chapter 2 were all concerned about discovery and included some mechanism for this purpose.

For example, during the development of CoBrA (see section 2.4) different protocols such as SLP [GPVD99], Jini [Sun99], UPnP SSDP [GC<sup>+</sup>99] and Salutation were analysed for discovery purposes [CFJ01] and found unsuitable due to:

- Lack of rich representation: the existing architectures lack of expressive languages, representation and tools for the broad range of service descriptions.
- Lack of constraint specification and inexact matching: most protocols require exact matching, with a simplistic notion of constraints. Lack of semantic matching.
- Lack of ontology support: for representing service descriptions and capabilities.

Finally Jini was selected as the best choice, despite being very platform specific (Java), thus limiting the flexibility and wide deployment of the whole system. Moreover, Jini matching is performed by comparing interface names not functionality, there is no reasoning mechanism and it uses absolute matching (e.g.: printer name and model, instead of “the nearest printer”).

In order to improve Jini's performance, CoBrA used the Ronin Agent Framework featuring additional mechanisms to augment its discovery capabilities.

Gaia also addressed the importance of discovery and the suitability of Semantic Web technologies to help here [MRCM03a] [RCAM<sup>+</sup>05]. Gaia employed a first approach to use Semantic Web technologies during discovery, by using class types that drive the search process, since classes represent service types from the authors' point of view.

There have been also novel approaches, such as Cooltown [KB01] which used "URL sensing" for discovery, based on IR/RF, barcodes, electronic tags or optical recognition. López de Ipiña carried out a similar approach in EMI<sup>2</sup> [LVG<sup>+</sup>06] by encoding tiny URLs in TRIP tags [LMH02] for device identification. Other Ubiquitous Computing projects have traditionally used protocols such as Multicast DNS (mDNS) [CK06b], DNS Service Discovery (DNS-SD) [CK06a] and UPnP SSDP [GC<sup>+</sup>99].

In all the cases, the activity that embodies the essence of the discovery process is matchmaking: how the query is resolved to identify suitable candidates.

Virtually all existing discovery protocols base the matchmaking in the device or service type, the ID or some concrete attribute values. This approach performs relatively well for simple systems, but as McGrath points out [McG05] "*keyword or string matching is relatively easy and efficient to implement [...] String matching performs well in limited cases: essentially when the vocabulary is controlled*".

Simple discovery protocols provide basic attribute-value matching mechanisms, where service type or ID can be also considered attributes. The expressive capability of such systems is thus restricted at a syntactic level.

The need for more intelligent discovery capabilities is again stressed in [ZMN05]: "*service discovery protocols must work in unfamiliar computing environments to achieve the goal of computing anytime and anywhere. Service discovery design for such environments is more challenging [...]. This means that service discovery protocols and the underlying computing infrastructure must have more intelligence*".

As described in subsection 1.3.1, Semantic Web technologies overcome the above limitations by describing information at a higher level, and it is specially suitable for unknown problems [Las06]. Since no Semantic Web-based discovery mechanisms for Ubiquitous Computing exist so far (see references in subsection 1.3.1), we assumed the task of designing a basic semantic discovery protocol suitable for our model.

## 3.2 Introduction to mRDP – Multicast Resource Discovery Protocol

Both as an important part of the SoaM architecture and as a broader goal itself we accomplished the design of a semantic-powered discovery protocol with the following requirements:

- In order to achieve spontaneity, no central registration server must be used. Thus, multicast communication will be applied.
- Since some devices can feature limited processing capabilities, two modes of operation must be provided: full semantic-powered operation, and basic operation.
- It must feature a highly expressive language for the queries.
- It should be able not only to provide searching capabilities about resources, but also to obtain URLs where more information about them can be downloaded.
- It must be based on the TCP/IP stack with the highest possible integration with HTTP (thus, taking advantage of HTTP security mechanisms).

The resulting design is mRDP – Multicast Resource Discovery Protocol, a UDP/HTTP based protocol for semantic powered searches in Ubiquitous Computing scenarios.

mRDP provides two different functions:

- Resource identification: search the network for resources meeting particular conditions. For example, “find all the devices located in `urn:uuid:room21`”.
- Resource description location: search the network for sources of information about a particular resource. For example, “where can I obtain information about `urn:uuid:tv1?`”.

While resource identification has already been explained, description location is a new feature that enables an mRDP client to locate information providers about a particular resource, given its URI.

For example, an mRDP client may want to know all the information related to `http://people.com/bobby` in the network, or maybe all the data related to `urn:uuid:tv1` stored by any entity. The mRDP client would issue

a location request about the URI, and every mRDP server containing some piece of information about that resource would reply with a URL where that information can be downloaded.

Attaching just the URL is more efficient than directly sending the whole volume of data in the reply for several reasons:

- It generates less traffic, since packets are significantly smaller than conveying all the information.
- The decision about eventually download the data is left to the client, which carries out the download if needed (maybe the required information has already been downloaded from other source, so there is not need to retrieve it again).
- The URL can be reused several times if the client needs to poll periodically the data about the resource, without searching the network again.

The two functions of mRDP, despite independent, can be coordinated in a complementary process: the mRDP client obtains the identification of resources meeting some conditions, and afterwards locates the information sources where more data about those resources can be downloaded.

### 3.2.1 Operation

The mRDP architecture declares two types of agents: mRDP clients and mRDP servers. Generally, both agents coexist within the same entity, querying and providing information, promoting a P2P architecture in the network.

During resource identification, (1) an mRDP client disseminates the query through the network about a particular resource matching some concrete conditions. Every mRDP server receives the query, processes it against its information model, that is, the RDF graph, and (2) returns the results with the URIs of the resources matching the query back to the client.

During resource description location, (3) an mRDP client disseminates a request about the resource throughout the network. Every mRDP server checks whether there is information available about that resource in its RDF graph and, if so, (4) it returns one or more URLs to the client, where data about the resource can be downloaded.

The interactions illustrating both functions are depicted in Figure 3.1. The client disseminates a query  $q$  about a resource to the network, where four mRDP servers are available, each one with an RDF graph representing

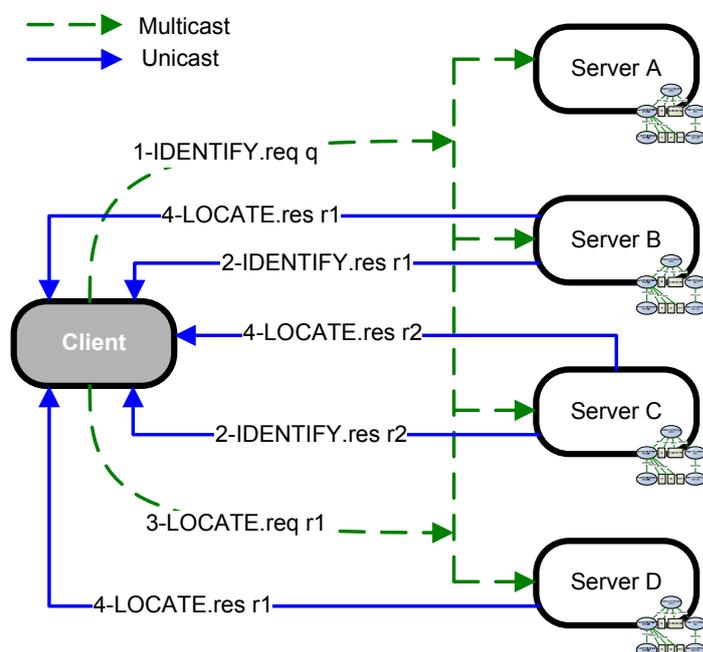


Figure 3.1: Example of mRDP operation.

the managed information. Two of them, B and C, can resolve the query successfully and provide the identification of two resources satisfying the query: B provides  $r_1$  and C provides  $r_2$ . Next, the client disseminates a request to retrieve all the available information about  $r_1$  in the network. Not only B and C reply providing at least one URL where all their stored information about  $r_1$  and  $r_2$  can be downloaded respectively, but D also replies, since it stores some pieces of information concerning  $r_1$  (although those data were not enough to match the query  $q$ ).

In order to save traffic and time, mRDP servers can also provide resource description location URLs in the response to the identification message (2), so the client does not need to perform a second interaction except in the case more information from other sources is required.

While client queries are always multicast, so they can use UDP as the transport protocol, the server replies are always unicast. These replies are not limited in size, and in fact, they can be very large, for example in a scenario where an mRDP server stores information about dozens of users, and the mRDP client disseminates a request to identify resources of type “user”.

Queries can be conveyed in UDP messages since they are small, but replies could be larger and require fragmentation or “chunking” if embodied in UDP datagrams. Moreover, in order to cope with network reliability, the client can issue several copies of the query, but if servers have to deal with UDP unreliability in every response, the complexity of the protocol increases unnecessarily.

Moreover, while queries must be readable for any server in order to check whether they can provide the desired information, responses could require a higher level of privacy and security. Attaching the appropriate security mechanisms to the resulting protocol would even create a more cumbersome communication scheme.

Fortunately there is another solution to the problem, backed up by existing and well-proven techniques and better integrated into our model: the client could attach a callback HTTP URI to the request, so that every mRDP server can send the response to that endpoint, using traditional HTTP communication, augmented with authentication mechanisms (basic or digest), or even HTTPS.

Some advantages of sending the response via HTTP back to the client are:

- Since HTTP is intended to be used in other parts of the SoaM architecture, reuse of libraries and minimisation of platform size is achieved by this strategy.
- HTTP Basic or Digest Authentication as well as HTTPS can be easily reused and implemented, thus taking advantage of existing standards.
- Since HTTP uses TCP as transport layer, reliability is intrinsically provided and messages are not limited in size.

Therefore, the mRDP communication architecture will use UDP multicast messages for sending the requests and HTTP callbacks for receiving the responses as illustrated in Figure 3.2.

An UML sequence diagram of the communication process is depicted in Figure 3.3. The steps involved are:

1. A client application requests the mRDP client module to issue a request to the network conveying a certain query to find matching resources. The mRDP server module on the server side receives the query and processes it against the internal information model. The response is generated and sent back to the client in an HTTP POST request, where the HTTP server module extracts the data and passes them to the client application.

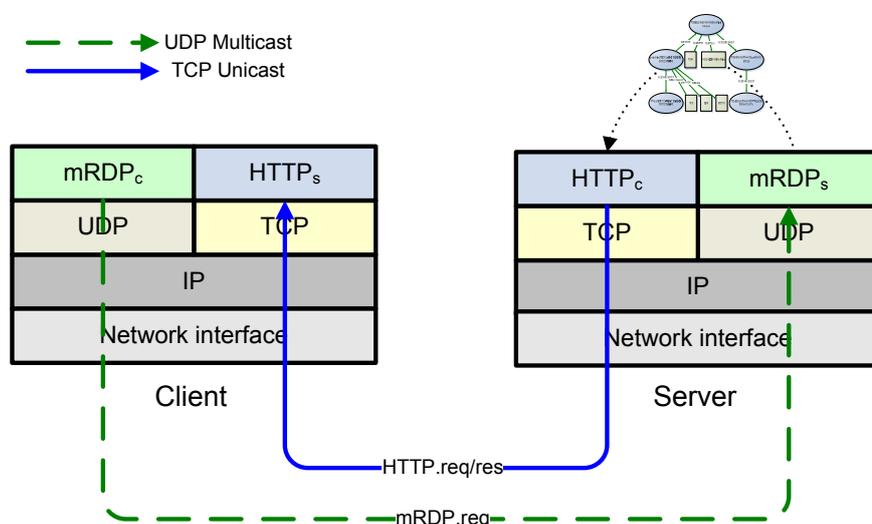


Figure 3.2: mRDP over the TCP/IP protocol stack.

2. If the client needs to find more information for a concrete resource, it requests the mRDP client module to issue a description location request to the network. Receiving servers managing information about the queried resource generate an HTTP POST callback targeted at the client HTTP server, containing their references for accessing resource data. The client HTTP server, in turn, passes the data to the client application.
3. Finally, the client application may use the references to retrieve the desired resource information from the HTTP server at the server, via a simple HTTP GET operation.

The step 2 can be avoided if the response to the identification request in step 1 already contains references provided by the server to download resource information, thus generating the sequence depicted in Figure 3.4.

Both mechanisms are complementary, since the server can include its references for retrieving resource information, while the client can issue an mRDP description location request to the network for obtaining more references from other servers that also manage information about the queried resource.

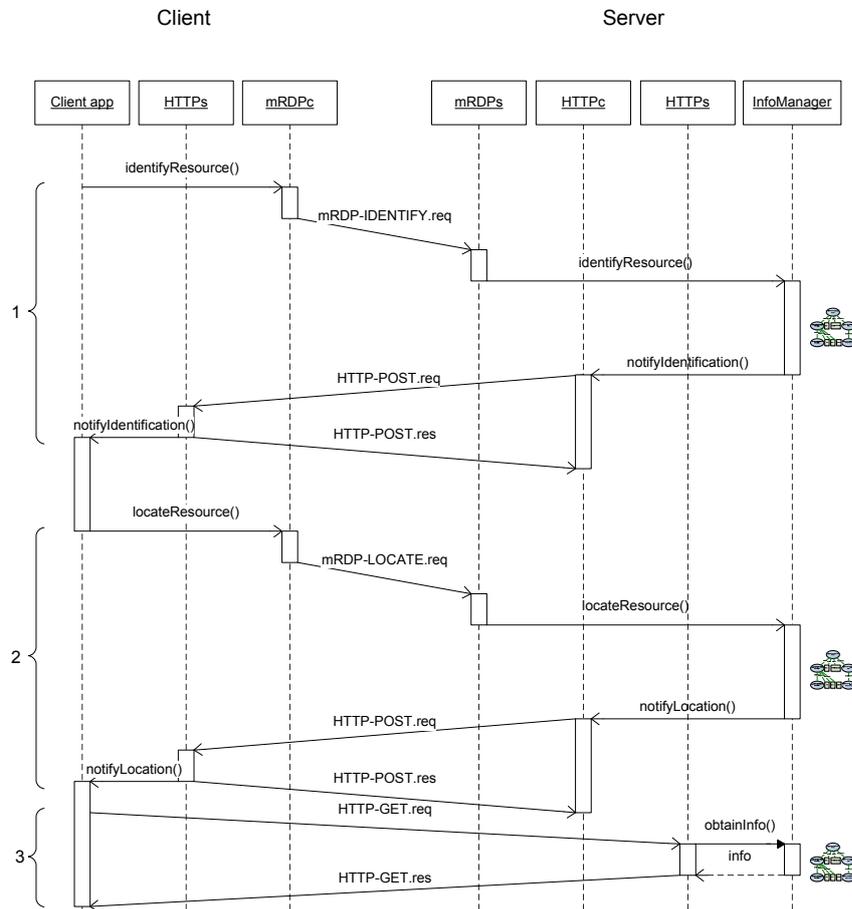


Figure 3.3: mRDP UML sequence diagram with all the interactions.

### 3.2.2 Resource identification

The purposes of resource identification request messages is to convey some kind of query, so that resources matching the query are identified. Different query languages can be used for this purpose, but SPARQL [Wor06b] is probably the best positioned candidate: SPARQL syntax provides rich levels of expressiveness for constructing conditions about resources in a RDF graph.

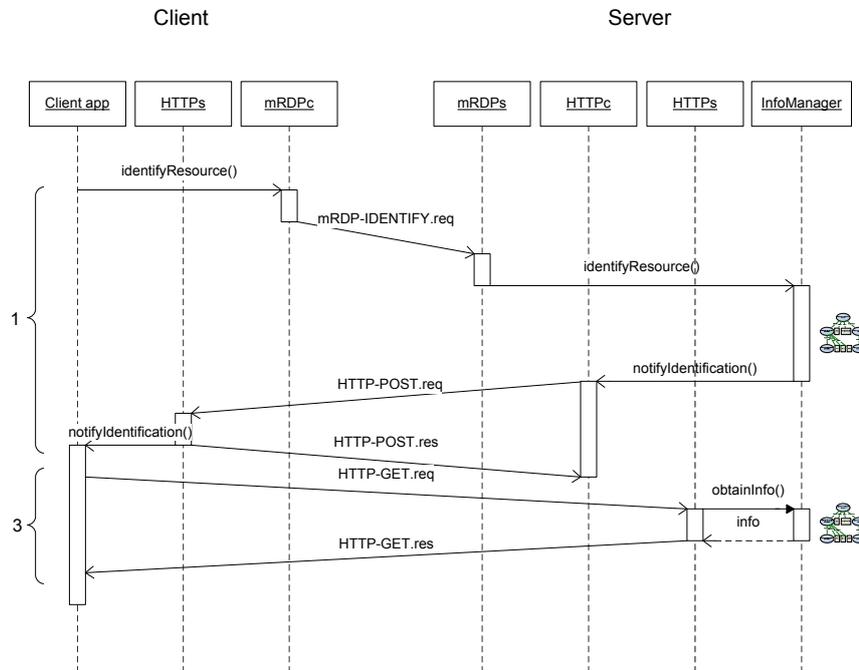


Figure 3.4: Optimised mRDP UML sequence diagram.

Obviously the main problem with SPARQL is that it requires a SPARQL engine in every mRDP server, which is clearly not possible in limited devices. Moreover, SPARQL provides a lot of additional constructions and keywords that are neither required nor useful for resource identification.

A simpler alternative to SPARQL must be provided, such that limited devices, maybe storing the information using an attribute-value scheme instead of an RDF graph, can interpret and process the queries, and generate replies when acting as mRDP servers.

### 3.3 Plant: Pattern Language for N-Triples

The most basic standardised RDF notation available is N-Triples [Wor04g]. N-Triples features a line-based, plain-text format and very simple grammar, a subset of Notation 3 [Ber06], for encoding an RDF graph.

An example document representing RDF information in N-Triples format is<sup>1</sup>:

```
<urn:uuid:tv1> <http://www.awareit.com/onto/2005/12/tv#channel> <http://www.bbc.co.uk/bbctwo> .
<urn:uuid:light1> <http://www.awareit.com/onto/2005/12/light#luminance> "4" .
```

No prefixes but absolute URIs must be used in N-Triples for identifying resources. Basically the N-Triples notation is the direct serialisation of every triple in the RDF graph. N-Triples allows typed literals and blank nodes.

N-Triples's simplicity makes it the best candidate for limited devices when dealing with RDF information.

However, there is an important drawback in N-Triples notation for our purpose: it can only express concrete RDF triples, neither patterns nor conditions. In order to support these kind of constructions, the N-Triples syntax must be extended to be able to produce expressions such as:

Listing 3.1: Example of triple patterns based on N-Triples.

```
1 ?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
  awareit.com/onto/2005/12/tv#TV>.
2 ?r <http://www.awareit.com/onto/2005/12/tv#channel> <http://www.bbc.co
  .uk/bbctwo>.
```

The above expression can be now considered a query to identify a resource `?r` meeting two statement patterns (conditions): `?r`'s type is TV and its current configured channel is BBC2.

In order to be able to construct this kind of expressions with variables, representing RDF triple patterns, N-Triples grammar must be extended. Therefore, we modified the `subject` and `object` productions that were originally defined in the N-Triples specification [Wor04g] as:

```
subject ::= uri-ref | nodeID
object  ::= uri-ref | nodeID | literal
```

To support variable constructs via a new `variable` production.

```
subject ::= uri-ref | nodeID | variable
object  ::= uri-ref | nodeID | literal | variable

variable ::= '?' name
```

<sup>1</sup>Apparent line breaks in the code are due to limitations in page width.

Now, triple patterns such as those illustrated in Listing 3.1 can be supported by this simple extension we called Plant (Pattern Language for N-Triples).

Very much as N-Triples represents a subset of Notation 3 expressiveness, Plant represents a subset of SPARQL expressiveness. In fact, the example of Listing 3.1 can be translated into SPARQL as:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX tv: <http://www.awareit.com/onto/2005/12/tv#>
SELECT ?r
WHERE { ?r rdf:type tv:TV .
        ?r tv:channel <http://www.bbc.co.uk/bbctwo> .
      }
```

However, Plant's simplicity makes it suitable for being processed by resource-limited devices without much computing requirements. In fact even simple devices, not using RDF graphs for knowledge information representation but attribute-value collections with URI namespaces for naming the attributes, can easily process Plant queries.

If the receiving server is able to process SPARQL queries, it can transform the Plant query into a SPARQL representation as illustrated above. If the server does not feature a SPARQL engine or lacks semantic processing capabilities, the following Plant Query Resolution Algorithm can be applied.

### 3.3.1 The Plant Query Resolution Algorithm

The Plant Query Resolution Algorithm is a very simple and straightforward variables unification algorithm optimised for dealing with RDF information and resolve Plant queries, although other query syntax could be used.

The algorithm is structured in the following steps:

1. Optionally, map the attribute-value pairs into RDF triples (if the information is not already in RDF form).
2. Identify all the variables in the whole set of Plant patterns.
3. For every Plant pattern, select the triples in the information model that meet such pattern and annotate the combination of valid values for the variables provided by the triple (\* for any value).
4. Substitute \* by the available values of the involved variable in combinations present in other Plant patterns.

5. Identify the combinations of values that match all the patterns: these combinations represent the solutions for the variables. If no combination is identified, the query cannot be resolved.

For example, the query “identify all the devices whose user is a woman” can be translated into Plant as three patterns:

Listing 3.2: An example Plant query.

```

1 ?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://pervasive
    .semanticweb.org/ont/2004/06/device#Device> .
2 ?r <http://pervasive.semanticweb.org/ont/2004/06/device#hasUser> ?u .
3 ?u <http://pervasive.semanticweb.org/ont/2004/06/person#gender> <http
    ://pervasive.semanticweb.org/ont/2004/06/person#Female> .
    
```

The resolution process is straightforward:

1. **Map the attribute-value pairs into RDF triples.** . Appropriate domain ontologies must be selected. Generally the attribute maps onto the predicate, the value onto the object, and the subject is the device itself. For example, the following attribute-value pairs in `urn:uuid:tv1`:

```

1 state = on
2 volume = 5
    
```

Can be mapped in RDF triples as:

```

1 <urn:uuid:tv1> tv:state <http://www.awareit.com/onto/2005/12/devices#
    stateOn> .
2 <urn:uuid:tv1> tv:volume "5" .
    
```

The current example is more complex since the device manages information about other entities. This information can be finally mapped onto RDF as:

```

<urn:uuid:hifil> rdf:type dev:Device .
<urn:uuid:tv1> rdf:type dev:Device .
<urn:uuid:light1> rdf:type dev:Device .
<urn:uuid:hifil> dev:hasUser <http://people.com/alice> .
<urn:uuid:tv1> dev:hasUser <http://people.com/bobby> .
<urn:uuid:tv1> dev:hasUser <http://people.com/chris> .
<http://people.com/alice> per:gender per:Female .
<http://people.com/bobby> per:gender per:Male .
<http://people.com/chris> per:gender per:Female .
    
```

**2. Identify all the variables in the whole set of Plant patterns.** The variables are  $?r$  and  $?u$ .

$?r$	$?u$

**3. For every Plant pattern, select the triples in the information model that meet such pattern and annotate the combination of valid values for the variables provided by the triple (\* for any value).** If patterns are referred to as Pattern 1, Pattern 2 and Pattern 3:

**Pattern 1:**

```
<urn:uuid:hifil> rdf:type dev:Device .
<urn:uuid:tv1> rdf:type dev:Device .
<urn:uuid:light1> rdf:type dev:Device .
```

$?r$	$?u$
$\langle \text{urn:uuid:hifil} \rangle$	*
$\langle \text{urn:uuid:tv1} \rangle$	*
$\langle \text{urn:uuid:light1} \rangle$	*

**Pattern 2:**

```
<urn:uuid:hifil> dev:hasUser <http://people.com/alice> .
<urn:uuid:tv1> dev:hasUser <http://people.com/bobby> .
<urn:uuid:tv1> dev:hasUser <http://people.com/chris> .
```

$?r$	$?u$
$\langle \text{urn:uuid:hifil} \rangle$	$\langle \text{http://people.com/alice} \rangle$
$\langle \text{urn:uuid:tv1} \rangle$	$\langle \text{http://people.com/bobby} \rangle$
$\langle \text{urn:uuid:tv1} \rangle$	$\langle \text{http://people.com/chris} \rangle$

**Pattern 3:**

```
<http://people.com/alice> per:gender per:Female
<http://people.com/chris> per:gender per:Female
```

$?r$	$?u$
*	$\langle \text{http://people.com/alice} \rangle$
*	$\langle \text{http://people.com/chris} \rangle$

**4. Substitute \* by the available values of the involved variable in combinations present in other Plant patterns.** The \* is unwrapped and the whole set of possible combinations is obtained:

	?r	?u
Pattern 1:	<urn:uuid:hifil>	<http://people.com/alice>
	<urn:uuid:hifil>	<http://people.com/bobby>
	<urn:uuid:hifil>	<http://people.com/chris>
	<urn:uuid:tv1>	<http://people.com/alice>
	<urn:uuid:tv1>	<http://people.com/bobby>
	<urn:uuid:tv1>	<http://people.com/chris>
	<urn:uuid:light1>	<http://people.com/alice>
	<urn:uuid:light1>	<http://people.com/bobby>
	<urn:uuid:light1>	<http://people.com/chris>
Pattern 2:	<urn:uuid:hifil>	<http://people.com/alice>
	<urn:uuid:tv1>	<http://people.com/bobby>
	<urn:uuid:tv1>	<http://people.com/chris>
Pattern 3:	<urn:uuid:hifil>	<http://people.com/alice>
	<urn:uuid:tv1>	<http://people.com/alice>
	<urn:uuid:light1>	<http://people.com/alice>
	<urn:uuid:hifil>	<http://people.com/chris>
	<urn:uuid:tv1>	<http://people.com/chris>
	<urn:uuid:light1>	<http://people.com/chris>

**5. Identify the combinations of values that match all the patterns: these combinations represent the solutions for the variables. If no combination is identified, the query cannot be resolved.** There are two combinations that appear in every pattern: these are the solutions.

?r	?u
<urn:uuid:hifil>	<http://people.com/alice>
<urn:uuid:tv1>	<http://people.com/chris>

Therefore, the values of ?r are the actual resources that meet the query “devices whose user is a woman”.

Some optimisations can be implemented during step 4, avoiding the substitution of all the possible values in the patterns and using the minimum possible subset (provided by Pattern 2 in the example).

This algorithm is straightforward to implement in limited devices and provides the required compatibility with our semantic-powered discovery mechanism<sup>2</sup>.

### 3.3.2 mRDP SPARQL queries

In addition to Plant, SPARQL queries are supported in mRDP as illustrated further in Listing 3.5. Only `SELECT` constructions are allowed in mRDP SPARQL queries since they are the only ones that can produce bounded variables with values (`CONSTRUCT` and `DESCRIBE` return a graph, and `ASK` returns a boolean).

Since a SPARQL `SELECT` query can include a number of variables, the one involved in the mRDP resolution is that referred in the request line of the protocol (see subsection 3.4).

SPARQL queries are much more powerful than Plant queries. They can express conditions and filters with logical connectives and a broad range of operators. The main drawback of using SPARQL is that resource-limited devices will not be able to process SPARQL queries, therefore it is always recommendable to use Plant queries whenever possible and SPARQL queries more sparingly.

## 3.4 mRDP message format

mRDP request messages typically convey a Plant query over UDP and are multicasted to the address 224.0.24.1<sup>3</sup> and UDP port 2773. The MIME type [FK05] for Plant queries is `application/com.awareit.plant`.

mRDP request messages follow an multi-line syntax, being extensible through new headers to provide additional semantics. As most of Internet protocols, ASCII characters 13 and 10 (CR LF) are used to separate lines which constitute the major divisions in the message format.

The Augmented BNF grammar [CO05] for mRDP messages is:

Listing 3.3: Augmented BNF grammar for mRDP messages.

```
1 request-message = request-line *(header CRLF) CRLF [body]
2 request-line = (c-identify / c-locate) SP version CRLF
3 c-identify = "IDENTIFY" SP variable
```

---

<sup>2</sup>Other alternative algorithms such as Rete were considered, but they generally require more time and memory to create the temporal structures for efficient matching.

<sup>3</sup>Not assigned by IANA as of November 2006.

```

4 c-locate = "LOCATE" SP absoluteURI
5 variable = "?" token
6 version = "mRDP/" number "." number
7 header = h-NSeq / h-Content-Type / h-Content-Length / h-Callback-URI
8 h-NSeq = "NSeq" ":" number
9 h-Content-Type = "Content-Type" ":" media-type
10 h-Content-Length = "Content-Length" ":" number
11 h-Callback-URI = "Callback-URI" ":" i-Callback 0*("," i-Callback)
12 i-Callback = absoluteURI [ ";" p-Callback-Type]
13 p-Callback-Type = "type" "=" absoluteURI
14 body = 1*OCTET
15
16 number = 1*DIGIT
17 media-type = type "/" subtype
18 type = token
19 subtype = token

```

Some of the above productions are widely used in Internet protocols. `SP`, `CRLF`, `OCTET`, `DIGIT` can be found in [CO05], as well as `token` and `absoluteURI` can be found in [FGM<sup>+</sup>99].

An mRDP request message is composed of a request line, zero or more headers and an optional body section. The request line's main element is the command, either an `IDENTIFY` command followed by the variable to resolve in the query or a `LOCATE` command with a resource URI. The request line ends with a protocol version number.

Headers follow the same format as traditional headers in protocols such as HTTP [FGM<sup>+</sup>99], also used in RTSP [SRL98] or SIP [RSCI<sup>+</sup>02]. There are four headers defined in mRDP:

- `NSeq`: request sequence number for detecting duplicate request messages from clients and matching requests and responses.
- `Content-Type`: MIME type of the message body section. Currently, mRDP supports two MIME types: `application/com.awareit.plant` for Plant queries, and `application/sparql-query` for SPARQL queries. This header must not appear if the message does not contain a body section (such as `LOCATE` messages).
- `Content-Length`: length in bytes of the message body section. This header must not appear if the message does not contain a body section (`LOCATE` messages).
- `Callback-URI`: the endpoint where the server must send the response back to the client. The `type` parameter can inform about the interface type used by this endpoint in the form of a namespace

URI. Several callback URIs can be provided by the client in this header, with the same or different interface types, in order to allow the server to build the response message in the most suitable format. If no interface type is provided, the default is `http://www.awareit.com/soam/2006/04/redelws#httpPost`, which is the only interface type for mRDP callbacks defined at the moment (see appendix C).

IDENTIFY messages must contain a body section where the query, either in Plant or SPARQL is conveyed, while LOCATE messages must not contain a body.

For example, a message conveying the Plant query in Listing 3.2 could be<sup>4</sup>:

**Listing 3.4: An example mRDP resource identification message.**

```
1 IDENTIFY ?r mRDP/1.0
2 NSeq: 23
3 Content-Type: application/com.awareit.plant
4 Content-Length: 314
5 Callback-URI: http://169.254.0.3/mrdpendpoint
6
7 ?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://pervasive
   .semanticweb.org/ont/2004/06/device#Device> .
8 ?r <http://pervasive.semanticweb.org/ont/2004/06/device#hasUser> ?u .
9 ?u <http://pervasive.semanticweb.org/ont/2004/06/person#gender> <http
   ://pervasive.semanticweb.org/ont/2004/06/person#Female> .
```

The same message using SPARQL for the query:

**Listing 3.5: An example mRDP resource identification message with SPARQL.**

```
1 IDENTIFY ?r mRDP/1.0
2 NSeq: 23
3 Content-Type: application/sparql-query
4 Content-Length: 327
5 Callback-URI: http://169.254.0.3/mrdpendpoint
6
7 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
8 PREFIX dev: <http://pervasive.semanticweb.org/ont/2004/06/device#>
9 PREFIX per: <http://pervasive.semanticweb.org/ont/2004/06/person#>
10 SELECT ?r
11 WHERE { ?r rdf:type dev:Device .
```

---

<sup>4</sup>Note that the IP address 169.254.0.3 belongs to the link-local range 169.254.0.0–169.254.255.255, reserved by IANA for IP auto-configuration [IAN02].

```

12     ?r dev:hasUser ?u .
13     ?u per:gender per:Female .
14 }

```

Listing 3.6 is an example of a `LOCATE` message broadcasted to the network in order to obtain information sources where more data about `urn:uuid:phone1` can be downloaded. Two possible callback endpoints are provided, one of them with the default interface explicitly declared (though this is not required):

Listing 3.6: An example mRDP resource description location message.

```

1 LOCATE urn:uuid:phone1 mRDP/1.0
2 NSeq: 57
3 Callback-URI: http://169.254.0.3/mrdpendpoint, http
    ://169.254.0.3:8081/altmrdpep?type=http://www.awareit.com/soam
    /2006/04/redelws#httpPost

```

Due to the unreliable nature of UDP, mRDP request messages must be sent three times in order to increase the probability of reaching the possible destinations in the case some UDP packet is lost. Three times for every message is a good balance between increasing reliability without generating much network traffic (other discovery protocols such as UPnP SSDP also send the messages three times).

### 3.5 ReDEL: Resource Description Endpoints Language

After resolving the variables in the resource identification message or locating the information sources in the resource description location message, the mRDP server constructs the reply, which is an HTTP request to the callback URI provided by the client and conveying the required data in a suitable format.

The client receives the replies via the provided `Callback-URI`, processing them afterwards. The replies themselves are very simple, they just need to express the resources matching the query and how to access available descriptions. The format should be straightforward, so that limited devices can process it without much overload.

We have designed a simple mark-up language called ReDEL – Resource Description Endpoints Language, for annotating the information contained in the replies to `IDENTIFY` and `LOCATE` mRDP messages.

A possible HTTP callback conveying a ReDEL response to the query shown in Listing 3.4 is the following:

Listing 3.7: An example of HTTP callback conveying ReDEL payload.

```
1 POST /mrdpendpoint HTTP/1.0
2 Host: 169.254.0.3
3 NSeq: 23
4 Content-Type: application/com.awareit.redel+xml
5 Content-Length: 589
6
7 <?xml version="1.0" encoding="UTF-8"?>
8 <redel xmlns="http://www.awareit.com/soam/2006/04/redel"
9     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
10    xsi:schemaLocation="http://www.awareit.com/soam/2006/04/redel
11    http://www.awareit.com/soam/2006/04/redel.xsd">
12
13   <resource uri="urn:uuid:tv1">
14     <location url="http://169.254.0.12/description" type="http://www
15     .awareit.com/soam/2006/04/srdfws#httpGet"/>
16     <location url="http://169.254.0.12/sparql" type="http://www.w3.
17     org/2005/08/sparql-protocol-query/#queryHttpPost"/>
18   </resource>
19 </redel>
```

Lines 13–16 embody the information related to the resource that matched the query. For example, `urn:uuid:tv1` received the request message and identified itself as a possible solution to the query, thus generating the above response for the client and including two URLs in lines 14 and 15 where the client can retrieve extended information about `urn:uuid:tv1` itself.

The first URL provides information about `urn:uuid:tv1` when requested through a simple HTTP GET message, while the second URL implements the SPARQL protocol HTTP POST binding as described in [Wor06a]. The default in case no interface is specified is the basic HTTP GET represented by `http://www.awareit.com/soam/2006/04/srdfws#httpGet`.

The mRDP client provides an HTTP interface for receiving this kind of callbacks. Besides simple ReDEL payload over HTTP POST messages as in the previous example, a SOAP interface can be provided by the client. Both the complete ReDEL XML Schema and ReDEL WSDL can be found in appendix C.

The minimum requirements any implementation must support at the client side are:

- Creating `application/com.awareit.plant` queries.
- Disseminating the queries to the multicast IP address 224.0.24.1 and multicast UDP port 2773.

- Reserving an HTTP callback URI for receiving HTTP POST messages with ReDEL payload.
- Processing ReDEL documents with the solution.

And at the server side:

- Joining the multicast IP address 224.0.24.1 and listening to the multicast UDP port 2773.
- Processing `application/com.awareit.plant` queries.
- Generating ReDEL documents with the solution.
- Issuing HTTP POST callbacks for conveying the ReDEL payload.

### 3.6 Example of advanced uses of semantic queries

As commented earlier, a major advantage of using RDF/OWL support instead of basic attribute-value matching is that semantic processing can be performed to resolve the query. For instance, revisiting a slightly modified version of the previous example in this section:

```
<urn:uuid:hifil> rdf:type dev:Device .
<urn:uuid:tv1> rdf:type tv:TV .
<urn:uuid:light1> rdf:type light:Light .
<urn:uuid:hifil> dev:hasUser <http://people.com/alice> .
<urn:uuid:tv1> dev:hasUser <http://people.com/bobby> .
<urn:uuid:tv1> dev:hasUser <http://people.com/chris> .
<http://people.com/alice> rdf:type rel:Mother .
<http://people.com/bobby> per:gender per:Male .
<http://people.com/chris> per:gender per:Female .
```

If a client issues the query “identify all the devices whose user is a woman”, a strict matching would not yield any result, since only `<urn:uuid:hifil>`'s type is considered a `dev:Device`, and there is not any evidence that its user's genre (`<http://people.com/alice>`) is `per:Female`.

There are two mechanisms to support intelligent reasoning in mRDP servers when resolving the queries:

1. Ontologies, based on description logics.
2. Domain inference rules, based on forward chaining.

In order to illustrate the first mechanism, let us suppose that an ontology is available declaring that:

```
tv:TV rdfs:subClassOf dev:Device .
```

In order to illustrate the second mechanism, let us suppose that the following inference rule is known by the server:

```
(x rdf:type rel:Mother) ⇒ (x per:gender per:Female)
```

Therefore, the processing of both existing ontologies and knowledge domain rules, prior to query evaluation would produce the following statements that would be added to the knowledge base:

```
<urn:uuid:tv1> rdf:type dev:Device .  
<http://people.com/alice> per:gender per:Female .
```

The first statement is produced by the ontological reasoning mechanism, while the second one is produced by applying the domain inference rule. Once reasoning is performed in this way, the query would be successfully resolved to `<http://people.com/alice>` and `<http://people.com/chris>`.

### 3.7 Performance evaluation of lexical and semantic discovery with mRDP

The benefits of semantic discovery compared to traditional text-based matching are clear: more refined results are obtained by interpreting the information relationships. Only a subset of those results, or even none, could be obtained using non-semantic mechanisms. However, what is the impact in performance for semantic queries?

In order to test the performance of the matchmaking mechanism in mRDP with and without reasoning, we executed the above search against 10 mRDP servers (Pentium 1.86 GHz) powered with a Java implementation of the Plant Query Resolution Algorithm. During the first round, the servers were configured to apply both ontology and domain rules reasoning. During the second round, no form of semantic reasoning was performed.

Every test was executed 8 consecutive times. The first execution provokes the initialisation of internal structures and, thus, takes more time, so we isolated it. Second and subsequent executions produce more stabilised measures.

Absolute values are not representative since they depend on system configuration, so we analysed relative measures. In this scenario with few rules and basic ontological relationships, lexical matchmaking is about 2.82

times faster than semantic matchmaking in the first execution, when the initialisation takes place (see Figure 3.5).

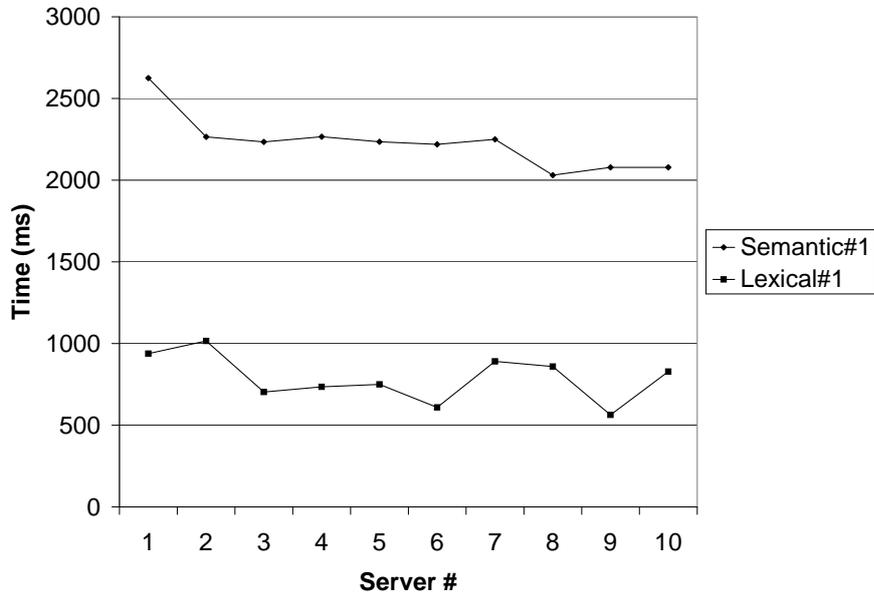


Figure 3.5: Matchmaking performance during the first execution in mRDP discovery.

Subsequent executions are much more efficient, since internal structures have been already generated. In this case, lexical matchmaking performs even better, around 5.55 times faster than semantic matchmaking after stabilisation (see Figure 3.6). As more complex ontologies and domain rules are provided, the time devoted to reasoning increases exponentially.

As expected, semantic matchmaking is more time-consuming than the simple lexical version. However, it is much more powerful than the traditional approach in order to find resources that meet some concrete criteria. Semantic discovery is able to browse the roots of the information to unleash the real meaning, create relationships and produce the right answers.

Semantic matchmaking can be activated in computer platforms that are capable of producing results without a significant delay, while lexical matchmaking can be performed in more resource-limited devices. Moreover, we consider that the performance of our implementation of the Plant Query Resolution Algorithm can be greatly improved.

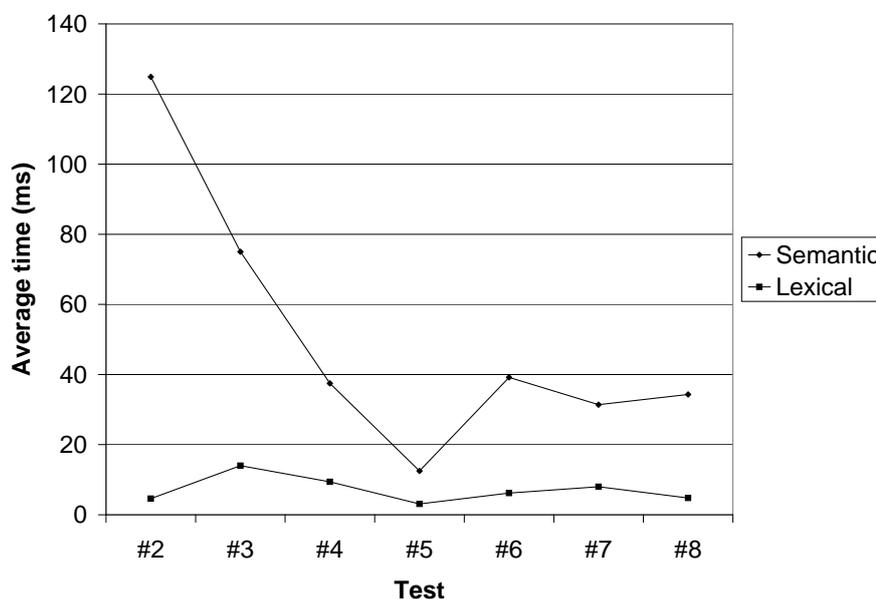


Figure 3.6: Stabilised matchmaking performance in mRDP discovery.

Despite demanding more time and higher computing requirements, we deem semantic discovery mechanisms to provide the degree of intelligence expected in Ambient Intelligence scenarios.

mRDP is able to provide this functionality while also supporting traditional lexical matchmaking, thus integrating limited and advanced platforms within one unique discovery model.

### 3.8 Comparative analysis

mRDP provides a simple and powerful way for performing semantic queries in Ubiquitous Computing environments, taking advantage and reusing HTTP infrastructure. Several discovery mechanisms for Ubiquitous Computing have been proposed in the past, none of them widely accepted. Edwards published a comparative [Edw06] about SSDP, Jini, Bluetooth SDP, SLP, Bonjour, Salutation, INS and Ninja SSDS, including also infrared and RFID mechanisms. The comparison was based on the criteria of topology, transport, scope, search and security.

We reproduce in Table 3.1 the comparison table with an additional row for mRDP<sup>5</sup>.

mRDP exhibits some distinct factors from other alternatives, such as its powerful semantic search capabilities, and the use of well-proven and reliable HTTP-based security infrastructure.

We deem mRDP to be the best candidate for intelligent discovery in pervasive computing scenarios, and also a valuable and promising alternative in other networking environments.

---

<sup>5</sup>The rows for eSquirt (infrared) and RFID are not included, since they are not relevant for the current research.

System	Topology	Transport	Scope	Search	Security
Simple Discovery (SSDP)	Peer-to-peer (P2P)	Unicast HTTP, multicast HTTP	Subnet	Type or ID	Authentication, access control
Jini	Hybrid	Unicast TCP, multicast UDP	Subnet, bridgeable	Type, ID or attribute	Jini/Java security mechanisms
Bluetooth Discovery (SDP)	Peer-to-peer (P2P)	Link manager Protocol (LMP) and Logical Link Control and Adaptation Protocol (L2CAP)	Proximity (~10 meters)	Type attribute (universally unique identifier [UUID] only)	Link-level or service-level encryption, authentication
Service Protocol	Peer-to-peer (P2P) or directory	Unicast TCP, multicast UDP	Subnet, bridgeable	Type attribute ( Lightweight Directory Access Protocol v.3 search predicates)	Optional service authentication
Bonjour	Peer-to-peer (P2P)	Multicast DNS (mDNS), DNS service discovery (DNS-SD)	Subnet	Type	Optional IP security (IPsec), DNS security extensions (DNSsec)
Salutation	Peer-to-peer (P2P) or directory	Open Network Computing remote procedure call (ONC RPC) over arbitrary transports	Depends on transport	Type or attribute	RPV authentication
Intentional Service (INS)	Decentralized, weakly consistent directories	Unicast UDP	Administrative	Attribute domain	None
Ninja Secure Service (SSDS)	Directory	Authenticated Method Invocation (RMI)	Wide area (through hierarchical directories)	XML-based descriptions	Capability-based access control
<b>Multicast Discovery (mRDP)</b>	<b>Peer-to-peer (P2P)</b>	<b>Unicast HTTP, multicast UDP</b>	<b>Subnet, bridgeable</b>	<b>Semantic queries, attribute-based queries</b>	<b>HTTP-based security (HTTPS and HTTP Authentication)</b>

Table 3.1: Comparison of current discovery systems and mRDP.

# A Theoretical Model for Context-Aware Reactivity

*“The Power of Context is an environmental argument.  
It says that behavior is a function of social context.”*

*Malcolm Gladwell  
The Tipping Point, 2000*

**C**ONSIDERING the research goals established in chapter 1 and the analysis of related initiatives, we have designed a theoretical and architectural model called SoaM (Smart Objects Awareness and Adaptation Model). The SoaM architecture is intended to meet the proposed criteria better than any of the analysed architectures, and accomplish our general research goal.

But SoaM is not just an architecture. We have tried to identify the theoretical basis underlying the context-awareness process under our premises. In this chapter we present this theoretical basis that will shape the foundations of the architecture.

## 4.1 Passively influencing the environment

The vision of Ubiquitous Computing and Ambient Intelligence, as referred in chapter 1, promotes proactive environments that perceive users' surrounding information, often referred to as context, and react in the appropriate way to facilitate users' activities. As stated before, the

most valuable resource in such environments is not computing power, communication or storage capabilities, but user attention [GSSS02] [SG02] [SG03].

From our point of view, environment adaptation to user needs can be achieved from two different perspectives:

1. **Active influence:** any mechanism in which the agent explicitly commands other agents or objects to change their state or perform an action. Examples of active mechanisms are configuration processes and operation invocation techniques such as CORBA, RMI or SOAP.
2. **Passive influence:** any mechanism in which an agent disseminates certain information, expecting that other agents perceive that information and change their state or perform an action at their discretion in order to create a more adapted environment [VL04].

While active influence represents traditional “command and control” mechanisms, agents applying passive influence do not command the target agents or objects to do anything concrete; they simply modify the context expecting the others to react, changing their state in a positive (expected or maybe surprisingly unexpected) way [VL05] (see Figure 4.1).

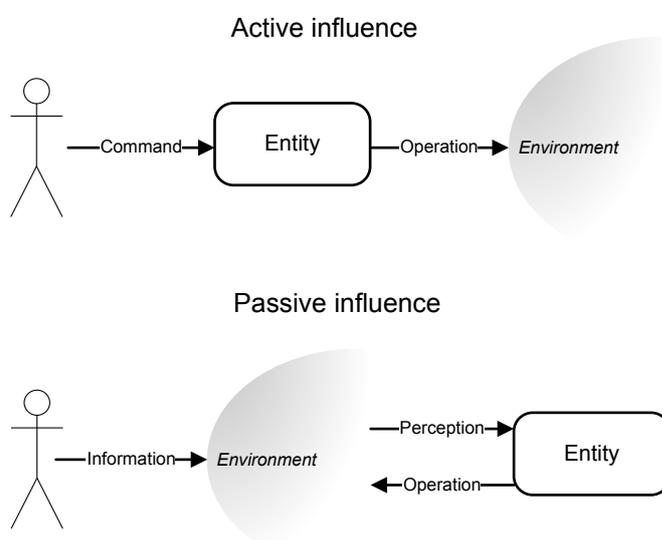


Figure 4.1: Active and passive influence.

For instance, by exhibiting our presence in front of an automatic door, we influence the door behaviour, which opens immediately. Maybe when

watching a concrete TV show the ambient lights of the living-room adjust themselves automatically to fit the intensity and colours of the screen, eliminating reflections and creating a more comfortable environment.

Passive influence works at the periphery of user attention, without disturbing the user, without being explicitly commanded, silently working in the background, non-intrusively, but probably their effects are less predictable. Passive influence is at the core of context-awareness by promoting autonomous behaviour on devices.

Passive mechanisms are also complementary to active ones and can serve to automatically adapt the environment in a initial phase, while allowing users to customise it later via explicit interaction. For instance, when a user enters a car, the temperature, radio station and driving settings, could be automatically configured to his preferences/characteristics without any explicit command, being the user free to change them explicitly afterwards.

These passive influence strategies have not been very much explored in pervasive computing scenarios; most of existing systems use several forms of active mechanisms, such as WebServices/SOAP (UPnP [UPn03], Task Computing [MLPS03] [MPL03] [SHP03], WSAMI [IST<sup>+</sup>05]), Jini, CORBA, and so forth. But as such, they require explicit user intervention continuously to control the environment.

For example, if a user wants to open a door using an active influence model, he can look up for the “door device” in the PDA and invoke the “open” service.

In order to obtain the same results using a passive influence model, there are two techniques available:

- **Context modification:** if devices are context aware, it is possible to influence their behaviour by modifying the context information (see Figure 4.2). For example, if a door opens automatically when a person is situated less than 1 meter away, just getting into these limits causes the door to open. The user does not command anything, but the door reacts autonomously. Probably, this door features a built-in internal rule relating the “open” operation to the data provided by a presence sensor. But generally it is not possible to dynamically change this behaviour in order to, for example, opening only during working hours, or depending on the weather. The door is context-aware but features *static behavioural rules*.
- **Behavioural profiles dissemination:** the behaviour of context-aware devices can be dynamically altered and modified by other entities in such a way that the device can be influenced to react to any concrete stimulus in the desired way. For example, the door’s continuous

behaviour can be altered by the rule “if the weather is sunny the door should remain open during daylight hours”. This technique modifies the entities’ internal behaviour and prepares them to react appropriately to context modification, thus working at a higher level (see Figure 4.3). Existing entities in the environment can influence each other by disseminating behavioural profiles, achieving a greater degree of context-awareness and reactivity than applying context modification alone.

During our research we also use the term “adaptation profiles” for “behavioural profiles” interchangeably, since they enable an entity to adapt dynamically to environmental changes.

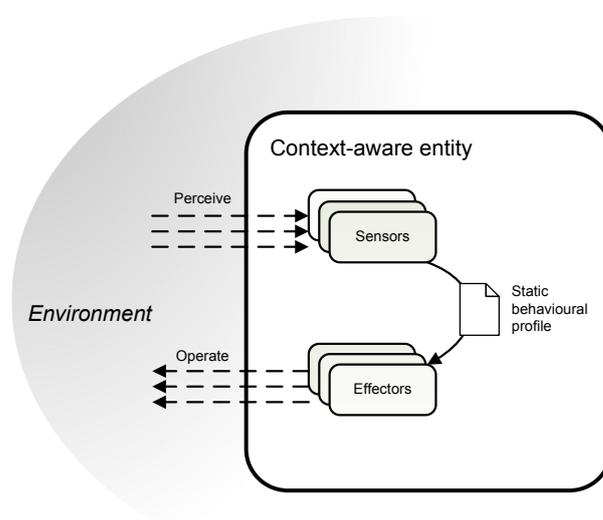


Figure 4.2: A context-aware entity featuring static behavioural reactivity to context modification.

Some of the architectures analysed such as CoBrA [CFJ03a], Gaia [RC03b] or Semantic Spaces [GPZ04b] only provide the traditional context modification based reactivity. They do not provide any form of “behaviour dissemination”, nor any means for existing agents in the environment to dynamically change others’ behaviour. In case that mechanism was provided, it could not be placed in the involved devices, but in the central component of the architecture that accumulates all the information. This limitation is again originated from their centralised nature.

The dissemination technique is complementary to traditional context modification based reactivity: it is a preliminary preparation step for

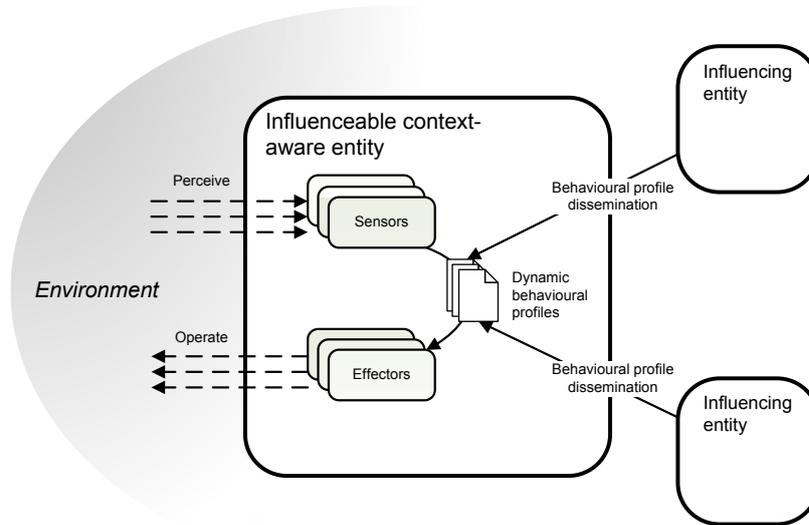


Figure 4.3: A context-aware entity featuring influenceable behavioural reactivity to context modification.

configuring a context-aware device with a required particular behaviour for a period of time, in order to react appropriately when the context changes.

A context-aware door featuring context modification reactivity is good, but if the door's behaviour can be dynamically augmented to address specific behavioural requirements, its context-aware reactivity improves dramatically.

Thus, the ability of passively influencing other entities not only via context modification but also through "behavioural profiles" dissemination will become a differentiation factor and a vehicle in our model to achieve a higher degree of context-awareness.

## 4.2 A set-theory based approach for context-aware reactivity

A very large number of Ubiquitous Computing systems, as well as non-ubiquitous ones, lack from a theoretical background that supports the decisions taken during the design phase. Although defining theoretical foundations is not the main concern of our research, we deem it important to support the resulting design.

Therefore, we have tried to identify the factors that take part in how an entity can perceive context information and adapt dynamically to environmental changes.

Undoubtedly, context awareness is the key concept that embraces both perceiving context information and reacting to it accordingly, maybe generating new context information that will originate further reactions in other entities.

During our research, we found that set theory was a suitable formalism to explain and illustrate the major concepts related to context identification and reactivity from our point of view, and thus, we built our theoretical framework around this mathematical area.

#### 4.2.1 Environment-oriented approach

Context information can be classified into *knowledge domains*, which are in turn formed by individual *knowledge domain items* that represent particular properties or attributes within that domain.

The following are some examples of knowledge domains and constituent knowledge domain items:

- Light domain: light state (on / off), luminance level, light color, ...
- TV domain: state (on / off), current channel, current volume, current mode (teletext, TV), ...
- Door domain: locked (true / false), opening degree, ...
- User activities: current activity code, current activity starting time, ...

In the following formula,  $I^E$  represents the whole set of context knowledge domain items  $i_1, i_2, \dots$ , available in the environment  $E$ , as indicated above.

$$I^E = \{i_1, i_2, i_3, \dots, i_n\} \quad (4.1)$$

Of course, not every item in a concrete context domain is required to be available in the environment  $E$ : for a particular space the “luminance level” may be part of the context information, while “light colour” may not.

At a particular moment of time  $t$ , these items adopt concrete values (or sets of values if the items are multi-valued):

$$\begin{aligned}
 i_1(t) &= v_1 \\
 i_2(t) &= v_2 \\
 i_3(t) &= v_3 \\
 &\dots \\
 i_n(t) &= v_n
 \end{aligned}
 \tag{4.2}$$

At that moment of time  $t$ , the whole environment can be characterised by the concrete values adopted by the constituent knowledge domain items. We represent this set of values as  $I^E(t)$ .

$$I^E(t) = \{i_1(t), i_2(t), i_3(t), \dots, i_n(t)\} = \{v_1, v_2, v_3, \dots, v_n\} \tag{4.3}$$

The evolution of one particular knowledge domain item in the environment depends on its current value, and the influences (constraints) exerted on it by existing entities at the concrete moment of time  $t$ . The constraints exerted over the item  $i$  at the moment of time  $t$  are represented as the set  $C(i, t)$ .

$$i(t+1) = f(i(t), C(i, t)) = f(v, C(i, t)) \tag{4.4}$$

For example, the next value of “temperature” depends on the current value and the existing influences applied by heating and cooling systems, human activity and so on. Those influences or constraints can be represented as desired conditions:

$$C(\text{temperature}, \text{now}) = \{> 23^\circ\text{C}, < 27^\circ\text{C}, = 25^\circ\text{C}\} \tag{4.5}$$

Integrating formulas 4.3 and 4.4, the state of the environment at time  $t+1$  can be expressed as:

$$\begin{aligned}
 I^E(t+1) &= \{i_1(t+1), i_2(t+1), i_3(t+1), \dots, i_n(t+1)\} \\
 &= \{f(v_1, C(i_1, t)), f(v_2, C(i_2, t)), \dots, f(v_n, C(i_n, t))\} \\
 &= F(I^E(t), C(I^E, t))
 \end{aligned}
 \tag{4.6}$$

The formula 4.6 comprehends a fundamental point in our model: the future state of the environment depends on the current state of the environment and all the influences exerted on it at a concrete moment of time, which leads to the following proposition:

**Proposition 1.** Influencing constraints are the driving factor of environmental change.

### 4.2.2 Entity-oriented approach

A concrete environment is context-aware if it is, in turn, composed by individual entities, which are context-aware and, thus, can drive the adaptation process.

Any context-aware entity pursues a major goal: to adapt its behaviour accordingly to changes in the environment, that is, to changes in perceived context information. The adaptation behaviour can be represented in different forms, such as a program or a rule-based conditional action [LK01] [BEP06] [SRC05].

In order to perform this task, a context-aware entity must be able to perceive some context information, but also to alter other context information, which is the basis for reactivity.

For instance, a temperature control system features a temperature sensor and an internal clock for perceiving the current temperature and time, but it features also some effectors that operate over the heating system to set the appropriate temperature value.

Therefore, a context-aware entity can perceive information on some knowledge domain items of  $I^E$  and operate over the same or other knowledge domain items. In the previous example, the temperature control system perceives the current temperature and the time, but operates only on the temperature item.

An intelligent TV system can turn itself off automatically if nobody is located in the room. In this case the perception domain is “presence” and the operation domain is “TV state”.

A context-aware entity can perceive context information directly using built-in sensors or indirectly via communication channels with other entities that provide that information. For instance, a mobile phone can detect directly whether the user is engaged in a conversation using the phone, and perceive user’s location indirectly by receiving positioning data from the mobile operator systems.

Similarly, the current context information can be changed directly via built-in effectors or indirectly via invocation of remote services on other entities that perform the required operation.

Thus, the capabilities of every context-aware entity can be characterised by four sets of knowledge domain items:

- $P_d$ : direct perception capabilities. The set of knowledge domain items the entity is able to perceive directly.

- $P_i$ : indirect perception capabilities. The set of knowledge domain items the entity is able to perceive indirectly.
- $O_d$ : direct operation capabilities. The set of knowledge domain items the entity is able to operate directly.
- $O_i$ : indirect operation capabilities. The set of knowledge domain items the entity is able to operate indirectly.

A context-aware entity's capabilities can be expressed as:

$$e_c = (P_d, P_i, O_d, O_i) \quad (4.7)$$

If direct or indirect perception / operation capabilities were not relevant in a particular model, the sets could be reduced to two:

$$\begin{aligned} P &= P_d \cup P_i \\ O &= O_d \cup O_i \end{aligned} \quad (4.8)$$

As well as the entity's capabilities characterisation is simplified:

$$e_c = (P, O) \quad (4.9)$$

If the environment is populated by a number of entities  $a, b, c, \dots$ :

$$E = \{a, b, c, \dots\} \quad (4.10)$$

And these entities exhibit perception capabilities  $P^a, P^b, P^c, \dots$ , the whole set of environment perception capabilities is:

$$P^E = P^a \cup P^b \cup P^c \cup \dots = I^E \quad (4.11)$$

Which is the context information types populating the environment  $E$ , and it is the same set of knowledge domains represented in formula 4.1 as  $I^E$ .

But now there is also another set  $O^E$ , which embraces all the knowledge domains items that can be *operated* in the environment:

$$O^E = O^a \cup O^b \cup O^c \cup \dots \quad (4.12)$$

$O^E$  represents the set of environmental information that can be altered, changed, adapted, which is the focus of the reactive behaviour. This set is

identical or a subset of the environmental information that can be perceived (maybe some context information can be perceived but not altered such as “weather forecast” or “presence of people”):

$$O^E \subseteq I^E = P^E \quad (4.13)$$

Inferred from the previous formulas is the notion that an environment  $E$  populated by individual entities inherits their intrinsic capabilities.

$$E_c = (P^E, O^E) \quad (4.14)$$

**Proposition 2.** The set of perception and operation capabilities exhibited by any environment is composed by the aggregation of perception and operation capabilities, respectively, provided by its constituent entities.

### 4.2.3 Managed constraints

We have already presented informally the term *constraint* but for the purposes of the following descriptions, a more formal definition is provided.

**Constraint** : the state of being restricted or confined within prescribed bounds<sup>1</sup>.

For the purposes of our research we provide a working definition, more related to the other concepts presented above:

**Proposition 3.** A constraint is a particular restriction over the set of possible values a concrete knowledge domain item is able to acquire.

For instance, some constraints on the “temperature” might be [ $> 23^\circ C$ ], [ $< 27^\circ C$ ], [ $= 19^\circ C$ ] or [ $= 25^\circ C$ ], illustrated in Figure 4.4.

Constraints can be represented as sets containing a concrete subset of all possible values the knowledge domain items can take. Of course, one-element sets such as [ $= 19^\circ C$ ] are possible.

During a reactive response, a particular context-aware entity  $e$  performs control on some knowledge domain items (which are declared in the entity’s

---

<sup>1</sup>Source: The American Heritage Dictionary of the English Language, Fourth Edition.

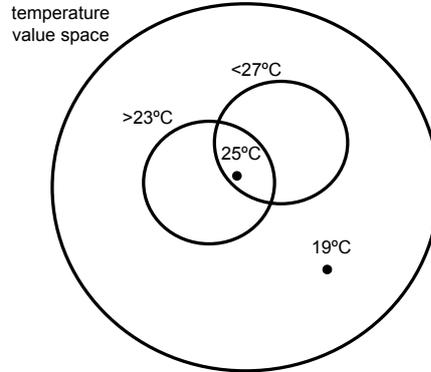


Figure 4.4: Constraints as subsets and elements.

operation capabilities, as explained above). In our model, this control process of the entity  $e$  is completely driven by the constraints managed by  $e$ .

For instance, a temperature control system may perform control on the “temperature” item, driven by the constraints  $[> 23^{\circ}C]$  and  $[< 27^{\circ}C]$ , thus trying to keep the current temperature within this threshold all the time. From the set theory point of view the goal is to find a value contained both in the  $[> 23^{\circ}C]$  and  $[< 27^{\circ}C]$  sets, that is, the intersection of both sets.

A problem arises when constraints, represented as sets, are disjoint, as the case of  $[= 19^{\circ}C]$  (a one-element set) in Figure 4.4. In the presence of conflicting constraints it is up to the entity to decide which conflict resolution strategy to apply (see subsection 5.1.3).

We use the term  $C^e(i, t)$  to represent the set of constraints managed by the entity  $e$  over the knowledge domain item  $i$  at a concrete moment of time  $t$ .

$$C^e(i, t) = \{co \mid co \text{ is a constraint of the entity } e \text{ on the knowledge domain item } i \text{ at the moment of time } t\} \quad (4.15)$$

And  $C^e(t)$  is the whole set of constraints managed by the entity  $e$  on all its operation domains.

$$C^e(t) = C^e(i_1, t) \cup C^e(i_2, t) \cup C^e(i_3, t) \cup \dots \cup C^e(i_n, t) \quad (4.16)$$

Thus, the term  $C(I^E, t)$  in formula 4.6, which represented the whole set of constraints driving the adaptation in environment  $E$ , is ultimately generated by the constraints managed by every entity  $e$  in such environment  $E$ .

$$C(I^E, t) = \{co | co \in C^e(t) \forall e \in E\} \quad (4.17)$$

Which is the logical inference: since constraints are managed by context-aware entities in the environment, these entities are in charge of adapting the whole environment. Thus, proposition 1 can be further refined as:

**Proposition 4.** Context-aware entities in the environment manage the constraints that drive environmental change.

#### 4.2.4 The context-awareness process

For the purposes of our research, the whole process of context-awareness is composed of five different activities any context-aware entity  $e$  must perform:

1. Context discovery: involves the discovery of sources of context information that are relevant for the entity. At this point, the set of perception capabilities for the entity  $e$  is created: direct perception capabilities (sources are built-in sensors) and indirect perception capabilities (sources are other entities).
2. Context retrieval: involves the retrieval of actual context information from the sources discovered in the previous step.
3. Reasoning: involves generation of new context information from the existing one, by applying several strategies such as description logics or rules.
4. Analysis: involves the interpretation of the whole set of context information, both obtained by external sources and inferred through reasoning in the previous phase, to determine the appropriate reactive behaviour. Constraints representing that behaviour are generated at this point.
5. Operation: involves the execution of the constraints determined in the previous phase. These constraints are matched against operation capabilities to determine whether the entity  $e$  is able to perform the operations (directly through built-in effectors or indirectly through other entities).

The following example is intended to clarify how the whole context-awareness process in our model works.

**Example 4.1.** Room21 is populated by a light control system (light1), a TV set (tv1) and a Hi-Fi music system (hifi1). All these devices are context-aware entities whose capabilities are expressed as follows:

$$\begin{aligned}
 P_d^{light1} &= \{\text{luminance, colour}\} \\
 O_d^{light1} &= \{\text{luminance, colour}\} \\
 P_d^{tv1} &= \{\text{tvstate (on / off), channel, user activity}\} \\
 O_d^{tv1} &= \{\text{tvstate (on / off), channel}\} \\
 P_d^{hifi1} &= \{\text{hifistate (on / off), volume, station, time}\} \\
 O_d^{hifi1} &= \{\text{hifistate (on / off), volume, station}\}
 \end{aligned} \tag{4.18}$$

The activities are carried out in this way:

1. Context discovery: devices discover each other and exchange capabilities information, so every one knows what knowledge domain items other objects are able to perceive, as well as what knowledge domain items can be altered by everyone. The sets  $P_i$  and  $O_i$  are created by every device:

$$\begin{aligned}
 P_i^{light1} &= \{\text{tvstate, channel, user activity, hifistate, volume, station, time}\} \\
 O_i^{light1} &= \{\text{tvstate, channel, hifistate, volume, station}\} \\
 P_i^{tv1} &= \{\text{luminance, colour, hifistate, volume, station, time}\} \\
 O_i^{tv1} &= \{\text{luminance, colour, hifistate, volume, station}\} \\
 P_i^{hifi1} &= \{\text{luminance, colour, tvstate, channel, user activity}\} \\
 O_i^{hifi1} &= \{\text{luminance, colour, tvstate, channel}\}
 \end{aligned} \tag{4.19}$$

2. Context retrieval: devices retrieve context information directly as stated by  $P_d$  and indirectly from other devices as stated by  $P_i$ .
3. Reasoning: the TV set is able to determine “user activity” information using predetermined built-in inference rules (via forward-chaining). If the TV state is “on” and Hi-Fi state is “off” the TV can generate the new context information “the user is watching TV” that can be made available to other devices during the next context retrieval cycle.
4. Analysis: the light system is configured to generate the constraint “light state must be off” whenever “the user is

watching TV” and “it is later than 10:00 pm” to facilitate the activity.

5. Operation: since the light system exhibits a direct capability operation on the “light state” knowledge domain item, it takes the responsibility of executing the constraint directly through its built-in effector and the light is turned off.

The Example 4.1 involves several devices providing different kinds of context information and discovering every other’s perception and operation capabilities, retrieving and generating new context information through reasoning, analysing the context information against a programmed behavioural rule and generating the adaptation constraints that must be honoured.

During the last phase, was not the light control system able to execute the constraint it would have sent it to the appropriate device if possible, or it would have ignored it if cannot be honoured by any of them.

This example illustrates how context-aware entities can share information and coordinate changes in the environment. But still there is one point left: how to represent the behavioural rules that generate constraints from the context information during the analysis activity.

#### **4.2.5 Behavioural profiles**

The reactive behaviour of context-aware entities can be represented using simple rules with preconditions and postconditions. We refer to these rules as “behavioural” or “adaptation profiles”, since they embody a facet of adaptive behaviour in the system. Behavioural profiles conform the basis for the dynamic context-aware reactivity discussed in section 4.1.

An example of behavioural profile could be “if temperature  $> 25^{\circ}\text{C}$  then window state = open”. This rule provokes an environmental adaptation, embodied in the postcondition, whenever the precondition is met.

Preconditions are evaluated against the current context information, and if matched, the behavioural profiles are activated and context-aware entities should perform the required operations for the postconditions to be honoured.

Since those operations are represented via constraints in our model, a mechanism is needed to resolve postconditions into a set of constraints that represent such postconditions.

Continuing the previous example, a constraint must be generated representing “window state = open” as the desired possible value of the knowledge domain item “window state”.

A behavioural profile is represented as:

$$\begin{aligned}
 bp &= (D_{PRE}, D_{POST}) \\
 D_{PRE} &= \{G_{PRE} | G_{PRE} \text{ is a condition on current context information}\} \\
 D_{POST} &= \{G_{POST} | G_{POST} \text{ is a condition desired for future context information}\}
 \end{aligned} \tag{4.20}$$

The point to note here is that both  $G_{PRE}$  and  $G_{POST}$  are sets themselves, since they represent possible values of the information (very much as constraints do). For example,

$$\begin{aligned}
 27^\circ C &\in G_{>25^\circ C} \\
 24^\circ C &\notin G_{>25^\circ C}
 \end{aligned} \tag{4.21}$$

We define the function  $active()$  to evaluate all the preconditions in a behavioural profile against current context information in order to determine if the behavioural profile must be activated.

$$active(I^E(t), bp) = \begin{cases} true & \text{if } \forall G_{PRE} \in D_{PRE}^{bp} \exists v \in I^E(t), v \in G_{PRE} \\ false & \text{otherwise} \end{cases} \tag{4.22}$$

That is, for every precondition  $G_{PRE}$  in the behavioural profile  $bp$  there is a piece of context information  $v$  such that  $v$  adopts one of the possible values defined by the precondition  $G_{PRE}$ , thus meeting it.

Once a behavioural profile has been activated, a set of constraints associated to the postconditions must be generated.

The  $G_1 \subseteq G_2$  relation represents the situation when a condition  $G_1$  is subsumed by other condition  $G_2$ , that is, all the values contained in  $G_1$  are also contained in  $G_2$ . For example:

$$\begin{aligned}
 G_{>27^\circ C} &\subseteq G_{>25^\circ C} \\
 G_{>25^\circ C} &\not\subseteq G_{>27^\circ C}
 \end{aligned} \tag{4.23}$$

If the temperature value is greater than  $27^\circ C$ , it is also surely greater than  $25^\circ C$ : all the values in the first set are also in the second. But the opposite is not true, since there are some values greater than  $25^\circ C$  that are not greater than  $27^\circ C$ .

The constraints obtained when the behavioural profile  $bp$  is activated under the context information  $I^E(t)$  can be expressed as:

$$C(I^E(t), bp) = \{co | \forall G_{POST} \in D_{POST}^{bp} \exists co, co \subseteq G_{POST}\} \quad (4.24)$$

Every constraint in this set meets a postcondition, being a subset of that postcondition since all the elements in the constraint are also elements in the postcondition.

For example, if the postcondition states that  $temperature > 25^\circ C$ , the constraint  $temperature > 27^\circ C$  honours the postcondition, being a subset of it as shown in Figure 4.5.

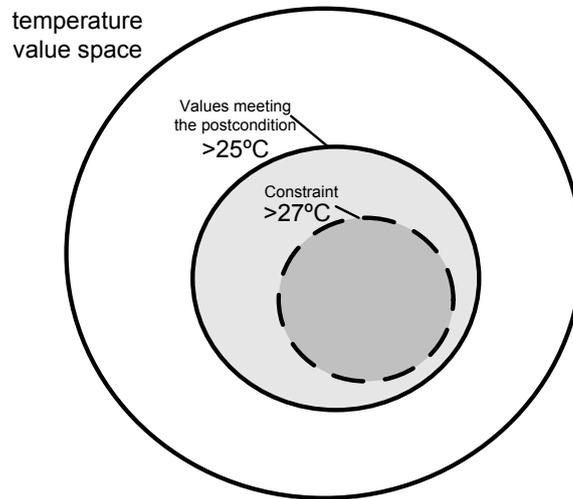


Figure 4.5: Valid values for a postcondition and one constraint representing a subset of it.

However, this set of constraints  $C(I^E(t), bp)$  is not useful in practice, since it has an infinite number of elements. Again, if the postcondition states that  $temperature > 25^\circ C$ , there are infinite constraints that match this postcondition (see Figure 4.6):

$$\begin{aligned}
 &> 25^\circ C \\
 &= 25.1^\circ C \\
 &= 25.2^\circ C \\
 &> 26^\circ C \\
 &> 27^\circ C \\
 &> 28^\circ C \\
 &\dots
 \end{aligned} \quad (4.25)$$

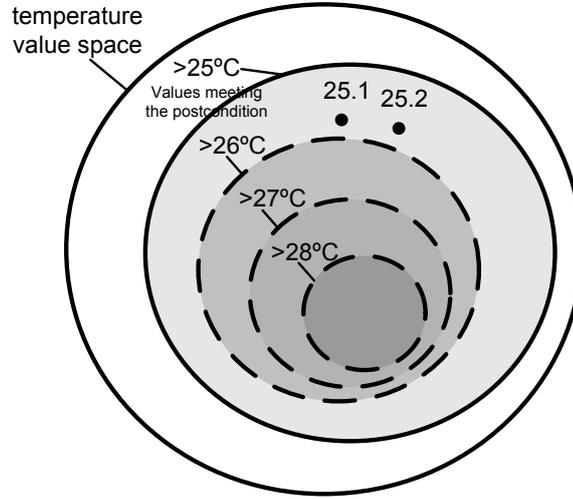


Figure 4.6: An infinite number of constraints as subsets of the postcondition.

For practical purposes, this problem can be solved by identifying the most broad constraint that meets the postcondition. That is,

$$co|co \subseteq G_{POST} \wedge (\nexists co'|co' \subseteq G_{POST} \wedge co \subseteq co') \quad (4.26)$$

That is, to find a subset  $co$  of the postcondition such as no other subset in the postcondition contains  $co$ .

In fact, there is only one possibility in the previous example:

$$co = [ > 25^\circ C ] \quad (4.27)$$

The most broad constraint that embraces a particular postcondition is the postcondition itself. So finally, the formula 4.24 becomes

$$C(I^E(t), bp) = \{ co|co = G_{POST} \wedge G_{POST} \in D_{POST}^{bp} \} \quad (4.28)$$

Which means that the set of constraints generated when the behavioural profile  $bp$  is activated under the context information  $I^E(t)$  is formed by the postconditions of  $bp$ . Our model generates constraints directly from postconditions whenever a behavioural profile is activated.

### 4.3 Semantic Web mapping

Once the theoretical basis is established, the following step is to map the described concepts onto an architectural solution. Since we selected the Semantic Web as the technological foundation for dealing with all the context-aware related requirements, it becomes necessary to identify Semantic Web -based techniques to represent the ideas presented in the previous section.

Particularly, the terms described above that need to find such a suitable representation are:

- Environment
- Entity
- Context information
- Knowledge domain
- Knowledge domain item
- Knowledge domain item value
- Perception capability
- Operation capability
- Constraint
- Adaptation profile
- Precondition
- Postcondition

#### 4.3.1 Environment and Entity

Environment and Entity do not have a particular representation using Semantic Web. Both are architectural concepts that can be directly assumed in our architecture. A context-aware entity can refer to a device, a process, an agent, and so forth, whatever is called the element that takes part in the context-awareness process and exhibits the features described in the theoretical model.

The only point noteworthy is that, following the Web architectural model [Wor04a] in which every resource must be referenced through a URI, the

context-aware entity must be identified unambiguously using a URI. Again, an entity can be a device or a built-in agent in the device, but if they need to be distinctly identified, two different URIs will be used for that purpose.

Some examples of valid URIs for entities are:

```
http://acme.com/products/tv/TVH23/8293984
http://www.nokia.com/phones/9890/SN0670334635
urn:uuid:9f22d136-deb4-4385-886c-47e7b3b60620
http://www.places.com/ivazquez-home/pda23
http://www.books.com/ISBN/1-58113-992-6/83930498
```

The main condition to assign a URI to a context-aware entity is that such assignment must be unique: there must not be another entity in the world with the same URI. This requirement makes the UUID (Universal Unique IDentifiers) URN namespace [LMS05] the most suitable alternative for assigning such sort of “unique serial numbers” to every device or process. Moreover, UUIDs can be allocated and set by the manufacturer of the entity during production, so the final user does not need to assign them (and even this may not be permitted).

### 4.3.2 Context information

This major concept is easily mapped onto an RDF graph representing the existing available information in the environment. This information is provided and RDF-annotated by contributing context-aware entities, creating a graph that represents the situation of the environment at a particular moment of time, associating knowledge domain items and their values.

For instance, continuing the Example 4.1, the light control system (`light1`), the TV set (`tv1`) and the Hi-Fi music system (`hifi1`), as stated by their perception capabilities, could provide the following information to the environment (in RDF/XML) [Wor04i]<sup>2</sup>:

Listing 4.1: RDF/XML representation of context information from the Example 4.1.

```
1 <?xml version="1.0"?>
2 <rdf:RDF
3   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4   xmlns="http://www.awareit.com/onto/examples/example2#">
5
6 <rdf:Description rdf:about="urn:uuid:light1"
```

<sup>2</sup>For the sake of clarity we use non-standard UUID URIs like `urn:uuid:light1` instead of standard ones such as `urn:uuid:a83b0f9d-3999-4cf2-993d-c40054602ec3`.

```
7   xmlns:lit="http://www.awareit.com/onto/2005/12/light#">
8   <lit:luminance rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
9     30
10  </lit:luminance>
11  <lit:color rdf:resource="http://www.awareit.com/onto/2005/12/light#
12    White"/>
13  <rdf:type rdf:resource="http://www.awareit.com/onto/2005/12/light#
14    Light"/>
15  </rdf:Description>
16  <rdf:Description rdf:about="urn:uuid:tv1"
17    xmlns:tv="http://www.awareit.com/onto/2005/12/tv#">
18    <tv:state
19      rdf:resource="http://www.awareit.com/onto/2005/12/devices#stateOn
20        "/>
21    <tv:channel rdf:resource="http://www.bbc.co.uk/bbctwo"/>
22    <rdf:type rdf:resource="http://www.awareit.com/onto/2005/12/tv#TV"/>
23  </rdf:Description>
24  <rdf:Description rdf:about="urn:uuid:hifil"
25    xmlns:sound="http://www.awareit.com/onto/2005/12/sound#">
26    <sound:state
27      rdf:resource="http://www.awareit.com/onto/2005/12/devices#stateOn
28        "/>
29    <rdf:type
30      rdf:resource="http://www.awareit.com/onto/2005/12/sound#
31        SoundSystem"/>
32    <sound:hasSound>
33    <sound:Sound rdf:about="urn:uuid:hifil_sound">
34    <sound:volume rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
35      8
36    </sound:volume>
37    <sound:station rdf:resource="http://www.bbc.co.uk/radio1"/>
38    </sound:Sound>
39    </sound:hasSound>
40    <device:hasTime
41      xmlns:device="http://www.awareit.com/onto/2005/12/devices#"
42      xmlns:time-entry=
43        "http://www.isi.edu/~pan/damlttime/time-entry.owl#">
44    <time-entry:Instant>
45    <time-entry:inCalendarClockDataType
46      rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
47      2003-11-05T14:00:00-8:00
48    </time-entry:inCalendarClockDataType>
49    </time-entry:Instant>
50    </device:hasTime>
51  </rdf:Description>
52 </rdf:RDF>
```

Lines 6–13 comprise the context information provided by `light1`, lines 15–21 those provided by `tv1` and lines 23–48 those by `hifi1`. In this case, the Hi-Fi system has been modelled as a device that generates an internal resource (lines 30–35, `hifi1_sound`) composed of the current station and volume.

For the sake of clarity, an RDF graph representing the knowledge conveyed in the Listing 4.1 is depicted in Figure 4.7. Note the unlabelled node near the bottom at the end of the edge `device:hasTime` from `hifi1` representing an instance of `time-entry:Instant`. Line 41 in the listing declaring this resource has no `rdf:ID` or `rdf:about` tags labelling the node, so it remains blank (anonymous).

The listing is an aggregation of different context informations provided by the environmental entities referred above. Each of these contributions could have been sent in different documents having every entity aggregated them in the form of Listing 4.1, in order to build a model of the context.

### 4.3.3 Knowledge domain

The idea of “ontology” in artificial intelligence fits perfectly to represent the “knowledge domain” concept. An ontology is a data model that represents a domain and the relationships among the objects in order to perform reasoning. A formal definition for ontologies is provided in appendix A.

In order to work with ontologies an ontology language is needed, and the obvious choice, based on Semantic Web technologies, is OWL (Ontology Web Language) [Wor04d] (see subsection A.2).

An ontology language enables engineers and developers to create domain vocabularies in order to represent the elements, classes of elements, properties and relationships among them for a particular knowledge domain.

While other architectures such as CoBrA or Semantic Spaces/SOCAM defined their own ontology vocabulary for the pervasive computing knowledge domain and based their operation primarily on that ontology (SOUPA and CONON respectively) in our model the reuse of existing ones is promoted. For example, if SOUPA, which in turn is an aggregation of newer and existing ontologies, is considered suitable for one concrete application, it may be seamlessly used in SoaM.

One of the major problems with ontologies, as with XML-based languages, is that engineers are reinventing them once and again instead of reusing and extending existing ones in order to create standard or accepted ontologies for concrete knowledge domains [Bon05].

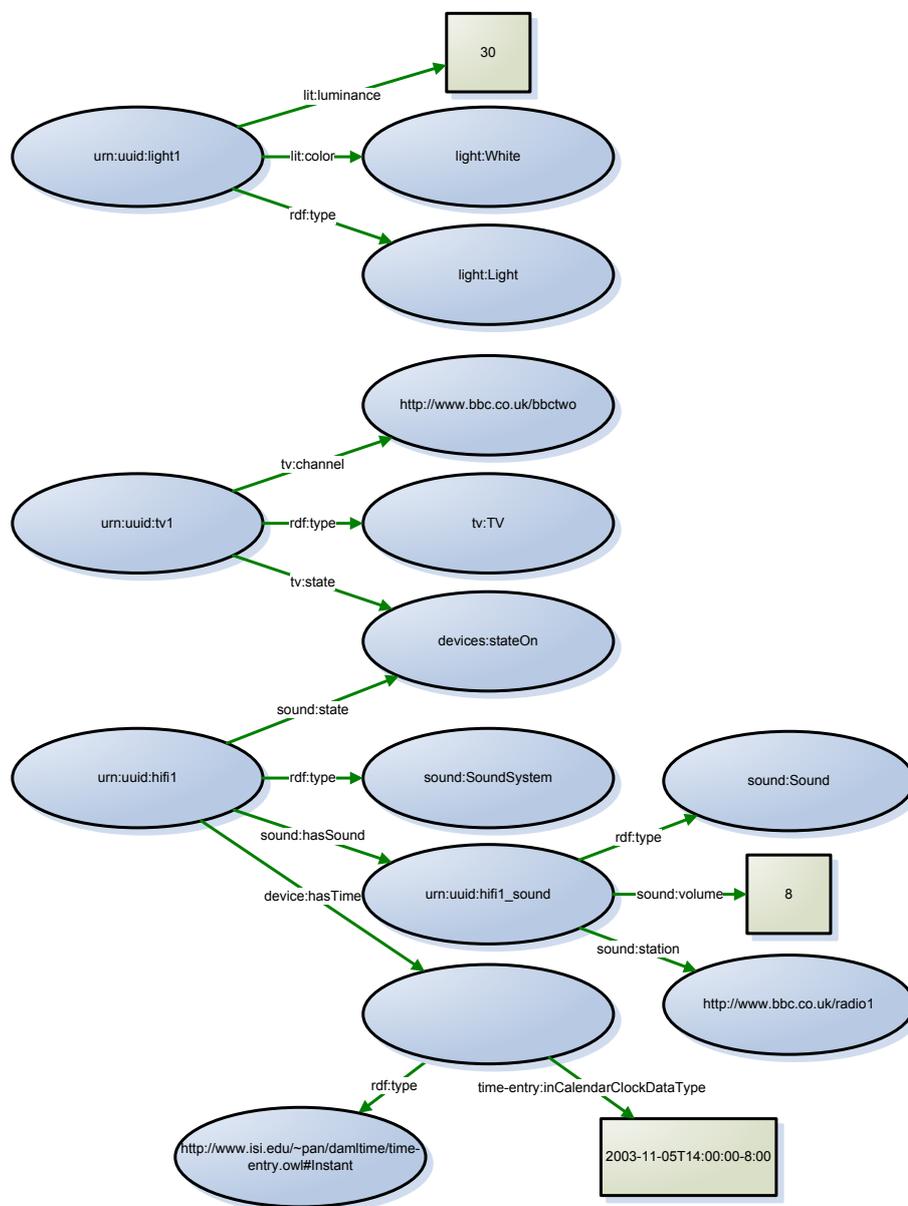


Figure 4.7: RDF graph representing the Listing 4.1.

SoaM will encourage the reuse of existing ontologies for every knowledge domain that partakes in the context-awareness process, and only in the case no ontology is provided, a developer would be required to create one explicitly. Ontology reuse is becoming a hot topic in the Semantic Web arena, and concepts successfully applied in HTML such as modularisation can also play a major role here [GBPK06].

In Listing 4.1 lines 41–46 comprise the representation of the time using an existing ontology called OWL-Time [HP04] [PH05] [Wor06c], thus demonstrating via an example how ontology reuse can be accomplished.

Ontologies, as any other kind of resource, are represented by their unique URIs, from where they can be usually downloaded.

#### 4.3.4 Knowledge domain item

While “knowledge domain” and “ontology” are easily mapped onto each other, “knowledge domain item” does not have an obvious mapping in the Semantic Web. A knowledge domain item is a particular type of information in the domain, for instance in the “TV” knowledge domain, “current channel” will be a “knowledge domain item”.

The problem is that RDF as mechanism for representing context information follows a structure based on (*subject, predicate, object*). Considering that “object” is a good candidate for mapping the “knowledge domain item value”, the item itself must be characterised via the “subject”, the “predicate”, or the combination of both. And this characterisation must be related to the knowledge domain in some way.

Let us suppose some examples of context knowledge items:

- Current temperature
- `hifil`’s volume
- `tv1`’s channel
- Current time
- `light1`’s colour
- ...

It is noticeable how context knowledge items refer to properties or attributes. Those attributes can exhibit a well-defined subject (such as `hifil`, `tv1` or `light1`) or an undetermined subject such as “current temperature” and “current time”. However, in these last cases the subjects exists, although no explicitly stated. In “current temperature” we really mean the “current temperature of a particular (probably the current) location or room”, thus maybe being “my location” the subject of the property. In the second case, we may mean “current time of the present location”, as provided by any entity located nearby.

Therefore, in our model, the knowledge domain items are represented by the combination of a subject and a predicate from the RDF point of view, being this subject more or less explicit, but existent anyway.

Some examples of context knowledge items are<sup>3</sup>:

```
<urn:uuid:light1> lit:luminance
<urn:uuid:light1> lit:color
<urn:uuid:tv1> tv:state
<urn:uuid:tv1> tv:channel
<http://people.com/bobby> task:isDoing
```

### 4.3.5 Knowledge domain item value

As commented above the object part of an RDF triple is able to map the present value of the knowledge domain item perfectly. This object / value can be either a literal value or another resource represented by its URI, as in:

```
<urn:uuid:light1> location:isLocatedIn <urn:uuid:room21>
```

Literal values in RDF are normally represented using XML Schema built-in datatypes [Wor04k] as in lines 8–10 or 31–33 of Listing 4.1 to represent an integer value, as well as in lines 42–45 to represent a date/time value:

```
<urn:uuid:instant1> time-entry:inCalendarClockDataType "2003-11-05T14:00:00-8:00"^^<http://www.w3.org/2001/XMLSchema#dateTime>
```

### 4.3.6 Perception capability

Perception capabilities were defined in subsection 4.2.2 as the set of knowledge domain items the entity is able to perceive, either directly or indirectly to refer the direct or indirect perception capabilities respectively.

Since knowledge domain items were mapped onto RDF as the combination of a subject and a predicate, an initial approach yields combinations of subjects and predicates, whatever the object is, as the way of representing perception capabilities.

---

<sup>3</sup>From now on we will use a Notation 3 (N3) -like syntax [Ber06]. For the sake of clarity, we will also use intuitive prefixes for ontology namespaces such as `lit:`, `tv:`, `sound:` and so forth.

For instance, to declare that `light1` is able to perceive its own luminance and colour, whatever their values are, we can use the following RDF triple template:

```
<urn:uuid:light1> lit:luminance *  
<urn:uuid:light1> lit:color *
```

This template represents the RDF triples of context information the entity is able to provide: it can generate RDF triples with `urn:uuid:light1` as subject, `lit:luminance` as predicate and *any* value (the actually measured one) as object.

If `tv1` was able to provide information about its own state, channel and Bob's activity:

```
<urn:uuid:tv1> tv:state *  
<urn:uuid:tv1> tv:channel *  
<http://people.com/bobby> task:isDoing *
```

It is noteworthy how a context-aware entity is not only able to provide information about itself, but also about other entities, objects or resources, as long as it features the mechanisms to perceive that information.

Thus, every context-aware entity could declare in this way its perception capabilities about context information (knowledge domain items).

However, this approach lacks the required flexibility in the cases where the subject of the RDF triple is unknown. Some examples are:

- “able to perceive any user’s presence”: the property / predicate is “presence”, but the subject, user, is presently unknown.
- “able to perceive current location’s temperature”: the location is undetermined, it will depend on where the thermometer is.
- “able to perceive any user’s activity”: can perceive whether the user is watching the TV or not, but once the user has been identified.

In this cases, the wildcard can be used in the “subject” part, therefore representing perception capabilities in the following way:

```
* location:isLocatedIn *  
* temp:hasTemperature *  
* task:isDoing *
```

If `room21` is able to determine whether a currently unknown user is in or not, being `room21` the only possibility for the “object” part of the RDF triple, this may be represented this way:

```
* location:isLocatedIn <urn:uuid:room21>
```

Meaning that this entity is able to provide perceived context information following the template “x is located in `room21`”.

However, in most cases the “object” part, representing the value, is the unknown factor and the one required to finally characterise the measure.

By applying the wildcard to the “predicate” part of the RDF triple, an entity can declare that is able to provide any kind of information about a concrete resource. For example:

```
<urn:uuid:hifi1> * *
```

But this kind of template is too loose and provides almost no hints about what the entity is actually able to perceive. If the entity is able to perceive any predicate of the “sound” knowledge domain related to `hifi1`, it may be better to use this representation:

```
<urn:uuid:hifi1> sound:* *
```

Thus limiting the number of predicates perceivable to those belonging to the sound domain. If `state` and `hasSound` were the only predicates for that domain, the last representation would be equivalent to:

```
<urn:uuid:hifi1> sound:state *  
<urn:uuid:hifi1> sound:hasSound *
```

In this example `hifi1` is also able to provide the volume and station of its internally generated sound `hifi1_sound`, the current time and its own type, so the complete set of direct perception capabilities could be:

```
<urn:uuid:hifi1> rdf:type *  
<urn:uuid:hifi1> sound:state *  
<urn:uuid:hifi1> sound:hasSound *  
<urn:uuid:hifi1> device:hasTime *  
<urn:uuid:hifi1_sound> sound:volume *  
<urn:uuid:hifi1_sound> sound:station *
```

Therefore, wildcards, when used in the appropriate places, create a template that represents the RDF triples an entity is able to generate out of environmental perceptions: its perception capabilities.

### 4.3.7 Operation capability

The representation of operation capabilities using Semantic Web technologies is analogous to the perception capabilities strategy. RDF triple

templates can be used to represent which context information an entity is able to alter through operations, either direct or indirectly.

For instance, following the previous example, `tv1` could declare its direct operation capabilities as:

```
<urn:uuid:tv1> tv:state *
<urn:uuid:tv1> tv:channel *
```

That is, `tv1` is able to adjust its state and channel as requested by other entities.

A more complex example involving `hifil` operation capabilities:

```
<urn:uuid:hifil> sound:state *
<urn:uuid:hifil_sound> sound:volume *
<urn:uuid:hifil_sound> sound:station *
```

`hifil` declares that is able to modify its own state, as well as the volume and station of the generated sound `hifil_sound`, but it is not able to change any of the other context information that were perceived, such as its type or current time.

### 4.3.8 Constraint

A constraint was defined as a particular restriction over the set of possible values a concrete knowledge domain item is able to acquire, and we went through several examples such as temperature [ $> 23^{\circ}C$ ], [ $< 27^{\circ}C$ ], [ $= 19^{\circ}C$ ] or [ $= 25^{\circ}C$ ].

Constraints, as explained, represent subsets of possible values adopted by a knowledge domain item, which we have modelled using Semantic Web technologies as the combination of “subject” and “predicate”. Thus, the constraint “the luminance of `light1` must be greater than 5” can be represented as:

```
<urn:uuid:light1> light:luminance ">" "5"
```

Or “the volume of the generated sound `hifil_sound` from `hifil` must be greater or equal than 5”:

```
<urn:uuid:hifil_sound> sound:volume "≥" "5"
```

And “the channel of `tv1` must be CNN and the volume less than 7”:

```
<urn:uuid:tv1> tv:channel "=" <http://www.cnn.com/CNN>
<urn:uuid:tv1> sound:volume "<" "7"
```

As it can be noticed, in order to represent constraints we add a fourth element to the RDF triple: a comparison operator. In order to simplify the notation the operator *equals* is considered as the default when no other is provided. So the previous example about the TV channel could also be represented as:

```
<urn:uuid:tv1> tv:channel <http://www.cnn.com/CNN>
```

### 4.3.9 Behavioural profile, precondition and postcondition

Behavioural profiles represent a facet of adaptive behaviour in the system formed by preconditions and postconditions in such a way that preconditions are evaluated against the current context information, and if matched, the behavioural profiles are activated and context-aware entities should perform the required operations for the postconditions to be honoured.

As seen above in our Semantic Web mapping, we represent conditions as RDF triples with an extra comparison operator. Thus, behavioural profiles' preconditions and postconditions are represented in this way.

For example, the profile “if *tv1* is on and *light1*'s luminance is less than 4, the sound volume of *hifi1* must be 0” can be represented as:

```
[  
<urn:uuid:tv1> tv:state <http://www.awareit.com/onto/2005/12/devices#  
  stateOn>  
<urn:uuid:light1> light:luminance "<" "4"  
]  
⇒  
[  
<urn:uuid:hifi1_sound> sound:volume "0"  
]
```

In order to carry out more complex behaviours, variables are needed. For instance, “if *tv1*'s volume is *x* then the volume of the sound generated by *hifi* must be the same”:

```
[  
<urn:uuid:tv1> sound:volume ?x  
]  
⇒  
[  
<urn:uuid:hifi1_sound> sound:volume ?x  
]
```

Variables actually generate RDF triple templates where the variable is the unknown part. In the last example, the context information is checked against the precondition, maybe an RDF triple honouring the template exists, such as:

```
<urn:uuid:tv1> sound:volume "4"
```

So, the variable *x* can be resolved and assigned that value.

At this time constraints are generated by performing the substitution on the postconditions:

```
<urn:uuid:hifil_sound> sound:volume "4"
```

Checking the operation capabilities of existing devices, *hifil* declared itself as able to change this context information directly:

```
<urn:uuid:hifil_sound> sound:volume *
```

So finally, *hifil* is assigned this constraint and operates the built-in effectors adjusting the volume at the desired level: the environment has changed according to the behavioural profile in a Semantic Web-based scenario.

## 4.4 Serialisation

Context information can be exchanged directly using RDF/XML as illustrated in Listing 4.1. However, the other structures (perception and operation capabilities, constraints and behavioural profiles with pre- and postconditions) require a suitable and structured representation so that entities can easily exchange this information.

For this purpose we have designed two XML-based languages:

- SoaM XML Datatypes: XML datatypes to represent the internal structure of capabilities, constraints and behavioural profiles.
- SoaM XML Exchange Messages: XML wrappers for conveying SoaM XML Datatypes over transport mechanisms such as HTTP.

We have designed these XML-based grammars in such a way that they are simple enough even for resource limited devices to be able to use them without much processor load.

The complete XML Schema definitions for these languages can be found in appendix D, but some paradigmatic examples are included in this section for the sake of understanding the flow of messages.

#### 4.4.1 Capabilities

As previously explained, capabilities are represented using the URIs for the concrete resources and predicates required, forming a template. In order to represent the wildcard value \* we have created a special URI:

```
http://www.awareit.com/soam/2005/12/soamonto#Any
```

For instance, a location system can declare the capability to provide the location of any *entity*:

Listing 4.2: Example of a capability using the *any* wildcard.

```

1 <perceptionCapability id="urn:uuid:loc_pcap">
2   <subject resource="http://www.awareit.com/soam/2005/12/soamonto#Any
   "/>
3   <predicate
4     resource="http://www.awareit.com/onto/2005/12/location#isLocatedIn
   "/>
5 </perceptionCapability>
```

The “object” part is optional, usually absent and so being equivalent to *any* or \* (e.g. in the previous case, any place).

The same example but involving a concrete object:

Listing 4.3: Fragment of a capability involving a concrete object.

```

1 <perceptionCapability id="urn:uuid:loc_cap">
2   <subject resource="http://www.awareit.com/soam/2005/12/soamonto#Any
   "/>
3   <predicate
4     resource="http://www.awareit.com/onto/2005/12/location#isLocatedIn
   "/>
5   <objectResource resource="urn:uuid:room21"/>
6 </perceptionCapability>
```

Which represents the case analysed in 4.3.6, where a location system can provide location information of *any* entity located in `room21`:

```
* location:isLocatedIn <urn:uuid:room21>
```

A more complex and complete example about `hifil`'s capabilities is:

Listing 4.4: Example capabilities in SoaM XML Datatypes.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <capabilitiesCollection owner="urn:uuid:hifil"
3   xmlns="http://www.awareit.com/soam/2006/02/soamdt"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://www.awareit.com/soam/2006/02/soamdt http
   ://www.awareit.com/soam/2006/02/soamdt">
```

```

6
7 <perceptionCapability id="urn:uuid:hifil_pcap1">
8   <subject resource="urn:uuid:hifil"/>
9     <predicate resource="http://www.awareit.com/onto/2005/12/sound#
      hasSound"/>
10 </perceptionCapability>
11
12 <perceptionCapability id="urn:uuid:hifil_pcap2">
13   <subject resource="urn:uuid:hifil"/>
14   <predicate
15     resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
16 </perceptionCapability>
17
18 <perceptionCapability id="urn:uuid:hifil_pcap3">
19   <subject resource="urn:uuid:hifil_sound"/>
20   <predicate resource="http://www.awareit.com/onto/2005/12/sound#
      volume"/>
21   <predicate resource="http://www.awareit.com/onto/2005/12/sound#
      station"/>
22 </perceptionCapability>
23
24 <operationCapability id="urn:uuid:hifil_ocap1">
25   <subject resource="urn:uuid:hifil"/>
26   <predicate resource="http://www.awareit.com/onto/2005/12/sound#
      hasSound"/>
27 </operationCapability>
28
29 <operationCapability id="urn:uuid:hifil_ocap2">
30   <subject resource="urn:uuid:hifil_sound"/>
31   <predicate resource="http://www.awareit.com/onto/2005/12/sound#
      volume"/>
32   <predicate resource="http://www.awareit.com/onto/2005/12/sound#style
      "/>
33 </operationCapability>
34
35 </capabilitiesCollection>

```

The element `capabilitiesCollection` comprises all the capabilities, both perception and operation related ones. Each of these capabilities is in turn composed of an undetermined number of elements `subject`, `predicate`, `ontology` or `object` (`objectLiteral` as well as `objectResource`).

The actual individual capabilities are obtained by “unwrapping” a concrete SoaM XML Datatypes capability element, thus generating all the possible combinations of the three groups: `subject`, `predicate` or `ontology`, and `objectLiteral` or `objectResource`.

For instance, the perception capability in lines 18–22 comprising one subject and two predicates is unwrapped as two capabilities, combining that subject with each of the predicates:

```
<urn:uuid:hifi1_sound> sound:volume *
<urn:uuid:hifi1_sound> sound:station *
```

If two subjects and two predicates were declared in this capability, the unwrapped number of actual capabilities would be 4; if two subjects and three predicates, the unwrapped number would be 6, and so forth.

This compressed form of representing capabilities in SoaM XML Datatypes saves space while retaining the same expressive power.

It is also noteworthy how each capability is assigned an unique `id` generated by the issuer in order to unambiguously identify the capability later if needed (e.g. managing, debugging, or logging)<sup>4</sup>.

Listing 4.5 represents a perception capability comprising all the predicates in a ontology (knowledge domain):

Listing 4.5: Fragment of a capability involving all the predicates in an ontology.

```
1 <perceptionCapability id="urn:uuid:hifi1_pcap">
2   <subject resource="urn:uuid:hifi1"/>
3   <ontology resource="http://www.awareit.com/onto/2005/12/sound"/>
4 </perceptionCapability>
```

Which in turn represents the notation in subsection 4.3.6:

```
<urn:uuid:hifi1> sound:* *
```

The `ontology` element is an abbreviation for all the predicates included in the referred ontology. It can be used in the cases where all the predicates are supported in the capability, in order to save space and enforce clarity. This notation can also benefit from ontology modularisation strategies [GBPK06], since embracing all the predicates in one concrete ontology is more feasible if the latter has a limited size (such as ontologies that are split up in different modules).

---

<sup>4</sup> The use of `xml:id` [Wor05c] here was proposed and rejected since it does not permit full URIs, just fragment identifiers relative to the base URI of the document. Then, the combination of `xml:id` and `xml:base` [Wor01b] to build full URIs was explored. Since the capability must have a unique identifier, thus using UUID in the `xml:base`, the resulting capability URI would be, for example:

```
urn:uuid:b9351a93-a08e-4588-b3e1-9b0058d62522#hifi1_pcap1
```

Which violates the UUID construction rules, where a fragment part is not permitted. Therefore, we finally opted for creating our own `id` attribute for every structure.

## 4.4.2 Constraints

The following example illustrates a message conveying two constraints, the first one can be read as “`hifil_sound` must have a volume level less than 7”, while the second can be read as “its station must be set to BBC Radio 4”:

Listing 4.6: An example of constraints usage.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <constraintsCollection xmlns="http://www.awareit.com/soam/2006/02/
   soamdt "
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://www.awareit.com/soam/2006/02/soamdt
5   http://www.awareit.com/soam/2006/02/soamdt "
6   owner="urn:uuid:hifil">
7
8   <constraint id="urn:uuid:hifil_con1"
9     requester="http://people.com/bobby"
10    subject="urn:uuid:hifil_sound"
11    predicate="http://www.awareit.com/onto/2005/12/sound#volume"
12    operator="http://www.awareit.com/soam/2005/12/soamonto#LessThan"
13    expires="PT5M">
14     <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int">
15       7
16     </objectLiteral>
17   </constraint>
18
19   <constraint id="urn:uuid:hifil_con2"
20     requester="http://people.com/bobby"
21     subject="urn:uuid:hifil_sound"
22     predicate="http://www.awareit.com/onto/2005/12/sound#station"
23     expires="PT10M">
24     <objectResource resource="http://www.bbc.co.uk/radio4"/>
25   </constraint>
26
27 </constraintsCollection>

```

The first noticeable aspect is the `owner` attribute in the root element (line 6), indicating the entity whose behaviour is being driven by these constraints. Every constraint has a `requester` attribute (lines 9 and 20) referencing the resource (device, person, process) who actually asked for that constraint to be honoured.

The constraint itself is represented via the `subject`, `predicate`, `operator` (optional) and `object` (`objectLiteral` or `objectResource`) parts. The first constraint features a typed literal as object, while the second features a resource.

The `expires` attribute informs about the time left for this constraint to expire (five minutes in the first case and 10 in the second), according to the `xs:duration` XML data type [Wor04k] whose lexical form is described in the standard ISO 8601 [Int04].

### 4.4.3 Behavioural profiles

Since behavioural profiles represent behaviour in the form of pseudo-rules, different existing XML-based languages could have been adopted, being the most suitable RuleML [BTW01] and SWRL (Semantic Web Rules Language) [HPSB<sup>+</sup>04].

However, we decided not to take these ongoing efforts as a basis and to design a particular representation for the behavioural profiles. The main reasons for this decision were:

- SWRL seemed more suitable for dealing with semantic information and resources, so RuleML was a weak candidate from the beginning.
- Neither SWRL nor RuleML are presently considered standards. The World Wide Web Consortium has recently appointed a new working group<sup>5</sup>, the Rule Interchange Format (RIF) Working Group, for designing a standardised web rules interchange language<sup>6</sup>.
- These languages consider rules as axioms. For instance, in the rule “if the light is on, then the TV is off”, whenever the precondition is met, the information generated from the postcondition is added to the knowledge base as a true fact. Behavioural profiles do not behave in this way: if the preconditions are met, entities must do the best effort for the postconditions to be met, but they may actually never be honoured under certain situations (e.g. no entity features such operation capability, no permissions, the TV is locked, and so forth). Therefore, behavioural profiles, despite featuring a rule-like syntax, cannot be conceptually considered traditional inference rules. While preconditions can be evaluated, postconditions generate constraints that are intended to be honoured, but cannot be acknowledged as current facts: they are *potential* future facts.

However, RuleML, SWRL and the new language resulting from the W3C RIF WG are suitable candidates for representing domain-related rules during

---

<sup>5</sup>November 2005.

<sup>6</sup>The RIF WG will develop the recommendation in three phases with expected outcomes in May 2007, June 2008 and June 2009 [Wor05b].

the reasoning phase of the context awareness process (see subsection 4.2.4), inferring new data from existing data and, thus, augmenting the context information prior to the analysis phase.

In order to illustrate our syntax, let us suppose a behavioural profile stating “if Bob is in `room21` then the temperature must be set at 24°C”. This profile can be represented in SoaM XML Datatypes as:

Listing 4.7: A simple behavioural profile.

```

1 <behavioralProfile id="urn:uuid:prof1" expires="PT2M"
2   requester="http://people.com/bobby">
3
4   <precondition id="urn:uuid:prof1_precl"
5     subject="http://people.com/bobby"
6     predicate="http://www.awareit.com/onto/2005/12/location#isLocatedIn
7       ">
8     <objectResource resource="urn:uuid:room21"/>
9   </precondition>
10
11   <postcondition id="urn:uuid:prof1_postcl"
12     subject="urn:uuid:room1"
13     predicate="http://www.awareit.com/onto/2005/12/temperature#
14       hasTemperature">
15     <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int">
16       24
17     </objectLiteral>
18   </postcondition>
19 </behavioralProfile>

```

It is important to remark how the behavioural profile itself, its preconditions and postconditions are each uniquely identified through the `id` attribute for future reference. The `expires` attribute attached to the profile informs about its validity period (2 minutes in the above example, line 1).

For information about how the postcondition in lines 10–16 can be transformed into a constraint, please refer to subsection 5.1.6.

The possibilities of behavioural profiles are dramatically increased by applying variables, improving the level of expressiveness and allowing complex behaviour to be articulated.

For example, the behaviour “if Bob is working with the laptop in a room with ambient sound, then the sound volume must be set at 3 and the radio station to BBC Radio 4”, can be represented as:

Listing 4.8: A more complex behavioural profile with variables.

```

1 <behavioralProfile id="urn:uuid:prof2" expires="PT2M"
2   requester="http://people.com/bobby">
3
4   <variable xml:id="x"/>
5   <variable xml:id="y"/>
6
7   <precondition id="urn:uuid:prof2_prec1"
8     subject="http://people.com/bobby"
9     predicate="http://www.awareit.com/onto/2005/12/task#isDoing">
10    <objectResource
11      resource="http://www.awareit.com/onto/2005/12/task#
12        WorkingWithLaptop"/>
13  </precondition>
14
15  <precondition id="urn:uuid:prof2_prec2"
16    subject="http://people.com/bobby"
17    predicate="http://www.awareit.com/onto/2005/12/location#isLocatedIn
18      ">
19    <objectVariable ref="x"/>
20  </precondition>
21
22  <precondition id="urn:uuid:prof2_prec3"
23    subject="#x"
24    predicate="http://www.awareit.com/onto/2005/12/sound#hasSound">
25    <objectVariable ref="y"/>
26  </precondition>
27
28  <postcondition id="urn:uuid:prof2_postc1"
29    subject="#y"
30    predicate="http://www.awareit.com/onto/2005/12/sound#volume">
31    <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int">
32      3
33    </objectLiteral>
34  </postcondition>
35
36  <postcondition id="urn:uuid:prof2_postc2"
37    subject="#y"
38    predicate="http://www.awareit.com/onto/2005/12/sound#station">
39    <objectResource resource="http://www.bbc.co.uk/radio4"/>
40  </postcondition>
41 </behavioralProfile>

```

This behavioural profile declares two variables, *x* and *y*, that will be used in its scope. It is noteworthy how `xml:id` can be now applied to identify the variables, since they are always used within the document scope and do not

require to be uniquely identified from external sources (so, UUIDs are not required to name them).

The variables are resolved against existing context information adopting concrete values. These values are exported onto the postconditions which are, by this operation, “transmuted” into constraints and can be sent to the appropriate entities.

For example, let us suppose a small excerpt of the context information at a particular moment of time to be:

```

1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:dc="http://purl.org/dc/elements/1.1/"
4   xmlns:task="http://www.awareit.com/onto/2005/12/task#"
5   xmlns:loc="http://www.awareit.com/onto/2005/12/location#"
6   xmlns:sound="http://www.awareit.com/onto/2005/12/sound#">
7
8   <rdf:Description rdf:about="http://people.com/bobby">
9     <task:isDoing
10      rdf:resource="http://www.awareit.com/onto/2005/12/task#
11      WorkingWithLaptop"/>
12     <loc:isLocatedIn rdf:resource="urn:uuid:room21" />
13   </rdf:Description>
14
15   <rdf:Description rdf:about="urn:uuid:room21">
16     <sound:hasSound>
17       <sound:Sound rdf:about="urn:uuid:hifil_sound">
18         <sound:volume rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
19           7
20         </sound:volume>
21         <sound:station rdf:resource="http://www.eitb.com/ei"/>
22       </sound:Sound>
23     </sound:hasSound>
24   </rdf:Description>
25 </rdf:RDF>

```

Lines 8–10 can be provided by the user’s laptop, which disseminates information about whether the user is working with it or not. Location information is provided by a location system, while sound information is provided by a Hi-Fi system.

If the behavioural profile is evaluated against this context information, the variables would be resolved as:

```

x = <urn:uuid:room21>
y = <urn:uuid:hifil_sound>

```

And the generated constraints would be:

Listing 4.9: Generated constraints from the behavioural profile of Listing 4.8.

```
1 <constraint id="urn:uuid:con1"
2   requester="http://people.com/bobby"
3   subject="urn:uuid:hifil_sound"
4   predicate="http://www.awareit.com/onto/2005/12/sound#volume"
5   expires="PT2M">
6 <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int">
7   3
8 </objectLiteral>
9 </constraint>
10
11 <constraint id="urn:uuid:cond2"
12   requester="http://people.com/bobby"
13   subject="urn:uuid:hifil_sound"
14   predicate="http://www.awareit.com/onto/2005/12/sound#station"
15   expires="PT2M">
16 <objectResource resource="http://www.bbc.co.uk/radio4"/>
17 </constraint>
```

And, hopefully, the context information would change eventually to fulfil the constraints.

Listing 4.8 illustrates how a relative complex context-aware behaviour can be embodied into a behavioural profile using SoaM XML Datatypes, in such a way that enables very rich expressions and, thus, smart environmental behaviour.

## 4.5 Summary

We consider the design of a theoretical framework as a basis for the subsequent architectural design to be a remarkable outcome that distinguishes our work from other initiatives.

We identified passive influence as a non-intrusive model to provide reactivity in embedded platforms attached to everyday objects. Context modification and behavioural profiles dissemination were also identified as practical mechanisms to implement the passive influence model.

We consider passive influence to be an interesting and novel concept to approach how devices and environments may react to different *stimuli*.

We proposed a set-theory based approach for representing the context-awareness process, providing some definitions, and identifying the main entities and concepts as well as their relationships via a number of propositions and formulas.

We designed a mapping in order to represent these relationships among concepts and influences in an environment from a Semantic Web point of view that would enable the integration of the theoretical model with a concrete implementation technology. It is noteworthy how the mapping process produces similar structures based on RDF triples, using templates and variable substitution.

Especially remarkable is the possibility of describing a whole knowledge domain as a capability via the construction `ontology:*`, as in `sound:*` to represent the ability of describing a resource using all the terms in a concrete domain. Finally, we designed an XML-based serialisation model for exchanging basic structures such as capabilities, constraints and behavioural profiles.

We consider our theoretical model to be highly consistent and appropriate to represent the context-awareness process, seamlessly integrated with the technologies we wanted to apply and a convenient foundation for the subsequent architectural design.



## SoaM Architecture

*“If you wish to make an apple pie from scratch,  
you must first invent the universe.”*

*Carl Sagan  
Cosmos, 1980*

**F**OLLOWING the theoretical principles depicted in the previous chapter, we have designed a distributed architecture that fulfills the main requirements and evaluation criteria established in section 2.1.

Part of the architectural design of SoaM, tightly related to the Semantic Web mapping of the theoretical concepts as well as the XML serialisation of the required structures, has already been introduced in section 4.4.

In this chapter we will focus on the features of SoaM that constitute the core of the architecture and the basis of its innovation compared to previous initiatives:

- Semantic Web technologies, mainly ontologies and vocabularies, for representing not only context information but also devices' behaviour.
- A completely decentralised communication model where intelligent semantic devices share a common knowledge of the environment, and react accordingly.
- A novel mechanism and protocol for semantic discovery in Ubiquitous Computing environments: mRDP.
- A unique, comprehensive and integrated communication model based on web technologies.

- A flexible mechanism to influence devices's behaviour by users and other devices.
- Feasible for implementation in embedded platforms.
- Extensible through additional elements to take advantage of available computing facilities in the environment.

These innovations materialise through the different outcomes of the architecture:

- Architectural elements: smobjects, orchestrators and BPinjectors.
- SoaM phases specification: the context-awareness process and reactivity in the SoaM architecture.
- SoaM Entity Management API – HTTP Binding: HTTP extensions for communication among architectural elements.
- SoaM topologies: the different deployment strategies for SoaM elements.
- SoaMonto: the SoaM support ontology.

The design process has been carried out using an iterative model, highly linked with the implementation activity, during which increasingly more complete versions of the prototype were implemented and tested, thus validating both the theoretical model and the architecture.

Using SoaM, devices are able to discover each other and their capabilities via mRDP; they are also able to share their perceptions in order to create a common semantic space of knowledge about what is happening in their environment. By interpreting and contextualising this shared information via Semantic Web technologies they can behave accordingly and react to the different situations and events. Moreover, their behaviour is not statically programmed, but users or other devices can inject new “behaviours” into them to augment their sensitivity to new *stimuli* and modify their reactivity mechanism at a higher level.

The following sections illustrate different aspects of the SoaM architecture.

### 5.1 Smobject

Semantic devices in SoaM should be able to perceive and share information in order to create a common knowledge space; they should analyse and interpret this context information, and carry out the appropriate behaviour.

These contex-aware devices must feature semantic processing capabilities as well as decentralised peer-to-peer discovery and communication mechanisms, while still being feasible for implementation in embedded platforms with limited resources.

A *smobject* (a portmanteau for “smart semantic object”) is the software agent we have designed for representing this context-aware device in the SoaM architecture. A sole smobject can act on behalf of a physical device, several devices or even part of a device.

Smobjects are able to perceive external *stimuli*, according to their perception capabilities, semantically annotate and share this context information with other smobjects in the environment, augmenting and interpreting this common knowledge through reasoning, and finally perform accordingly to their behavioural rules, thus carrying out a context-aware reactivity.

We have designed the internal architecture of the smobject as a composition of different functional components. These components can be grouped into two major sets:

- Base components: they provide the very basic operation for perceiving and sharing information with fellow smobjects and are always active.
- Awareness components: they provide context-aware and reasoning capabilities, becoming active only when autonomous intelligent reactivity from the smobject is required.

That is, base components contribute to the creation of a semantic context information space among smobjects in the environment, while awareness components embody the intelligent reactivity that emerges when smobjects reason and interpret this information space.

Regarding the awareness components activation, smobjects are intrinsically powered with reasoning capabilities but, since they are hosted in resource-constrained devices, their reasoning power must be economised and used only when required. For example, if other external entities are present in the environment, e.g. full computers, able to provide a higher degree of reasoning power without such resource limitations (memory, processor and battery), smobjects can transfer the reasoning responsibility to them (see 5.2).

If the awareness components are active, smobjects are able to perform the reasoning processes themselves, with no need from external elements, reasoning and interpreting the information to react accordingly.

This flexible scheme makes smobjects able to cope and adapt dynamically to a broad range of variable scenarios, trying always to obtain

the most from existing facilities in order to develop their context-aware behaviour.

The awareness-based operation mode contributes to the creation of intelligent environments populated by autonomous devices that collaborate spontaneously, and embraces the more important challenges of our research.

### 5.1.1 Base components

The goal of base components is to contribute to the creation of a common semantic information space among existing smobjects. They can also provide the means for the smobject to be controlled by a more intelligent external entity if required. In order to do so, they expose several services through their communication interfaces to other entities (see Figure 5.1):

- Discovery: the smobject replies to discovery requests issued by other entities.
- Capabilities retrieval: it provides descriptions about smobject perception and operation capabilities to requesting parties.
- Context information retrieval: it provides the context information captured by the smobject, as stated in its perception capabilities, to requesting parties.
- Constraints management: it provides management operations over the constraints that drive the smobject's behaviour as declared in its operation capabilities.

Smobjects can also access built-in sensors, actuators, communication ports and maybe storage facilities if available in the host device to carry out the required processes.

In some way, the smobject is designed as a “black box”, transforming a host device into a SoaM-compatible context-aware entity: a semantic device. Device sensors are declared via the perception capabilities, while actuators are represented by the smobject's operation capabilities, in both cases using the appropriate domain ontologies.

Context information provided by the smobject is a semantic representation of the actual information captured through the sensors, augmented with other relevant data such as device ID, type, manufacturer and so forth.

The constraints management interface is aimed at influencing the behaviour of the smobject by sending or removing relevant constraints

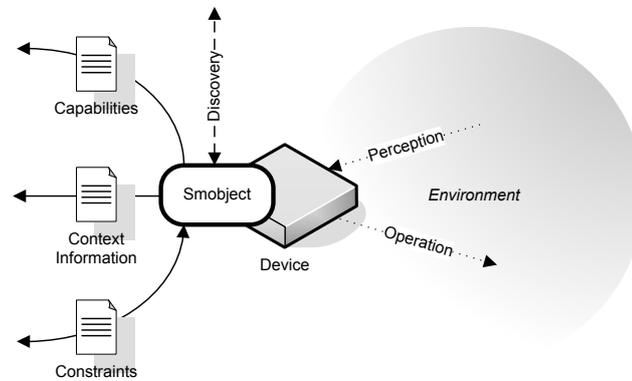


Figure 5.1: Smobject base components communication interfaces.

by external entities, ultimately affecting the actuators and eventually performing a change in the state of the device or environment.

The internal structure of the smobject base components is formed by several functional elements that carry out the required processes and ultimately connect device sensors and actuators to the communication interfaces as depicted in Figure 5.2.

We have classified the base components into three different groups:

- Base core components: the components that implement the main management functionality and its communication interfaces. These components use a discovery protocol (mRDP) and an HTTP interface for communicating with other smobjects and external entities.
- Platform interfaces: these components act as intermediaries between the base core components and the device specific hardware. They form the glue that sticks together and integrates base core components with the actual built-in platform components, such as sensors and actuators.
- Built-in platform components: the built-in low level specific elements that embody device's capabilities, generally formed by sensors and actuators.

This classification helps to understand the different layers for perceiving data through sensors and performing operations under request using the available actuators. This separation also simplifies the development and integration of different sensors and actuators into a device, and their connections with the base components, without affecting other elements.

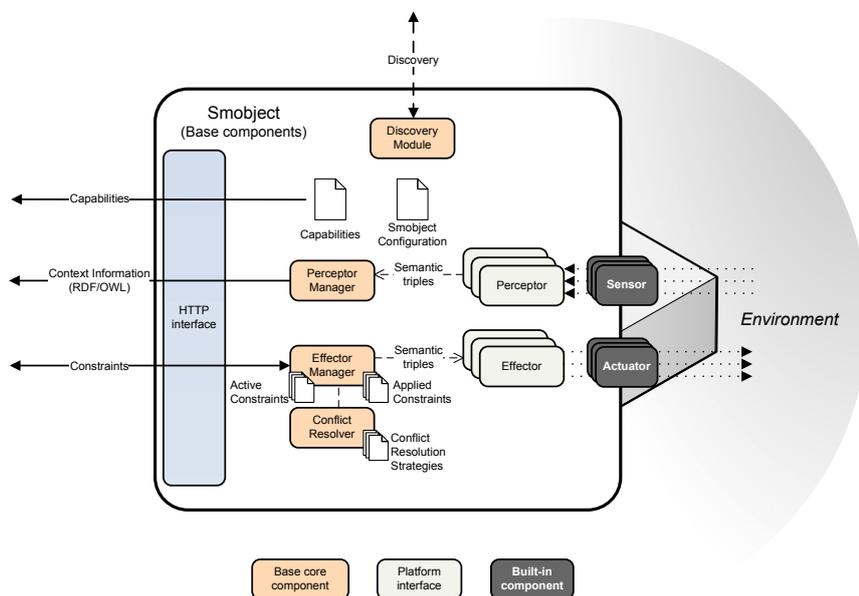


Figure 5.2: Smobject internal structure with base components.

### 5.1.2 Base core components

There are four base core components in a smobject:

- **Discovery Module:** is the component in charge of receiving search requests from other entities and replying accordingly. It implements the discovery mechanisms explained in chapter 3 and section 5.7.
- **Perceptor Manager:** is the component in charge of managing the perceptor interfaces, gathering all the semantic information from them, ultimately obtained via built-in device sensors, in order to make it available to requesting parties via the HTTP interface.
- **Effector Manager:** is the component in charge of managing the effector interfaces, which ultimately operate the built-in device actuators. The Effector Manager receives the constraints that drive smobject's behaviour (*active constraints*), resolves conflicts among them if required using the Conflict Resolver, to finally generate the *applied constraints* for the effector interfaces. This component also manages the constraints life-cycle, removing periodically those that expire. A discussion about active and applied constraints, and how they are related can be found in subsection 5.1.3.

- **Conflict Resolver:** is the component that performs conflict resolution among the set of received active constraints to generate the final set of applied constraints that will be effectively applied.

These components, performing the high-level management functions, are identical for every smobject whatever its purpose. Smobjects can behave very differently and be applied for a broad range of solutions by creating environment specific platform interfaces, as explained in the next subsection, however, the base core components (as well as the awareness components, explained in 5.1.5) remain immutable, providing management functions and platform-agnostic functionality.

### 5.1.3 Platform interfaces

We have designed two categories of platform interfaces for smobjects:

- **Perceptors:** they act as interfaces between the Perceptor Manager and the built-in device sensors. Perceptors are connected to device sensors using platform-specific libraries, APIs or system calls. Every perceptor implements at least one of the declared perception capabilities of the smobject. They collect the information read by the local-level sensor, annotate it semantically using the appropriate knowledge domain vocabulary (or ontology) depending on the information nature, and make that information available to the Perceptor Manager under request.
- **Effectors:** they act as interfaces between the Effector Manager and the built-in device actuators. Very much as perceptors, effectors are connected to device actuators using platform-specific libraries, APIs, system calls or any other mechanism. Every effector is responsible for at least one declared operation capability at the smobject. The Effector Manager provides the effectors with the constraints they are able to process depending on their operation capabilities. Afterwards, these logical effectors act upon the built-in device actuators in order to carry out the desired behaviour.

As it can be noticed, in order to avoid confusion we will use the following terminology from this point on:

- *Sensor:* the low-level, probably physical element, that captures the data from the source.
- *Actuator:* the low-level, probably physical element, that performs a concrete action or operation at the very end.

- *Perceptor*: the logical element in the smobject that provides a semantic interface to access a sensor.
- *Effector*: the logical element in the smobject that provides a semantic interface to operate an actuator.

It is noteworthy how the whole set of perception and operation capabilities of the smobject are functionally implemented by these logical perceptors and effectors, which are connected to the actual physical device sensors and actuators. Symmetrically, capabilities in a concrete smobject are defined by its attached perceptors and effectors. Therefore, perceptors and effectors act as *semantic gateways* to the physical components.

In order to make a concrete device capability available in the smobject, the perceptor and/or effector must be developed, deployed and registered in the smobject (via the “smobject configuration file”), no modification is needed in any of the other smobject components.

According to our design, all the perceptors implement a common interface called `IPerceptor` which is accessed by the Perceptor Manager. This interface provides a unique method:

```
triple[] perceive()
```

This method returns the whole set of triples representing an RDF graph that describe the semantic information captured by the perceptor via the built-in sensors. Typically, every perceptor is specialised in a concrete knowledge domain, attached to the sensors that capture information about that domain, and annotates semantically the captured information using the appropriate configured domain vocabularies. Therefore, a perceptor transforms raw data into semantic data via annotation.

For example, a smobject in a self-regulated heating system can contain a `TemperaturePerceptor` connected to a built-in temperature sensor. The `TemperaturePerceptor` polls the sensor periodically obtaining the current temperature and using a temperature vocabulary such as

```
http://www.awareit.com/onto/2005/12/temperature
```

to annotate the data. Whenever the perceptor is polled by the Perceptor Manager via the `perceive()` method, it returns just one triple, such as

```
<http://www.awareit.com/onto/2005/12/location#ThisLocation> <http://  
www.awareit.com/onto/2005/12/temperature#hasTemperatureCelsius>  
"23"
```

On the other hand, all the effectors share a common interface called `IEffector` which is accessed by the Effector Manager. This interface provides a unique method:

```
void operate (constraint[] state)
```

This method receives a set of constraints (a class derived from RDF triple<sup>1</sup>) representing the desired state of the information in the concrete knowledge domain the effector operates. The effector is in charge of translating this semantic-based representation into concrete low-level operations on the attached device actuators. Therefore, an effector transforms semantic data representing a goal state (constraint) into concrete operations for the built-in actuators to achieve that state.

For example, the `TemperatureEffector` could receive as parameter of `operate()` a single constraint about the temperature of the current location:

```
<http://www.awareit.com/onto/2005/12/location#ThisLocation> <http://
www.awareit.com/onto/2005/12/temperature#hasTemperatureCelsius>
"26"
```

Since this is a simple effector, it may only accept constraint with

```
http://www.awareit.com/onto/2005/12/temperature
#hasTemperatureCelsius
```

as predicate, so its task is limited to checking that the received predicate is such, and to operating the built-in actuator to configure “26°C” as the target temperature.

Figure 5.3 illustrates this example depicting the internal information flows involving transformation of raw data into semantic data in a smobject.

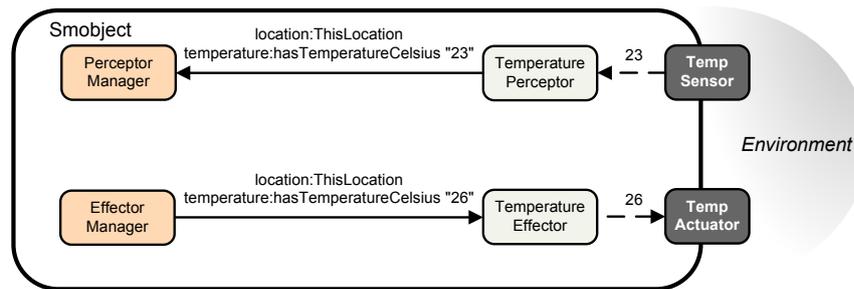


Figure 5.3: Example of internal perception and operation process in a smobject.

<sup>1</sup>A constraint is basically an RDF triple augmented with an optional operator property to represent ranges (e.g. “greater than 30”) and some other control data.

Perceptors and effectors are not required to be mutually isolated and some times they can be packaged into the same logical component, exhibiting the `IPerceptor` and `IEffector` interfaces simultaneously and being registered with both the Perceptor and Effector Manager. This is the desired configuration in the previous example, so resulting in a `TemperaturePerceptorEffector`, allowing the component to act as a control process: continuously reading the current temperature and acting (controlling) on the built-in actuator to meet the desired constraint.

The “smobject configuration file” contains basic information to initialise and operate the smobject, for example:

- Smobject ID (generally an UUID), class types and other information.
- Perceptors polling period by the Perceptor Manager.
- Correspondences between perceptors and perception capabilities.
- Correspondences between effectors and operation capabilities.
- Conflict resolution strategies for the operation capabilities.
- Any other data for smobject initialisation and operation.

An example configuration file is included in appendix G.

As stated before, perceptor and effector interfaces act as semantic gateways to the built-in platform-specific sensors and actuators, hiding the low-level details to other smobject components.

### Conflict resolution strategies

A smobject can be affected by more than one constraint on the same knowledge domain item at a concrete moment of time.

If the knowledge domain item is multivalued, that is, it can assume more than one value at the same time, no conflict arises and all the constraints can be honoured. An example on multivalued property could be the a `displayOption` property of an information kiosk, it can show an unlimited number of options to the user, so the following constraints do not pose any conflict and can be presented to the user:

```
<urn:uuid:panell1> display:displayOption <http://www.awareit.com/onto  
/2005/12/tourist#Monuments> .  
<urn:uuid:panell1> display:displayOption <http://www.awareit.com/onto  
/2005/12/tourist#Museums> .
```

```
<urn:uuid:panel1> display:displayOption <http://www.awareit.com/onto  
/2005/12/tourist#TypicalShopping> .
```

This would be the case of a tourist whose main interest when browsing information kiosks are monuments, museums and typical shopping spots in town. The user's preferences in this scenario are provided by his PDA or mobile phone, and processed by the kiosk, generating the previous constraints, which in turn may highlight the above mentioned options in the user interface for easy identification.

As well as multivalued properties, there are also functional properties which can assume only one value at a time. Examples of functional properties are `light:luminance` or `tv:channel`<sup>2</sup>. In these cases when several constraints are requested on the same knowledge domain item, some mechanism must be activated to generate the final value.

Conflict resolution strategies are intended to provide different mechanisms to cope with these situations by automatically obtaining the value the property must assume. The right strategy to apply depends on the property characteristics and the behaviour the designer of the smobject intends to apply.

Examples of conflicts are already present in existing devices and the strategy uses to be statically coded or hardcoded by the designer. For instance, a TV whose channel is simultaneously configured from the remote control and the TV set external buttons; or an electronic opening door whose "open" and "close" buttons are pressed at the same time.

From the smobject's point of view, a conflict resolution strategy can be attached to each operation capability, and thus, to each effector as established in the "smobject configuration file". By applying a strategy, a set of *applied constraints* is generated from the set of *active constraints* received by the device.

While applied constraints are those finally passed to the effector, the active constraints are still important for a concrete property, since their processing finally results in the applied constraints being generated for such property (which may be different from the all the active constraints, e.g. an average value).

Although smobject designers can provide particular strategies depending on the knowledge domain item, device and so forth, we have designed several generic built-in conflict resolution strategies intended for a wide range of situations:

---

<sup>2</sup>Except in the case the TV screen can be split up in a mosaic-like way to show multiple channels.

- First wins: the first received constraint takes precedence over subsequent ones and becomes the applied constraint, following a FIFO scheme. When this constraint expires, the next one following the reception order becomes the applied constraint.
- Last wins: the opposite of the “first wins” strategy, the last received constraint becomes the applied one, following a LIFO scheme. Everyday appliances usually behave this way (e.g. TV set with remote controls, washing machines, lights, and so on).
- Requester-based priority: all the constraints have a compulsory field called “requester” conveying the requester ID (a URI), which can be used in combination with a priority list to promote some constraints over others depending on the requester. In the case two requesters share the same priority a “first wins” or “last wins” strategy can be complementarily applied.
- Numeric average: if the object is a numerical value, all the active constraints are taken into account to generate the average value which results in the applied constraint.
- Numeric ranges weighted average: if the constraints also express ranges, they can be taken into account to generate the applied constraint. For instance, let’s consider the following active constraints:

```
location:ThisLocation temperature:hasTemperature "<" "20" .  
location:ThisLocation temperature:hasTemperature "30" .  
location:ThisLocation temperature:hasTemperature "<" "15" .  
location:ThisLocation temperature:hasTemperature ">" "20" .  
location:ThisLocation temperature:hasTemperature ">" "22" .  
location:ThisLocation temperature:hasTemperature ">" "25" .
```

We have designed a conflict resolution strategy that works as follows:

1. Identify all the constraints ranged with “<” and multiply their number as many times as the lower accompanying value. In the example there are 2 of such, [ $< 20$ ] and [ $< 15$ ], so multiplying  $2 \times 15 = 30$  (15 is the lower value among these).
2. Analogously, identify those constraints specified with “>” and multiply their number as many times as the larger value. In the example,  $3 \times 25 = 75$ .
3. Add the resulting values of the previous steps to those without range. In the example, 30 is provided without range, so  $30 + 75 + 30 = 135$ .

4. Divide the resulting value among the total amount of constraints to obtain the final value for the applied constraint. In the example,  $135 \div 6 = 22.5$ .

Using this strategy, ranges are weighted and an average value is generated. Of course, since some constraints may be disjoint, such as in the example, they cannot be fully honoured, but they still “attract” the final value towards their side.

Conflict resolution is difficult and very situation specific. These strategies are not intended to cover all the cases but to provide a basic functionality for typical scenarios, while still allowing smobject designers to create particular device- or domain-specific resolution strategies.

#### 5.1.4 Built-in platform components

These components are platform-specific and out of SoaM standardisation. Platform interfaces, perceptors and effectors, interact with these components as explained above in order to provide semantic gateways to them. Built-in platform components are generally sensors, actuators, and other device-specific elements, such as digital or analog inputs, outputs, or attached devices.

#### 5.1.5 Awareness components

Awareness components provide the smobject with intelligent capabilities and autonomous reactive behaviour. Through these components, the smobject is able to receive behavioural profiles from requesting parties, retrieve capabilities and context information from other neighbour smobjects, and finally perform reactive behaviour based on all this knowledge.

Awareness components expose an additional service through the communication interface: the behavioural profiles management service, which provides management operations over the profiles that represent the required behaviour.

Figure 5.4 depicts the complete smobject communication interfaces provided by both base and awareness components.

Awareness components act independently from base components and its activity is triggered by the existence of at least one behavioural profile, since their operation is associated to the management and processing of such.

The steps carried out by these components are:

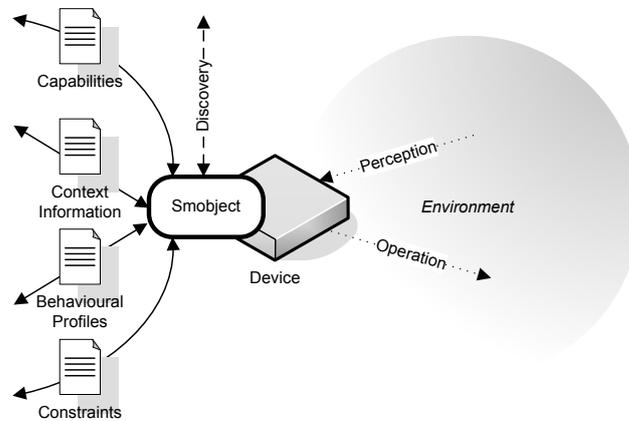


Figure 5.4: Complete smobject communication interfaces.

1. Listen for behavioural profiles: the existence of a profile is the event that triggers the context-awareness process in the smobject. If no behavioural profile is configured at the smobject, there is no need for the awareness process. Behavioural profiles can be natively provided with the smobject and/or injected externally.
2. Discover other existing smobjects in the environment, and retrieve their capabilities.
3. Retrieve context information from the Perceptor Manager and from other discovered smobjects.
4. Apply description logics based on domain knowledge ontologies to augment the context information.
5. Apply knowledge domain rules to augment the context information.
6. Resolve existing behavioural profiles against the augmented information obtaining the constraints.
7. Identify the constraints that can be honoured by the smobject by matching them against its operation capabilities and send the candidate constraints to the Effector Manager.
8. Manage and renew the constraints on the Effector Manager as needed.

There are four awareness-related components:

- **Profiles Manager:** manages the life cycle of the behavioural profiles in the smobject, removing them at expiration and renewing them under

request. It receives the profiles through the HTTP interface, passing them along to the Awareness Engine for processing.

- **Entity Manager:** periodically searches for other smobjects in the environment using installed discovery protocols. The Entity Manager is the component in charge of communicating with other smobjects, retrieving their capabilities and context information, and making the resulting knowledge available to the Awareness Engine.
- **Awareness Engine:** whenever a behavioural profile is present, the Awareness Engine retrieves the context information from the smobjects (including the local smobject) via the Entity Manager. Then, the Awareness Engine augments the context information through the Reasoner, and resolves the profile generating the constraints, which are passed along to the Constraints Manager.
- **Reasoner:** implements one or several reasoning mechanisms, such as description logics or inference rules. It receives context information directly obtained from environmental entities and returns that information augmented via reasoning.
- **Constraints Manager:** manages the life cycle of the constraints, dispatching, renewing and finally removing them when no longer required in the local smobject via the Effector Manager. Removal of constraints may have two different causes: either the behavioural profiles have not generated the constraint during the last resolution, or the original profile that generated the constraint has been removed or not renewed by the requester.

Figure 5.5 illustrates the internal structure of a smobject including both base and awareness components.

Therefore, the smobject behaviour is always driven by behavioural profiles at a higher level and constraints at a lower level. Behavioural profiles can be natively stored in the smobject or externally injected by an influencing entity. Constraints are always generated from behavioural profiles via a resolution process which can be performed internally by the smobject, or externally by an intelligent entity in the environment. We have designed one type of such external entity, called Orchestrator (see 5.2).

When the constraints are generated internally, the Awareness Engine is able to obtain them by resolving behavioural profiles against existing context information, applying the Profiles Resolution Algorithm.

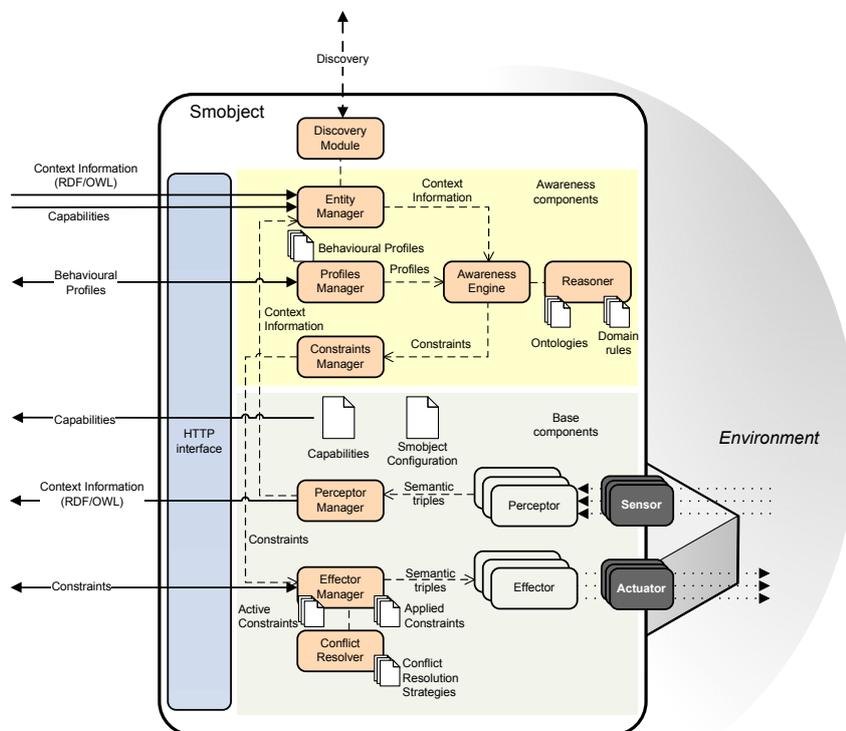


Figure 5.5: Complete Smobject internal structure.

### 5.1.6 The Profiles Resolution Algorithm

The process of resolving behavioural profiles, formed by preconditions and postconditions, against existing context information in order to obtain the constraints, is very similar to rule processing, since once preconditions are satisfied and resolved, postconditions are fully characterised.

In order to evaluate preconditions, we cannot make use of a complex rule engine due to platform limitations in the smobject. However, we have designed a lightweight mechanism to resolve queries with conditions and variables against existing RDF information: the Plant Query Resolution Algorithm presented in subsection 3.3.1.

The Profile Resolution Algorithm is an extension of the Plant Query Resolution Algorithm. Since a behavioural profile is basically a rule with some preconditions and postconditions, the preconditions can be

resolved using the Query Resolution Algorithm against existing context information, and the values of the variables obtained, can be substituted in the postconditions, generating the constraints.

For instance, let us revisit the simple example in Listing 4.8 represented in RDF/XML:

```
<behavioralProfile id="urn:uuid:prof2" expires="PT2M"
  requester="http://people.com/bobby">

  <variable xml:id="x"/>
  <variable xml:id="y"/>

  <precondition id="urn:uuid:prof2_prec1"
    subject="http://people.com/bobby"
    predicate="http://www.awareit.com/onto/task#isDoing">
    <objectResource
      resource="http://www.awareit.com/onto/task#WorkingWithLaptop"/>
    </objectResource>
  </precondition>

  <precondition id="urn:uuid:prof2_prec2"
    subject="http://people.com/bobby"
    predicate="http://www.awareit.com/onto/location#isLocatedIn">
    <objectVariable ref="x"/>
  </precondition>

  <precondition id="urn:uuid:prof2_prec3"
    subject="#x"
    predicate="http://www.awareit.com/onto/sound#hasSound">
    <objectVariable ref="y"/>
  </precondition>

  <postcondition id="urn:uuid:prof2_postc1"
    subject="#y"
    predicate="http://www.awareit.com/onto/sound#volume">
    <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int">
      3
    </objectLiteral>
  </postcondition>

  <postcondition id="urn:uuid:prof2_postc2"
    subject="#y"
```

```
    predicate="http://www.awareit.com/onto/sound#station">
    <objectResource resource="http://www.bbc.co.uk/radio4"/>
  </postcondition>
</behavioralProfile>
```

Applying the Plant Query Resolution Algorithm on this profile against the existing context information referred in subsection 4.4.3 would produce possible values for  $x$  and  $y$ . These values, can be substituted in the postconditions in order to generate the concrete constraints shown in Listing 4.9 and replicated here:

```
<constraint id="urn:uuid:con1"
  requester="http://people.com/bobby"
  subject="urn:uuid:hifil_sound"
  predicate="http://www.awareit.com/onto/sound#volume"
  expires="PT2M">
  <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int">
    3
  </objectLiteral>
</constraint>

<constraint id="urn:uuid:cond2"
  requester="http://people.com/bobby"
  subject="urn:uuid:hifil_sound"
  predicate="http://www.awareit.com/onto/sound#station"
  expires="PT2M">
  <objectResource resource="http://www.bbc.co.uk/radio4"/>
</constraint>
```

Therefore, the Profiles Resolution Algorithm produces the expected results, taking advantage of the already existent Plant Query Resolution Algorithm and adding an extra final step after variable solving: substituting the obtained values for the variables in the postconditions. By performing this final step constraints are generated (see section 6.2.1 for performance measures about the profile resolution process).

### 5.1.7 Selective and comprehensive context information collection

When collecting context information from existing smobjects in the environment, there are two possible strategies to follow:

- Comprehensive context collection: it consists on retrieving all the context information from every smobject present in the environment, so that the smobject creates an RDF graph representing all the available knowledge at a concrete moment of time. The drawback is the amount of network traffic that can be generated as well as a huge amount of information that can exceed the limited smobject storage facilities (memory).
- Selective context collection: it consist on identifying the smobjects that can provide the required context information for resolving the current behavioural profiles, and only collecting information from those smobjects. It is the best approach in terms of efficiency in network traffic and information retrieval, but it requires a previous process for selecting relevant smobjects.

At the initial steps of our design, we decided that smobjects should implement the selective strategy, applying the following criterion to decide whether a concrete smobject is relevant or not: if its perception capabilities declare the ability to provide information matching any precondition of any current behavioural profile, the smobject is considered relevant; ignored otherwise.

This filter selects smobjects providing context information required to evaluate existing behavioural profiles, which is in turn the main goal of the information collection.

Since only required information is requested, this strategy decreases the overall network traffic and saves processing cycles and RDF/XML parsing both at the requesting smobject and at the requested ones.

However smobjects are intelligent entities that apply reasoning processes over the context information. When reasoning process are applied it is difficult to identify all the required data for resolving a behavioural profile: since reasoning is based on rules, backward-chaining should be applied to find out the original facts that can potentially generate some context information via inference, and smobjects providing those original facts must be also identified.

Since backward-chaining is costly in terms of computing power, which is a scarce resource in embedded platforms, comprehensive context information collection can provide a better alternative in some scenarios.

Table 5.1 illustrates some criteria to select the most appropriate context information collection strategy<sup>3</sup>.

Collection strategy	Required computing power	Required memory space	Generated network traffic	Intelligence
Selective without backward-chaining	Medium	Low	Low	Medium
Selective with backward-chaining	High	Medium	Medium	High
Comprehensive	Low	High	High	High

Table 5.1: Comparison among possible context collection strategies.

The suitability of each strategy can be summarised as:

- Selective without backward-chaining: provides a conservative balance between intelligence and available resources.
- Selective with backward-chaining: optimises network traffic and memory, but it can only be applied if enough computing resources are available.
- Comprehensive: suitable in scenarios where the platform processor capabilities are limited, and the network and platform memory can cope with large amounts of context information.

In order to obtain the largest amount of context information collected and thus, maximise the intelligence of the system, so that the smobject is aware of any relevant information to reason upon and evaluate the profile, either a selective with backward-chaining or a comprehensive strategy must be applied.

Considering that network bandwidth is not generally a problem (context information requires less bandwidth than any multimedia system), the selected strategy depends on the hosting platform capabilities: computing power and memory.

---

<sup>3</sup>Keeping other factors stable, perceived intelligence depends on the amount of context information collected.

### 5.1.8 Optimising behavioural profiles resolution

We also decided to apply some optimisations in the resolution process at the smobject when receiving behavioural profiles: if the smobject's operation capabilities do not match any of the profile postconditions, the behavioural profile must be ignored and discarded.

When the smobject operation capabilities do not match any of the profile postconditions, there is no point in retrieving context information from other smobjects, evaluating the information and resolving the profile, obtaining the constraints, to find out, at that moment, that none of the constraints is suitable for execution at the smobject.

Upon reception of a behavioural profile, the smobject performs a filtering against its operation capabilities, discarding the profile if none of its postconditions could be honoured by the smobject when resolved into constraints.

This optimisation saves a lot of network traffic by avoiding all the messages involved in profile resolution, basically context information requests to other smobjects, whenever there is no point in doing so.

### 5.1.9 Intelligence and reasoning at the smobject

The embedded reasoner at the smobject, powered with ontologies and domain rules, can successfully interpret situations that were not previously resolvable without reasoning.

For instance, a profile so ambiguous for a machine such as “if Bob is sleeping at home, the ambient must be quiet”, can be resolved via description logics ontologies and forward-chaining over knowledge domain rules involving more atomic facts that can determine Bob's state (e.g. “if  $x$  last movement was 10 minutes ago and it is about midnight then  $x$  is sleeping”). Applying ontologies about location and places, if “Bob is in the room” and “the room is contained in home”, the transitive property of location produces the fact that “Bob is at home”.

Similarly, “ambient must be quiet” is probably not a constraint acceptable by any smobject, but rules such as “if the sound level is less than 2 and all the appliances are muted, then the ambient is quiet” can produce more concrete constraints injectable into smobjects (the premises of the above rule).

High-level ambiguous or “naturally expressed” profiles can be processed by the reasoning engine at the smobject contributing to a higher perceived intelligence in surrounding objects.

### Augmenting smobject intelligence through ontologies and description logics

As we have already discussed, description logics can contribute to better interpret and analyse context information by generating additional facts obtained through the application of ontologies.

Smobjects can share and apply ontologies to augment the context information they have obtained from fellow smobjects. The reasoner we have designed at the smobject is composed of two subelements: the MiniOwlReasoner and the MiniRuleReasoner. The MiniOwlReasoner is not a full ontology reasoner, which would be too big for an embedded platform, but a limited, yet powerful Semantic-Web oriented rule-engine.

The MiniOwlReasoner carries out the following steps:

1. It loads available ontologies.
2. It filters the ontologies, selecting the constructions the reasoner is able to deal with. The constructions the smobject's reasoner currently supports in our prototype are:
  - `rdfs:subClassOf`
  - `owl:sameAs`
  - `owl:TransitiveProperty`
  - `owl:SymmetricProperty`
  - `owl:inverseOf`

Which are the most common and used ontological predicates to create relationships among concepts in ontologies. It is noteworthy how any kind of transitive or symmetrical property can be processed by the MiniOwlReasoner; these kind of properties intrinsically embed a considerable amount of intelligence in any ontology. The MiniOwlReasoner implements a subset of OWL Lite, but it is more complex and powerful than other proposals such as RDF++ [Las06] by Lassila and almost equivalent to OWL Tiny [Bec04].

3. It generates a rule for each construction, which is added to the rules base. The rules are specifically generated depending on the construction arguments. For example, if `isLocatedIn` is declared as a transitive property, a particular rule matching RDF statements with the `isLocatedIn` property is generated and added to the rules base. This mechanism performed much better in terms of time consumed during reasoning than other alternatives we also tested.

4. When context information needs to be augmented via ontologies, the stored rules in the rule base are iteratively applied until no more additional information is generated.

We deem this strategy to provide a good balance between intelligence and limited resources availability, and the MiniOwlReasoner can be further improved in the future to cope with additional constructions if the host platform is powerful enough to support the work load.

### Augmenting smobject intelligence through domain rules

The MiniRuleReasoner is able to generate new facts from existing data (context information) by applying production rules.

The possibilities of this approach are remarkable: if a smobject stores rules about different knowledge domains, it can generate new data from existing context information and, thus, be able to resolve behavioural profiles that were previously discarded, since required information was not available.

For example, let us consider the following profile:

```
[<http://people.com/bobby> loc:isLocatedIn ?place]
[<http://people.com/bobby> task:isDoing tv:watchingTv]
⇒
[?place light:luminance "2"]
```

Basically this profile requests a faint light in a concrete room if `http://people.com/bobby` is watching the TV there.

While a location system can provide information to evaluate the first precondition, and the light control system can operate the postcondition, it is more difficult to create devices able to detect the different activities of the user in order to evaluate the second precondition. Interpreting whether the user is watching the TV or not can be somehow tricky.

However, this information could be inferred with a certain confidence degree from other available and easier-to-obtain facts. For instance, we could infer that the user is probably watching the TV if the following statements are true:

1. There is a device of type “TV set”.
2. The device is in a room.
3. There is a person.

4. The person is in the same room as the TV set.
5. The TV is on.

We could express this knowledge in the form of rule as:

```
[?tv rdf:type tv:TV]
[?tv location:isLocatedIn ?place]
[?person rdf:type per:Person]
[?person location:isLocatedIn ?place]
[?tv tv:state dev:stateOn]
⇒
[?person task:isDoing tv:watchingTv]
```

Therefore, the fact that the user is watching the TV can be derived from many more basic low level facts that can be obtained from existing devices: the location system can provide information about locations and the fact that a concrete ID belongs to a person; while the TV set can provide information about its state and type.

The light control system in the room could now successfully process the previously depicted adaptation profile from the information provided by the TV set and the location system, to adjust the lights properly.

### Implications of reasoning processes in performance

There are several issues to consider when applying reasoning processes in smobjects hosted in limited platforms:

- If the selective context information strategy is applied, the rules-enabled smobject must collect not only all the context information to evaluate profile preconditions, but also all the context information to evaluate rule premises using backward-chaining, thus generating more network traffic and processing requirements.
- Another approach, of course, could be applying the comprehensive context information collection strategy. In this way, smobjects would collect information from all others ignoring the perception capabilities. As already mentioned, the drawback of this approach is a larger amount of network traffic generated as well as the memory load in the smobject.
- As the amount of rules increases, the resolution process is more costly and takes more computing resources, specially in more limited

devices. Intelligence and required platform resources are bounded to each other.

- A mechanism should be defined for ontologies and rule exchange and discovery among smobjects, since built-in default static rules are not sufficient to cope with all the scenarios of everyday problems. The good news is that mRDP can be successfully used for ontologies and rules discovery, thus no additional protocol / infrastructure is required (see section 5.3).

Considering all these issues and for experimental purposes we decided to promote intelligence at the smobject by supporting both ontologies-based and rules-based reasoning: the smobject collects context information using a selective strategy, but a comprehensive strategy could also be configured. We decided not to incorporate backward-chaining to still deal with limited computing resources, keeping the balance between intelligence and platform requirements.

We also decided that the main role in augmenting context information via reasoning should be located at the requester side, not at the provider side. That is, the context information provider is not required to perform any kind of reasoning to augment context information before delivering it to requesting parties, but the latter should be in charge of this task since they are the users of the information and they can apply any configured reasoning mechanism at their discretion.

This scheme penalises requesters that poll surrounding smobjects very frequently and releases these smobjects, whose main role is providing context information, from reasoning activities that are not their main concern since they do not apply obtained conclusions.

In this way, every requester can self-regulate its polling period not to overload its reasoning mechanisms and to achieve a proper balance between awareness / reactivity, and energy consumption. On the other hand, non-reactive smobjects are not required to perform reasoning, so they can feature less platform requirements and save more energy.

#### 5.1.10 Smobjects as context-aware entities

Designed in such a way, the smobject is a full context-aware entity as defined in subsection 4.2.2:

*Any context-aware entity pursues a major goal, which is to adapt its behaviour accordingly to changes in the environment, that is, to changes in perceived context information.*

As mentioned earlier, the smobject can augment this context information by applying ontologies and rules, in order to fully interpret the deep relationships among the concepts.

Driven by the expressive behavioural profiles, the smobject collects context information directly from the environment and indirectly from other existent smobjects, applies reasoning augmenting this information, resolves the profiles, and finally generates the constraints that ultimately affect its behaviour and maybe change the environment.

Smobjects are sensitive to environmental changes via both its direct perception capabilities, represented by platform interfaces connected to built-in sensors; and its indirect perception capabilities, represented by fellow smobjects in the neighbourhood whose information is collected. Smobjects implement the desired behaviour through direct operation capabilities.

Thus, following the notation used in subsection 4.2.2, the smobject capabilities  $s_c$  can be expressed as:

$$e_c = (P_d, P_i, O_d, \emptyset) \quad (5.1)$$

Each smobject perceives information provided by itself and every other smobject, acting over its direct operation capabilities. Since every single smobject behaves this way, the result is that smobjects operate their effectors, and thus built-in actuators, as required.

Therefore, in a concrete scenario populated by a number of smobjects, all of them are aware of each other, exhibiting a coordinated reactive behaviour according to their behavioural profiles and embodying the concept of *semantic device*.

### **The awareness process in the smobject**

The existence of a behavioural profile triggers the context-awareness process. The smobject can be deployed with a number of pre-configured or static behavioural profiles (with no expiration period) that represent the permanent behavioural rules, the core behaviour, of the smobject. But any existing agent in the environment can also inject behavioural profiles into the smobject, augmenting its behaviour and making it more sensitive to other *stimuli*.

The awareness process starts with the smobject discarding non-operable profiles, then discovering other smobjects and retrieving their capabilities, and finally collecting the context information. The Profiles Manager

periodically checks existing profiles, removing them upon expiration if no renewal is requested by the influencing entity.

The Awareness Engine manages all the awareness process, identifying the suitable smobjects needed to evaluate profile preconditions from those provided by the Entity Manager. Requesting the information from them, augmenting the information through the reasoning processes, resolving each profile against the augmented context information, thus, generating the constraints, and finally passing them along to the Constraints Manager.

These newly generated constraints are checked against current ones by the Constraints Manager:

- If the constraint already exists, no further action is carried out
- If the constraint is new, not present among the current constraints, it is injected into the Effector Manager
- If a current constraint is not in the new set, it is removed from the Effector Manager, meaning that the originating influence no longer exists

Moreover, the Constraints Manager periodically renews the constraints in the Effector Manager as needed. From this point, the base components carry out their task as explained in 5.1.1.

Figure 5.6 depicts graphically the context awareness process in an environment populated by a number of smobjects.

As mentioned above, profiles may not always come from external sources. Smobjects can be created and preconfigured with some static profiles that embody smobjects behaviour, and are continuously active. In this case, the awareness components are also active all the time, checking the profiles against existing context information and carrying out the desired behaviour.

In this way, upon startup, the smobject is already populated with some profiles that define its behaviour and tries to discover other smobjects that provide information to evaluate preconditions. For example, a TV set can feature a preconfigured profile to allow or deny watching some TV shows depending on the user age or the time; or a web browser in a mobile phone can automatically show the homepage and most common services of the user's location, provided by an available location system.

In these cases, both the TV set and the mobile phone are context-aware entities, powered by their internal smobjects, with some preconfigured behaviour that makes them smarter for the user's point of view.

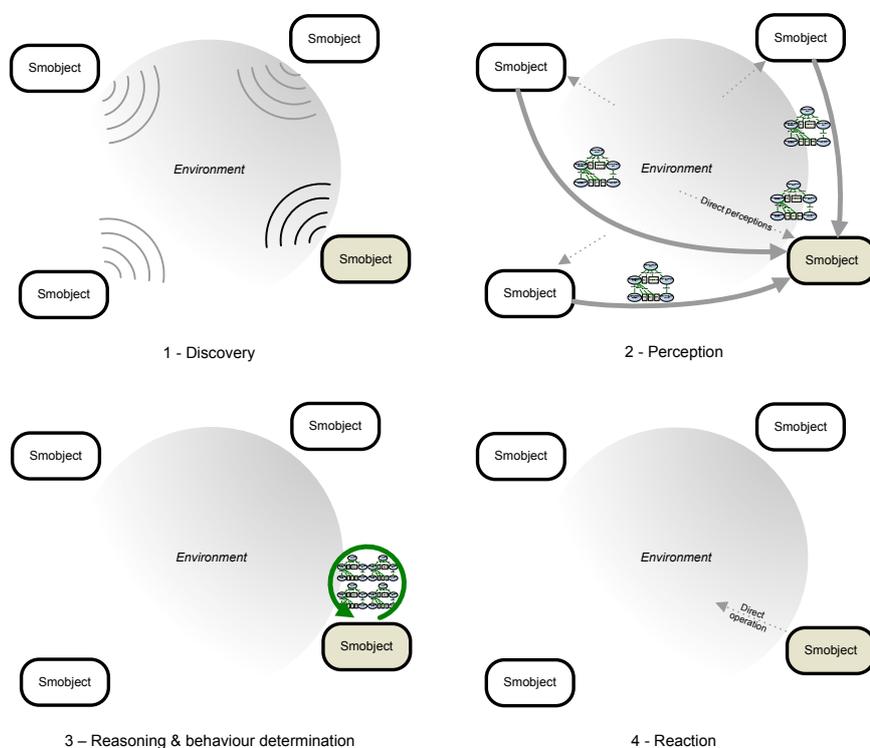


Figure 5.6: The context awareness process for smobjects.

### Passively influencing the smobject-powered environment

As presented in the theoretical model (see 4.1), we identified two mechanisms for provoking a change in context-aware entities: context modification and behavioural profiles injection.

We have designed smobjects as context-aware entities, in such a way that their behaviour can be modified dynamically in both ways:

- **Context modification:** the behaviour carried out by a smobject depends on the perceived context information. By altering the context, we can change its behaviour. For example, if a TV smobject reacts differently depending on user's preferences, modifying the preferences provoke a change in the smobject's behaviour.

The conclusion is that it is possible to modify the environment indirectly, applying the passive influence model, by exhibiting or hiding information by the client (a user or other smobject) as needed.

- Behavioural profiles injection: smobjects’s behaviour can be altered even at a higher level by injecting behavioural profiles that enable them to be aware and react to other particular *stimuli*. Behavioural profiles augment smobjects’ behaviour and make them more sensitive and reactive to surrounding context information.

Therefore, smobjects can be dynamically influenced by any other entity, being it a user or a smobject acting on behalf of a user or a device, by exhibiting the appropriate context information and / or reconfiguring smobjects’ behaviours for the intended goals.

### 5.1.11 An example scenario

Let us consider the following behavioural profile belonging to the user Alice: “if I am working with my laptop in a room with a sound system, such sound system must play classical music with the volume set to 2, and the light luminance of the room to 5”.

The SoaM XML Datatypes representation of this profile is shown in Listing 5.1.

Listing 5.1: Behavioural profile for the example scenario.

```

1 <behavioralProfile id="urn:uuid:prof2" expires="PT2M" requester="http
  ://people.com/alice">
2
3   <variable xml:id="vroom"/>
4   <variable xml:id="vsoundsys"/>
5   <variable xml:id="vsound"/>
6
7   <precondition id="" subject="http://people.com/alice" predicate="
  http://www.awareit.com/onto/2005/12/task#isDoing">
8     <objectResource resource="http://www.awareit.com/onto/2005/12/
  task#WorkingWithLaptop"/>
9   </precondition>
10
11  <precondition id="" subject="http://people.com/alice" predicate="
  http://www.awareit.com/onto/2005/12/location#isLocatedIn">
12    <objectVariable ref="vroom"/>
13  </precondition>
14
15  <precondition id="" subject="#vsoundsys" predicate="http://www.
  awareit.com/onto/2005/12/sound#hasSound">
16    <objectVariable ref="vsound"/>
17  </precondition>
18
19  <precondition id="" subject="#vsoundsys" predicate="http://www.
  awareit.com/onto/2005/12/location#isLocatedIn">

```

```

20     <objectVariable ref="vroom"/>
21 </precondition>
22
23 <postcondition id="" subject="#vsound" predicate="http://www.
24   awareit.com/onto/2005/12/sound#volume">
25   <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int
26     ">3</objectLiteral>
27 </postcondition>
28
29 <postcondition id="" subject="#vsound" predicate="http://www.
30   awareit.com/onto/2005/12/sound#style">
31   <objectResource resource="http://www.awareit.com/onto/2005/12/
32     sound#ClassicalMusic"/>
33 </postcondition>
34
35 <postcondition id="" subject="#vroom" predicate="http://www.
36   awareit.com/onto/2005/12/light#luminance">
37   <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int
38     ">5</objectLiteral>
39 </postcondition>
40 </behavioralProfile>

```

Let us suppose four different smobjects in a concrete room as depicted in Figure 5.7: the light control system, the Hi-Fi system, a location system and Alice’s laptop. Alice enters the room, using the access control in the entrance that keeps track of the users inside and it is connected to the location system, which also has a database of appliances and their locations.

When Alice turns on her laptop, the internal smobject is able to provide the information “Alice is working with me (the laptop)”, along with other authorised details about Alice and the laptop itself that can be used by surrounding smobjects. The laptop also acts as Alice’s user-agent (BPinjector), discovering existing smobjects (step 1) in the environment and injecting Alice’s profiles into them (step 2), including again the laptop itself.

If the profile were disseminated to every smobject, they would check it against their capabilities, keeping it in the case of the Hi-Fi and light control systems, and discarding it in the case of the laptop and the location system, since their behaviour is not affected by the profile.

The Hi-Fi and light control systems discover each other as well as other smobjects (step 3), retrieve their capabilities and finally each of them notice how the information to evaluate the profile is provided by three different smobjects: the location system, the Hi-Fi system and the laptop. So, they start polling the context information from these sources (step 4) and evaluating it against the profile, generating the following constraints:

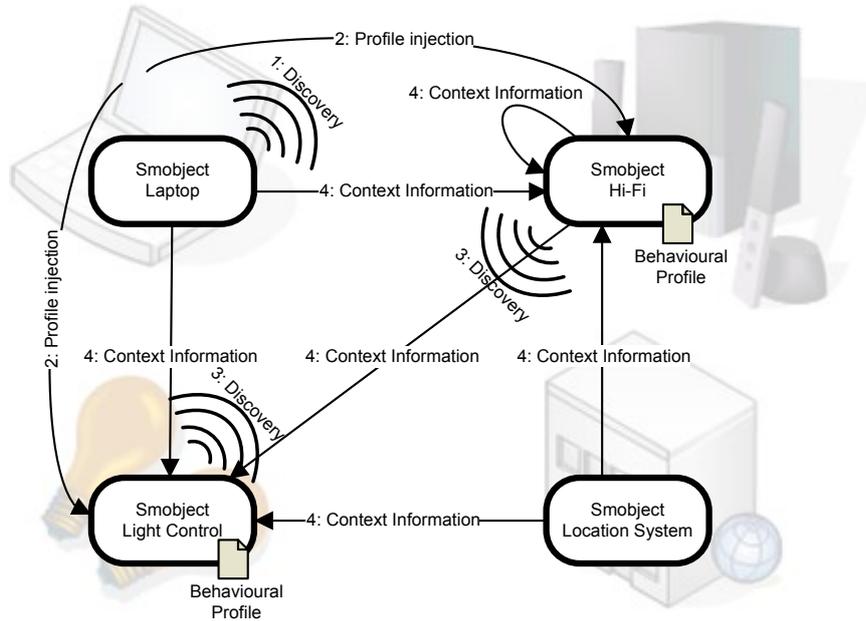


Figure 5.7: Example scenario with four smobjects.

```
<urn:uuid:hifil_sound> sound:volume "3".
<urn:uuid:hifil_sound> sound:style sound:ClassicalMusic.
<urn:uuid:room21> light:luminance "5".
```

The Hi-Fi system operates the first two constraints, discarding the third, while the light control system performs inversely, thus finally configuring the system to Alice’s goals.

The laptop must renew the profiles in the smobjects periodically, otherwise, without external influences, the smobjects are free to carry out a different behaviour as adopting an energy saving mode in the case of the Hi-Fi system.

### 5.1.12 Advanced perceptors and effectors

We would like to remark that one of the most noteworthy features of the smobject is its modular architecture for the platform interfaces: perceptors and effectors. Preserving the rest of the architecture unmodified, any designer can create his/her own specific smobject just by plugging-in the

appropriate perceptors and effectors for a particular purpose, and making the appropriate changes in the configuration file.

The rest of the modules remain unaltered, since they integrate with perceptors and effectors dynamically, on the fly, as specified in the configuration file.

This flexible plug-in architecture makes it possible to create advanced perceptors and effectors with additional properties. We have identified and tested two of these properties:

- Internet connection: perceptors can use the Internet as “sensor” system. For instance, we can create a perceptor that connects to a weather information website, downloads the weather forecast for the current day, and annotates the data using the appropriate ontology to make them available to surrounding smobjects in case some behavioural profile must be activated depending on the weather. Other examples of relevant information that can be accessed and “semantized” through perceptors are: vehicles traffic data, events in town, news feeds, user’s mailbox or even stock market quotes.

In this way, it is possible to build the user’s context both from very local and location-constrained data captured by smobjects in the ubiquitous network, as well as from more global information sources, enabling the creation of profiles such as “if I am watching TV and a new email arrives, show an alert on TV”.

Similarly, effectors can also connect to the Internet to perform some kind of action on remote entities. For instance, “if I am watching TV at time  $x$ , create an appointment at  $x$  in my shared calendar and label it as busy”.

- Proxy-tised external reasoners: combining the Internet connection capability with reasoning, a perceptor can act as a reasoner proxy, accessing an Internet hosted reasoner, sending the context information and obtaining the results, which are in turn returned to the Perceptor Manager. However, while architecturally correct, this approach is more costly in terms of bandwidth and response time. The advantage is that even a smobject hosted in a limited device without enough processing power to perform reasoning, can emulate it via an external agent.

## 5.2 Orchestrator

We have designed an optional entity in the SoaM architecture called *Orchestrator*, generally hosted in an advanced computing platform, that extends the functionality of the awareness components of the smobject to take advantage of more resources, and more intelligent and powerful reasoning mechanisms.

Basically, orchestrators are formed by the same individual modules as the awareness components, behaving the same way, except that the “Awareness Engine” is renamed as “Orchestrator Module” and features the capability of *orchestrating* (controlling) external smobjects.

Orchestrators are named this way, because they perceive and coordinate existing smobjects in the environment in order to implement a more refined context-awareness process. The orchestrator is the agent where the reasoning can be fully exploited without resource limitations, and pushed to its limits, thus, is the entity that can exhibit more intelligence in the SoaM architecture, along with more platform requirements.

Another important difference between smobjects and orchestrators is that while the Constraints Manager at the smobject awareness components discards constraints not suitable for the local smobject, the Constraints Manager at the Orchestrator processes all the constraints, finding the appropriate smobjects to inject them into.

Therefore, the main role of the orchestrator is to process behavioural profiles for the environment, applying advanced reasoning mechanisms, resolve the profiles into constraints and send those constraints to the appropriate smobjects.

The orchestrator resembles similar central components in other architectures such as CoBrA's Context Broker or Gaia's Lookup and History services. However, SoaM's orchestrator is a completely optional component, a reasoning and orchestrating facility that can provide more intelligence to the environment but is not required at all for the smobjects to operate: they can still collaborate autonomously and spontaneously.

An orchestrator exposes several services through its communication interfaces:

- **Discovery:** replies to discovery requests issued by other entities (orchestrators and BPinjectors).
- **Context information retrieval:** provides to requesting entities the context information collected by the orchestrator from all the

managed smobjects, augmented after ontological and knowledge domain reasoning.

- Behavioural profiles management: provides management operations over the profiles that represent the required environmental behaviour the orchestrator is in charge of.

Orchestrators do not only provide these services, but they also request some from other entities:

- Discovery: orchestrators search the network continuously for smobjects and retrieve their capabilities.
- Context Information retrieval: orchestrators request context information from smobjects in order to build a comprehensive knowledge base of the environment over which reasoning can be performed.
- Constraints management: orchestrators inject constraints into smobjects to drive their behaviour, managing these constraints as needed.

Figure 5.8 depicts the communication interfaces of the orchestrator.

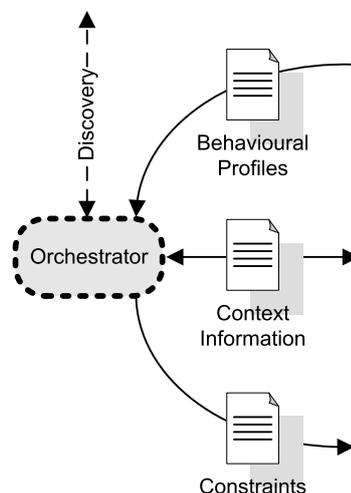


Figure 5.8: Orchestrator communication interfaces.

The orchestrator implements a more sophisticated context-awareness process by managing and orchestrating the smobjects. The steps carried out by the orchestrator are:

1. Listen for behavioural profiles: the reception of a profile is the event that triggers the context-awareness process. The orchestrator is a reasoning and orchestrating facility, it does not feature any pre-configured behavioural profiles since it does not perform any action.
2. Discover existing smobjects.
3. Retrieve context information from available smobjects.
4. Apply description logics, domain rules and any other kind of complex reasoning mechanism to augment the context information.
5. Resolve existing behavioural profiles against the augmented context information obtaining the constraints.
6. Identify the smobjects whose operation capabilities can honour the generated constraints and inject these constraints into them.
7. Manage and renew the constraints influence on the smobjects as needed.

These activities are carried out continuously by the orchestrator, whenever at least one behavioural profile exists.

The orchestrator functional components are depicted in Figure 5.9. The internal architecture resembles the awareness components except for two differences:

- The Orchestrator Module can feature not only ontologies-based and rules-based reasoning, but also more advanced mechanisms such as fuzzy logic, neural or bayesian networks.
- The communication flow that linked the Constraints Manager to the Effector Manager in the smobject architecture is substituted by a link between the former and the Entity Manager, since the constraints are injected through HTTP communication into external smobjects (as opposite to internal communication between awareness and base components within a smobject).

While increasing intelligence in smobjects via domain rules and ontologies must be used sparingly and its performance carefully evaluated in the hosting platform, the approach when it comes to orchestrators is completely the opposite: their mission is both to augment and enrich context information as much as possible through reasoning as well as to coordinate smobjects behaviour to implement the required goals. Orchestrators, if present, are devoted to reasoning without limitations

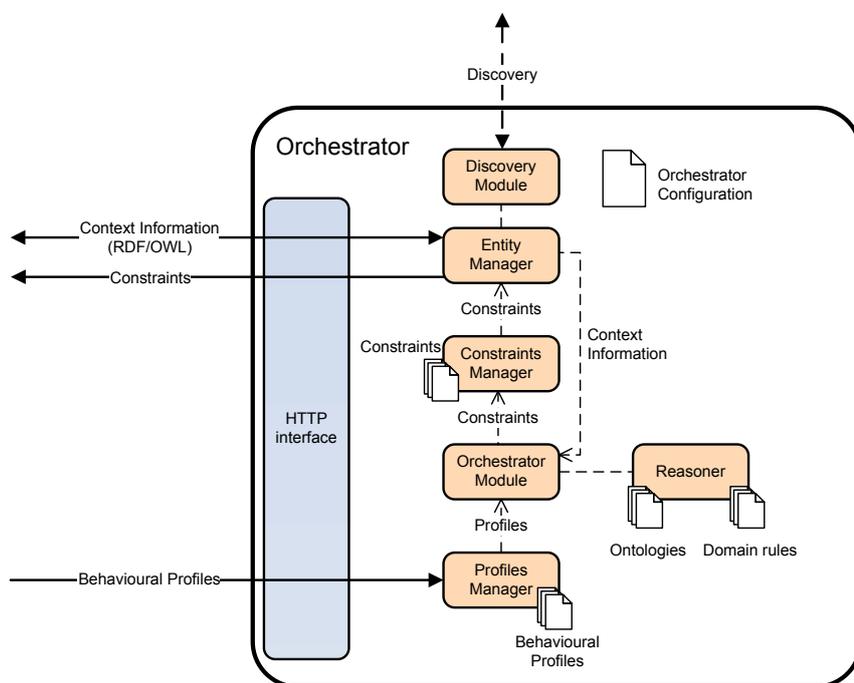


Figure 5.9: Orchestrator internal structure.

for the sake of other smobjects, and to creating apparent higher level intelligence in the environment.

Since the domain rules can be chained as already explained and there may be dependences among different ontologies when performing description logics reasoning, the orchestrator can adopt one of the two previously commented strategies:

- Comprehensive context information collection, in order to have the maximum amount of information available before reasoning.
- Selective context information collection, analysing both domain rules and ontologies to find out the minimum set of context information required, identifying the smobjects able to provide it and polling only these smobjects.

The limitations of computing power and memory that were a major concern in smobjects to select the best strategy are not such an issue for the orchestrator.

Whatever the strategy selected, it must be granted that all relevant information is generated during reasoning in order to process the profile.

We opted for a comprehensive information collection strategy in the orchestrator for our implementation (see chapter 6).

### 5.2.1 Example scenario with orchestrator

Revisiting the scenario depicted previously in subsection 5.1.11, now incorporating one orchestrator which can be provided by the existing laptop, the relationships among the entities are shaped in a different way as illustrated in Figure 5.10.

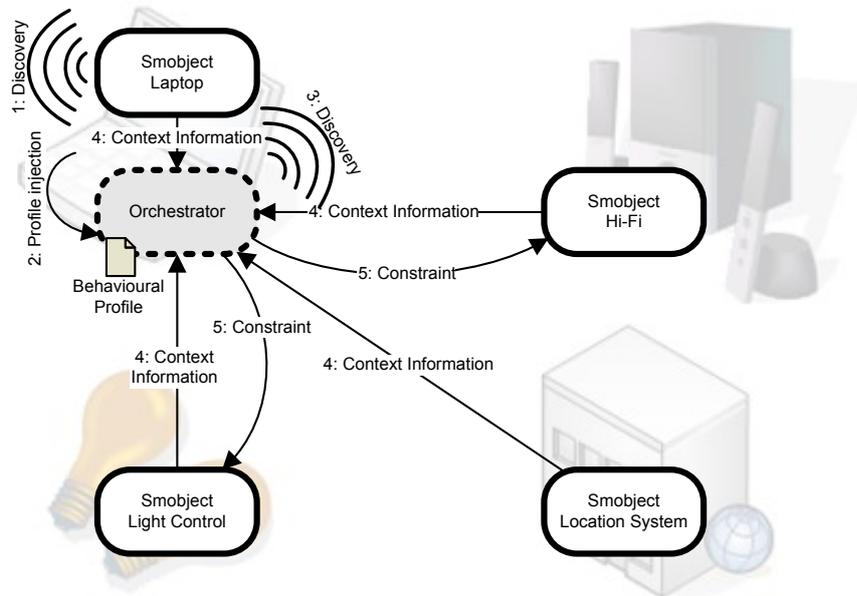


Figure 5.10: Example scenario with one orchestrator coordinating four smobjects.

Alice’s user agent with her profiles, that is, the BPinjector (see section 5.5), discovers the presence of an orchestrator (step 1) and injects the behavioural profiles into it (step 2). Since the smobjects do not receive any profile, their awareness components remain idle and only the base components are active.

The presence of an orchestrator releases existing smobjects from the reasoning process, saving resources for context information provision and constraints operation.

The orchestrator discovers (step 3) and collects all the context information (step 4) from the surrounding smobjects; it applies the relevant ontologies and domain rules (downloading them from a website or searching them through the network via mRDP) and resolves the profiles, generating the constraints.

Constraints are injected by the Constraints Manager module of the Orchestrator into the appropriate smobjects (step 5), as stated by their operation capabilities, achieving the desired reactivity.

### **5.2.2 Reasoning at the orchestrator**

The orchestrator, hosted in a non resource-limited platform, is potentially able to implement more sophisticated reasoning mechanisms than the smobject, including backward-chaining or fuzzy logic.

Moreover, the orchestrator could be able to store registries with all the interactions, and context-information snapshots that took place in the system over time, and apply neural or bayesian networks to identify recurrent patterns and predict situations before they take place, thus fulfilling the anticipatory character of Ambient Intelligence.

The integration of ontologies, domain rules, fuzzy logic or other reasoning mechanisms able to deal with uncertainty in the reasoner makes the orchestrator to emulate in a higher degree how people reason, and how to process “naturally expressed” desired behaviours in profiles.

## **5.3 Ontologies and domain rules discovery for Ubiquitous Computing**

One of the most challenging issues when it comes to dealing with OWL ontologies or domain rules is how a semantic engine, such as the reasoners embedded in smobjects or in the orchestrator, is able to obtain the ontologies and rules.

For example, if existing smobjects can provide information about location, user profiles in the enterprise and available surveillance systems, how can the reasoner identify, locate and apply the appropriate ontologies and rules for augmenting the information via inference and create new relationships among the concepts?

In the case of locating ontologies, which must be explicitly declared in the RDF document using `<owl:imports>`[Wor04e], current semantic reasoners<sup>4</sup> generally provide two complementary means:

- Provision of a copy of the ontology in local storage.
- Downloading the ontology dynamically via HTTP.

When an `<owl:imports>` clause is found while parsing an RDF document, the usual mechanism starts by looking in the local storage for the referred ontology, then loading it. If the ontology is not stored, an HTTP GET request is issued to the ontology URI where the OWL document can be downloaded. In order to accomplish this second step, the reasoner must be connected to the Internet.

Since it is quite difficult to determine beforehand which ontologies are needed to process RDF graphs about arbitrary knowledge domains generated by different entities, and thus, store the required ontologies previously in the system, dynamic downloading of ontologies is a must. Moreover, downloaded ontologies can be cached in the local storage for subsequent accesses.

However, when it comes to Ubiquitous Computing scenarios is not always feasible to have an Internet connection available, so this situation must not be taken for granted. For example, in open public spaces, travelling by car or at restaurants, public Internet connection is not always available, and sometimes requires explicit configuration by the user, which breaks our serendipitous principle.

For knowledge domain rules, the problem becomes more evident since there is no standardised means of locating and retrieving rules related to a particular knowledge domain in a concrete scenario. Generally, these rules are stored by the reasoning engine and applied as needed, without any further dynamic mechanism for retrieving other rules about that domain or about other domains.

Without Internet connection, orchestrators and semantic engines in general, are not able to locate and download new ontologies and domain rules, thus limiting their ability to generate new valuable information and RDF triples from existing knowledge. Moreover, there is no means of identifying suitable domain rules or URLs where they can be downloaded from, related to concrete knowledge domains: semantic engines check all available rules, just in case.

---

<sup>4</sup>Such as the default Jena2 OWL reasoner document manager.

Our approach for solving these problems is based on several premises that SoaM fulfils:

- Since smobjects provide context information about different knowledge domains, depending on attached perceptors and effectors, they could also provide related ontologies and rules documents to requesting parties.
- Since mRDP is a general resource discovery protocol, it can be used to locate ontologies and domain rules through the SoaM network, without the need for an Internet connection. This is not possible using UPnP SSDP or any other traditional service discovery protocol.

For example, if a TV set is manufactured with a built-in smobject providing information about the TV state, configured stations or show type in digital TV, related ontologies and even domain rules can also be stored in the smobject in order to make them available to existing reasoners in the environment.

After retrieving context information from existing smobjects, the reasoner identifies the `<owl:imports>` provided and issues an mRDP `LOCATE` multicast message:

```
LOCATE http://www.awareit.com/onto/2005/12/tv mRDP/1.0
NSeq: 67
Callback-URI: http://169.254.0.3/mrdpendpoint
```

Possible locations of the ontology in the network will be provided via ReDEL responses to the callback URI. Then, the reasoner downloads the ontologies as indicated in ReDEL, applying them during reasoning.

Identifying and locating possible domain rules is more difficult. First, there is no framework for integration of rules languages with RDF and OWL, although RuleML and SWRL have been proposed (see subsection 4.4.3) and standardisation is on its way at the W3C<sup>5</sup>.

We have designed some classes and properties in SoaMonto, the SoaM ontology (see section 5.9), to allow association of rules with knowledge domains represented by ontologies using their URIs.

Moreover, rules can involve different knowledge domains at the same time. For instance the already analysed rule “if  $x$  last movement was 10 minutes ago and it is about midnight then  $x$  is sleeping”, involves user

---

<sup>5</sup>As of December 2006.

movement, time and user activities (sleeping), and can be attached to the three different domains.

The next excerpt associates a rule with the TV and task domains, using the appropriate ontology URIs and the SoaMonto ontology predicate `soamonto:involves`:

```
<http://www.awareit.com/rules/2005/12/tv_basicrules#rule2> soamonto:
  involves <http://www.awareit.com/onto/2005/12/tv>
<http://www.awareit.com/rules/2005/12/tv_basicrules#rule2> soamonto:
  involves <http://www.awareit.com/onto/2005/12/task>
```

Since TV domain-related information is provided by a smobject, the reasoning engine can search the network for rules involving the TV domain, in case they can be applied to generate new knowledge.

We have designed a two-step discovery process: first the reasoner issues an `mRDP IDENTIFY` message with a Plant query to identify rules in the network involving the TV domain:

```
IDENTIFY ?r mRDP/1.0
NSeq: 57
Content-Type: application/com.awareit.plant
Content-Length: 101
Callback-URI: http://169.254.0.3/mrdpendpoint

?r <http://www.awareit.com/soam/2005/12/soamonto#involves> <http://www
  .awareit.com/onto/2005/12/tv> .
```

In this example, no explicit representation of the rule is requested. However, if the reasoning engine is only able to process rules in some concrete language, for instance SWRL, the message conveying the Plant query in this case would be:

```
IDENTIFY ?r mRDP/1.0
NSeq: 57
Content-Type: application/com.awareit.plant
Content-Length: 194
Callback-URI: http://169.254.0.3/mrdpendpoint

?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.
  org/2003/11/swrl#Imp> .
?r <http://www.awareit.com/soam/2005/12/soamonto#involves> <http://www
  .awareit.com/onto/2005/12/tv> .
```

This last Plant query embodies two conditions: the resource type must be `swrl:Imp`, which is the class for rules in SWRL; and, the resource must involve the TV domain.

All the nodes storing rules of that type, would return a ReDEL response with identification and the location where more information about the rule can be downloaded, possibly including the rule itself or a link to the rule file, depending on the rules language used. Alternatively, an `mRDP LOCATE` request can be issued to find other possible locations of the rule or file. Once downloaded, the rule can be applied.

Defining the structure and use of rules languages is out of the scope of SoaM; reasoning engines can build and disseminate appropriate Plant requests for identifying and downloading individual rules or rule files, depending on the allowed granularity of the language.

Since we wanted to maintain smobjects design simple, they are not expected to create an RDF graph including every single stored rule; the smobject designer can group rules in files with associations to knowledge domains, and make them available for download by other smobjects. The only requirement is that every file must be identified with a URI, as any other resource, for the purposes of Plant queries, and annotated with `soamonto:involves` and `rdf:type` predicates for the query resolution.

For example, a TV smobject designer could store a rules file in RuleML 0.9 format and annotate it as:

```
<http://www.sony.com/tv/rules/basictrrules-01> rdf:type <http://www.ruleml.org/0.9/onto#RuleSet>
<http://www.sony.com/tv/rules/basictrrules-01> soamonto:involves <http://www.awareit.com/onto/2005/12/tv>
```

When requested about the location of such resource (a RuleML 0.9 rules file<sup>6</sup>) the smobject can generate a reply with ReDEL payload including the URL where such file is provided by the smobject. For example, if the smobject has IP address 169.254.0.12:

```
POST /mrdpendpoint HTTP/1.0
Host: 169.254.0.12
NSeq: 58
Content-Type: application/com.awareit.redel+xml
Content-Length: 438

<?xml version="1.0" encoding="UTF-8"?>
```

---

<sup>6</sup>As of December 2006, there is not any ontology for representing RuleML rule sets.

```
<redel xmlns="http://www.awareit.com/soam/2006/04/redel"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.awareit.com/soam/2006/04/redel
  http://www.awareit.com/soam/2006/04/redel.xsd">

  <resource uri="http://www.sony.com/tv/rules/basictvrules-01">
    <location url="http://169.254.0.12/basictvrules-01.ruleml"/>
  </resource>
</redel>
```

Therefore, smobjects can store ontologies and rules about the knowledge domains they deal with, so that reasoning engines in the ubiquitous network can discover and download those ontologies and rules in order to apply them in the reasoning process.

In this way, reasoning engines at smobjects and orchestrators insert an additional step between local storage file search and Internet download. This step represents the capability of the network to store and share ontologies and rules: ubiquitous network search using mRDP.

## 5.4 Smobjects-only versus orchestrator-powered scenarios

There are some remarkable differences to be aware of when deciding whether to use or not an orchestrator in the environment.

From our point of view, the following are some of the major advantages of orchestrator-powered topologies:

- More intelligence: the orchestrator is able to provide more advanced reasoning mechanisms over context information due to the absence of platform limitations.
- Reduced network traffic: the orchestrator acts as an information attractor, a central point where context information converges. This feature reduces the traffic compared to the smobjects-only scenario.

In the worst case where all the smobjects require context information from every other, the amount of connections to collect the data is  $n \times (n - 1)$ , being  $n$  the number of smobjects. In a medium case, where any smobject must collect information from 50% of the available smobjects, the number of connections would be  $\frac{n \times (n-1)}{2}$ .

In the presence of an orchestrator, the amount of connections to collect context information is just  $n$ , one connection per smobject from the orchestrator.

Figure 5.11 compares the number of connections required as the amount of smobjects increases.

- The use of an orchestrator demands less platform requirements at the smobjects, since only base components are required: context information collection, reasoning and behavioural profiles resolution is performed by the orchestrator.

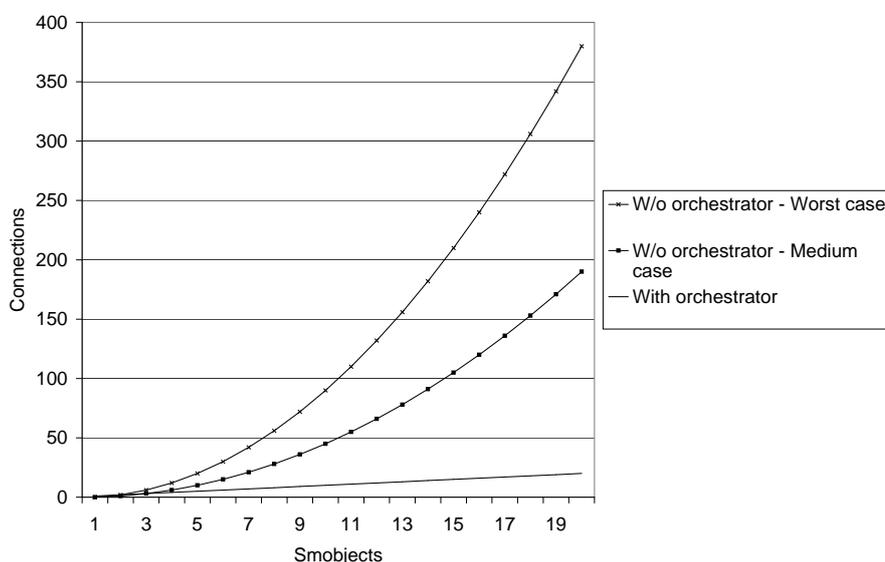


Figure 5.11: Number of connections for retrieving context information with and without orchestrator.

On the other hand, among the major advantages of smobjects-only topologies we would remark:

- Seamless deployment: the cooperation among smobjects emerges naturally anywhere as they discover each other, there is no need for deploying or providing a central point of control. This feature supports the requirement about the serendipitous nature of Ubiquitous Computing, while promoting distributed dynamic intelligence in the environment.
- Fault tolerance: smobjects-only networks do not rely on a single central point of control, but the responsibilities are distributed among

all the nodes. Any node is equally important and a single failure does not interrupt the process. Orchestrator-powered topologies become out of operation if the orchestrator fails.

The obvious conclusion is that both topologies are useful and must be activated depending on the circumstances: if an orchestrator is available, it must be used to coordinate environmental behaviour with more intelligent and traffic-optimisation features; in the absence of such orchestrator, smobjects are still able to form a context-awareness network autonomously and provide the required intelligent behaviour.

### 5.5 BPinjector

We have designed an additional SoaM entity called *BPinjector* (behavioural profiles injector) that acts on behalf of a user or other entity, disseminating behavioural profiles through the environment in order to obtain personalised behaviour and reactivity from available smobjects.

The BPinjector does not expose any service to other entities, acting solely as a client. Its internal architecture is very simple, easily embeddable in resource-constrained platforms, as illustrated in Figure 5.12.

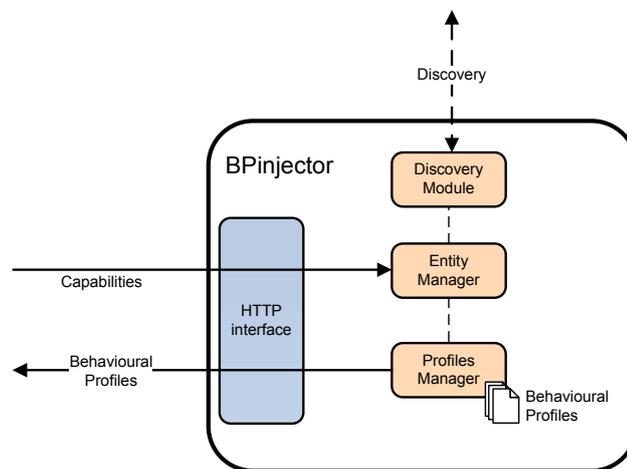


Figure 5.12: BPinjector internal structure.

The BPinjector is composed of three different modules:

- **Discovery Module:** periodically searches the environment for orchestrators or smobjects, using the provided discovery protocols.

- Entity Manager: manages the information related to the entities found in the network, orchestrators and smobjects, retrieving their capabilities in the latter case.
- Profiles Manager: injects the client's behavioural profiles into the appropriate entities, renewing them as needed.

The BPinjector is the entity that triggers the whole process of personalised behavioural change and adaptation in the smobjects of the environment by injecting the profiles. The concrete steps performed are:

1. Search the network for orchestrators and smobjects.
2. If more than one orchestrator is found, select one. Inject or renew the profiles (if needed) in the orchestrator. Go to step 1.
3. If no orchestrator is found, retrieve smobjects' capabilities and inject or renew the appropriate profiles (if needed) depending on their operation capabilities. Go to step 1.

First, the BPinjector tries to find orchestrators in the network. If these are found, profiles are injected into one of them, which will be in charge of performing the reasoning procedures as well as orchestrating the smobjects.

In the case that more than one orchestrator is found and if mRDP is used for discovery, the BPinjector can locate and download additional descriptions of the orchestrators from the network to check their features and select the most appropriate.

The SoaMonto ontology described in section 5.9, declares an extensible set of classes and predicates for describing orchestrators and their available reasoners, so that the BPinjector can make the selection based on criteria such as the reasoning mechanism, or whether the reasoner is local or remote.

In the case no orchestrator is found, the BPinjector retrieves smobjects' capabilities and injects the appropriate behavioural profiles into the correspondent smobjects.

There are two situations managed by the BPinjector and supported in the process as described previously:

- An orchestrator shows up when profiles have already been injected in smobjects. In this case, the BPinjector injects the profiles into the orchestrator, and since step 3 is never reached again, the profiles at the smobjects are not renewed. Upon profile expiration in smobjects, the

orchestrator is already exerting influence over them, so the constraints continue active without any perceived gap; the Effector Manager at the smobject is still receiving the constraints, now sent by the orchestrator's Constraints Manager.

- An orchestrator leaves the network. In this case, step 3 is executed and behavioural profiles are directly injected into the appropriate smobjects. The Constraints Manager in the smobject awareness components sends the constraints to the Effector Manager of the local smobject, and no gap in the behaviour is perceived.

However, there is one scenario in which the above mentioned situations motivate a discontinuity in the smobject operation: when the advanced reasoner in the orchestrator (powered with fuzzy logic or neural networks) generates new facts in the context information that lead to profile resolution and thus, constraints. The limited reasoner at smobjects may not augment the context information as much as the orchestrator's reasoner, so profile resolution is performed based on a smaller knowledge and some constraints may not be generated.

This outcome is consequent with the model, since the orchestrator's leaving is, after all, a loss of intelligence in the environment

BPinjectors could be embedded in wearables, phones, PDAs, or any other item the user carries with him anywhere and anytime in order to achieve a continuous influence over the environment and the surrounding devices.

This influence forces surrounding objects to react and behave accordingly to the user's preferences without any explicit command. The user presence influences the surrounding devices or, from a different point of view, devices are sensitive to user's preferences about their behaviour<sup>7</sup>.

### 5.5.1 The smobject as BPinjector

The BPinjector acts in behalf of a concrete client that desires to exert influence on the surrounding environment. In general terms, the client can be a user, a process, a device or even a smobject.

In the case of a user, although behavioural profiles can be created and edited manually, it is better to use some kind of software wizard or end-user programming technique to generate them automatically (e.g. based on user selections in a graphical interface).

---

<sup>7</sup>A scenario with a BPinjector disguised in a work jacket is explained in section 5.6.2.2.

Since ontologies are usually well documented in natural language using the annotation properties of OWL, and they have been used in other projects in the past for helping the user to make selections, for instance Task Computing (see subsection 2.3.3), they can also be used to help the user to create behavioural profiles without dealing with hard-to-remember and hard-to-process URIs.

Pre-configured or statically stored behavioural profiles act as a local BPinjector in smobjects. For example, a newly manufactured TV can be provided with the ability to adjust its screen brightness dynamically depending on existing luminance in the environment (which can make user watching difficult). The good news is that complying with the SoaM awareness model, the TV does not need to feature a built-in light sensor, but any smobject with a light perceptor can provide the information for the TV to react.

Multiple examples are possible where smobject-powered devices can rely on information provided by other fellow smobjects to implement context-awareness:

- A web browser whose homepage is automatically configured, depending on user's location.
- A TV set that reduces its volume automatically, if the user is making a phone call.
- A sculpture whose colour changes depending on the user's favourite football team performance.
- An office door that locks when nobody is in to prevent an unauthorised party entering.
- A fixed phone which redirects the phone calls to a mobile phone, depending on whether the user is or not at home as reported by the access control system.

The possibility of the BPinjector being a smobject is noteworthy, because of its side effects, which lead to a brand new range of SoaM applications. Having a BPinjector attached to a smobject means that the smobject takes an active role in deciding how other smobjects should behave, which basically means that the smobject is able to influence others' behaviours through the profiles. An example is the SmartPlants scenario depicted in chapter 1, where the plants actively influence how the lights, air-conditioning and heating systems behave in order to preserve their health.

## 5.6 Interactions

As mentioned earlier, the BPinjector performs in different ways depending on the existence of orchestrators in the environment. The following UML sequence diagrams illustrate the interactions among the different entities of the SoaM architecture in smobjects-only and orchestrator-powered scenarios.

In both cases, it is supposed there are two smobjects, called Smobject 1 and Smobject 2, and perceived information is such that profiles preconditions are immediately met and constraints are generated.

The first case, represented by Figure 5.13, illustrates the smobjects-only scenario. The steps carried out by the entities are:

1. The BPinjector searches for orchestrators. Since none is found, issues a request for smobjects and discovers two of them, retrieving their capabilities.
2. After checking smobjects' operation capabilities and find out the profiles that are suitable for them, the BPinjector injects the profiles.
3. The BPinjector performs discovery and injection periodically in case other smobjects or orchestrators show up.
4. Upon profile reception, the awareness components in each smobject become active.
5. The smobjects discover each other (although represented as a single arrow with two heads to save space, there are two interactions here, from one smobject towards the other), replying the discovery message with an HTTP POST request conveying ReDEL payload, and retrieving each other's capabilities.
6. Every smobject retrieves the required context information (provided by itself and the other smobject).
7. Every smobject performs reasoning over that information, augmenting it.
8. Every smobject evaluates the profiles and executes the constraints that match its operation capabilities.
9. Every smobject renews internally the constraints as needed.
10. Smobjects perform the discovery, context information retrieval, reasoning, profile resolving and constraints renewal periodically, checking changes in context information.

- The BPinjector removes the profiles from the smobjects, the awareness components become inactive and constraints are removed internally.

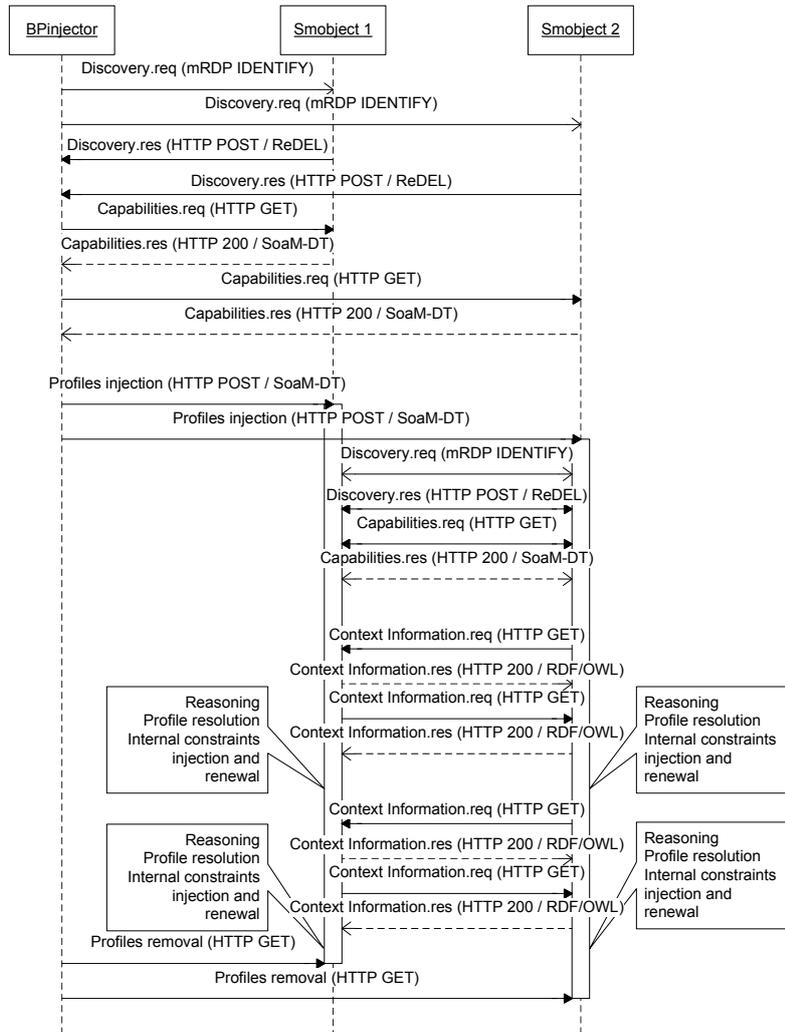


Figure 5.13: Interactions in a smobjects-only scenario.

The second case, represented by Figure 5.14, illustrates the orchestrator-powered scenario. The steps carried out by the entities are:

1. The orchestrator periodically searches for existing smobjects and retrieves their capabilities. This step could be delayed until a behavioural profile is received, but maintaining a list of existing smobjects speeds up the subsequent process.
2. The orchestrator periodically updates the list of existing smobjects.
3. The BPinjector searches for orchestrators, finds one and injects the profiles into it.
4. The BPinjector performs discovery periodically in case a more advanced orchestrator (featuring more complex reasoning) is found, or the current one leaves the network.
5. Upon profile reception, the orchestration module becomes active.
6. The orchestrator periodically retrieves context information from existing smobjects.
7. The orchestrator performs reasoning over that information, resolves the profiles and generates the constraints.
8. Constraints are injected into smobjects, according to their operation capabilities, and renewed as needed.
9. The BPinjector removes the profiles from the orchestrator, which in turn removes the constraints from the smobjects.

These examples show a particular interaction possibility among SoaM entities. However, there is a remarkable number of situations that can produce variations in these interactions, and that have been already mentioned along this chapter:

- Smobjects and orchestrators joining and leaving the network unexpectedly, thus reorganising profiles distribution by the BPinjector.
- Smobjects joining or leaving the network, thus providing or removing certain capabilities and context information awareness from the environment, which in turn leads to the successful or failed processing of a profile.
- Behavioural profiles whose preconditions are not matched at the beginning but after a certain period, when context information changes, thus generating the constraints some time after profile injection.
- Removal of constraints due to changes in context information that makes profile preconditions no longer valid.

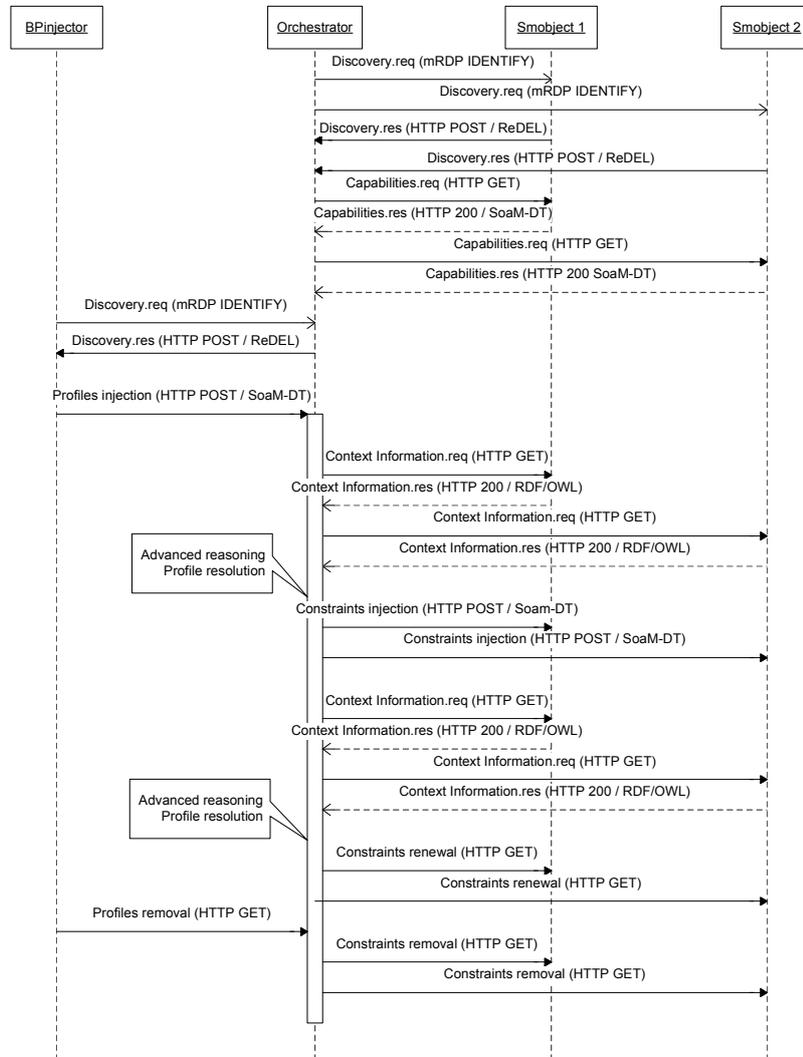


Figure 5.14: Interactions in a orchestrator-powered scenario.

- Behavioural profiles renewal, thus renewing the awareness behaviour in a concrete orchestrator or smobject.
- Behavioural profiles expiration, triggering associated constraints expiration and removal if they are no longer needed or generated by other profiles.

## 5.7 SoaM Discovery

The goal of SoaM discovery is to find the entities that can take part in SoaM activities throughout the network. BPinjectors use the discovery service to find orchestrators in the network, orchestrators use it to find smobjects, while smobjects can make use of it to find other smobjects. Therefore, searched entities are always either orchestrators or smobjects.

Every orchestrator exposes its functionality through an interface with three different services:

- **Discovery:** replies to discovery requests issued by other entities (orchestrators and BPinjectors).
- **Context information retrieval:** provides to requesting entities the context information collected by the orchestrator from all the managed smobjects, augmented after ontological and knowledge domain reasoning.
- **Behavioural profiles management:** provides management operations over the profiles that represent the required environmental behaviour the orchestrator is in charge of.

Smobjects expose their functionality through five different services in a very similar way:

- **Discovery:** the smobject replies to discovery requests issued by other entities.
- **Capabilities retrieval:** it provides descriptions about smobject perception and operation capabilities to requesting parties.
- **Context information retrieval:** it provides the context information captured by the smobject, as stated in its perception capabilities, to requesting parties.
- **Constraints management:** it provides management operations over the constraints that drive the smobject's behaviour as declared in its operation capabilities.
- **Behavioural profiles management service:** it provides management operations over the profiles that represent the required behaviour.

In order to provide these services we have created communication endpoints for each of them. The purpose of SoaM discovery is not only

to find orchestrators and smobjects in the network, but also to find their communication endpoints.

We have experimentally applied two different discovery protocols in SoaM for different evaluation purposes: UPnP SSDP and mRDP.

SSDP is not only used in UPnP as explained in section 2.2, but also is the discovery protocol applied in many other Ubiquitous Computing architectures such as Task Computing.

Despite the lack of standardisation approval in most of its protocols, including SSDP, UPnP is widely used by manufacturers in many computer platforms and devices. Extending these existing devices with SoaM, in order to “semanticise” them was an added challenge in our research, since we could not only reuse already existing devices and functionality but also increase it to make them smarter: semantic UPnP devices.

Therefore, one alternative for SoaM was to be 100% compatible with UPnP, taking existing UPnP devices and augmenting them with SoaM functionality, to make them more intelligent while still being accessible as normal UPnP devices to other UPnP entities.

On the other hand, mRDP is able to provide semantic discovery, which is much more powerful than SSDP.

We designed two different alternatives, so that SoaM can be deployed in both types of scenarios:

- Existing UPnP devices can be converted into smobjects and integrated in SoaM by slightly altering the SSDP layer and implementing the SoaM services.
- New “native” SoaM devices can implement the more powerful and efficient mRDP discovery protocol as well as the SoaM services.

### **5.7.1 SoaM discovery with UPnP integration and SSDP extensions**

Our first challenge in this approach was to model UPnP devices as smobjects without losing their UPnP nature. UPnP devices exhibit a unique device type. They can become smobjects by changing their type to `urn:awareit-com:device:smobject:1`, following the naming requirements provided by UPnP specification.

However, this alternative results in the device losing its native type, for example `urn:schemas-upnp-org:device:BinaryLight:1` in the case of a simple light. We wanted the UPnP device to share both natures at the same time: becoming a smobject while still retaining its initial device type unaltered.

The solution was to represent the smobject as an embedded device within the root device (see Figure 5.15). Since the UPnP specification allows a root device to contain any number of embedded devices, the smobject nature can be embodied in this way, without affecting the original device type.

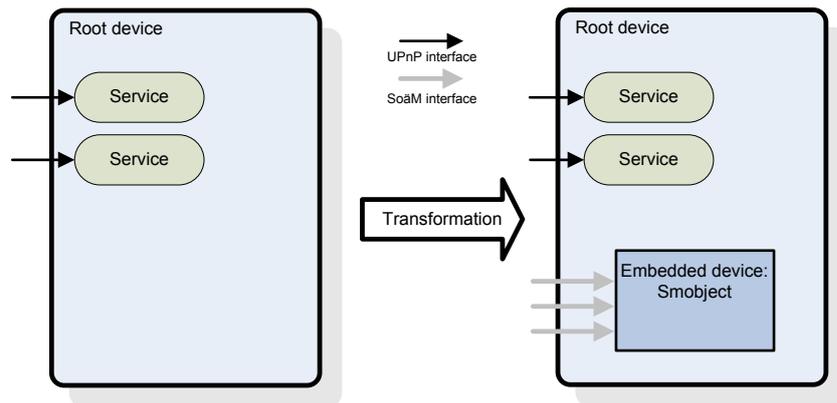


Figure 5.15: Transformation of a UPnP device into a SoaM UPnP device.

The smobject UPnP device is not populated by any UPnP services, since SoaM services are provided by non-UPnP interfaces, but the root device still provides the same UPnP services as the original device, thus remaining unaffected.

The problem now is how to notify SoaM entities about SoaM endpoints in the device using UPnP mechanisms. Since SSDP is the UPnP discovery protocol, we have extended SSDP to provide this information. SSDP was briefly described in section 2.2 and the complete specification can be found in [GC<sup>+</sup>99].

SSDP uses three headers to inform about the device type:

- **NT (Notification Type)**: the type of the device for notifications.
- **ST (Search Target)**: the type of the device for searching.
- **USN (Unique Service Name)**: a composite identifier formed by the device unique ID and its service type.

We have defined the values of these headers for smobjects as:

```
NT / ST: urn:awareit-com:device:smobject:1
USN: uuid:device-uuid::urn:awareit-com:device:smobject:1
```

And the values for orchestrators as:

```
NT / ST: urn:awareit-com:device:orchestrator:1
USN: uuid:device-uuid::urn:awareit-com:device:orchestrator:1
```

In order to provide the SoaM communication endpoints in the discovery messages, we have augmented SSDP specification with four new headers. The ABNF grammar of these headers is:

**Listing 5.2: ABNF grammar for SoaM SSDP extensions.**

```
1 h-soam-cap = "SoaM-Capabilities" ":" absoluteURI
2 h-soam-con = "SoaM-Constraints" ":" absoluteURI
3 h-soam-prof = "SoaM-Profiles" ":" absoluteURI
4 h-soam-info = "SoaM-Information" ":" absoluteURI
```

Smobjects will notify their capabilities, constraints, profiles and information communication endpoints through the correspondent headers, while orchestrators will use the “profiles” and “information” headers.

Listing 5.3 and Listing 5.4 illustrate a SoaM SSDP extended search interaction to find smobjects in the network, with one smobject replying.

**Listing 5.3: UPnP Control Point request message.**

```
1 M-SEARCH * HTTP/1.1
2 HOST: 239.255.255.250:1900
3 MAN: "ssdp:discover"
4 MX: 3
5 ST: urn:awareit-com:device:smobject:1
```

**Listing 5.4: UPnP Device response messages.**

```
1 HTTP/1.1 200 OK
2 Cache-Control: max-age=60
3 EXT:
4 Location: http://192.168.2.1/igd.xml
5 Server: DSL router/1.02 UPnP/1.0 UPnP-Device-Host/1.0
6 ST:uuid:00000000-0000-0001-0000-000d54a55be6
7 USN: uuid:00000000-0000-0001-0000-000d54a55be6
8
9 HTTP/1.1 200 OK
10 Cache-Control: max-age=60
11 EXT:
12 Location: http://192.168.2.1/igd.xml
13 Server: DSL router/1.02 UPnP/1.0 UPnP-Device-Host/1.0
14 ST: urn:awareit-com:device:smobject:1
15 USN: uuid:00000000-0000-0001-0000-000d54a55be6::urn:awareit-com:device
    :smobject:1
16 SoaM-Capabilities: http://http://192.168.2.1/capabilities
17 SoaM-Constraints: http://http://192.168.2.1/constraints
```

```

18 SoaM-Information: http://http://192.168.2.1/information
19 SoaM-Profiles: http://http://192.168.2.1/profiles

```

Since the smobject is an embedded UPnP device, it generates two replies. While `ST` and `USN` headers provide information for identifying the device and its type as smobject, the last four headers inform about the endpoints for communicating with the smobject following the SoaM Entity Management API – HTTP Binding (see section 5.8).

The receiving entity can use these endpoints for carrying out SoaM communication processes with the smobject.

### 5.7.2 SoaM discovery with mRDP and SoaMonto

mRDP can be used without alterations as discovery protocol for SoaM. In this case, resource identification queries using Plant can be disseminated to the network in order to identify the entities that are either smobjects or orchestrators.

Listing 5.5 contains a Plant query over mRDP to identify smobjects in the network.

Listing 5.5: Example of a `IDENTIFY` mRDP message during SoaM mRDP Discovery.

```

1 IDENTIFY ?r mRDP/1.0
2 NSeq: 23
3 Content-Type: application/com.awareit.plant
4 Content-Length: 110
5 Callback-URI: http://169.254.0.3/mrdpendpoint
6
7 ?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
   awareit.com/soam/2005/12/soamonto#Smobject> .

```

SoaMonto, the SoaM ontology described in section 5.9, declares OWL classes such as “smobject” or “orchestrator”, that are used for constructing the query. One of the advantages of mRDP is its expressiveness for the search process. For example, a client can issue a request to find sound-system smobjects used by a concrete person as illustrated in Listing 5.6.

Listing 5.6: Complex smobject discovery message in mRDP.

```

1 IDENTIFY ?r mRDP/1.0
2 NSeq: 23
3 Content-Type: application/com.awareit.plant
4 Content-Length: 316
5 Callback-URI: http://169.254.0.3/mrdpendpoint
6

```

```
7 ?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
  awareit.com/soam/2005/12/soamonto#Smobject> .
8 ?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
  awareit.com/onto/2005/12/sound#SoundSystem> .
9 ?r <http://pervasive.semanticweb.org/ont/2004/06/device#hasUser> <http
  ://people.com/bobby> .
```

Replies are sent to the provided callback URI using HTTP POST messages as explained in subsection 3.5. A possible response is depicted in Listing 5.7.

Listing 5.7: Example of a HTTP POST message during SoaM mRDP Discovery.

```
1 POST /mrdpendpoint HTTP/1.0
2 Host: 169.254.0.3
3 NSeq: 23
4 Content-Type: application/com.awareit.plant
5 Content-Length: 454
6
7 <?xml version="1.0" encoding="UTF-8"?>
8 <redel xmlns="http://www.awareit.com/soam/2006/04/redel"
9   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
10  xsi:schemaLocation="http://www.awareit.com/soam/2006/04/redel
11  http://www.awareit.com/soam/2006/04/redel.xsd">
12
13   <resource uri="urn:uuid:hifi1">
14     <location url="http://169.254.0.13/description" type="http://www
15     .awareit.com/soam/2006/04/srdfws#httpGet"/>
16   </resource>
17 </redel>
```

Orchestrators are discovered in a very similar way, using the Plant query:

```
?r <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
  awareit.com/soam/2005/12/soamonto#Orchestrator> .
```

Since SoaM concepts, such as “smobject” and “orchestrator” are defined in the ontology SoaMonto, the discovery process can be seamlessly integrated into a semantic-based query model such as Plant or SPARQL.

In order to obtain the communication endpoints for the discovered smobject, the URL provided in the ReDEL payload of the callback can be accessed (line 14 of Listing 5.7). Listing 5.8 shows the query and the response containing the RDF description of the smobject, including the communication endpoints using SoaMonto.

Listing 5.8: Example of smobject RDF description retrieval.

```
1 GET /information HTTP/1.0
2 Host: 169.254.0.13
3 Accept: application/rdf+xml, */*
4
5 HTTP/1.0 200 OK
6 Date: Wed, 21 Jun 2006 10:15:19 GMT
7 Cache-Control: max-age=21600
8 Expires: Wed, 21 Jun 2006 16:15:19 GMT
9 Last-Modified: Thu, 15 Dec 2005 14:59:50 GMT
10 Content-Length: 1209
11 Content-Type: application/rdf+xml
12
13 <?xml version="1.0"?>
14 <rdf:RDF
15     xmlns:owl="http://www.w3.org/2002/07/owl#"
16     xmlns="http://www.awareit.com/example2#"
17     xmlns:dev="http://pervasive.semanticweb.org/ont/2004/06/device#"
18     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
19     xmlns:soamonto="http://www.awareit.com/soam/2005/12/soamonto#"
20     xml:base="http://www.awareit.com/example2">
21
22     <owl:Ontology rdf:about="">
23         <owl:imports rdf:resource="http://www.awareit.com/soam/2005/12/
24             soamonto"/>
25     </owl:Ontology>
26
27     <rdf:Description rdf:about="urn:uuid:hifil">
28         <rdf:type rdf:resource="http://pervasive.semanticweb.org/ont
29             /2004/06/device#Device"/>
30         <rdf:type rdf:resource="http://www.awareit.com/onto/2005/12/sound#
31             SoundSystem"/>
32         <rdf:type rdf:resource="http://www.awareit.com/soam/2005/12/
33             soamonto#Smobject"/>
34
35         <soamonto:capabilitiesUri>http://169.254.0.13/capabilities</
36             soamonto:capabilitiesUri>
37
38         <soamonto:constraintsUri>http://169.254.0.13/constraints</soamonto
39             :constraintsUri>
40
41         <soamonto:informationUri>http://169.254.0.13/information</soamonto
42             :informationUri>
43
44         <soamonto:profilesUri>http://169.254.0.13/profiles</soamonto:
45             profilesUri>
46
47         <dev:hasUser rdf:resource="http://people.com/bobby"/>
48     </rdf:Description>
```

41  
42

```
</rdf:RDF>
```

It is remarkable how this reply does not provide just the communication endpoints, but also a rich description of the device, including the types (lines 27, 28 and 29) and even information about the user (line 39) that can be used by the receiving entity.

### 5.7.3 Comparison

Two discovery mechanisms have been designed for SoaM: the first one based on UPnP integration via SSDP extensions, and a second one based on mRDP semantic queries and SoaMonto. Table 5.2 compares advantages and drawbacks of both mechanisms.

Mechanism	Advantages	Drawbacks
SoaM SSDP extensions	Integration with already existent UPnP devices Simplicity	No security mechanisms Neither complex queries nor semantic discovery Communication not reliable (UDP)
mRDP+SoaMonto	Reliable replies (TCP) HTTPS and HTTP-based authentication Advanced queries and semantic discovery	Slightly more complex

Table 5.2: Comparison of SoaM SSDP extensions and mRDP as SoaM discovery mechanisms.

Both SSDP and mRDP require around 21 IP packets for a single search process encapsulating UDP datagrams or TCP segments:

- SSDP:
  - 1 UDP datagram for the search request (repeated 3 times)
  - 2 UDP datagrams for replying every request (repeated 3 times each), in the case no request packet is lost.
- mRDP:
  - 1 UDP datagram for the search request (repeated 3 times)

- 9 IP packets for HTTP callback (3 for connection opening, 1 for the HTTP request, 1 for the HTTP reply and 4 for TCP connection closing)
- 9 IP packets more for the resource information retrieval in a very similar way.

Since replies are reliable and requests have a sequence number, replies are not repeated even in the case the same request is received more than once.

UPnP specification states that SSDP packets should be sent more than once and three times at most to maximise the possibilities of reaching the destination without increasing significantly network traffic. If SSDP packets are sent just twice, the total number of IP packets for a search would be reduced to 10 (but the possibility of loss would increase).

TCP connection closing in mRDP can involve just 3 TCP segments, so reducing the number of IP packets to 19.

The requirement of mRDP about providing an HTTP server in the client for receiving the callbacks could be perceived as a disadvantage, but UPnP also requires an HTTP server in every UPnP device, so both alternatives feature this issue.

## 5.8 SoaM Entity Management API

As mentioned and illustrated earlier, the SoaM architecture fulfils our research goals by making extensive use of HTTP for communication purposes and proving how an advanced Ubiquitous Computing architecture can be designed based on this protocol. Even during discovery when mRDP is applied, we maximise the use of HTTP by providing a reliable means for callbacks; only multicasting is not possible using this protocol.

The reuse of HTTP everywhere in the smobject architecture promotes a small software footprint, embeddable in limited-resource devices as well as it provides a powerful and consistent communication model.

HTTP is used in several different activities involving SoaM entities and data types. As explained in section 4.4, we have designed two XML-based languages whose structures form the payload of most HTTP messages; except for context information messages, which convey RDF/XML, and mRDP callbacks that use ReDEL. The XML grammars in form of XML Schemas for these two languages, SoaM XML Datatypes and SoaM XML Exchange Messages, are included in appendix D.

However, not only the payload must be defined but also the relationships among the different possible calls, the HTTP endpoints, and the URI formats provided for requesting the services, that is, the web service description for SoaM.

As a first step, we have identified and defined the operations that can be performed over the different entities for requesting services. These are:

- Smobject:
  - Context Information [*retrieve*]: from smobjects, orchestrators and any other entity.
  - Capabilities [*retrieve*]: from smobjects, orchestrators and BPinjectors.
  - Behavioural profiles [*retrieve, add, renew and remove*]: from orchestrators and BPinjectors.
  - Constraints [*retrieve, add, renew and remove*]: from orchestrators.
  
- Orchestrator:
  - Context Information [*retrieve*]: from smobjects, orchestrators, BPinjectors and any other entity.
  - Behavioural profiles [*retrieve, add, renew and remove*]: from BPinjectors.

BPinjectors are not included as service providers, since they act as clients, requesting services from others.

We have labelled context information as being accessible from any entity because it is provided in RDF/XML format, so that it can be collected from any other Semantic Web infrastructure and integrated into business processes.

Context information and capabilities services do not provide advanced management mechanisms over HTTP; they are just endpoints where these structures can be downloaded from.

However, behavioural profiles and constraints can be managed from external sources via HTTP operations, so that the correspondent endpoints must provide facilities for requesting the four operations referred above: retrieve, add, renew and remove.

In order to specify the HTTP interface for invoking these operations, we have applied the widely used Web Services Description Language (WSDL)[Wor01a]. However, limited devices hosting smobjects are not

expected to be able to process and deserialise SOAP [Wor03] messages. That is the reason we have also designed an alternative, more compact HTTP interface based on simple HTTP GET and POST requests, less expressive than SOAP, but easier to process by limited devices.

The HTTP interface for managing behavioural profiles and constraints, provided in both more expressive (SOAP binding) and more compact (simple HTTP binding, GET and POST) versions, constitute the SoaM Entity Management API.

In the next subsection, we describe the compact version. The full WSDL description, including the SOAP version, is provided in appendix E.

### 5.8.1 HTTP binding operations and messages

Let us suppose that the endpoint URLs for behavioural profiles and constraints management provided by a smobject (the same applies to the orchestrator in the case of behavioural profiles) are respectively

```
http://169.254.0.3/profiles
http://169.254.0.3/constraints
```

For the HTTP binding, we have created a simple mechanism to identify the required operation, based in adding the appropriate suffix to the URI:

- Retrieve: (nothing). E.g. `http://169.254.0.3/constraints`
- Add: (?add). E.g. `http://169.254.0.3/constraints?add`
- Renew: (?renew). E.g. `http://169.254.0.3/constraints?renew`
- Remove: (?remove). E.g. `http://169.254.0.3/constraints?remove`

Using this mechanism, only one management endpoint URL is required for the concrete element type (constraint or profile), and operations are identified by adding the correspondent suffix as explained and illustrated below.

#### Retrieve

The retrieve operation is performed by issuing a simple HTTP GET request on the endpoint URL. The requested entity must reply with the SoaM XML Datatypes message conveying the results, or RDF/XML in the case of context information.

For example, the messages involved in retrieving the constraints using the previous URL are illustrated in Listing 5.9.

Listing 5.9: Example of constraints retrieval.

```
1 GET /constraints HTTP/1.0
2 Host: 169.254.0.3
3 Accept: text/xml,application/xml,application/com.awareit.soamdt+xml
  , */*
4 Connection: close
5
6 HTTP/1.0 200 OK
7 Content-Type: application/com.awareit.soamdt+xml
8 Date: Tue, 11 Jul 2006 14:12:01 GMT
9 Content-Length: 629
10
11 <?xml version="1.0" encoding="UTF-8" ?>
12 <constraintsCollection xmlns="http://www.awareit.com/soam/2006/02/
   soamdt" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
   schemaLocation="http://www.awareit.com/soam/2006/02/soamdt http://
   www.awareit.com/soam/2006/02/soamdt" owner="urn:uuid:tempControll
   ">
13
14 <constraint id="urn:uuid:9248857817233744" requester="http://people.
   com/bobby" subject="urn:uuid:room1" predicate="http://www.awareit.
   com/onto/2005/12/temperature#hasTemperature" expires="PT1M9S">
15   <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int
   ">24</objectLiteral>
16 </constraint>
17
18 </constraintsCollection>
```

### Add

The “add” operation injects several instances of the data structure, behavioural profile or constraint, into the HTTP endpoint for processing. Every time we mentioned “injection” during this chapter we meant an add operation.

The “add” operation is carried out by issuing an HTTP POST request conveying the profiles in the body, on the endpoint URL augmented with the operation suffix ?add, for example

```
http://169.254.0.3/profiles?add
```

Listing 5.10 illustrates the injection of two behavioural profiles into a smobject, using the correspondent endpoint.

Listing 5.10: Example of constraints injection with add.

```

1 POST /profiles?add HTTP/1.0
2 Content-Length:2352
3 Connection:close
4 User-Agent:awareIT.com/HTTP Java client 1.0
5 Host:169.254.0.3
6
7 <?xml version="1.0" encoding="UTF-8" ?>
8 <behavioralProfilesCollection xmlns="http://www.awareit.com/soam
   /2006/02/soamdt" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xsi:schemaLocation="http://www.awareit.com/soam/2006/02/
   soamdt http://www.awareit.com/soam/2006/02/soamdt">
9 <behavioralProfile id="urn:uuid:prof2" requester="http://people.com/
   bobby" expires="PT2M0S">
10 <variable xml:id="x2"/>
11 <variable xml:id="y2"/>
12 <precondition id="" subject="http://people.com/bobby" predicate="
   http://www.awareit.com/onto/2005/12/task#isDoing">
13 <objectResource resource="http://www.awareit.com/onto/2005/12/
   task#WorkingWithLaptop"/>
14 </precondition>
15 <precondition id="" subject="http://people.com/bobby" predicate="
   http://www.awareit.com/onto/2005/12/location#isLocatedIn">
16 <objectVariable ref="x2"/>
17 </precondition>
18 <precondition id="" subject="#x2" predicate="http://www.awareit.
   com/onto/2005/12/sound#hasSound">
19 <objectVariable ref="y2"/>
20 </precondition>
21 <postcondition id="" subject="#y2" predicate="http://www.awareit.
   com/onto/2005/12/sound#volume">
22 <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int
   ">3</objectLiteral>
23 </postcondition>
24 <postcondition id="" subject="#y2" predicate="http://www.awareit.
   com/onto/2005/12/sound#style">
25 <objectResource resource="http://www.awareit.com/onto/2005/12/
   sound#ClassicalMusic"/>
26 </postcondition>
27 </behavioralProfile>
28 <behavioralProfile id="urn:uuid:prof3" requester="http://people.com/
   bobby" expires="PT2M0S">
29 <variable xml:id="x3"/>
30 <variable xml:id="y3"/>
31 <precondition id="" subject="http://people.com/bobby" predicate="
   http://www.awareit.com/onto/2005/12/task#isDoing">
32 <objectResource resource="http://www.awareit.com/onto/2005/12/
   task#Talking"/>
33 </precondition>

```

```
34 <precondition id="" subject="http://people.com/bobby" predicate="
    http://www.awareit.com/onto/2005/12/location#isLocatedIn">
35 <objectVariable ref="x3"/>
36 </precondition>
37 <precondition id="" subject="#x3" predicate="http://www.awareit.
    com/onto/2005/12/sound#hasSound">
38 <objectVariable ref="y3"/>
39 </precondition>
40 <postcondition id="" subject="#y3" predicate="http://www.awareit.
    com/onto/2005/12/sound#volume">
41 <objectLiteral datatype="http://www.w3.org/2001/XMLSchema#int
    ">0</objectLiteral>
42 </postcondition>
43 </behavioralProfile>
44 </behavioralProfilesCollection>
45
46 HTTP/1.0 200 OK
47 Content-Length:534
48 Date:Tue, 11 Jul 2006 14:38:51 GMT
49 Content-Type:application/com.awareit.soammsg+xml
50
51 <?xml version="1.0" encoding="UTF-8"?>
52 <soamResultsCollection xmlns="http://www.awareit.com/soam/2006/02/
    soammsg" xmlns:soamdt="http://www.awareit.com/soam/2006/02/soamdt"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    schemaLocation="http://www.awareit.com/soam/2006/02/soammsg http
    ://www.awareit.com/soam/2006/02/soammsg">
53
54 <soamResult ref="urn:uuid:prof2" code="200" expires="PT2M0S">Accepted
    </soamResult>
55 <soamResult ref="urn:uuid:prof3" code="200" expires="PT2M0S">Accepted
    </soamResult>
56
57 </soamResultsCollection>
```

The first remarkable aspect is the format of the response, whose body contains a `soamResultsCollection` with individual results for every data structure conveyed in the request. In the example, two profiles were sent, so the response is populated with two `soamResult` elements, each referencing the appropriate profile and informing about:

- The status code associated to the profile. In both cases is 200, which means that the profile has been accepted.
- The expiration time configured for the profile. In both cases, 2 minutes, as requested. However, the smobject or orchestrator, could have reduced the accepted period not to exceed the maximum allowed expiration period.

- A text message, for debugging purposes.

The response message body uses SoaM XML Exchange Messages, whose root element is `soamResultsCollection`.

### Renew and remove

These operations can be performed either by HTTP GET or HTTP POST, but there is an important difference: using HTTP GET only one managed entity (constraint or profile) can be renewed or removed at a time, while using HTTP POST several entities can be renewed or removed.

This difference originates from the fact that HTTP POST requests contain a message body in which a `soamEntityIdsCollection` element can be embedded with multiple `soamEntityId` containing the unique URIs for the constraints or profiles.

On the other hand, HTTP GET operations are not allowed to contain a message body, so the constraint or profile is identified via URL encoding using the query part of the URL with the parameter `soamEntityId`.

Either if HTTP POST or GET are used, the reply contains a `soamResultsCollection` with individual results for each addressed resource, although in the case of HTTP GET only one result will be provided.

For example, Listing 5.11 illustrates a renew interaction using HTTP GET.

Listing 5.11: Example of constraint renewal using HTTP GET.

```
1 GET /constraints?renew&soamEntityId=urn:uuid:constraint1 HTTP/1.0
2 Connection:close
3 User-Agent:awareIT.com/HTTP Java client 1.0
4 Host:169.254.0.3
5
6 HTTP/1.0 200 OK
7 Content-Length:459
8 Date:Thu, 13 Jul 2006 15:52:27 GMT
9 Content-Type:application/com.awareit.soammsg+xml
10
11 <?xml version="1.0" encoding="UTF-8"?>
12 <soamResultsCollection xmlns="http://www.awareit.com/soam/2006/02/
13   soammsg" xmlns
14   :soamdt="http://www.awareit.com/soam/2006/02/soamdt" xmlns:xsi="http
15   ://www.w3.or
16   g/2001/XMLSchema-instance" xsi:schemaLocation="http://www.awareit.com/
17   soam/2006/
18   02/soammsg http://www.awareit.com/soam/2006/02/soammsg">
19   <soamResult ref="urn:uuid:constraint1" code="200" expires="PT2M0S">
```

```
17     Renewed
18   </soamResult>
19 </soamResultsCollection>
```

On the other hand, Listing 5.12 illustrates a remove interaction using HTTP POST, to remove two constraints, where the second constraint was not found at the smobject.

**Listing 5.12: Example of constraint removal using HTTP POST.**

```
1 POST /constraints?remove HTTP/1.0
2 Connection:close
3 User-Agent:awareIT.com/HTTP Java client 1.0
4 Host:169.254.0.3
5
6 <?xml version="1.0" encoding="UTF-8"?>
7 <soamEntityIdsCollection xmlns="http://www.awareit.com/soam/2006/02/
   soammsg" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi
   :schemaLocation="http://www.awareit.com/soam/2006/02/soammsg
8 http://www.awareit.com/soam/2006/02/soammsg.xsd">
9   <soamEntityId ref="urn:uuid:constraint1"/>
10  <soamEntityId ref="urn:uuid:constraint2"/>
11 </soamEntityIdsCollection>
12
13 HTTP/1.0 200 OK
14 Content-Length:576
15 Date:Thu, 13 Jul 2006 15:54:26 GMT
16 Content-Type:application/com.awareit.soammsg+xml
17
18 <?xml version="1.0" encoding="UTF-8"?>
19 <soamResultsCollection xmlns="http://www.awareit.com/soam/2006/02/
   soammsg" xmlns
20 :soamdt="http://www.awareit.com/soam/2006/02/soamdt" xmlns:xsi="http
   ://www.w3.or
21 g/2001/XMLSchema-instance" xsi:schemaLocation="http://www.awareit.com/
   soam/2006/
22 02/soammsg http://www.awareit.com/soam/2006/02/soammsg">
23   <soamResult ref="urn:uuid:constraint1" code="200">
24     Removed
25   </soamResult>
26   <soamResult ref="urn:uuid:constraint2" code="404">
27     Not found
28   </soamResult>
29 </soamResultsCollection>
```

## 5.9 SoaMonto: the SoaM support ontology

We have created SoaMonto, the SoaM support ontology to describe semantically all the entities and concepts that take part in SoaM. SoaMonto defines 14 classes, 16 properties and 12 individual instances. Some of these elements are widely used across SoaM in different parts of the process, while others can only be applied to specific situations.

We have already introduced the use of SoaMonto during the discovery process in order to find smobjects or orchestrators by disseminating a Plant query, which is a semantic query after all, through the network. The use of `soamonto:Any` was also introduced in section 4.4 for representing wildcards in capabilities.

We designed SoaMonto to provide a means for representing some information and structures about the SoaM architecture in a semantic way, so that intelligent processes with reasoning mechanisms can take advantage of this information to produce new conclusions.

The main advantage of this approach is that SoaM-related management data become context information *per se* and can be used to describe a smobject, an orchestrator or any other SoaM concept. Using SoaMonto, the SoaM architecture itself becomes semantically self-descriptive.

For example, behavioural profiles and constraints are exchanged using SoaM XML Datatypes; but a concrete smobject can also include the profiles or constraints that drive its behaviour as part of the semantic information provided as if it was captured by a virtual “Smobject Perceptor”.

Very much in the same way, using RDF and the SoaM ontology, an orchestrator can describe which smobjects have been discovered, the set of managed behavioural profiles and even which constraints have been injected in which smobjects.

SoaM XML Datatypes and SoaMonto are not competing mechanisms, but rather they serve for different purposes: while SoaM XML Datatypes are designed for data structures exchange via HTTP messages as explained earlier, SoaMonto defines semantic relationships among SoaM elements in order to represent available knowledge about the system in a concrete moment of time.

The full specification of SoaMonto in RDF/XML is included in appendix F.

### 5.9.1 SoaMonto classes

We have designed 14 classes representing concepts in SoaM. They form the SoaMonto taxonomy:

- **Capability:** extends `rdf:Statement`. It represents a smobject capability.
- **Entity:** represents a SoaM entity.
- **Smobject and Orchestrator:** they are subclasses of `Entity` to represent a particular smobject or orchestrator.
- **Condition:** a subclass of `rdf:Statement`, augmented with an `operator` property.
- **Variable:** represents a variable for conditions.
- **Operator:** represents an operator for conditions and constraints.
- **Constraint:** declared as subclass of `Condition`, since basically a constraint is a fully resolved condition (a condition with no `Variable` resources).
- **BehaviouralProfile:** represents a profile relating preconditions and postconditions.
- **Rule:** represents a domain knowledge rule.
- **Reasoner:** represents a reasoner attached to a smobject or orchestrator.
- **LocalReasoner and RemoteReasoner:** subclasses of `Reasoner`.
- **ReasoningMechanism:** represents a reasoning mechanism of a `Reasoner`, such as an OWL DL reasoner, an OWL Lite reasoner, a generic rule reasoner and so forth.

Every remarkable element taking part in SoaM can be represented using these classes. In fact `Smobject` and `Orchestrator` are used during the discovery process with mRDP (see subsection 5.7.2); and `Reasoner`, `LocalReasoner`, `RemoteReasoner`, `ReasoningMechanism` and `Rule` can be used to characterise the reasoner in the smobject or orchestrator as well as to identify rules in the ubiquitous network (see subsection 5.3).

Other advanced semantic searches are possible. For example, if a BPinjector needs to find orchestrators and smobjects in the network, instead

of issuing two different requests for each type, it could issue just one request for a type `Entity`<sup>8</sup>.

### 5.9.2 SoaMonto properties

We have designed 16 properties to create relationships among SoaMonto concepts. The following list includes their domains and ranges:

- `hasPerceptionCapability` and `hasOperationCapability` [`domain = Smobject`, `range = Capability`]: declares the capabilities of a concrete smobject.
- `hasPrecondition` and `hasPostcondition` [`domain = BehaviouralProfile`, `range = Condition`]: declares the preconditions and postconditions of a behavioural profile.
- `operator` [`domain = Condition`, `range = Operator`]: declares an operator for the condition (or constraint, since it is a subclass of the former).
- `capabilitiesUri` and `constraintsUri` [`domain = Smobject`, `range = xsd:anyURI`]: these are data type properties containing the endpoint URI of capabilities and constraints for the smobject.
- `informationUri` and `profilesUri` [`domain = Entity`, `range = xsd:anyURI`]: these are endpoints for context information and profiles management associated to SoaM entities, and thus its subclasses (smobject and orchestrator).
- `isConstrainedBy` [`domain = Smobject`, `range = Constraint`]: declares the constraints that currently drive a smobject's behaviour.
- `observesBehaviour` [`domain = Entity`, `range = BehaviouralProfile`]: declares the profiles honoured by an entity (smobject or orchestrator).
- `manages` [`domain = Orchestrator`, `range = Smobject`]: declares the smobjects managed by an orchestrator.
- `managedBy` [`domain = Smobject`, `range = Orchestrator`]: declares the orchestrator that manages a concrete smobject. It is the inverse of `manages`.

---

<sup>8</sup>For this request to be successful, the receiving entities must be able to perform ontological processing based on SoaMonto.

- `involves` [domain = Rule, range = owl:Ontology]: associates a concrete rule with related ontologies.
- `usesReasoner` [domain = Entity, range = Reasoner]: declares the reasoners used by a smobject or an orchestrator.
- `applies` [domain = Reasoner, range = ReasoningMechanism]: declares the reasoning mechanisms used by a reasoner.

These properties enable the creation of rich relationships among instances of the above SoaMonto classes: declaring which profiles are being honoured by a concrete smobject or orchestrator, specifying all the relevant details about the reasoners, declaring the active constraints in a smobject, and so forth.

In this way, the full description of a concrete SoaM element can be obtained and interpreted.

### 5.9.3 SoaMonto instances

We have designed 12 individual instances of SoaMonto classes, required in the model. This list can be extended with new instances as needed:

- `Any` [is a owl:Thing]: represents the wildcard in capabilities.
- `Equals`, `NotEquals`, `GreaterThan`, `GreaterOrEqualsThan`, `LessThan` and `LessOrEqualsThan` [is a Operator]: represent different SoaM built-in operators for conditions and constraints.
- `GenericRules`, `OwlFull`, `OwlDL`, `OwlLite` and `Rdfs` [is a ReasoningMechanism]: represent different reasoning mechanisms to characterise reasoners.

These built-in predefined instances of SoaMonto may be used throughout the architecture as explained earlier.

### 5.9.4 Examples

Let us suppose that a concrete smobject represents its internal state in terms of active profiles and constraints using SoaMonto, augmenting its perceived context information with these data. In this way, when the smobject is queried, it does not only provide semantised data about sensorial perceptions, but also its internal state as mentioned.

For example, if the smobject’s behaviour was driven by one behavioural profile and one constraint, the resulting RDF graph, including an hypothetically perceived temperature value, could be that depicted in Figure 5.16.

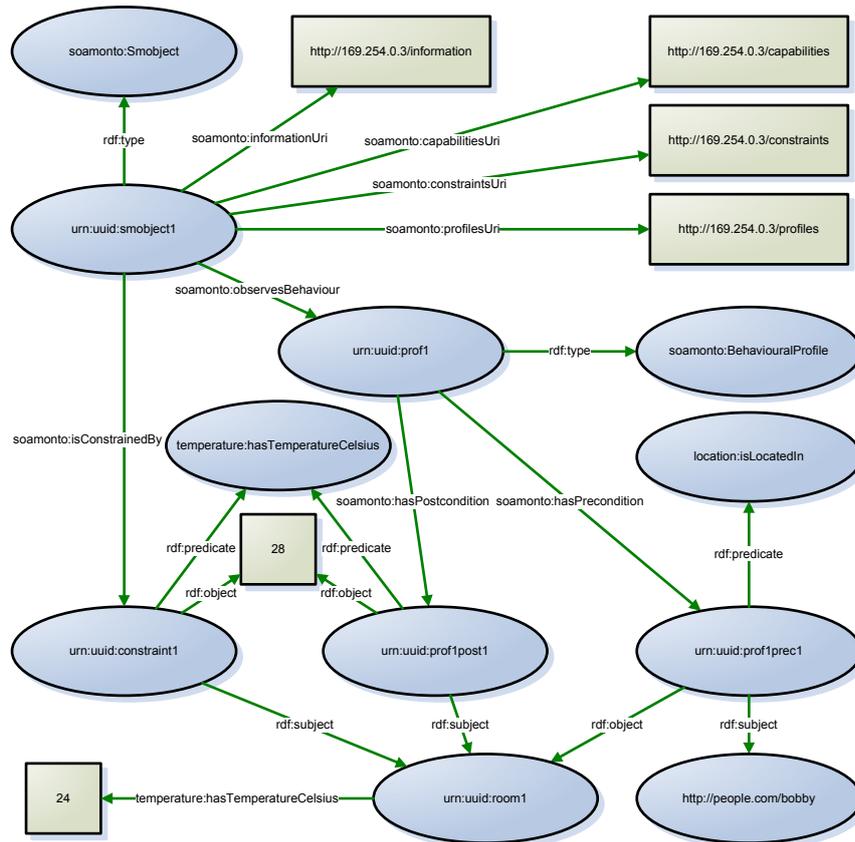


Figure 5.16: Context information provided by a smobject, including SoaMonto data.

The upper part of the RDF graph provides information about the smobject, its type and its communication endpoints. The medium and lower right part represents the behavioural profile and the constraint currently active. The profile is formed by one precondition (“if Bobby is located in room1”) and one postcondition (“then room1 must have temperature 28°C”).

The current perceived information is depicted in the lower left part, asserting that “room1 has a temperature of 24°C” (thus, the constraint has not been fully met yet, but probably the smobject is increasing the temperature steadily though effectors / actuators).

For the sake of clarity in this example we omitted the declaration of other relationships and concepts that would be present normally such as perception and operation capabilities and some types.

We would like to remark again that SoaMonto does not only play an important role in the whole architecture, used in different parts, but it may also be fully exploited to represent SoaM-related context information for self-describing purposes, as shown in the example.

Since SoaMonto is a fundamental ontology, SoaM entities (smobjects and orchestrators), should always have it available in the local storage for the built-in reasoner.

In this way, the retrieval of capabilities, constraints and profiles could be carried out by applying SoaMonto and RDF/XML serialisation through the context information interface, instead of using the retrieval operation defined in the SoaM Entities Management API – HTTP Binding, explained in section 5.8.

However, the increment in the amount of resulting context information, higher RDF processing requirements at the smobject side, and the obligation of keeping on providing the SoaM Entities Management API – HTTP Binding for the rest of operations (add, renew and remove), disallowed this approach<sup>9</sup>.

## 5.10 Comparative analysis

We have tested the two possible topologies of SoaM against the evaluation criteria, with the following results:

**Decentralisation:** while the smobjects-only architecture is completely decentralised, the orchestrator-powered topology fails to fulfill this criterium as other similar architectures. SoaM smobjects: High; SoaM orchestrator: Low.

**Reasonability:** smobjects feature a limited level of reasoning, applying ontologies (a subset of OWL Lite) and rules, while the orchestrator-powered topology features a full reasoner, that can be even extended with new mechanisms. SoaM smobjects: Medium; SoaM orchestrator: High.

---

<sup>9</sup>Further discussion about this is provided in section 7.2.

**Context-awareness:** both topologies feature a high level of reactivity to changes in context information supported by dynamic behavioural profiles. SoaM smobjects: High; SoaM orchestrator: High.

**Technological Consistency:** the exclusive application of web technologies to design the majority of aspects of SoaM such as behavioural profiles, context-information descriptions, capabilities and so on, contributes to an unprecedented technological consistency, similar to that provided in UPnP. Same APIs, protocol layers and framework facilities can be used for multiple activities both at the smobjects and at the orchestrator. SoaM smobjects: High; SoaM orchestrator: High.

**Standards Adherence:** SoaM strictly adheres to existing standards such as HTTP, XML, RDF, OWL and so forth. Only mRDP related aspects are specifically designed for SoaM and yet highly integrated with other existing technologies such as SPARQL. SoaM smobjects: High; SoaM orchestrator: High.

**Device Implementation Cost:** as a distinguishing aspect, SoaM is targeted to embedded platforms. Components such as the SmobjectBase and SmobjectAware can be easily implemented limited devices with minimum requirements, only UPnP is lighter. SoaM smobjects: Medium; SoaM orchestrator: Medium.

**Environment Deployment Cost:** the smobjects-only topology does not require any kind of preparation in the environment. Smobjects dynamically discover each other and collaborate spontaneously to honour their behavioural profiles. On the other hand, the orchestrator-powered topology requires an orchestrator to be deployed in the environment previously. SoaM smobjects: Low; SoaM orchestrator: High.

**Lightness:** the SmobjectBase component is very light and easy to implement in resource constrained devices, and the SmobjectAware component can perform reasoning in embedded platforms with little extra computing power. However, the orchestrator cannot be implemented in such platforms. SoaM smobjects: Medium; SoaM orchestrator: Low.

**Autonomy:** both topologies are completely autonomous and self-manageable. No user intervention is required as the activities are carried out automatically by smobjects and the orchestrator after discovery. SoaM smobjects: High; SoaM orchestrator: High.

The Table 5.3 compares the values obtained by the different technologies analysed in chapter 2 with those provided by the two possible topologies of

SoaM. The weighted final values have been obtained following the same procedure as in the original comparative (e.g. the two economical criteria are interpreted as negative figures, thus promoting low values).

SoaM promotes a balance between decentralisation and intelligence, being light enough for implementation in embedded devices and maintaining low implementation and deployment costs. It meets our criteria naturally, ranking exceptionally well in some of them.

As already mentioned, SoaM distinguishes itself from other alternatives by leveraging spontaneous collaboration and semantic information exchange among devices that can semantically interpret and react to changes in context information. SoaM is a Ubiquitous Computing architecture powered by semantic devices.

<b>Criterion</b>	<b>UPnP</b>	<b>Task Computing</b>	<b>CoBra SOUPA</b>	<b>Gaia</b>	<b>Semantic Spaces SOCAM</b>	<b>SoaM smobjects-only</b>	<b>SoaM orchestrator-powered</b>
<b>Decentralisation</b>	Medium(8)	Low(4)	Low(4)	Low(4)	Low(4)	High(12)	Low(4)
<b>Reasonability</b>	None(0)	Low(4)	High(12)	Very High(16)	High(12)	Medium(8)	High(12)
<b>Context-awareness</b>	None(0)	Low(4)	Medium(8)	High(12)	High(12)	High(12)	High(12)
<b>Technological Consistency</b>	High(9)	High(9)	Low(3)	Low(3)	Medium(6)	High(9)	High(9)
<b>Standards Adherence</b>	Medium(4)	High(6)	High(6)	High(6)	Medium(4)	High(6)	High(6)
<b>Device Implementation Cost</b>	Low(-2)	High(-6)	High(-6)	High(-6)	High(-6)	Medium(-4)	Medium(-4)
<b>Environment Deployment Cost</b>	Low(-2)	Medium(-4)	High(-6)	High(-6)	High(-6)	Low(-2)	High(-6)
<b>Lightness</b>	High(3)	Low(1)	Low(1)	Low(1)	Low(1)	Medium(2)	Low(1)
<b>Autonomy</b>	Low(1)	Low(1)	High(3)	High(3)	High(3)	High(3)	High(3)
<b>TOTAL</b>	21	19	25	33	30	46	37

Table 5.3: Analysis of SoaM and other technologies against the evaluation criteria.



## Prototypes and Evaluation

*“That was its trial run, the first time it had ever been tested [. . . ]  
This was the first full-scale operational run of the ENIAC.”*

*Mike Hally  
Electronic Brains, 2006*

**W**<sup>E</sup> already mentioned how the design process was highly linked with the implementation activity, during which increasingly more complete versions of the prototypes were implemented and tested, thus validating both the theoretical model and the architecture.

The feedback and results we obtained contributed to improve architectural designs as these prototypes were progressively built.

The iterative implementation process consisted in the following steps (see Figure 6.1):

1. Selection of the target (module / component / functionality) of the architecture to implement.
2. Implementation of the target.
3. Execution of unit tests of the target and development of required corrections.
4. Execution of integrated tests of the target and development of required corrections.
5. Execution of performance tests of the target, and development of corrections to improve efficiency.

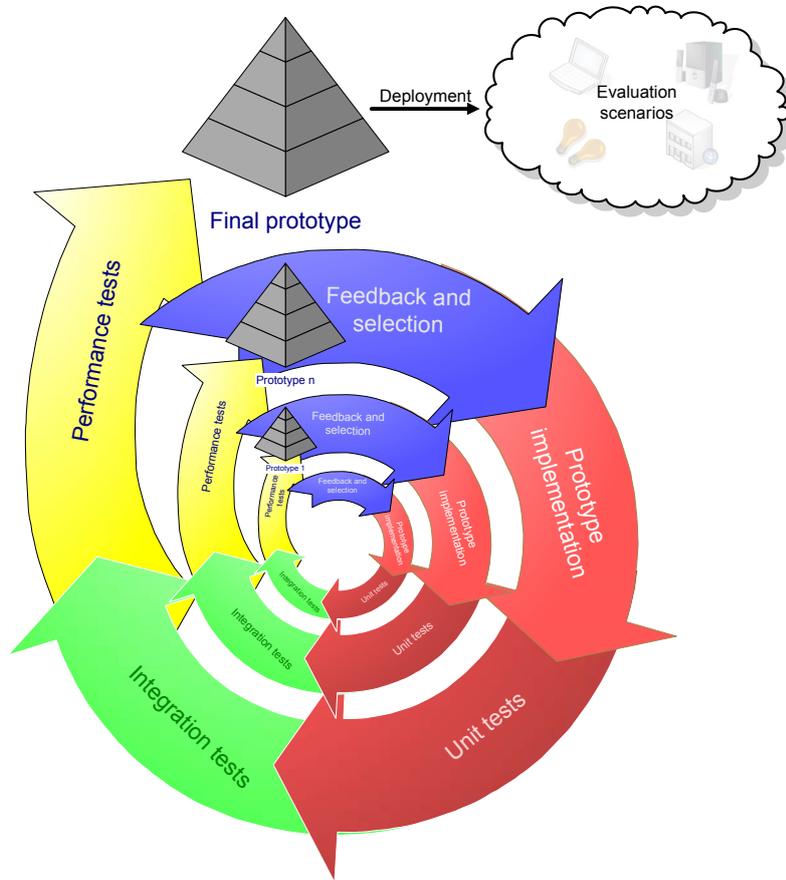


Figure 6.1: Incremental prototyping and testing process in SoAM.

Finally, we recreated some evaluation scenarios and carried out end-to-end performance tests throughout the whole system, with smobjects hosted in an assortment of commercial embedded platforms.

Performance results heavily depend on the computing power of the platform, specially when it comes to dealing with somehow complex information structures in limited devices, but we think that our evaluation results can provide some valuable information for future semantic devices developers.

## 6.1 Prototyping

The order in which the main components of the architecture were implemented and tested in the prototypes depended inversely on the degree of uncertainty about possible implementation problems. Therefore, we started with the simpler and less problematic components, augmenting the prototype by adding more complex ones, gradually.

The order in which the elements of SoaM architecture were implemented is:

1. Sobject base components
2. SoaM UPnP SSDP extensions
3. Orchestrator with semantic reasoning engine
4. BPinjector
5. mRDP client and server
6. Sobject awareness components
7. Sobject embedded reasoner

The implementation of the first three elements (base components, UPnP SSDP extensions and the orchestrator) enabled us to validate the orchestrator-powered architecture, and thus, the awareness process without any kind of semantic processing in the sobject, whose feasibility was one of our main concerns at that time. The BPinjector contributed to end-to-end validation of the system .

This first high-level iteration of the architecture already retained some of the major principles and requirements we addressed at the beginning of our research (such as reasonability, context-awareness and standards adherence) while still lacking of some others (such as decentralisation and a higher degree of technological consistency).

The more difficult components to implement in the embedded platform were the mRDP discovery client and server, the awareness components and the reasoning engines (MiniOWLReasoner and MiniRuleReasoner). mRDP was initially implemented and tested in PC platforms, but some adaptation and redesign was needed to make it work in more resource-constrained devices as explained further below.

The awareness components implementation was one of the most difficult activities, since involved RDF processing and profiles resolution at the

smobject; we did not know whether it was going to work with an acceptable performance before testing it. Finally, the reasoning engines implementation was specially time consuming due to several optimisations in order to save processing times and memory.

The development language and software platform was Java, because one of our main concerns was to create an easy-to-port middleware that could be migrated to different underlying computing systems, so that other researchers in the future were able to design and create smobjects just reusing the core components. We even did not know if, during the development of the project, other hardware platforms were going to be available, so that the implementation could be easily migrated.

This decision proved to be correct when, at the final steps of the testing, several embedded platforms were available and prototype porting was as easy as copying and configuring the smobject middleware. At the end of prototyping, smobject components have been successfully hosted in six different computing platforms:

- ARM7-based ConnectCore 7U with  $\mu$ Clinux
- ARM9-based ConnectCore 9U with Linux
- Intel XScale PXA255 Gumstix with Linux
- Intel Pentium Mobile with Windows XP
- Intel Pentium Mobile with Linux
- Intel Core Duo with Mac OS

Regarding the orchestrator component, since most of existing Semantic Web libraries are developed in Java, the orchestrator can reuse existing reasoning engines developed and maintained by open-source communities.

In the following subsections we describe how the implementation process was carried out for each major component.

### 6.1.1 Smobject base components

The initial hardware and software platform for hosting the smobject was the Digi ConnectCore 7U (earlier, FS Forth UNC20) with an ARM7 microprocessor at 55 MHz., a  $\mu$ Clinux kernel and a stripped-down Java Virtual Machine called mi|k|a.

The ConnectCore 7U also featured 8 Mb Flash and 16 Mb RAM, 2 serial interfaces and Ethernet connectivity, easily convertible into Wi-Fi

connectivity, in a relatively small form factor of  $6.28 \text{ cm} \times 1.85 \text{ cm} \times 1.04 \text{ cm}$  ( $12.08 \text{ cm}^3$ , see Figure 6.2).



Figure 6.2: Image of the ConnectCore 7U platform that hosted the smobject.

All the base components were implemented following the smobject architectural designs, along with additional helper classes and functions. Managing an XML parser to work in this platform was specially difficult, since Java platform libraries required by most of parsers are not included in mi|k|a.

After trying with several candidates, the MinML parser seemed to work. We augmented this parser by developing a subsystem for initially parsing results into memory and looking for XML elements and attributes there, releasing the memory afterwards. This enabled us to build high-level XML functions for parsing and generating XML documents.

Multicast networking, needed both for SSDP and mRDP, required a special configuration of the kernel that had to be recompiled.

The use of the serial port from Java is a typical problem in many computing platforms, since is very specific to the particular hardware and operating system involved, and it took some time before it worked in the ConnectCore 7U.

One of the most remarkable aspects of the implementation architecture was the design of a pluggable protocol subsystem: since there were at least two possible communication protocols for SoaM SSDP extensions and mRDP, we designed an independent mechanism, so that any protocol honouring an specific interface could be plugged as communication protocol for the smobject.

The result is that, not only SSDP extensions and mRDP can be used in the smobject, but any other protocol or communication mechanism, including Bluetooth or Zigbee -compatible ones could be used for discovery and communication.

Moreover, the architecture promotes the use of several protocols at the same time, by plugging them into the Entity Manager, which searches for

smobjects throughout installed protocols, keeping track and communicating with them.

For testing purposes, several helper perceptors and effectors were designed:

- `PerceptorFile`: convenience perceptor that reads information from a file, simulating a sensor, and makes it available in a semantic way.
- `EffectorFile`: convenience effector that uses a file as actuator, writing the desired actions, and thus logging the behaviour.
- `PerceptorHttp`: similar to the `PerceptorFile`, but reading the information from a remote URL, acting as a sensor.
- `EffectorHttp`: similar to the `EffectorFile`, but writing the actions on a remote process accessible via an URL, and simulating an actuator.

These peceptors and effectors enabled us to perform a broad range of simulations and to design smobjects that emulated different devices as illustrated in the examples throughout this dissertation. Changes in context information were performed by altering the perception files, either manually or by an automated process.

These perceptors are not only suitable for testing purposes but also for real execution environments: since they take a file as default input, only an external process is required to write that file with the real perceived information to have an end-to-end complete system. For example, if a camera surveillance system provides an URL where the captured image and movement information is posted, a particular `PerceptorHttp` can be designed to download these data and semantically annotate them with the appropriate ontology.

We also developed a small HTTP server and client for the base components. The HTTP server featured a handler-based architecture, so that different handlers could be associated to different URL prefixes: when the server received the request, examined the URL and delivered it to the appropriate handler, which generated a response for the server, that passed it back to the original requesting client.

This strategy enabled the use of a single HTTP server from different modules and became even more important when we added the mRDP protocol, since discovery replies are HTTP requests that must be notified to the mRDP client.

We had to create a new HTTP client from scratch, since the one provided in mi|k|a did not manage the HTTP connections properly, leading to disruptions and wasting resources.

### 6.1.2 UPnP SSDP extensions

Our approach for implementing the discovery protocol was to reuse as much as possible existing UPnP SSDP libraries, modifying the code conveniently to support our SoaM SSDP extensions commented in subsection 5.7.1.

In order to achieve this goal we selected the CyberLink Development Package for UPnP Devices as base platform, adapting the source code to make it run with mi|k|a and carrying out the modifications to implement our extensions.

This task was specially troublesome, since features such as multicast communication or intensive networking were tested then for first time in the ConnectCore 7U, along with modifications to CyberLink original code.

By adopting existing UPnP libraries we also wanted to prove that existing UPnP devices could be augmented into smobjects, without losing their UPnP features.

### 6.1.3 Orchestrator

The orchestrator reused our SoaM UPnP SSDP extensions implemented so that it was able to discover and communicate with surrounding smobjects.

The orchestrator is the agent in charge of collecting the context information from smobjects, reasoning over it, receiving and resolving behavioural profiles against augmented context information, and orchestrating smobjects reactivity by sending and managing the constraints.

We implemented all the modules described in section 5.2. Since the orchestrator was intended to run on non resource-constrained devices, we avoided all the problems derived of using limited Java virtual machines and a full development platform was available.

We implemented the four main modules that compose the structure of the orchestrator (ProfilesManager, ConstraintsManager, SmobjectsManager and the Orchestration Module) in such a way that they could be run in two different modes: as different concurrent subprocesses or sequentially within the same subprocess.

As concurrent subprocesses the overall performance was better since, for example, the SmobjectsManager could update the list of smobjects, while the Constraints Manager renewed constraints on existing ones. Our design dealt with possible concurrency problems derived from simultaneous access from different execution threads.

The orchestrator must also implement reasoning mechanisms. We created a pluggable architecture for reasoners, very similar to the designs

we used for the discovery protocols, so that any kind of reasoner could be used without affecting other modules.

We used Jena 2 Semantic Web Framework for developing all the Semantic Web related activities in the orchestrator, such as serialising and deserialising RDF/XML, performing declarative logics reasoning or even rule reasoning, since Jena provides several built-in reasoners.

In order to resolve the profiles, we came up with a mechanism that could take advantage of the Jena generic rule reasoner:

1. The profile was transformed into a rule, using the Jena rule syntax and taking profile variables into account.
2. The reasoner was provided with the profile (in the form of rule) and the augmented context information (after applying description logics and domain rules).
3. The reasoner generated the new context information, even more augmented.
4. The orchestrator discarded those statements that were part of the previous context information, thus obtaining just the new ones (deductions).
5. The statements were transformed into constraints.

Once the constraints were obtained, they were passed along to the ConstraintsManager in charge of injecting, renewing and removing constraints in smobjects as needed.

The orchestrator reused the HTTP server and client developed for the smobject. The client was used for communicating with the smobjects, while the server featured a unique handler for dealing with profile injections from BPinjectors.

### **6.1.4 BPinjector**

For the BPinjector we reused the SoaM UPnP SSDP extensions developed for the smobject base components, extending them with the ability to search for orchestrators. We also provided the additional function of injecting the profiles into the orchestrator or the smobjects.

The ProfilesManager module of the BPinjector had to be designed from scratch since it has nothing to do with the ProfilesManager at the smobject. The latter is in charge of checking profile expiration and removing the

profile if required. However, the module at the BPinjector performs the inverse task: checking possible profile expiration to dispatch renewal messages.

The BPinjector, as a standalone surrounding elements influencing component, is intended to be embedded into wearables and portable devices such as cellular phones or PDAs.

### 6.1.5 mRDP client and server

The mRDP client and server development was conceived as a different project from the beginning, since it was intended to be used beyond the scope of SoaM. We deem mRDP suitable for a broad range of applications involving identifying and searching semantic information throughout the network<sup>1</sup>.

However, since the first tests with mRDP were to be carried out within SoaM, we tried to design the system in such a way that it would be simple enough to be hosted in limited platforms as well as easily extensible to take advantage of advanced computing resources in more powerful platforms, particularly making use of Jena 2 and its SPARQL query engine in personal computers.

Therefore, the architecture of mRDP involves not only an mRDP client, a server and an HTTP server for callbacks (the same HTTP server used for smobjects, featuring an mRDP handler thus reusing components and limiting platform size), but also several additional elements that provide the basis for extensibility.

In fact, we created two abstractions called *query model* and *location model*. The query model abstraction allows the integration of different query models, depending on the platform characteristics. We developed two query models: the Plant query model and the SPARQL query model. Query models are attached to the mRDP server during its initialisation, so that the query is dispatched to the appropriate query model upon reception.

Obviously, in the smobject implementation of mRDP only the Plant query model is provided, while the more powerful and resource-demanding SPARQL query model is used in the orchestrator.

The Plant query model implements the Plant Query Resolution Algorithm described in subsection 3.3.1. Since program size is a critical

---

<sup>1</sup>We created project pages for mRDP in SourceForge <http://sourceforge.net/projects/mrdp> and <http://mrdp.awareit.com>.

factor in smobjects, we were able to code the Plant Query Resolution Algorithm in less than 10 Kb of Java bytecode including helper classes.

The other abstraction is the location model, which is a generator of URLs where information about resources can be downloaded. A location model takes a URI resource as input and produces a URL where information about the resource can be downloaded. We designed three types of location models:

- `SimpleLocationModel`: it returns URLs associated with resource URIs previously in an ad hoc way.
- `HttpPrefixUriLocationModel`: it creates URLs by adding the resource's URI suffix at the end of a URL prefix. We created an HTTP server handler called `HttpPlantHandler` for dealing with requests about this kind of URL at the HTTP server.
- `SparqlLocationModel`: it generates URLs complying with the SPARQL protocol specification [Wor06a]. We also implemented an HTTP handler called `HttpSparqlHandler` for dealing with requests about this kind of URLs at the HTTP server.

These abstractions facilitate the creation of query and location models adapted to platform resources. As mentioned, we created models appropriate for smobjects and complete PC platforms.

In order to test semantic searches and its corresponding replies, we developed a special client called “mRDP Browser”, able to send any kind of Plant query through the network, receive the replies from existing information sources and retrieve resource descriptions from the attached locations.

For example, during an early test we started five smobjects in the network and used the mRDP Browser to discover them by multicasting the Plant query:

```
?resource <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.awareit.com/soam/2005/12/soamonto#Smobject>.
```

This one is the very same Plant query a BPinjector would issue to discover surrounding smobjects.

After discovery the mRDP Browser is able to browse through the RDF descriptions of the smobjects. Figure 6.3 depicts a screenshot of this process.

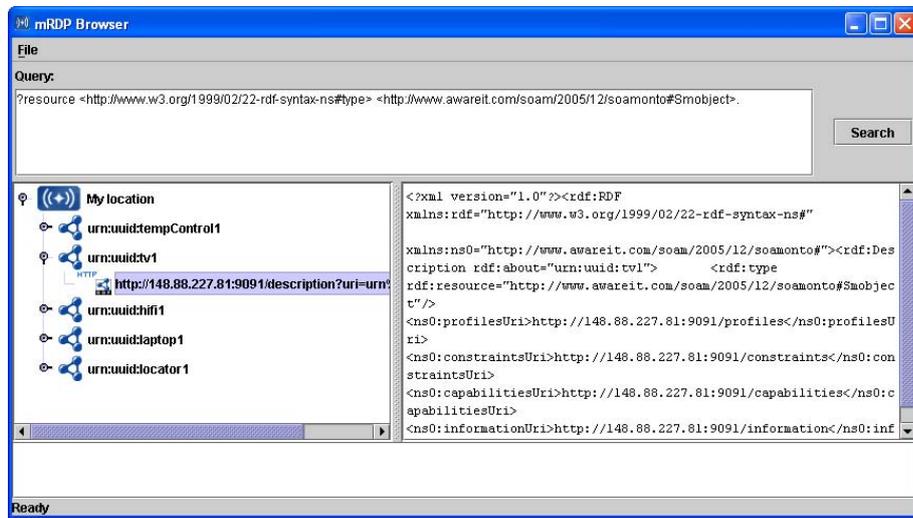


Figure 6.3: The mRDP Browser browsing through located smobjects in the network.

### 6.1.6 Smobject awareness components

All the smobject awareness components were implemented following the architecture described in subsection 5.1.5.

Although some of these components perform similarly to those present in the orchestrator, all the algorithms, specially those related to the processing of semantic information, required recoding, since Jena 2 cannot be run on limited platforms. Jena 2 libraries exceed 10 MB of disk space, while the ConnectCore 7U features only 8 MB Flash and the whole mi|k|a platform size is 1.37 MB.

Therefore, we developed a small Semantic Web library for the smobject. This library took advantage of the MinML parser used for the base components, combined with an RDF parser developed by Megginson Technologies in 2001 whose size is just 13 Kb. Unfortunately, the project was discontinued and the parser did not work properly, e.g. it did not take into account XML namespaces, so we had to correct and adapt it to finally make it work.

Although the RDF parser provided a means for deserialising RDF/XML, it did not provide any mechanism for serialisation, so we had to develop extra functions for serialising RDF triples into RDF/XML, taking into account XML namespaces as well as adding RDF datatypes parsing capabilities to the RDF parser for successfully interpreting literal types.

A small set of classes was also developed with the purpose of representing concepts such as RDF triples or RDF stores to provide some means of managing RDF information.

As already mentioned, reasoning processes were carried out by two small engines: `MiniOwlEngine` and `MiniRuleEngine`. In fact, the `MiniOwlEngine` transforms ontological relationships among semantic concepts into programmatic rules that can be processed by the rule engine implemented in the resolution algorithm. The `MiniRuleEngine` transforms rules expressed using a particular syntax we created for representing domain rules into programmatic rule structures for the resolution algorithm.

While profile resolution at the orchestrator was carried out by transforming the profile into a rule and applying the Jena 2 generic rule reasoner, the process at the smobject involved again the application of the resolution algorithm we already designed and used as a rule engine.

Combining the XML parser, the RDF parser, the RDF serialiser, the classes for managing RDF triples, the implementation of the reasoners and resolution algorithm, we managed to create a small semantic web framework, a kind of “Jena Lite”, now suitable for the smobject ConnectCore 7U platform.

The whole smobject platform, including the base and awareness components, was finally implemented in 300 Kb Java bytecode (165 Kb JAR size), distributed as specified in Table 6.1 (not including specific perceptors or effectors).

<i>Component name</i>	<i>Size (bytecode)</i>
HTTP Server and client	35 Kb
XML API	87 Kb
Base components	46 Kb
mRDP libraries	32 Kb
RDF parser	24 Kb
RDF API	19 Kb
Awareness components	57 Kb

Table 6.1: Smobject components size.

### 6.1.7 Prototyping issues

The main challenges and problems during the prototyping process were essentially derived from the platform features and limitations. It took a remarkable amount of time to solve some of these problems, generally motivated by the immaturity of the platform that resulted in a lack

of existing documentation and experiences by other developers. mi|k|a developed unexpected behaviours that were not found in the same Java code running in a desktop computer as mentioned below.

Based on our contact with the platform providers and existing feedback, we believe we have developed the most complex application ever run on Java for ConnectCore 7U, at least publicly disclosed, so far.

The low performance obtained using the awareness components during the initial tests encouraged us to optimise a lot of activities and to use basic Java libraries. In fact, the smobject components can be run in any Java 1.2 platform, which dates back to 1998.

Among the main problems we found developing the prototype were:

- The lack of an automatic garbage collector in mi|k|a forces to carefully establish “checkpoints” where garbage collection must be performed, otherwise the system never releases memory and eventually crashes.
- Network libraries in mi|k|a have a number of differences compared to the standard Java network library, so we had to adapt CyberLink UPnP to make it work in the ConnectCore 7U. Moreover, the structure of the CyberLink API made somehow difficult to incorporate our SoAM SSDP extensions. Several modifications at different classes had to be carefully done to add a single new capability.
- Making an XML parser work with the RDF parser, while having a small footprint, was even more difficult, since the RDF parser required SAX2 parsers.
- We also selected the parser in terms of performance (see subsection 6.2.1) by running several tests on the ConnectCore 7U platform. The one that ranked best was MinML, which is a SAX1 parser, but finally we could make it work by providing a SAX2 adapter for it.
- The RDF parser did not support RDF datatypes, local XML namespaces or the XML base namespace. We had to develop all this features.
- Although mi|k|a provided an implementation of the standard Java HTTP client class, we found unexpected behaviours when reading network data, mainly connection break-ups and data losses. Since, network traffic analysis reported that the data were arriving properly, we suspected of the HTTP client class and developed another from scratch, which worked perfectly.

Despite having developed two alternative communication stacks at the beginning, one based on SoAM SSDP extensions and other based in mRDP,

after developing the awareness components and make them work with mRDP, we discontinued the use of SSDP extensions.

The advantage of using UPnP was to readapt existing UPnP devices to make it work with SoaM extending their capabilities. Once this goal was achieved, and being UPnP further developments halted by the UPnP Forum, we focused our efforts in mRDP semantic discovery which is more challenging and promising.

### 6.1.8 Second generation prototypes

Shortly after the first generation of prototypes were deployed and tested in ConnectCore 7U-based platforms, we achieved to port the smobject middleware to more powerful and even smaller embedded alternatives: ConnectCore 9U and Gumstix Connex 400xm.

#### ConnectCore 9U

The ConnectCore 9U is pin-to-pin compatible with ConnectCore 7U, but hosting a much more powerful ARM9-based processor at 180 MHz, more than three times faster than the CC7U, 16 Mb RAM and 16 Mb flash ROM. The CC9U operating system consists on a Linux-based kernel, while the CC7U had a  $\mu$ Clinux kernel.

Regarding to Java platforms for CC9U we ported the open-source JVM-like Kaffe and also tested the mi|k|a version. The latter was remarkably faster and it was the one we used for performance evaluation. Moreover, it did not feature some of the limitations of the CC7U version (e.g. garbage collection).

#### Gumstix Connex 400xm

However, the ultimate selected platform for deploying the prototypes in real world-like scenarios was the Gumstix Connex 400xm. This platform features an impressive Intel XScale PXA255 at 400 MHz, 16 MB flash ROM and 64 MB RAM in a  $8.0 \text{ cm} \times 2.0 \text{ cm} \times 0.63 \text{ cm}$  ( $10.08 \text{ cm}^3$ ) form factor, even smaller than Digi's ConnectCore alternatives.

Table 6.2 compares size, computing power and the resulting “computing power-per-cm<sup>3</sup>” ratio for several platforms<sup>2</sup>, being the Gumstix clearly the best choice for embedding more intelligence in small sized artifacts.

Platform	Computing power	Size	Power-per-cm <sup>3</sup>
Laptop computer	2.33 GHz	2596.21 cm <sup>3</sup>	0.89 MHz/cm <sup>3</sup>
PDA	416 MHz	150.80 cm <sup>3</sup>	2.76 MHz/cm <sup>3</sup>
Digi ConnectCore 7U	55 MHz	12.08 cm <sup>3</sup>	4.55 MHz/cm <sup>3</sup>
Digi ConnectCore 9U	180 MHz	12.08 cm <sup>3</sup>	14.90 MHz/cm <sup>3</sup>
Gumstix Connex 400xm	400 MHz	10.08 cm <sup>3</sup>	39.68 MHz/cm <sup>3</sup>

Table 6.2: Relation among computing power and platform size.

The Gumstix components family allows easy and rapid prototyping and a wide range of daughterboards are available, including Wi-Fi, GPS or audio support (see Figure 6.4).

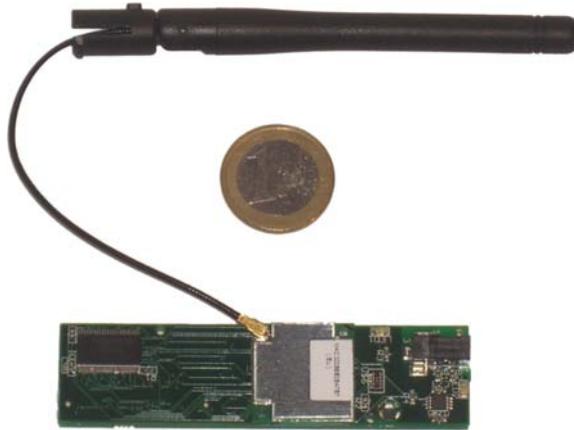


Figure 6.4: Image of the Gumstix 400xm with Wi-Fi card and antenna that hosted the second generation of smobjects.

<sup>2</sup>Although the embedded platforms require additional components to be fully operational for our prototyping goals, e.g. a wireless adapter and a battery, they still remain as the best choice.

Moreover, a Java virtual machine called JamVM is provided, so the smobject middleware was hosted into Gumstix Connex 400xm naturally.

### 6.1.9 Integration with wireless sensor networks

Although we developed a number of domain-specific perceptors and effectors, we would like to note the implementation of a Wireless Sensor Network perceptor that was used in several scenarios.

This perceptor was able to connect to a Crossbow Motes Wireless Sensor Network through a gateway. The network was formed by nodes capturing temperature, light, sound, and other parameters. The perceptor polled the sensor network periodically, retrieving the values provided by individual sensor nodes distributed along a room or building, and applying concrete OWL vocabularies to semanticise the information.

Therefore, a smobject containing this perceptor was able to share the information with other fellow smobjects in the environment, which were able to react accordingly to changes in temperature, light and any other semantically annotated parameter provided by the perceptor.

This example proves the integration capabilities of smobjects with other state-of-the-art Ubiquitous Computing-related technologies such as wireless sensor networks, and how to carry out semantic annotation and sharing of low-level data provided by other architectures.

## 6.2 Evaluation

The following phase in our research was to deploy and evaluate a number of scenarios to face the real challenges of implementing intelligence through the SoaM model in everyday devices and environments.

We decomposed evaluation into two different but complementary approaches:

- Performance tests: these tests focused on obtaining performance measures about different technical aspects of the SoaM architecture. We tested different configurations of the entities in a basic scenario.
- Scenarios tests: these tests focused on deploying real reasoning-demanding scenarios, similar to those depicted at the initial phases of our research, populated by smobject-powered semantic devices. The final goal was to demonstrate how SoaM is able to provide the technology to create Ambient Intelligence environments.

### 6.2.1 Performance tests

We carried out the following activities:

- XML and RDF parsing: since XML/RDF parsing is one of the most costly activities in terms of time and required memory, we obtained several measures that enabled us to select the XML parser that best performed at this stage.
- Context-awareness and reactivity: these tests were aimed at measuring the performance of smobjects and orchestrators when polling surrounding smobjects for context-information, and reacting appropriately in order to carry out the required behaviour. These tests measured the following aspects:
  - Context information retrieval: the amount of time required for an entity (smobject or orchestrator) to retrieve context information from surrounding smobjects as needed in order to evaluate the behavioural profiles.
  - Reasoning: the amount of time required for an entity to augment the context information via ontological and domain rules reasoning, in order to create a comprehensive knowledge base about the environment.
  - Behavioural profile resolution: the amount of time required for an entity to resolve existing behavioural profiles against the augmented context information in order to generate the constraints.
  - Constraints operation: the amount of time required for a smobject to process received constraints and to be ready for operating them. The actual execution time of a constraint was not included here as it is very domain dependent (e.g. changing the current room temperature can take several minutes).

The addition of the above measures provides an estimation of the total reactivity time: the amount of time required from the moment some piece of information changes in the environment until the associated reactivity is initiated, including all the reasoning processes.

- Impact of context information awareness in network traffic load: we also evaluated the dependence between context information update periods and its impact on network traffic. The obvious conclusion is that the shorter polling period, the higher network traffic load, but we obtained a recommendation threshold for selecting the most suitable polling period depending on the number of deployed smobjects and the network bandwidth.

### XML and RDF parsing

We carried out these tests to determine which of three possible XML parsers that could be run at mi|k|a / ConnectCore 7U exhibited a better performance along with a balanced size, when parsing XML and RDF.

In order to execute the XML tests we designed a collection of four behavioural profiles in a single XML document. The document comprised a total amount of 44 XML elements and 86 XML attributes.

We tested three different parsers: MinML (10 Kb), Ælfred (23 Kb) and Saxon-Ælfred (32 Kb). Figure 6.5 illustrates graphically the amount of time required for the parsers to parse the document ten consecutive times during the same run. It is noteworthy how the first and second tests take remarkably more time than the others, something usual in Java, since classes are dynamically loaded as they are required. After the third test the average time shows more uniformity.

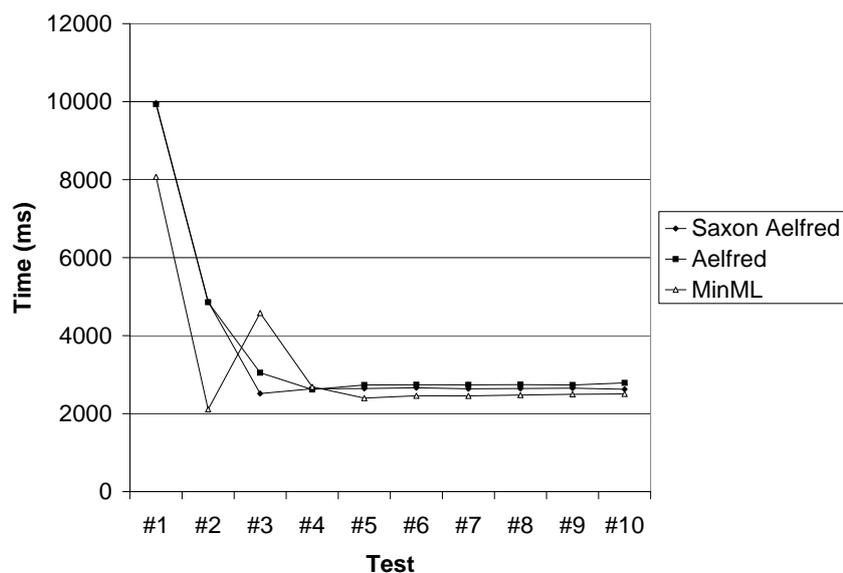


Figure 6.5: Performance of the different XML parsers parsing an XML document in the ConnectCore 7U platform.

The second test linked the RDF parser with the three different XML parsers in order to obtain the best combination in terms of performance. We used an RDF/XML document formed by 29 RDF triples of information which summed up 44 XML elements and 57 XML attributes. Figure 6.6 depicts the results obtained in ten consecutive parsings with each combination. Again, first parsings take more time as already explained.

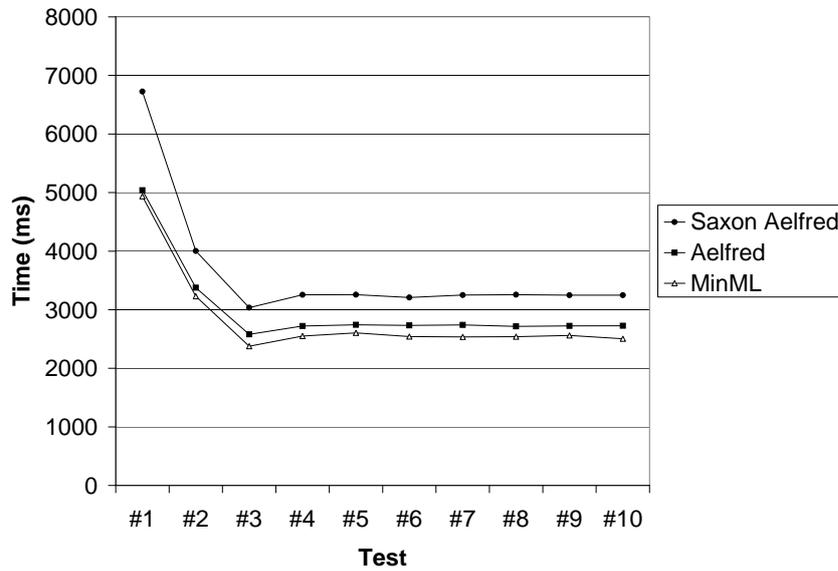


Figure 6.6: Performance of the different XML parsers parsing an RDF/XML document in the ConnectCore 7U platform.

In both cases, the MinML parser performed slightly better than the others. Moreover, MinML is remarkably lighter than other alternatives, still supporting our whole set of requirements. This advantage is depicted graphically in Figure 6.7, representing the number of XML documents, like the one provided, that can be parsed per minute and kilobyte of the parser’s file size, where MinML clearly outperforms the others.

**Context-awareness and reactivity**

Several performance aspects of the prototype implementation were evaluated both for the smobjects-only and for the orchestration-powered topologies.

A testing scenario resembling a home environment was deployed including smobjects for representing a simulated location system, a TV set, a temperature control system, an “alerter” in a laptop and a Hi-Fi system. A number of behavioral profiles were designed to obtain automatic environmental reactivity depending on certain situations, for example “turn off the TV if I leave the TV room” or “alert me if I am leaving home and the weather forecast says it is going to rain” (this information was obtained by a perceptor in the laptop smobject that retrieved the forecast

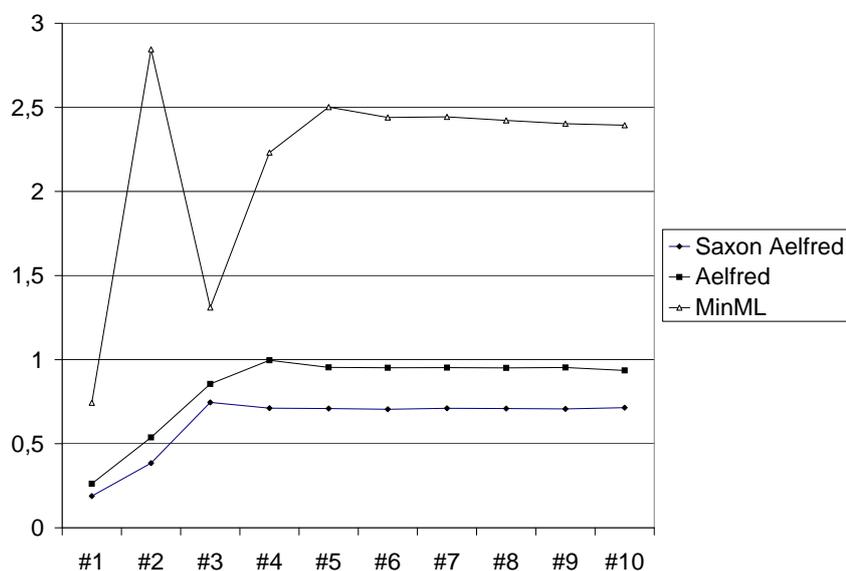


Figure 6.7: Performance of the different XML parsers in relation to its size.

from *Weather.com*, annotated it semantically and made it available to other surrounding smobjects).

Several ontologies were reused, mainly from SOUPA [CPFJ04], and others were designed from scratch to embody the knowledge of some domains such as the TV domain, the weather domain and the location domain, the latter including concepts such as room, building or town.

A particularly interesting issue about the ontology we created for the location domain is the application of the transitive property *islocatedIn* and its inverse *contains* as well as the symmetric property *nearby* to infer the location relationships among the deployed elements. In fact, the majority of behavioral profiles we designed demanded certain reactivity of smobjects depending on their location; for example, to turn on an Hi-Fi system in a certain room near the user's location.

These situations promoted the application of ontologies and description logics reasoning in order to successfully obtain the required reactivity. In the concrete scenarios described above an average of 30-50 triples were selectively collected from surrounding smobjects, and augmented around 50% once OWL reasoning was applied (mainly due to inferences obtained through the location ontology). This "augmented" context information was then matched against the previously injected behavioral profiles in order to determine the constraints that finally represented the desired environmental reactions.

The results of the tests using the completely decentralized smobjects-only architecture are depicted in Figure 6.8, illustrating the amounts of time required by the smobject to perform every activity.

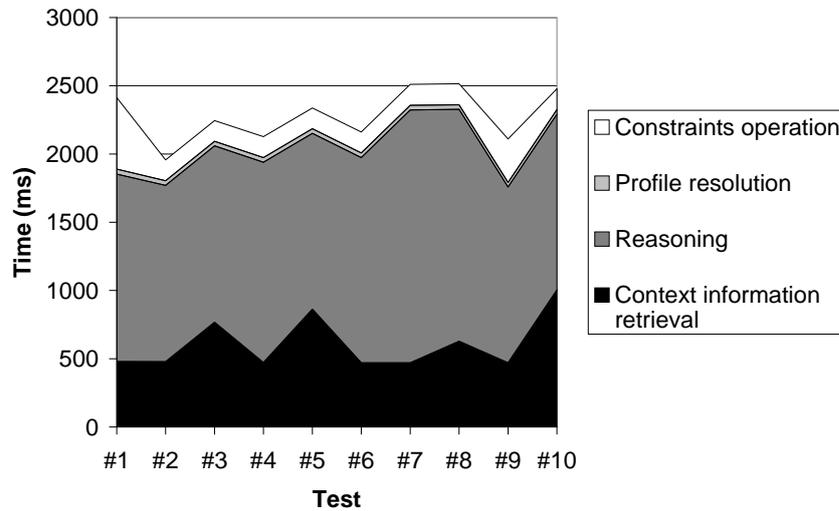


Figure 6.8: Performance measures of the smobjects-powered CC9U topology.

As already mentioned two activities take most of the operating time: context information retrieval, which includes HTTP communication to poll surrounding smobjects and RDF parsing of the received data; and description logics reasoning.

Our tests also pointed out that the performance of a smobject may be severely affected by external or platform issues such as the garbage collection process, or being intensively polled by other entities during a short period of time. This kind of situations arises in limited devices, very sensitive to additional work load, and must be taken into account when a constant reactivity response time is required.

While the smobject implementation in the ConnectCore 7U (CC7U) platform exhibited the above mentioned limitations, the ConnectCore 9U (CC9U) prototype performed exceptionally well. Therefore, we deployed the smobject-only topology with ConnectCore 9U devices hosting the SmobjectBase and SmobjectAware components, while the orchestrator-powered topology was deployed in an scenario populated by more limited ConnectCore 7U devices hosting SmobjectBase components coordinated by the central orchestrator.

In the smobjects-only topology, the average time required by a CC9U smobject to complete a full cycle including retrieving context information from other smobjects, generating new triples through reasoning, resolving the behavioral profiles against the context information and executing these constraints, was around 2.28 seconds for the above described scenario.

The orchestrator-powered topology performed similarly, around 2.59 seconds, being remarkably slowed by the limited CC7U smobjects, where XML processing and HTTP communication at the context information retrieval phase are very costly in terms of time (see Figure 6.9).

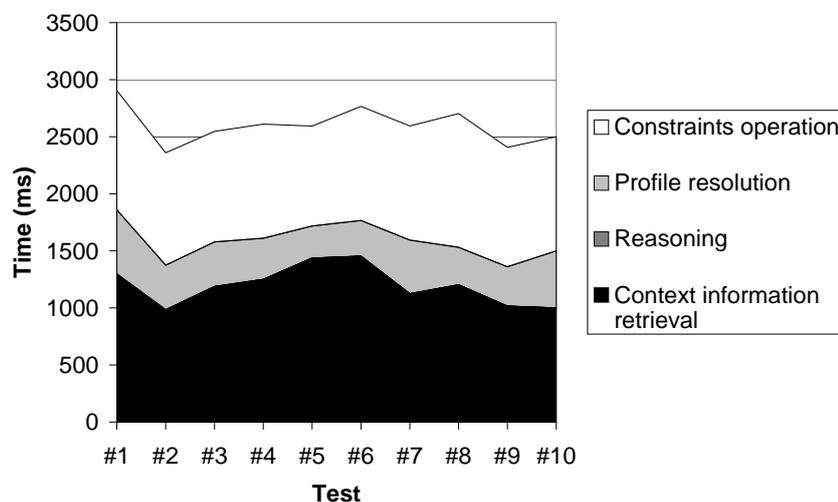


Figure 6.9: Performance measures of the orchestrator-powered CC7U topology.

The orchestrator ran in a Pentium-M 1.86 GHz with 1 GB RAM and enjoyed all the advantages of full computing resources. It is especially remarkable how the use of the Jena API for OWL reasoning reduced the reasoning time to an almost imperceptible amount, due to the extensive optimizations for representing the RDF graph in memory used by Jena. However, profile resolution performed relatively bad compared to the other activities. The approach applied here was to convert the profile into a rule to feed the general rule engine provided by Jena, but this mechanism can probably be optimized in the future.

Since constraints operation is an activity carried out in the CC7U smobject, thus being slowed again by this resource-constrained platform, the performance of this phase is not remarkably improved.

Figure 6.10 compares the performance of both topologies for the described scenario in absolute terms, while Figure 6.11 compares them in terms of relative time required for each activity to be carried out.

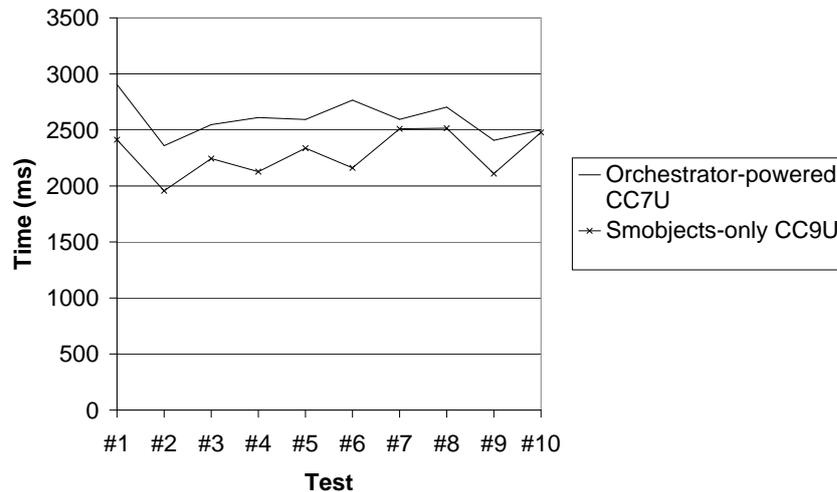


Figure 6.10: Comparison of performance measures for the smobjects-only CC9U and orchestrator-powered CC7U topologies.

### Impact of context information awareness in network traffic load

As we mentioned in section 5.4, the smobjects-only scenario may generate a lot of network traffic. In the worst case where all the smobjects require context information from every other, the overall amount of connections to collect the data is  $n \times (n - 1)$ , being  $n$  the number of smobjects.

Short polling periods promote promptly reactivity but also require a higher network bandwidth as well as better performance in smobjects to process a higher amount of inbound and outbound requests. Therefore, the factors to consider when deciding the polling period are:

- The number of deployed smobjects
- The upper limit of traffic (messages/second) the whole network or individual smobjects are able to process

Figure 6.12 and Figure 6.13 illustrate a 3-D and a two dimensional projection of the relationships among polling period, number of smobjects

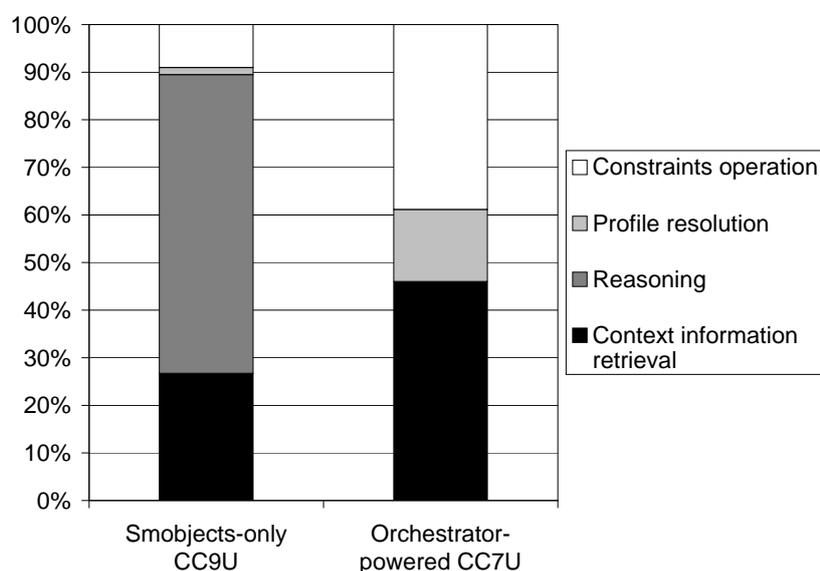


Figure 6.11: Comparison of relative effort for the activities in the smobjects-only CC9U and orchestrator-powered CC7U topologies.

and network capacity in messages/second for the “worst case” smobjects-only topology.

In this example we considered 20 (a very conservative value) as the maximum number of messages/second a network can support. This threshold can be noticed in yellow colour in the 3-D view and even more clearly in the two dimensional projection.

As the number of smobjects increase, the polling period must also increase (reducing the amount of requests) in order to maintain the stability of the messages/second ratio. For a large number of smobjects the polling period and, thus, device reactivity is severely affected. For example, in an scenario populated with 30 smobjects the polling period must not be shorter than 43.5 seconds.

Again, we must remark that these figures illustrate the worst-case scenario for a smobjects-only topology where all smobjects poll each other. In a more natural situation, where every smobject polls an average of 20% of surrounding smobjects, the above scenario with 30 smobjects can support an average polling period of 9 seconds.

Figure 6.14 and Figure 6.15 illustrate analogous concepts from a single smobject’s point of view.

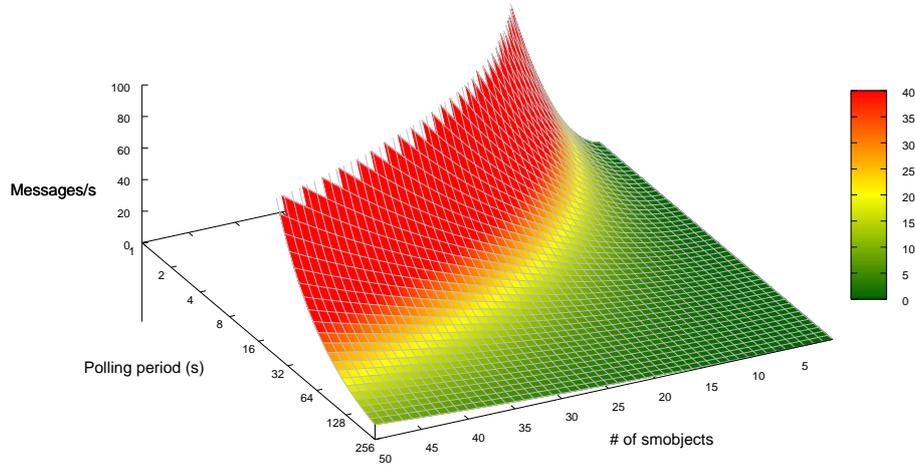


Figure 6.12: 3-D view of relationships among relevant factors for smobjects-only networking.

In this case we marked in yellow the maximum number of messages/second an individual smobject is able to process. This value depends on the concrete platform performance; for the simulation we selected a conservative value of 6 messages/second.

As the number of polled smobjects increase, the polling period must increase too. For example, 3.33 seconds for polling 20 smobjects.

Obviously, the conclusion is that the polling period must be fine-tuned in order to optimise the reactivity periods to the particular characteristics of the deployed scenario in terms of network bandwidth and smobjects platform performance.

### 6.2.2 Scenarios tests

In order to validate the hypothesis, we selected the first three scenarios described in section 1.5 and faced their design and deployment. These environments were populated by smobject-powered semantic devices, able to perform reasoning processes over exchanged context information in order to carry out different flavours of intelligent behaviour.

For each of these scenarios we accomplished the following steps:

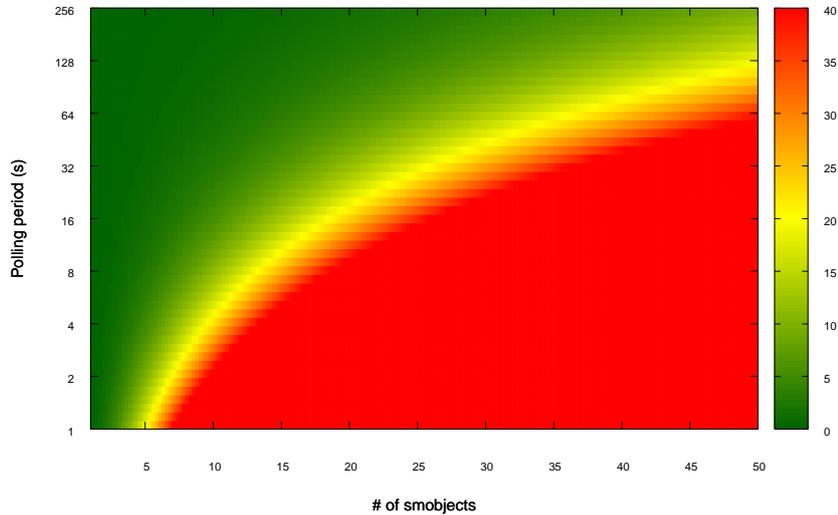


Figure 6.13: Two dimensional projection of the relationships among relevant factors for smobjects-only networking.

- Goals statement: we identified the major goals of the scenario in terms of SoaM features evaluation.
- Scenario design: we designed the scenario domain aiming to represent real-world situations. This task generally involved semantic devices identification with perceptors and effectors, how the devices connected with other hardware or software systems, ontologies reuse or design, and rules and smart behaviour design.
- Implementation: we implemented the different elements for the scenario. This task generally involved implementing particular perceptors and effectors, connection with external systems or hardware, ontologies, vocabularies, domain rules and behavioural profiles.
- Deployment: we deployed and configured the required elements in the environment.
- Evaluation: finally, we monitorised how the whole system worked and how individual semantic devices exhibited intelligent behaviour reacting to different situations.

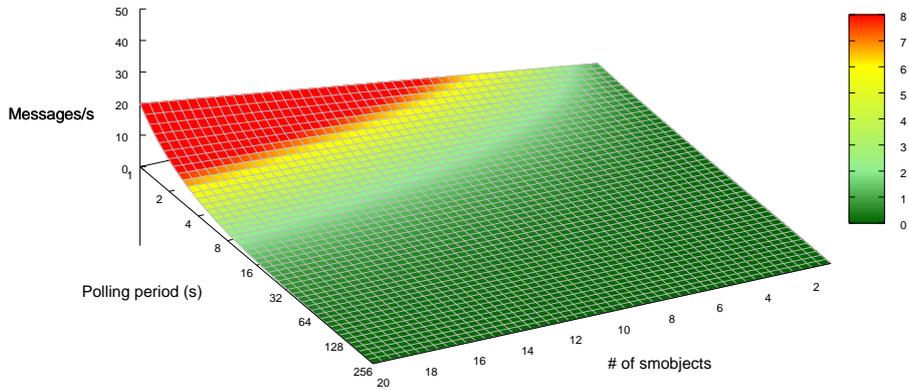


Figure 6.14: 3-D view of relationships among relevant factors for a single smobject.

We applied this process through three paradigmatic scenarios depicting possible applications of SoaM:

- SmartPlants: autonomous objects that interact with their environment.
- Aware-Umbrella: a reactive device integrating local and global communication.
- WorkSafe: a protective working agent enforcing user care in dangerous environments.

For all these scenarios we deployed the second generation smobject prototype shown in Figure 6.16. This prototype used a Gumstix embedded platform with the smobject middleware, Wi-Fi capabilities, audio output, a small speaker and a battery. The audio output daughterboard and the speaker were removed in the situations where they were not required, reducing the overall size.

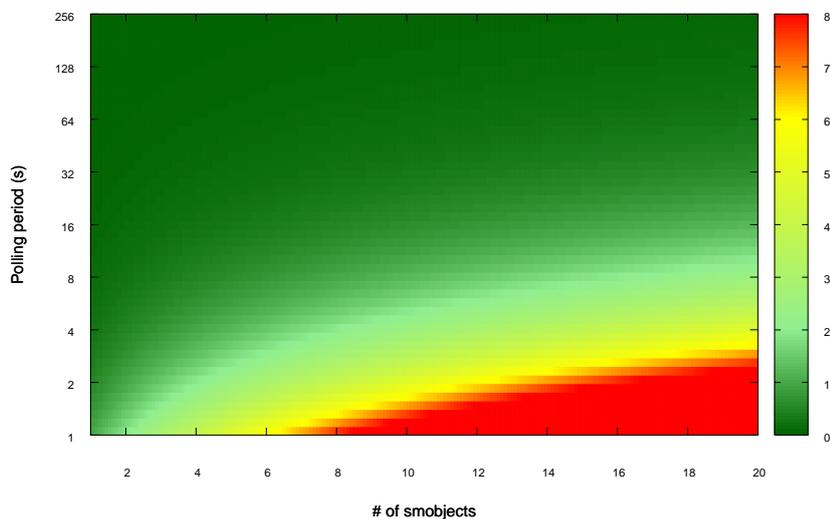


Figure 6.15: Two dimensional projection of the relationships among relevant factors for a single smobject.



Figure 6.16: The complete smobject prototype with Wi-Fi, audio and battery.

### **SmartPlants: autonomous objects that interact with their environment**

One of the scenarios we envisioned at the beginning of the research was to create an artifact that could be attached to real objects, augmenting their

perceptions and providing them with intelligent capabilities. An additional challenge was to attach this kind of artifact to living entities, such as plants, in a way that could result in intelligent behaviour carried out by the entities from the user's point of view.

Creating this kind of “smart plants” raised several new important implications such as:

- They could become first-class citizens in the environment, rather than passive elements. They could inject their preferences into their location to influence temperature, humidity of lighting settings.
- They could be perceived as autonomic systems [KC03] in a twofold view: as normal living beings they try to survive and adapt to environmental conditions; but also, as augmented intelligent entities they can interact and communicate with surrounding objects to create a more suitable and healthy environment.

We wanted these smart plants to be aware of relevant environmental conditions, and we provided them with the ability to disseminate their preferences to their location, as well as actively communicate with the user, asking her or him for help.

For example, when a plant is placed in an unknown room, it can discover existing temperature and lighting control systems, and inject its behavioural profiles to influence temperature and lighting according to its requirements. If the plant perceives that other nearby place is better conditioned (e.g. more illuminated) it can ask the user to be moved and placed there.

In order to evaluate the scenario, we developed the following components:

- A Crossbow Motes wireless sensor network with nodes featuring temperature and light sensors throughout a room. We also attached one of these motes to the plant to make it aware of its own location conditions.
- Ontologies and vocabularies for the location, temperature, lighting and sensor domains.
- Domain rules extrapolating the temperature obtained by a sensor node to the room area where it is located (see Listing 6.4).
- A `PerceptorMotes` connected to the wireless sensor network that captured the information provided by the sensor nodes and semantically annotated it using the mentioned ontologies.

- An augmented plant pot with the second generation smobject prototype in a plastic case (see Figure 6.17).
- An `EffectorTTSAIerter` in the smobject using a text-to-speech embedded engine able to synthesise human-like voice. Using this effector in the pot's smobject, the plant could alert the user using its voice, thus "talking" to the user.
- Appropriate behavioural profiles for the plant to influence the surrounding environment as well as to drive its own behaviour which basically consisted in alerting the user, using the synthesised voice, about the plant's needs.



Figure 6.17: The smobject prototype in the plant protected with a plastic case.

As already mentioned, the smobject prototype in the plant was placed in a plastic case for protection. Several sensor nodes were distributed across the environment and one was placed in the plant to measure its ambient conditions.

The plant was able to inject behavioural profiles into surrounding temperature and light control systems to adjust them as required in order to fulfill the plant's requirements. This system allowed the user to distribute a number of such plants in any environment without worrying about plants requirements, since they were able to autonomously establish their best

living conditions releasing the user from this burden. This was the way a smart plant could actively influence the surrounding environment in which it was located.

But the plant was also reactive via a number of statically configured behavioural profiles, in such a way that it was continuously aware of the temperature and light measures about different nearby locations provided by the wireless sensor network, and asked the user to be moved to the most suitable place using a synthesised voice. This was the way a smart plant was aware of the most suitable location in terms of environmental conditions and reacted asking the user to be reallocated.

**Example** Listing 6.1 illustrates a behavioural profile disseminated by the smart plant to request some concrete ambient conditions (20°-30°C temperature, and 50-90 % humidity) to smobjects in the environment.

Listing 6.1: Behavioural profile disseminated by the smart plant to adapt the environment.

```

1  <behavioralProfile id="urn:uuid:profbptoil" expires="PT1000M"
2      requester="urn:uuid:plant1">
3
4      <postcondition id="" predicate="http://www.awareit.com/onto
5          /2005/12/light#luminance" operator="http://www.awareit.com
6          /soam/2005/12/soamonto#GreaterThan" subject="http://www.
7          awareit.com/onto/2005/12/location#ThisLocation">
8          <objectLiteral>50</objectLiteral>
9      </postcondition>
10
11     <postcondition id="" predicate="http://www.awareit.com/onto
12         /2005/12/light#luminance" operator="http://www.awareit.com
13         /soam/2005/12/soamonto#LessThan" subject="http://www.
14         awareit.com/onto/2005/12/location#ThisLocation">
15         <objectLiteral>90</objectLiteral>
16     </postcondition>
17
18     <postcondition id="" predicate="http://www.awareit.com/onto
19         /2005/12/temperature#hasTemperature" operator="http://www.
20         awareit.com/soam/2005/12/soamonto#GreaterThan" subject="
21         http://www.awareit.com/onto/2005/12/location#ThisLocation
22         ">
23         <objectLiteral>20</objectLiteral>
24     </postcondition>
25
26     <postcondition id="" predicate="http://www.awareit.com/onto
27         /2005/12/temperature#hasTemperature" operator="http://www.
28         awareit.com/soam/2005/12/soamonto#LessThan" subject="http
29         ://www.awareit.com/onto/2005/12/location#ThisLocation">

```

```

16     <objectLiteral>30</objectLiteral>
17     </postcondition>
18
19 </behavioralProfile>

```

Listing 6.2 illustrates an example of statically configured behavioural profile for a smart plant. This profile provoked the plant to ask to be moved whenever the plant's location featured more than 30°C and there was a place with suitable temperature and lighting conditions. Similar profiles were configured for other ambient conditions.

Listing 6.2: Example of native behavioural profile for the smart plant.

```

1 <behavioralProfile id="urn:uuid:profal" expires="PT1000M"
  requester="urn:uuid:plant1">
2   <variable xml:id="otherLocation"/>
3   <variable xml:id="otherLocationLabel"/>
4
5   <precondition id="" predicate="http://www.awareit.com/onto
  /2005/12/temperature#hasTemperature" operator="http://www.
  awareit.com/soam/2005/12/soamonto#GreaterThan" subject="
  urn:uuid:plant1">
6     <objectLiteral>30</objectLiteral>
7   </precondition>
8
9   <precondition id="" predicate="http://www.awareit.com/onto
  /2005/12/light#luminance" operator="http://www.awareit.com
  /soam/2005/12/soamonto#GreaterThan" subject="#
  otherLocation">
10    <objectLiteral>50</objectLiteral>
11  </precondition>
12
13  <precondition id="" predicate="http://www.awareit.com/onto
  /2005/12/light#luminance" operator="http://www.awareit.com
  /soam/2005/12/soamonto#LessThan" subject="#otherLocation">
14    <objectLiteral>90</objectLiteral>
15  </precondition>
16
17  <precondition id="" predicate="http://www.awareit.com/onto
  /2005/12/temperature#hasTemperature" operator="http://www.
  awareit.com/soam/2005/12/soamonto#GreaterThan" subject="#
  otherLocation">
18    <objectLiteral>20</objectLiteral>
19  </precondition>
20
21  <precondition id="" predicate="http://www.awareit.com/onto
  /2005/12/temperature#hasTemperature" operator="http://www.
  awareit.com/soam/2005/12/soamonto#LessThan" subject="#
  otherLocation">
22    <objectLiteral>30</objectLiteral>

```

```

23     </precondition>
24
25     <precondition id="" predicate="http://www.w3.org/2000/01/rdf-
26         schema#label" subject="#otherLocation">
27         <objectVariable ref="otherLocationLabel"/>
28     </precondition>
29
30     <postcondition id="" predicate="http://www.awareit.com/onto
31         /2005/12/alerting#alertsAbout" subject="urn:uuid:plant1">
32         <objectResource resource="urn:uuid:prof0_alert"/>
33     </postcondition>
34
35     <postcondition id="" predicate="http://www.awareit.com/onto
36         /2005/12/alerting#level" subject="urn:uuid:prof0_alert">
37         <objectResource resource="http://www.awareit.com/onto
38             /2005/12/alerting#warning"/>
39     </postcondition>
40
41     <postcondition id="" predicate="http://www.awareit.com/onto
42         /2005/12/alerting#message" subject="urn:uuid:prof0_alert">
43         <objectLiteral>Please, take me to ?(otherLocationLabel).
44             The temperature is better for me there.</objectLiteral
45             >
46     </postcondition>
47
48     <postcondition id="" predicate="http://www.awareit.com/onto
49         /2005/12/alerting#title" subject="urn:uuid:prof0_alert">
50         <objectLiteral>Hey</objectLiteral>
51     </postcondition>
52
53     <postcondition id="" predicate="http://www.awareit.com/onto
54         /2005/12/alerting#frequency" subject="urn:uuid:prof0_alert
55         ">
56         <objectLiteral>30</objectLiteral>
57     </postcondition>
58
59 </behavioralProfile>

```

The plant was aware of existing temperature and lighting values in different parts of the scenario via the PerceptorMotes which provided real-time measures of deployed wireless temperature and light sensors. Information about sensors' location and labels was configured at this perceptor, basically mapping sensor IDs to their locations.

Listing 6.3 illustrates a concrete piece of context information at a particular moment of time. We can notice how the plant's temperature captured by the plant's sensor was out of the required bounds in the behavioural profile (30°C), while the temperature and lighting

conditions measured by the sensor “near the window” matched the plant’s requirements. This situation provoked the plant to ask the user to be placed “near the window”.

Listing 6.3: Excerpt of integrated context information obtained from several sources by the smart plant.

```
1 <?xml version="1.0"?>
2 <rdf:RDF
3   xmlns="http://www.awareit.com/example2#"
4   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5   xmlns:owl="http://www.w3.org/2002/07/owl#"
6   xmlns:loc="http://www.awareit.com/onto/2005/12/location#"
7   xmlns:light="http://www.awareit.com/onto/2005/12/light#"
8   xmlns:temp="http://www.awareit.com/onto/2005/12/temperature#"
9   xmlns:sensor="http://www.awareit.com/onto/2005/12/sensor#"
10  xml:base="http://www.awareit.com/example2">
11
12  <owl:Ontology rdf:about="">
13    <owl:imports rdf:resource="http://www.awareit.com/onto/2005/12/
14      location"/>
15  </owl:Ontology>
16
17  <rdf:Description rdf:about="urn:uuid:plant1">
18    <owl:sameAs rdf:resource="urn:uuid:mote5"/>
19  </rdf:Description>
20
21  <loc:Room rdf:about="urn:uuid:smartlab">
22    <rdfs:label>SmartLab</rdfs:label>
23  </loc:Room>
24
25  <loc:RoomArea rdf:about="urn:uuid:door_area">
26    <loc:isLocatedIn rdf:resource="urn:uuid:smartlab"/>
27    <loc:contains rdf:resource="urn:uuid:mote1"/>
28    <loc:nearby rdf:resource="urn:uuid>window_area"/>
29    <loc:nearby rdf:resource="urn:uuid:tv_area"/>
30    <loc:nearby rdf:resource="urn:uuid:meeting_area"/>
31    <rdfs:label>The door</rdfs:label>
32  </loc:RoomArea>
33
34  <loc:RoomArea rdf:about="urn:uuid>window_area">
35    <loc:isLocatedIn rdf:resource="urn:uuid:smartlab"/>
36    <loc:contains rdf:resource="urn:uuid:mote2"/>
37    <loc:nearby rdf:resource="urn:uuid:door_area"/>
38    <loc:nearby rdf:resource="urn:uuid:tv_area"/>
39    <loc:nearby rdf:resource="urn:uuid:meeting_area"/>
40    <rdfs:label>The window</rdfs:label>
41  </loc:RoomArea>
42
43  <loc:RoomArea rdf:about="urn:uuid:tv_area">
```

```
43 <loc:isLocatedIn rdf:resource="urn:uuid:smartlab"/>
44 <loc:contains rdf:resource="urn:uuid:mote3"/>
45 <loc:nearby rdf:resource="urn:uuid>window_area"/>
46 <loc:nearby rdf:resource="urn:uuid:door_area"/>
47 <loc:nearby rdf:resource="urn:uuid:meeting_area"/>
48 <rdfs:label>The TV</rdfs:label>
49 </loc:RoomArea>
50
51 <loc:RoomArea rdf:about="urn:uuid:meeting_area">
52 <loc:isLocatedIn rdf:resource="urn:uuid:smartlab"/>
53 <loc:contains rdf:resource="urn:uuid:mote4"/>
54 <loc:nearby rdf:resource="urn:uuid>window_area"/>
55 <loc:nearby rdf:resource="urn:uuid:door_area"/>
56 <loc:nearby rdf:resource="urn:uuid:tv_area"/>
57 <rdfs:label>The meeting area</rdfs:label>
58 </loc:RoomArea>
59
60 <sensor:Mote rdf:about="urn:uuid:mote1">
61 <temp:hasTemperature rdf:datatype="http://www.w3.org/2001/
62 XMLSchema#int">16</temp:hasTemperature>
63 <light:luminance rdf:datatype="http://www.w3.org/2001/XMLSchema#
64 int">80</light:luminance>
65 </sensor:Mote>
66
67 <sensor:Mote rdf:about="urn:uuid:mote2">
68 <temp:hasTemperature rdf:datatype="http://www.w3.org/2001/
69 XMLSchema#int">23</temp:hasTemperature>
70 <light:luminance rdf:datatype="http://www.w3.org/2001/XMLSchema#
71 int">60</light:luminance>
72 </sensor:Mote>
73
74 <sensor:Mote rdf:about="urn:uuid:mote3">
75 <temp:hasTemperature rdf:datatype="http://www.w3.org/2001/
76 XMLSchema#int">12</temp:hasTemperature>
77 <light:luminance rdf:datatype="http://www.w3.org/2001/XMLSchema#
78 int">90</light:luminance>
79 </sensor:Mote>
80
81 <sensor:Mote rdf:about="urn:uuid:mote4">
82 <temp:hasTemperature rdf:datatype="http://www.w3.org/2001/
83 XMLSchema#int">32</temp:hasTemperature>
84 <light:luminance rdf:datatype="http://www.w3.org/2001/XMLSchema#
85 int">40</light:luminance>
86 </sensor:Mote>
87
88 <sensor:Mote rdf:about="urn:uuid:mote5">
89 <temp:hasTemperature rdf:datatype="http://www.w3.org/2001/
90 XMLSchema#int">38</temp:hasTemperature>
```

```
82 <light:luminance rdf:datatype="http://www.w3.org/2001/XMLSchema#
    int">60</light:luminance>
83 </sensor:Mote>
84
85 </rdf:RDF>
```

Lines 16–18 were provided by the smart plant itself identifying its embedded wireless sensor ID (*mote5*); lines 20–58 were provided by the room’s smobject, informing about the different areas in the room, the location relationships among them, and the wireless sensors placed in each one; and lines 60–83 are provided by the wireless sensor network’s smobject.

Listing 6.4 illustrates the two rules loaded by the smart plant to infer that a temperature or luminance obtained by a sensor node located at a concrete room area can be assumed as the temperature or luminance value for that location.

**Listing 6.4:** Domain rules associating sensor nodes measures to their location.

```
1 ?place <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
    awareit.com/onto/2005/12/location#RoomArea> .
2 ?mote <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
    awareit.com/onto/2005/12/sensor#Mote> .
3 ?mote <http://www.awareit.com/onto/2005/12/location#isLocatedIn> ?
    place .
4 ?mote <http://www.awareit.com/onto/2005/12/temperature#hasTemperature>
    ?temp .
5 ->
6 ?place <http://www.awareit.com/onto/2005/12/temperature#hasTemperature
    > ?temp .
7
8 ****
9
10 ?place <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
    awareit.com/onto/2005/12/location#RoomArea> .
11 ?mote <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
    awareit.com/onto/2005/12/sensor#Mote> .
12 ?mote <http://www.awareit.com/onto/2005/12/location#isLocatedIn> ?
    place .
13 ?mote <http://www.awareit.com/onto/2005/12/light#luminance> ?lux .
14 ->
15 ?place <http://www.awareit.com/onto/2005/12/light#luminance> ?lux .
```

Lines 1–4 and 10–13 represent the premises, while lines 6 and 15 provide the conclusions.

As previously mentioned, the combined deductions obtained from the ontologies, especially the location ontology, and the domain rules augment the available context information that enables the smart plant to correctly deduce whether a more appropriate location in the room exists and ask for a reallocation.

### **Aware-Umbrella: a reactive device integrating local and global communication**

An additional challenge for SoaM identified earlier was the ability to seamlessly integrate local and global information sources in order to augment local intelligence and knowledge by injecting externally obtained context information. The most probable, but not unique, source for this information is the Internet, and particularly, available dynamic web services.

Our goal in this scenario was to design some kind of smart object that could be aware of both environment-provided and Internet-provided information in order to take decisions and look more intelligent from users' perspective.

Our choice was to create a smart umbrella that could obtain current weather information from both surrounding sensors and the Internet, as well as to obtain the weather forecast through the Internet for the next hours. The smart umbrella reacted when the user was leaving home without taking it by issuing a synthesised voice alert.

We attached the second generation smobject prototype to the umbrella (see Figure 6.18), and developed the following additional components:

- Ontologies and vocabularies for the weather and location domains.
- A `PerceptorWeatherDotCom` that connected to the `Weather.com` website and issued a request to obtain the current weather as well as the forecast for the next hours about the current location. The information was obtained in an XML-based language provided by `Weather.com` and dynamically transformed into a semantic form using the weather ontology designed for this purpose. This information was made available to other existing smobjects in the environment, including the umbrella.
- Crossbow Motes wireless nodes emulating rain sensors and proximity sensors to detect when the umbrella was taken from the umbrella stand or the user was leaving home.

- Appropriate behavioural profiles for the umbrella to be aware of current and future weather conditions, its location, or whether the user was leaving home, in order to suggest the user to take it.

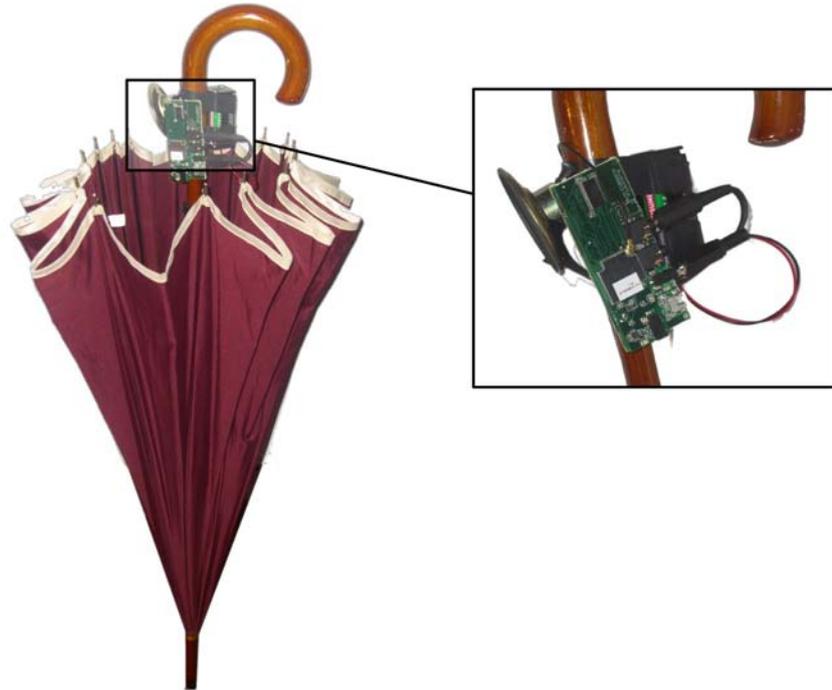


Figure 6.18: The smobject prototype in the umbrella.

**Example** Listing 6.5 illustrates a native behavioural profile for the umbrella: the umbrella suggests the user to be taken when the following (pre-)conditions occur:

- The umbrella has not been taken from the umbrella stand (it remains there, lines 6–8).
- The main door is open, this information can be provided by the door itself, based on the value obtained from a proximity sensor located at the main door (lines 18–20).
- The weather forecast announces rain (lines 26–28).

Listing 6.5: Example of native behavioural profile for the aware umbrella.

```

1   <behavioralProfile id="urn:uuid:prof_umbrella2" expires="PT20M"
2       requester="urn:uuid:umbrella">
3       <variable xml:id="maindoor"/>
4       <variable xml:id="area"/>
5
6       <precondition id="" predicate="http://www.awareit.com/onto
7           /2005/12/location#isLocatedIn" subject="urn:uuid:umbrella
8           ">
9           <objectResource resource="urn:uuid:umbrella_stand1"/>
10          </precondition>
11
12          <precondition id="" predicate="http://www.awareit.com/onto
13              /2005/12/location#isLocatedIn" subject="#maindoor">
14              <objectResource resource="http://www.awareit.com/onto
15                  /2005/12/location#ThisLocation"/>
16              </precondition>
17
18          <precondition id="" predicate="http://www.w3.org/1999/02/22-
19              rdf-syntax-ns#type" subject="#maindoor">
20              <objectResource resource="http://www.awareit.com/onto
21                  /2005/12/door#MainDoor"/>
22              </precondition>
23
24          <precondition id="" predicate="http://www.awareit.com/onto
25              /2005/12/door#state" subject="#maindoor">
26              <objectResource resource="http://www.awareit.com/onto
27                  /2005/12/door#Open"/>
28              </precondition>
29
30          <precondition id="" predicate="http://www.awareit.com/onto
31              /2005/12/location#isLocatedIn" subject="http://www.awareit
32              .com/onto/2005/12/location#ThisLocation">
33              <objectVariable ref="area"/>
34              </precondition>
35
36          <precondition id="" predicate="http://www.awareit.com/onto
37              /2005/12/weather#forecast" subject="#area">
38              <objectResource resource="http://www.awareit.com/onto
39                  /2005/12/weather#Rain"/>
40              </precondition>
41
42          <postcondition id="" predicate="http://www.awareit.com/onto
43              /2005/12/alerting#alertsAbout" subject="urn:uuid:umbrella
44              ">
45              <objectResource resource="urn:uuid:prof0_alert"/>
46              </postcondition>

```

```
34     <postcondition id="" predicate="http://www.awareit.com/onto
35         /2005/12/alerting#level" subject="urn:uuid:prof0_alert">
36         <objectResource resource="http://www.awareit.com/onto
37             /2005/12/alerting#information"/>
38     </postcondition>
39
40     <postcondition id="" predicate="http://www.awareit.com/onto
41         /2005/12/alerting#message" subject="urn:uuid:prof0_alert">
42         <objectLiteral>Please, take me, it is going to rain.</
43             objectLiteral>
44     </postcondition>
45
46     <postcondition id="" predicate="http://www.awareit.com/onto
47         /2005/12/alerting#title" subject="urn:uuid:prof0_alert">
48         <objectLiteral>Information</objectLiteral>
49     </postcondition>
50
51 </behavioralProfile>
```

Whenever all these conditions happen the umbrella built-in voice synthesiser recommends the user to take it (lines 30–44, and especially 38–40).

An excerpt of integrated context information that provokes this situation is represented in Listing 6.6.

**Listing 6.6:** Excerpt of integrated context information obtained from several sources by the aware umbrella.

```
1 <?xml version="1.0"?>
2 <rdf:RDF
3     xmlns="http://www.awareit.com/example#"
4     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
5     xmlns:owl="http://www.w3.org/2002/07/owl#"
6     xmlns:door="http://www.awareit.com/onto/2005/12/door#"
7     xmlns:loc="http://www.awareit.com/onto/2005/12/location#"
8     xmlns:weather="http://www.awareit.com/onto/2005/12/weather#"
9     xml:base="http://www.awareit.com/example">
10
11 <rdf:Description rdf:about="urn:uuid:maindoor">
12     <rdf:type rdf:resource="http://www.awareit.com/onto/2005/12/door#
13         MainDoor"/>
14     <door:state rdf:resource="http://www.awareit.com/onto/2005/12/door
15         #Open"/>
16     <loc:isLocatedIn rdf:resource="urn:uuid:home"/>
17 </rdf:Description>
18
19 <rdf:Description rdf:about="urn:uuid:umbrella">
20     <loc:isLocatedIn rdf:resource="urn:uuid:umbrella_stand1"/>
21 </rdf:Description>
```

```
20
21 <rdf:Description rdf:about="urn:uuid:home">
22   <rdf:type rdf:resource="http://www.awareit.com/onto/2005/12/
23     location#Building"/>
24   <owl:sameAs rdf:resource="http://www.awareit.com/onto/2005/12/
25     location#ThisLocation"/>
26   <loc:isLocatedIn rdf:resource="urn:uuid:bilbao"/>
27 </rdf:Description>
28
29 <rdf:Description rdf:about="urn:loccode:SPXX0016">
30   <rdf:type rdf:resource="http://www.awareit.com/onto/2005/12/
31     location#Town"/>
32   <owl:sameAs rdf:resource="urn:uuid:bilbao"/>
33   <weather:acurrent rdf:resource="http://www.awareit.com/onto
34     /2005/12/weather#Rain"/>
35 </rdf:Description>
36 </rdf:RDF>
```

Lines 11–15 are provided by the smobject at the main door; lines 17–19 are provided by the smobject connected to the wireless sensor network and particularly to the proximity sensor at the umbrella stand; lines 21–25 and 27–29 are provided by the room, while line 30 is provided by the `PerceptorWeatherDotCom` located at the umbrella, but could be located at the room’s smobject as well<sup>3</sup>.

We can notice how the `owl:sameAs` predicate and the location relationships between the main door, the building and the town contribute to the proper identification of the place and the associated weather forecast, so the behavioural profile is positively evaluated and the postconditions executed.

### **WorkSafe: a protective working agent enforcing user care in dangerous environments**

We also wanted to demonstrate how SoaM could be used in heterogeneous working environments to provide workers with intelligent safety measures. We designed an scenario populated with different electrical tools, a number of containers filled with flammable and chemical products and sensorised workwear. All these objects were transformed into smobjects, so they were able to exchange information about themselves and their perceptions.

---

<sup>3</sup>SPXX0016 is the `Weather.com` code for Bilbao.

The workwear jacket (see Figure 6.19) included a built-in prototyped BPinjector that disseminated several profiles through the environment with the purpose of alerting the user whenever some dangerous situation happened.

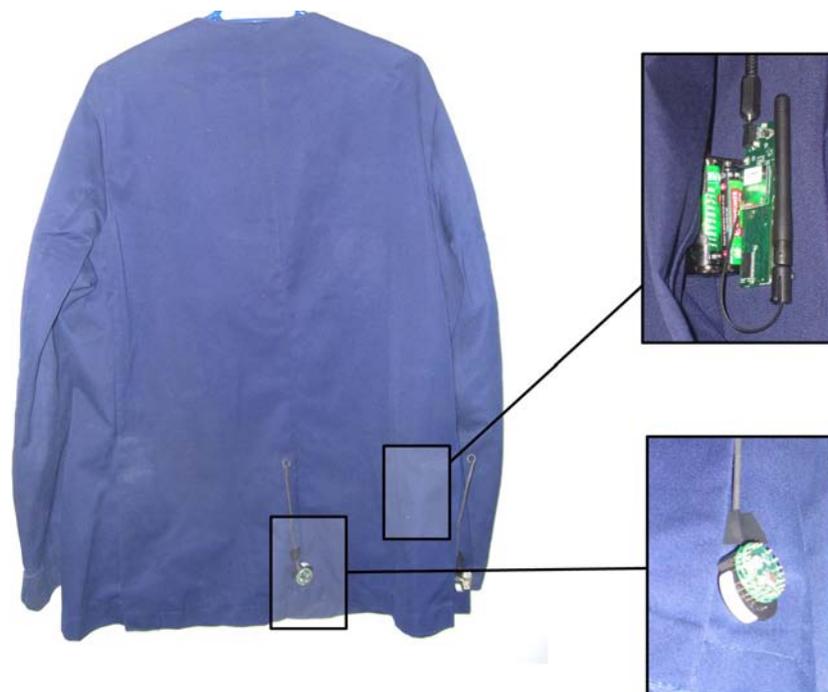


Figure 6.19: The workwear jacket with two wireless accelerometers to detect body orientation, and the smobject prototype (covered by fabric) with the BPinjector software.

For this scenario we developed the following components:

- Ontologies and vocabularies for danger, electrical, flammable and chemical domains.
- Domain rules for the above domains.
- Wireless sensor nodes to detect manipulation of flammable and chemical containers (via accelerometers or proximity sensors in the lid).
- A sensorised jacket with wireless sensor nodes to detect whether the user was laying in the ground, using accelerometers, and a built-in BPinjector.
- A number of behavioural profiles representing different kinds of reactivity when some dangerous situation happened. For example, if a

container with flammable products was opened while an electrical tool was being used (see Listing 6.7), the tool switched off automatically, or if a user fell down in a dangerous environment (e.g. while manipulating chemical products), an alarm was issued to a monitoring centre.

- An `EffectorSmsAlerter` that was able to send SMS messages to a remote monitoring service to alert about possible harms to workers. Alternatively, a `EffectorTrayAlerter` was also provided.

This kind of intelligent environment works in the periphery of attention to preserve workers' safety, not only alerting about possible dangerous situations but also actively reacting as the case with flammable and electrical tools.

**Example** Listing 6.7 illustrates a behavioural profile that makes an electrical tool to switch off when a flammable product is located nearby.

Listing 6.7: Example of a native behavioural profile for an electrical tool.

```

1 <behavioralProfile id="urn:uuid:prof_discon_electric_app1" expires
2   = "PT20M" requester="http://people.com/bobby">
3
4   <variable xml:id="material"/>
5
6   <precondition id="" predicate="http://www.awareit.com/onto
7     /2005/12/location#nearby" subject="urn:uuid:electric_tool
8     ">
9     <objectVariable ref="material"/>
10    </precondition>
11
12    <precondition id="" predicate="http://www.w3.org/1999/02/22-
13      rdf-syntax-ns#type" subject="#material">
14      <objectResource resource="http://www.awareit.com/onto
15        /2005/12/danger#InflammableMaterial"/>
16    </precondition>
17
18    <postcondition id="" predicate="http://www.awareit.com/onto
19      /2005/12/device#state" subject="urn:uuid:electric_tool">
20      <objectResource resource="http://www.awareit.com/onto
21        /2005/12/device#StateOff"/>
22    </postcondition>
23
24 </behavioralProfile>

```

A fine-grained location system must be available in the environment for generating facts containing the `loc:nearby` predicate.

The behavioural profile shown in Listing 6.8 represents the reactivity to carry out when a worker is laying on the floor and a chemical product was being manipulated. It is noteworthy the relative complexity of this profile, and the need for five variables, some of them used to compose the alarm message (lines 53–55).

**Listing 6.8: Example of a behavioural profile for sending an alarm whenever a worker collapsed in a dangerous environment.**

```
1      <behavioralProfile id="urn:uuid:profal" expires="PT20M" requester
2          ="http://people.com/bobby">
3          <variable xml:id="alerter"/>
4          <variable xml:id="risk"/>
5          <variable xml:id="riskLabel"/>
6          <variable xml:id="worker"/>
7          <variable xml:id="workerLabel"/>
8
9          <precondition id="" predicate="http://www.awareit.com/onto
10             /2005/12/location#isLocatedIn" subject="#alerter">
11             <objectResource resource="http://www.awareit.com/onto
12                 /2005/12/location#ThisLocation"/>
13         </precondition>
14
15         <precondition id="" predicate="http://www.w3.org/1999/02/22-
16             rdf-syntax-ns#type" subject="#alerter">
17             <objectResource resource="http://www.awareit.com/onto
18                 /2005/12/alerting#Alerter"/>
19         </precondition>
20
21         <precondition id="" predicate="http://www.w3.org/1999/02/22-
22             rdf-syntax-ns#type" subject="http://www.awareit.com/onto
23                 /2005/12/location#ThisLocation">
24             <objectResource resource="http://www.awareit.com/onto
25                 /2005/12/danger#DangerousLocation"/>
26         </precondition>
27
28         <precondition id="" predicate="http://www.awareit.com/onto
29             /2005/12/danger#healthRiskLevel" subject="http://www.
30             awareit.com/onto/2005/12/location#ThisLocation">
31             <objectVariable ref="risk"/>
32         </precondition>
33
34         <precondition id="" predicate="http://www.w3.org/2000/01/rdf-
35             schema#label" subject="#risk">
36             <objectVariable ref="riskLabel"/>
37         </precondition>
38
39         <precondition id="" predicate="http://www.w3.org/1999/02/22-
40             rdf-syntax-ns#type" subject="#worker">
```

```

30         <objectResource resource="http://pervasive.semanticweb.org
31           /ont/2004/06/person#Person"/>
32     </precondition>
33     <precondition id="" predicate="http://www.awareit.com/onto
34       /2005/12/location#isLocatedIn" subject="#worker">
35       <objectResource resource="http://www.awareit.com/onto
36         /2005/12/location#ThisLocation"/>
37     </precondition>
38     <precondition id="" predicate="http://www.awareit.com/onto
39       /2005/12/location#position" subject="#worker">
40       <objectResource resource="http://www.awareit.com/onto
41         /2005/12/location#Laying"/>
42     </precondition>
43     <precondition id="" predicate="http://www.w3.org/2000/01/rdf-
44       schema#label" subject="#worker">
45       <objectVariable ref="workerLabel"/>
46     </precondition>
47     <postcondition id="" predicate="http://www.awareit.com/onto
48       /2005/12/alerting#alertsAbout" subject="#alerter">
49       <objectResource resource="urn:uuid:prof0_alert"/>
50     </postcondition>
51     <postcondition id="" predicate="http://www.awareit.com/onto
52       /2005/12/alerting#level" subject="urn:uuid:prof0_alert">
53       <objectResource resource="http://www.awareit.com/onto
54         /2005/12/alerting#critical"/>
55     </postcondition>
56     <postcondition id="" predicate="http://www.awareit.com/onto
57       /2005/12/alerting#message" subject="urn:uuid:prof0_alert">
58       <objectLiteral>Warning: the worker ?(workerLabel) is
59         laying on the ground at a location with a '? (riskLabel
60         )' risk level.</objectLiteral>
61     </postcondition>
62     <postcondition id="" predicate="http://www.awareit.com/onto
63       /2005/12/alerting#title" subject="urn:uuid:prof0_alert">
64       <objectLiteral>Alarm</objectLiteral>
65     </postcondition>
66     <postcondition id="" predicate="http://www.awareit.com/onto
67       /2005/12/alerting#frequency" subject="urn:uuid:prof0_alert
68       ">
69       <objectLiteral>30</objectLiteral>
70     </postcondition>

```

```
64
65 </behavioralProfile>
```

For this scenario we chose to deploy an `EffectorSmsAlerter` so the alert could reach a remote monitoring facility. Alternatively, we also deployed an `EffectorTrayAlerter` that displayed a message in a screen at the monitoring centre<sup>4</sup> (see Figure 6.20).

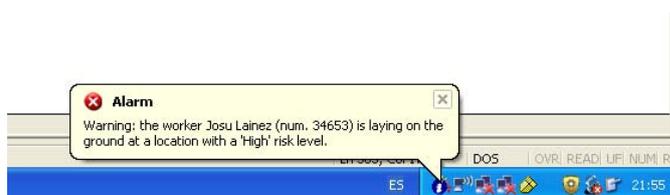


Figure 6.20: An `EffectorTrayAlerter` at a monitoring centre that generates an alarm if a worker collapsed in a dangerous environment.

Several domain rules about dangerous situations provided higher level information to evaluate context knowledge. For example, a domain rule related a concrete range in the workwear jacket’s accelerometers to the worker’s position, and the risk level of a room was inferred based on a contained material or the nature of the tasks taking place there, as illustrated in Listing 6.9.

Listing 6.9: Domain rules associating risk levels of task and materials to their location.

```
1 ?place <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
   awareit.com/onto/2005/12/location#Room> .
2 ?person <http://www.awareit.com/onto/2005/12/location#isLocatedIn> ?
   place .
3 ?person <http://www.awareit.com/onto/2005/12/task#isDoing> ?task .
4 ?task <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
   awareit.com/onto/2005/12/danger#DangerousTask> .
5 ?task <http://www.awareit.com/onto/2005/12/danger#healthRiskLevel> ?
   risk .
6 ->
7 ?place <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
   awareit.com/onto/2005/12/danger#DangerousLocation> .
8 ?place <http://www.awareit.com/onto/2005/12/danger#healthRiskLevel> ?
   risk .
9
10 ****
11
```

<sup>4</sup>The location’s id was not included in this example to facilitate reading.

```
12 ?place <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
    awareit.com/onto/2005/12/location#Room> .
13 ?material <http://www.awareit.com/onto/2005/12/location#isLocatedIn> ?
    place .
14 ?material <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
    www.awareit.com/onto/2005/12/danger#DangerousMaterial> .
15 ?material <http://www.awareit.com/onto/2005/12/danger#healthRiskLevel>
    ?risk .
16 ->
17 ?place <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.
    awareit.com/onto/2005/12/danger#DangerousLocation> .
18 ?place <http://www.awareit.com/onto/2005/12/danger#healthRiskLevel> ?
    risk .
```

### Other scenarios

We also created some other simpler scenarios such as:

- Souvenir-aware Google Earth: we designed an application that showed a place somehow related to an object (usually a souvenir), whenever the object was placed in a stand near the screen. The system included a Google Earth browser controlled (via an effector) by an smobject that was aware of the objects placed nearby; this perception was provided by another smobject connected to an RFID reader in the stand, and the objects were tagged using RFID tags (see Figure 6.21).
- Gesture-enabled Web browser: we placed several wireless accelerometers in shirt sleeves in such a way that a user could make certain information to appear in a computer, depending on his arms gestures. The system was designed using a smobject connected to the Crossbow Motes wireless sensor network and an effector that controlled the Web browser.

These examples do not deal with high amounts of data or reasoning processes; however they illustrate how even simple reactive systems can be easily designed and deployed with SoaM, so that objects in the environment can be aware of each other.

## 6.3 Conclusions

A complete evaluation process was carried out involving the creation of prototypes and their deployment in experimental scenarios demanding Ambient Intelligence capabilities in order to validate the hypothesis.

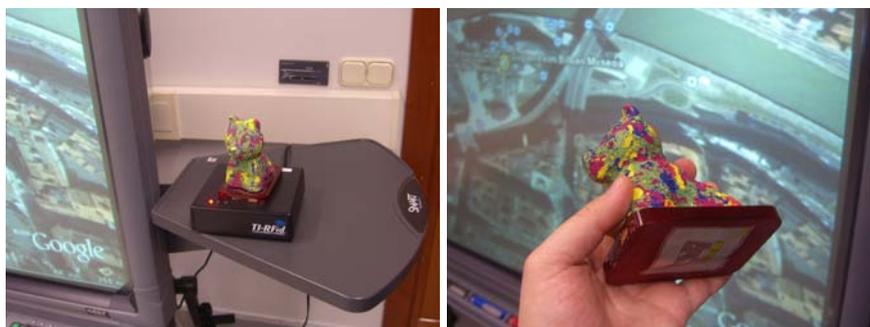


Figure 6.21: The souvenir-aware Google Earth display (the RFID reader and a tagged souvenir).

Managing the SoaM components to work in the limited ConnectCore 7U was specially difficult. However, the economy and compactness of the developed middleware contributed to easy porting and higher performance results in more powerful platforms, as it turned out in the last prototypes.

The most advanced smobject prototype was built upon the Gumstix platform and featured Wi-Fi connectivity and autonomous batteries. We consider that the next platform evolution should feature integrated semantic capabilities optimised for the platform (mRDP as a native platform protocol and a built-in semantic reasoner), so that performance will be remarkably increased.

Intensive testing and evaluation was carried out in the above described and other similar scenarios to demonstrate how the SoaM architecture could be applied to satisfy a broad range of requirements involving intelligent behaviour from surrounding objects and the environment. We also considered the advantages of the different deployment topologies for SoaM depending on the type and constraints of the scenario.

For evaluation, we tried to choose real world -like scenarios where intelligent behaviour from existing objects could boost perceived intelligence at the users' service, thus fulfilling the fundamental ideas underlying the hypothesis.

Designing these scenarios involved a mixture of hardware and software prototyping as well as integration of complementary technologies, such as wireless sensor networks or XML Web Services, which may behave as "raw information" sources for SoaM-provided semantic processes.

## Conclusion

*“When the industrial revolution came, we didn’t go to motorspace. The motors came to us as refrigerators to store our food and cars to transport us. This kind of transition is exactly what I expect will happen with computers and communications: they will come into our lives, and their identities will become synonymous with the useful tasks they perform.”*

*Michael L. Dertouzos  
The Future of Computing, Scientific American, 1999*

**A**FTER presenting our work and the evaluation results, it is now time to revisit once again our initial hypothesis:

*To prove that devices based on semantic technologies provide the level of context-awareness, intelligence and adaptability required in smart environments.*

We consider that the functional evaluation of the experimental scenarios carried out by deploying the SoaM architecture validates this hypothesis entirely. Smobjects, as semantic devices, provide the level of context-awareness, intelligence and adaptability required for Ubiquitous Computing / Ambient Intelligence environments:

- Context-awareness: smobjects are context-aware entities. They represent and exchange context information with fellow devices in a natural way, extending their own individual perception capabilities. They react accordingly to changes in the environment honouring their behavioural profiles.

- **Intelligence:** smobjects are able to perform reasoning through ontologies and domain inference rules. This feature empowers them with the ability to augment and interpret context information in a more advanced way than other alternatives.
- **Adaptability:** smobjects's behaviour can be dynamically modified to make them aware of new perceptions and reactive to different *stimuli*.

Moreover, smobjects feature a high degree of autonomy: they can spontaneously discover each other, share context information and develop their reactive behaviour in a completely unattended manner and with no central component required.

SoaM is a decentralised architecture with spontaneous collaboration by context-aware reactive entities that exhibit semantic reasoning mechanisms. Moreover, it can be further powered with an orchestrator at the environment for better performance and higher intelligence, while smobjects still remain simple enough for implementation in limited devices.

Figure 7.1 graphically illustrates a comparison of the two SoaM topologies and the other architectures.

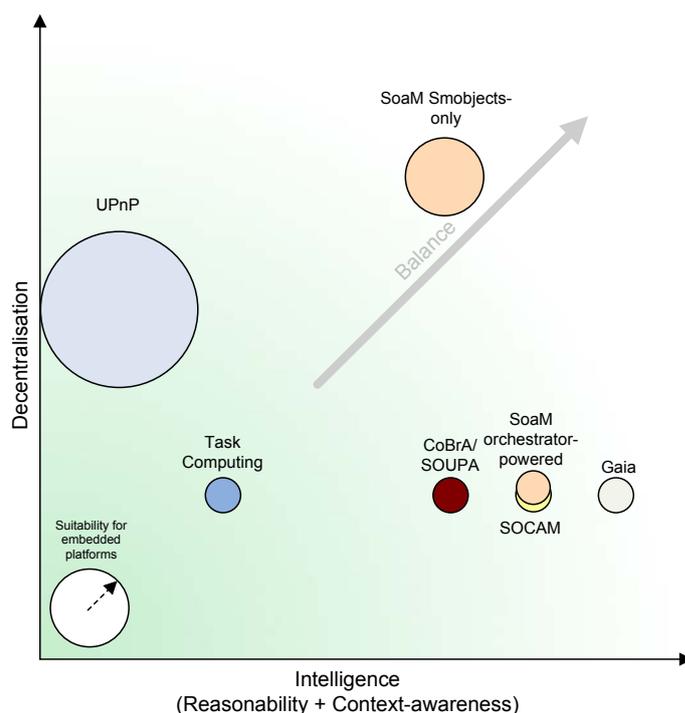


Figure 7.1: Graphical comparison of SoaM and other architectures.

While the orchestrator-powered topology ranks in a similar way to other centralised architectures, the smobjects-only topology exhibits the more balanced degree among decentralisation, intelligence and suitability for being implemented in embedded platforms. This approach fully leverages the concepts of serendipity and autonomic computing promoted by semantic devices.

Not only the goals have been accomplished, but SoaM also features a number of remarkable characteristics:

#### **Use of the web model (URI, HTTP, HTTPS) for communication**

SoaM heavily relies on the web information model: the URI as identification mechanism for concepts, entities and other resources; HTTP as underlying communication protocol, allowing extensibility through additional headers, or HTTPS for secure communication; and XML as the language for serialising the information and data structures during transmission.

The only phase where HTTP does not comply with SoaM requirements is during the process of discovery, where multicast capabilities are required. Our proposal to fill this gap, mRDP, takes advantage of HTTP, by using it for the callbacks.

These features make SoaM a projection of the web communication model into the Ubiquitous Computing world, further than simply using HTTP or browsing, but at a deeper level: creating an ambient-scale web of knowledge populated by cooperative devices.

#### **Use of Semantic Web technologies for context interpretation and reasoning (RDF, OWL)**

All the knowledge representation techniques rely on Semantic Web technologies: RDF is used for context information representation, while OWL is used for ontologies and vocabularies. In SoaM we used a customised representation for domain rules since the work of the World Wide Web Consortium on creating a standardised language for rules (Rule Interchange Format) is still on its way.

Our research extends current knowledge by incorporating novel elements, such as a lightweight query language for discovery in networks populated by limited devices: Plant. Plant and ReDEL provide a convenient means for issuing semantic search requests throughout an environment in order to find the devices or resources that match certain conditions.

The exclusive use of Semantic Web technologies in SoaM provides seamless integration with the other web standards (previous initiatives lacked of this integration as discussed in chapter 2), and they are considered one of the most potential technologies for representing and exchanging knowledge in distributed computing networks.

### **Seamless integration of local and global information sources**

The possibility of accessing external information sources by environmental devices, integrating and exchanging this knowledge among them enables SoaM to extend the context-awareness concept beyond the local scope, promoting the concept of social devices.

Architecturally, there is no much difference in smobject perceptors providing semantic information obtained from local sensors or global Internet sources. Thus, integration of traffic data, weather information, TV guides, local news, and so forth, is absolutely natural in SoaM and smobject-enabled devices can share and react appropriately to context changes.

### **Decentralised architecture and spontaneous adaptability to a highly variable number of devices**

While some real world environments are relatively static from the point of view of its constituent elements, some other environments are definitely highly dynamic with objects and devices joining and leaving the digital ecosystem continuously.

We faced this challenge in SoaM and tried to create an architectural model that allowed devices to discover the semantic capabilities provided by their neighbours at any time. The smobjects-only topology is a fully decentralised architecture that makes the whole system self-manageable and self-configuring, without user intervention.

We think that our approach entirely complies with the spontaneous nature of Ubiquitous Computing, which was one of the main features that previous initiatives lacked.

### **Flexible and suitable semantic discovery mechanism**

The problem of discovery has been a hot topic in Ubiquitous Computing for many years. None of the current discovery mechanisms seemed suitable for supporting the intelligent features required by SoaM.

We even adapted a popular discovery protocol, such as SSDP, to support semantic features, but finally we decided to face the challenge of designing a lightweight semantic discovery protocol: mRDP.

We consider the design of mRDP as a remarkable result of our research. It has been conceived for being used not only in Ubiquitous Computing scenarios, but also in more general traditional computing networks for the semantic discovery of resources.

mRDP enables smobjects to search for fellow devices that honour complex queries involving semantic relationships in their descriptions, while featuring a small footprint suitable for embedded devices.

### **Behavioural programming model based on documents, not on code, along with mechanisms to modify this behaviour dynamically according to the context**

One of the strengths and main differences of SoaM compared to other approaches is the semantic-based behavioural model. Smobjects can adapt their behaviour dynamically as required by users or other clients, even smobjects, based on the concept of behavioural profiles.

This approach strengthens the vision of devices whose behaviour is not completely determined statically from the beginning, but they can be adapted to perceive (indirectly, via other devices) new sets of *stimuli* and react to them in an appropriate manner.

Thus, a user can make its umbrella not only “weather-aware” but also “main door-aware” or “car state-aware” in order to get a visual clue about the need to take the umbrella; a different user can make its wrist watch “location-aware” or “activity-aware” in order to deactivate a preconfigured alarm, and so forth.

Since there are no constraints about what a smobject-powered device can perceive, there are no limits about the perceptions to which the device can autonomously react, as long as the appropriate behavioural profiles are provided.

### **Dynamic mechanism to allow users to influence devices’ behaviour, as well as for devices to influence each other’s behaviour**

The behavioural profiles model enables environmental agents to influence each other and dynamically adapt the behaviour of existing entities to their requirements (as long as they have the appropriate credentials if a security model is used).

Profile injection is the basic mechanism provided in SoaM to make entities to influence each other. This process conceptually represents something similar to a “firmware update” on smobjects, updating their behaviour to make them more aware and reactive to different perceptions.

In the experimental scenarios, we have not only explored the more traditional situation involving a user influencing his surrounding devices, but also a case where smart plants influenced their location to meet their requirements.

We think that intelligent devices will become first-class actors in our environments in very similar ways.

### **Distributed reasoning among several participating entities**

A form of OWL reasoning, a subset of OWL Lite, is performed at the smobjects, both for resolving Plant queries and analysing context information prior to behavioural profiles resolution. As described earlier, this subset of OWL is enough for most of the situations that occur in Ubiquitous Computing scenarios. In other cases, where more powerful reasoning processes are required, the SoaM architecture allows an orchestrator to be deployed in the environment in order to provide OWL DL or a larger subset of OWL Full reasoning.

While OWL reasoning at the smobjects is more limited than at the orchestrator, we consider it to better represent the philosophy of distributed reasoning: no central elements, but individual entities sharing and reasoning over a set of context information in order to take their decisions in an autonomous manner and react intelligently.

### **Reuse existing and future ontologies conveying knowledge about different domains as well as a means for discovering this knowledge**

We wanted SoaM not only to be extensible, but also to take advantage of existing work already carried out in the area of applying Semantic Web technologies, specially ontologies, in Ubiquitous Computing.

We have designed SoaMonto as a support ontology for SoaM but any domain ontology for particular applications and scenarios can be reused. We proved this point both reusing existing ontologies, such as SOUPA [CPFJ04] or Time-OWL[PH05], and designing specific ones for our evaluation scenarios, such as the basic weather ontology.

We have also designed a mechanism, based on mRDP and SoaMonto, for dynamic discovery and retrieval of ontologies and domain rules from other entities in the environment.

### **Feasible for implementation in embedded platforms, attachable to everyday objects**

Our work would not have been successful if after developing the theoretical basis, designing the architectural elements and the protocols, and implementing the designs in a programming language, we had found the platform requirements to be unacceptable for the aimed embedded platforms.

In order to overcome this problem, the prototyping phase was carried out in iterative steps, where increasingly more complex prototypes were built on the most limited platform (ConnectCore 7U) to guarantee its portability to most powerful ones. The algorithms and modules were checked and redesigned several times to improve their performance and to reduce their size.

The result of the process is that we successfully deployed the smobject middleware in different commercial embedded computing platforms. The separation of SmobjectBase and SmobjectAware components makes even easier the task of selecting the appropriate level of intelligence to embed into a platform depending on its computing power.

We consider feasibility for embedded computing platforms as an important issue not covered in previous initiatives that generally required a central server to perform reasoning and coordination tasks.

On the other hand, SoaM takes a more difficult but also more powerful approach: to delegate these tasks to the objects populating the environment, since they are now powered with a basic level of intelligence.

## **7.1 Contributions**

We would like to remark again the four main outcomes of our research:

- mRDP and Plant: a semantic discovery protocol that can be applied in Ubiquitous Computing and local networking scenarios.
- The SoaM theoretical model: a technology-independent model on how entities influence each other and how required behaviours are honoured.

- The SoaM architecture: a theoretical model -based design featuring different entities, namely smobjects, orchestrators and BPinjectors, in order to create intelligent devices and environments.
- Prototypes and evaluation results of the SoaM model and architecture based on real world -like scenarios.

The first three outcomes are aligned with the specific goals of our research, satisfying the general goal, while the fourth outcome is built upon them in order to validate the hypothesis.

The modular architecture, formed by the triad SmobjectBase, SmobjectAware and orchestrator, enables the Ambient Intelligence architect to place the intelligent mechanisms at the convenient entity, depending on the circumstances.

It is even possible to have some devices hosting only the SmobjectBase component. These devices do not carry out any adaptive behaviour; they just act as semantic information providers for other smobjects. Devices featuring sensors but not actuators are suitable candidates to host the SmobjectBase component alone, thus resulting in reduced platform requirements.

SoaM also empowers designers with a flexible framework to create the solution that best fits a concrete scenario, depending on available computing platforms. Moreover, the spontaneous discovery and operation features of SoaM makes deployment and configuration completely automatic.

We would also like to note again the concept of *semantic device* as an important outcome for facing the design of intelligent ubiquitous objects.

### 7.1.1 Discussion

We believe that a scientific research must also be self-critic. One of the most important issues we faced throughout the research process was to keep a proper balance between intelligence and simplicity in smobjects: too much intelligence is not feasible in limited devices, while too much simplicity may result in a worthless system.

Below, there is some discussion about several aspects of our architecture.

#### Security issues

Although we have not focused our research on any specific security mechanism, we were continually concerned about this issue. As explained

thoroughly, SoaM applies the web communication model in all the exchange flows among participating entities; therefore, both HTTP [FGM<sup>+</sup>99] and HTTPS [Res00] can be used for information transfer.

The only point where secure communications cannot be applied is in the initial multicast packet of mRDP discovery, which anyway is intended to reach every host, and thus security is not a concern at that stage. Moreover, the callback address contained in the packet can be an HTTPS URL, thus enabling a secure channel for the replies.

For the sake of simplicity we have represented all the previous examples as HTTP interactions. However, HTTPS can seamlessly replace HTTP and provide authentication via client and server certificates, authorisation and secure communication.

Device manufacturers can provide smobjects with built-in certificates containing the smobject UUID among other data. Smobjects exchange the certificates during the negotiating phase of TLS [DR06] in order to obtain a symmetric key to encrypt further transmissions. The process is very similar to how traditional HTTPS is used nowadays in the Web, except that client certificates should also be used here to validate the requesting party<sup>1</sup>, and those certificates should use the smobject UUID as the name instead of IP address or hostname, since the latter are dynamic.

In fact, this scheme is supported in HTTPS [Res00]:

*For instance, a client may be connecting to a machine whose address and hostname are dynamic but the client knows the certificate that the server will present.*

Authorisation can be easily implemented using ACL (Access Control Lists) at the smobjects. Once the other party's certificate has been obtained, the smobject can check whether it is allowed to operate or not.

In this way, SoaM could support the following security services among others:

- Only accepting behavioural profiles from authorised parties.
- Only accepting constraints issued by authorised parties.
- Only providing context information to authorised parties.

In any case, further prototyping must be carried out to validate this security model and identify further implications.

---

<sup>1</sup>Something that is carried out in the traditional web using the user / password mechanism.

### **Network traffic optimisation in the smobjects-only topology**

The smobjects-only topology is completely decentralised, where each smobject creates the required information flows with others in order to evaluate its behavioural profiles. We already carried out some optimisations such as selective smobject polling and information caching.

However, at the initial stages of design, we considered subscription systems instead of polling, so that smobjects could subscribe to changes in information managed by others. We finally rejected this alternative due to several reasons:

- Subscriptions management would add even more complexity to the tasks already performed by smobjects, both at the subscriber and notifier sides. Two extra modules would be required at every smobject for managing its subscriptions on others, as well as others' subscription on the smobject.
- Evaluation of conditions adds an additional workload in smobjects who are not responsible for honouring a concrete behavioural profile. Using eventing, context conditions would be evaluated at the *information providers*, not at the *reactivity providers*. This situation leads to unbalanced responsibilities. For example, a simple sensor system without any reactivity (no *SmobjectAware* component) may have to evaluate a lot of conditions requested by other systems, making it unnecessarily more complex than initially required.
- If the smobject features a reasoning mechanism, it cannot rely on events provided by surrounding devices: it needs raw access to the original information, in order to infer new data. In this case, eventing could only be a valid solution if the notifier guarantees that is able to perform an equivalent reasoning to that in the subscriber, adding even a more unfair workload at its side.

### **Gap between design and available technology**

There is a gap between our vision and available technology. We would have preferred to have full computing platforms able to run the Jena Semantic Framework featuring the size and power consumption of existing wireless sensor network nodes.

Since this goal was a chimera, we tried to find a proper balance among computing power, reasoning capabilities and platform lightness. We do think that the platform sizes of ConnectCore 7U / 9U or the Gumstix are small enough for prototyping semantic gadgets.

Moreover, the main challenge of SoaM was creating a model for the collaboration of semantic devices. The SmobjectBase component can be implemented in almost any architecture featuring TCP/IP communication, while the SmobjectAware component, with semantic reasoning capabilities, requires just a little more computing power.

Since the beginning of the age of computing, we have witnessed some trends regarding computing platforms, such as:

- Platform size reduction.
- Improved power-saving technologies and longer battery life, even for IP networking.
- Higher computing power and storage.
- Augmented communication capabilities.
- Price reduction.

These trends have been even more notable in the mobile computing market during the last years, and there are no reasons to believe that they are not going to continue in this way.

SoaM was not designed just for the short-term: we pretend to apply it widely during the next years, adapting it to available platforms with reduced size and price as well as improved capabilities, in order to create semantic-powered Ambient Intelligence scenarios.

### 7.1.2 Publications

During the research process we tried both to collect feedback from other prestigious researchers and groups related to our topic and to disseminate our experiences by submitting obtained results to different conferences and workshops.

A lot of effort was devoted to attending congresses and presenting papers in order to obtain late-breaking results and feedback about different parts of our architecture. A remarkable milestone was the W3C Workshop on the Ubiquitous Web, held in Tokyo, Japan, in March 2006, where we were invited to present a position paper about our vision for the Ubiquitous Web and the synergies with the Semantic Web activity [VAL06].

Networking with colleagues and researchers was one of our main concerns. For instance, as a result of some conversations during the mentioned W3C workshop we were able to advance on the work on the semantic discovery protocol.

One of the first practical results of our research was a theoretical model for context-aware reactivity, so that we could design a whole architecture complying with it.

The design process was carried out using an iterative approach, highly linked with the implementation activity, during which increasingly more complete versions of the prototypes were implemented and tested, thus validating both the theoretical model and the architecture.

We designed and created prototype implementations in order to evaluate our results and deployed real world -like scenarios. This phase was exceptionally time and resource consuming but, of course, absolutely necessary for the validation of the hypothesis.

Feedback about the research results were obtained from different sources:

1. We obtained quantitative measures of different aspects during the evaluation and tried to improve the designs until acceptable performance was achieved.
2. We designed challenging scenarios based on our architecture in order to evaluate its suitability for real world deployment.
3. We presented partial and final results in congresses, workshops, other scientific events, publications and even a blog<sup>2</sup> in order to get comments from the research community.

We consider all of them necessary and important in order to get complementary feedback about SoaM.

Our publications also represent the outcomes and contribution of our research, illustrating in some way the evolution of the process from the very initial steps to its final form:

- Juan Ignacio Vazquez, López de Ipiña, and Iñigo Sedano. SoaM: A Web-powered architecture for designing and deploying pervasive semantic devices. *IJWIS - International Journal of Web Information Systems*, (to be published), 2007.
- Juan Ignacio Vazquez, Iñigo Sedano, and Diego López de Ipiña. Evaluation of orchestrated reactivity of smart objects in pervasive Semantic Web scenarios. In *Proceedings of The Second International Workshop on Semantic Web Technology For Ubiquitous and Mobile*

---

<sup>2</sup><http://www.awareit.com/blog>

*Applications (SWUMA'06) at the 17th European Conference of Artificial Intelligence (ECAI 2006)*, August 2006.

- Juan Ignacio Vazquez, Diego López de Ipiña, and Iñigo Sedano. SoaM: An environment adaptation model for the Pervasive Semantic Web. In *Proceedings of the 2nd Ubiquitous Web Systems and Intelligence Workshop (UWSI 2006), colocated with ICCSA 2006. Lecture Notes in Computer Science - LNCS, volume 3983*, pages 108117, May 2006.
- Juan Ignacio Vazquez, López de Ipiña, and Iñigo Sedano. A passive influence model for adapting environments based on semantic preferences. In *Proceedings of the International Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB 2006), at Pervasive 2006*, May 2006.
- Juan Ignacio Vazquez, Joseba Abaitua, and Diego López de Ipiña. The Ubiquitous Web as a model to lead our environments to their full potential. In *Proceedings of the W3C Workshop on the Ubiquitous Web. World Wide Web Consortium*, 2006. Position paper.
- Juan Ignacio Vazquez and Diego López de Ipiña. Empowering wireless UPnP devices with WebProfiles. In *Proceedings of the 10th IFIP International Conference on Personal Wireless Communications*, 2005.
- Juan Ignacio Vazquez and Diego López de Ipiña. Webprofiles: A negotiation model for user awareness in personal area networks. In *Proceedings of MobiQuitous 2005 - The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 373-383, 2005.
- Juan Ignacio Vazquez and Diego López de Ipiña. Environment adaptation meeting user preferences. In *Proceedings of Ambient Intelligence and (Everyday) Life*, 2005.
- Juan Ignacio Vazquez and Diego López de Ipiña. An HTTP-based context negotiation model for realizing the user-aware Web. In *Proceedings of the 1st International Workshop on Innovations in Web Infrastructure at the 14th International World Wide Web Conference (WWW2005)*, 2005.
- Juan Ignacio Vazquez and Diego López de Ipiña. A language for expressing user-context preferences in the Web. In *WWW'05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 904-905, New York, NY, USA, 2005. ACM Press.
- Juan Ignacio Vazquez and Diego López de Ipiña. An interaction model for passively influencing the environment. In *Adjunct Proceedings of*

*EUSAI 2004: the 2nd European Symposium on Ambient Intelligence, 2004.*

## 7.2 Future research and challenges

We have explored the possibility of integrating Ubiquitous Computing and Semantic Web technologies in order to create an architecture populated by semantic devices and realising the Ambient Intelligence vision.

From our point of view, there are at least two main pathways for research communities interested in continuing experimenting with these approaches:

1. Exploring other forms of semantic devices.
2. Exploring and extending the SoaM model and architecture.

During the next subsections we briefly explain and suggest different potential research activities to carry out in these areas.

### 7.2.1 Exploring other forms of semantic devices

Regarding the first approach, there are several possibilities we can anticipate:

#### **Semantic Sensor Networks**

We have already integrated wireless sensor networks (WSN) into our model by creating a suitable perceptor connected to a Crossbow Motes WSN. However, the real challenge would be to enable existing WSN technologies to work directly with RDF representation of captured data, and provide these data to requesting parties.

This possibility is being explored by Ni et al. [NZM<sup>+</sup>05], although no further improvements seem to have been made so far. Obviously, platform constraints are even more important in WSN nodes than in other ubiquitous artifacts, thus making the implementation very challenging.

Moreover, if it is possible to implement it, how will other aspects of such crucial importance at these nodes, such as battery life or operational simplicity, be affected?

## Semantic Web Services

Semantic Web Services (SWS) and recommendations such as OWL-S [Wor04b] are also a hot topic in coordination and orchestration of multiple processes, especially in the business domain. There have been several attempts to apply SWS to Ubiquitous Computing, such as Task Computing described in section 2.3, but they all require a central traditional computer in order to perform all the processing.

Embedding SWS in devices could enable workflows of carefully planned and coordinated activities in a serendipitous way among all participating devices. Service composition and synchronisation of activities are hot issues here.

But again, platform limitations need to be considered when exploring these opportunities. The use of Semantic Web Services adds an extra layer above reasoning, which makes it even more difficult to implement in limited devices.

### 7.2.2 Exploring and extending the SoaM model and architecture

Regarding the second approach, during our research we identified some possible evolutions of SoaM that can be explored in the near future:

- Investigate more deeply the possibilities and consequences resulting from using the management information, annotated via SoaMonto, for reactivity purposes. That is, using SoaM internal information as part of the context information. Is it possible to perform coordination or choreographic reactivity without any further extension, just using SoaMonto?
- Smobjects and orchestrators retrieve context information from other smobjects. Although they are not intended to provide large amounts of data, a query mechanism to retrieve only particular chunks of context information would reduce the network traffic. How can this scheme impact in smobjects' performance? Is it feasible to use SPARQL for querying context information in resource-limited smobjects? Is it possible to extend Plant and use it not only for discovery but also for context information retrieval?
- Semantic devices will be fully realised when they feature a full-powered reasoning engine. This statement brings back the previous discussion about the balance between computing power and resource-constrained devices. Is it possible to overcome these limitations?

- Interactions with wearable computing: is it possible for users to embody their profiles in wearable objects, such as watches, jewels or clothes, and disseminate them in order to achieve personalised environmental behaviour?
- Integration of RFID for profile or context information dissemination. Is it possible to substitute profile dissemination for “profile sensing”, storing the profiles in RFID tags?
- Extend the expressiveness of SoaM XML Datatypes to create richer and more powerful behavioural profiles, while still trying to keep the resolution process simple enough. At the same time, profile creation and editing should be very easy for users via friendly user-interfaces (end-user programming).
- Design and evaluate a security model as the one explained in subsection 7.1.1 in order to validate it and identify other implications.
- Automatic generation of behavioural profiles by analysing previous interactions. In this way, behaviours do not need to be manually created by humans, but devices can identify patterns by exploiting information from past interactions and anticipate the behaviour required by users.
- Dynamically adjustable polling period when retrieving context information: depending on available network bandwidth and surrounding smobjects performance.

### 7.3 Final remarks

At the beginning of our research we faced several ambitious challenges. We have already mentioned them in this chapter and they are represented in our research goals.

We have designed a model and an architecture for creating semantic devices, which are social electronic objects willing to collaborate with others in order to help users in their everyday activities.

From the results of our work, its future challenges, discussions with leading organisations, and the research agendas of technology platforms such as ARTEMIS and PROMETEO<sup>3</sup>, we foresee that new paths will be

---

<sup>3</sup>ARTEMIS is the European Technology Platform for Embedded Systems, while PROMETEO is its Spanish “mirror”.

opened during the next years to explore the implications of embedded intelligence, specially semantics, in Ambient Intelligence scenarios.

We expect the results of our research to lay a significant basis and an experience for designing semantic devices as first-class citizens of next-generation intelligent environments.



# Bibliography

- [Aar02] Emile Aarts. *Ambient Intelligence in HomeLab*. Philips Research, 2002.
- [ADB<sup>+</sup>99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.
- [App05] Apple Computer, Inc. *Bonjour: Connect computers and electronic devices automatically, without any configuration*, April 2005. Technology Brief.
- [Bec04] Dave Beckett. Swad-europe deliverable 3.11: Developer workshop report 4 - workshop on semantic web storage and retrieval. Technical report, World Wide Web Consortium, 2004.
- [BEP06] François Bry, Michael Eckert, and Paula-Lavinia Pătrânjan. Querying composite events for reactivity on the web. In *Proceedings of the International Workshop on XML Research and Applications (XRA) in conjunction with Asia-Pacific Web Conference*, volume 3842 of LNCS, 2006.
- [Ber06] Tim Berners-Lee. *Notation 3: An readable language for data on the Web*. World Wide Web Consortium, March 2006. Online at <http://www.w3.org/DesignIssues/Notation3.html>.
- [BHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284(5):28–37, May 2001.

- [BHS05] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. *Lecture Notes in Artificial Intelligence*, 2605:228–248, January 2005.
- [BK01] John Barton and Tim Kindberg. *The cooltown user experience*. Hewlett-Packard, 2001. Technical Report HPL-2001-22.
- [BKK<sup>+</sup>05] Chris Bussler, Edward Kilgarriff, Reto Krummenacher, Francisco Martin-Recuerda, Ioan Toma, and Brahmananda Sapkota. *WSMX Triple-Space Computing*. DERI - Digital Enterprise Research Institute, June 2005. WSMO Working Draft D21.v0.1.
- [BL99] Tim Berners-Lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Harper San Francisco, 1999.
- [BLFM98] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. *Uniform Resource Identifiers (URI): Generic Syntax*, 1998. IETF RFC 2396.
- [BMK<sup>+</sup>00] Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven A. Shafer. Easyliving: Technologies for intelligent environments. In *Proceedings of HUC '00: the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 12–29, London, UK, 2000. Springer-Verlag.
- [Bon05] Elena Paslaru Bontas. Using context information to improve ontology reuse. In *Proceedings of the Doctoral Workshop at the 17th Conference on Advanced Information Systems Engineering CAiSE'05*, 2005.
- [BPR99] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - a FIPA-compliant agent framework. In *Proceedings of the Practical Applications of Intelligent Agents*, 1999.
- [BTW01] Harold Boley, Said Tabet, and Gerd Wagner. Design rationale for ruleml: A markup language for semantic web rules. In *Proceedings of SWWS'01, The first Semantic Web Working Symposium*, pages 381–401, 2001.
- [Bus05] Christoph Bussler. A minimal triple space computing architecture. In *Proceedings of the 2nd WSMO Implementation Workshop*, 2005.
- [CAG00] J. Cohen, S. Aggarwal, and Y. Y. Goland. *General Event Notification Architecture Base: Client to Arbiter*, 2000. Internet Draft.

- [CF02] Harry Chen and Tim Finin. Beyond distributed ai, agent teamwork in ubiquitous computing. In *Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices, AAMAS-2002*, Bologna, Italy, July 2002.
- [CFJ01] Harry Chen, Tim Finin, and Anupam Joshi. Dynamic service discovery for mobile computing: Intelligent agents meet jini in the aether. *Baltzer Science Journal on Cluster Computing*, pages 343–354, February 2001.
- [CFJ03a] Harry Chen, Tim Finin, and Anupam Joshi. An intelligent broker for context-aware systems. In *Adjunct Proceedings of Ubicomp 2003*, pages 183–184. UbiComp, UbiComp, October 2003.
- [CFJ03b] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. In *Workshop on Ontologies and Distributed Systems. IJCAI-2003*, August 2003.
- [CFJ03c] Harry Chen, Tim Finin, and Anupam Joshi. Semantic web in a pervasive context-aware architecture. *Artificial Intelligence in Mobile System 2003*, pages 33–40, October 2003.
- [CFJ03d] Harry Chen, Tim Finin, and Anupam Joshi. Using owl in a pervasive computing broker. In *Workshop on Ontologies in Agent Systems, AAMAS-2003*, Melbourne, Australia, July 2003.
- [CFJ04a] Harry Chen, Tim Finin, and Anupam Joshi. A context broker for building smart meeting rooms. In Craig Schlenoff and Michael Uschold, editors, *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAI Spring Symposium*, pages 53–60, Stanford, California, March 2004. AAI, AAI Press, Menlo Park, CA.
- [CFJ04b] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review*, 18(3):197–207, May 2004.
- [CFJ04c] Harry Chen, Tim Finin, and Anupam Joshi. Semantic web in the context broker architecture. In *Proceedings of the Second Annual IEEE International Conference on Pervasive Computer and Communications*. IEEE Computer Society, March 2004.

- [CFJ<sup>+</sup>04d] Harry Chen, Tim Finin, Anupam Joshi, Filip Perich, Dipanjan Chakraborty, and Lalana Kagal. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 8(6), November 2004.
- [CFJ05] Harry Chen, Tim Finin, and Anupam Joshi. *Ontologies for Agents: Theory and Experiences*, chapter The SOUPA Ontology for Pervasive Computing. Whitestein Series in Software Agent Technologies. Springer, July 2005.
- [CGK05] Eleni Christopoulou, Christos Goumopoulos, and Achilles Kameas. An ontology-based context management and reasoning process for ubicomp applications. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and Ambient Intelligence*, pages 265–270, New York, NY, USA, 2005. ACM Press.
- [CGZK04] Eleni Christopoulou, Christos Goumopoulos, Ioannis Zaharakis, and Achilles Kameas. An ontology-based conceptual model for composing context-aware applications. In *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management colocated with Ubicomp 2004: the Sixth International Conference on Ubiquitous Computing*, 2004.
- [Che04] Harry Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, December 2004.
- [CK04] Eleni Christopoulou and Achilles Kameas. Using ontologies to address key issues in ubiquitous computing systems. In *EUSAI*, volume 3295 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2004.
- [CK05] Eleni Christopoulou and Achilles Kameas. Gas ontology: an ontology for collaboration among ubiquitous computing devices. *International Journal of Human-Computer Studies*, 62(5):664–685, 2005.
- [CK06a] Stuart Cheshire and Marc Krochmal. *DNS-Based Service Discovery*, August 2006. IETF Draft draft-cheshire-dnsext-dns-sd-04.txt.
- [CK06b] Stuart Cheshire and Marc Krochmal. *Multicast DNS*, August 2006. IETF Draft draft-cheshire-dnsext-multicastdns-06.txt.
- [CO05] Dave Crocker and Paul Overell. *Augmented BNF for Syntax Specifications: ABNF*, October 2005. IETF RFC 4234.

- [CPC<sup>+</sup>04] Harry Chen, Filip Perich, Dipanjan Chakraborty, Tim Finin, and Anupam Joshi. Intelligent agents meet semantic web in a smart meeting room. In *Proceedings of the Third International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS 2004)*, New York City, NY, July 2004.
- [CPFJ04] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, Boston, MA, August 2004.
- [CPJ<sup>+</sup>02] Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Tim Finin, and Yelena Yesha. A reactive service composition architecture for pervasive computing environments. Technical report, University of Maryland, Baltimore County, March 2002.
- [CTS<sup>+</sup>01] Harry Chen, Sovrin Tolia, Craig Sayers, Tim Finin, and Anupam Joshi. Creating context-aware software agents. In *First GSFC/JPL Workshop on Radical Agent Concepts*, Greenbelt, MD, USA., September 2001. NASA Goddard Space Flight Center.
- [DAM02] DAML Services Coalition. *DAML-S: Semantic Markup for Web Services*. DARPA, 2002. Whitepaper.
- [DAS99] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A context-based infrastructure for smart environments. In *Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99)*, pages 114–128, 1999.
- [DAS01] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:97–166, 2001.
- [Den02] Peter J. Denning, editor. *The invisible future: the seamless integration of technology into everyday life*. McGraw-Hill, Inc., New York, NY, USA, 2002.
- [Der99] Michael L. Dertouzos. The future of computing. *Scientific American*, August 1999.
- [Dey00] Anind K. Dey. Enabling the use of context in interactive applications. In *CHI '00: Extended abstracts on Human factors in computing systems*, pages 79–80, New York, NY, USA, 2000. ACM Press.

- [Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.
- [DR06] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol. Version 1.1*, April 2006. IETF RFC 4346.
- [DS05] Martin Duerst and Michel Suignard. *Internationalized Resource Identifiers (IRIs)*, 2005. IETF RFC 3987.
- [DZPJ05] Li Ding, Lina Zhou, Tim Finin, and Anupam Joshi. How the semantic web is being used: an analysis of foaf documents. In *Proceedings of the 38th International Conference on System Sciences*, January 2005.
- [Edw06] W. Keith Edwards. Discovery systems in ubiquitous computing. *IEEE Pervasive Computing*, 5(2):70–77, 2006.
- [Eur01] European Commission IST Advisory Group (ISTAG). Scenarios for ambient intelligence in 2010. Technical report, EU Commission, 2001.
- [Eur03a] European Commission IST Advisory Group (ISTAG). Ambient intelligence: from vision to reality. Technical report, EU Commission, 2003.
- [Eur03b] European Commission IST Advisory Group (ISTAG). IST research content. Technical report, EU Commission, 2003.
- [FDW<sup>+</sup>04] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall, and Stephen Pink. Supporting service discovery, querying and interaction in ubiquitous computing environments. *Wireless Networks*, 10(6):631 – 641, November 2004.
- [Fen04] Dieter Fensel. Triple-space computing: Semantic web services based on persistent publication of information. In Finn Arve Aagesen, Chutiporn Anutariya, and Vilas Wuwongse, editors, *Proceedings of the IFIP International Conference on Intelligence in Communication Systems*, volume 3283 of *Lecture Notes in Computer Science*, pages 43–53. Springer-Verlag, November 2004.
- [FGM<sup>+</sup>99] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, 1999. IETF RFC 2616.
- [FHBH<sup>+</sup>99] John Franks, Phillip M. Hallam-Baker, Jeffery L. Hostetler, Scott D. Lawrence, Paul J. Leach, Ari Luotonen, and Lawrence C. Stewart. *HTTP Authentication: Basic and Digest Access Authentication*, June 1999. IETF RFC 2617.

- [FHH03] Richard Fikes, Pat Hayes, and Ian Horrocks. *DAML Query Language (DQL) Abstract Specification*, 2003.
- [fIPA02] Foundation for Intelligent Physical Agents. *FIPA ACL Message Structure Specification*. Foundation for Intelligent Physical Agents, 2002.
- [FK05] Ned Freed and John C. Klensin. *Media Type Specifications and Registration Procedures*, December 2005. IETF RFC 4288.
- [FRP<sup>+</sup>05] Cristina Feier, Dumitru Roman, Axel Polleres, John Domingue, Michael Stollberg, and Dieter Fensel. Towards intelligent web services: Web service modeling ontology (wsmo). In *Proc. of the Int’al Conf on Intelligent Computing (ICIC) 2005*, 2005.
- [GBPK06] Bernardo Cuenca Grau, Evren Sirin Bijan Parsia, and Aditya Kalyanpur. Modularity and web ontologies. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR2006)*, 2006.
- [GC<sup>+</sup>99] Yaron Y. Goland, Ting Cai, et al. *Simple Service Discovery Protocol/1.0. Operating without an Arbiter*, 1999. Internet Draft.
- [GPVD99] Erik Guttman, Charles Perkins, John Veizades, and Michael Day. *Service Location Protocol, Version 2*, June 1999. IETF RFC 2608.
- [GPZ04a] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A bayesian approach for dealing with uncertain contexts. In G. Kotsis, editor, *Proceedings of the 2nd International Conference on Pervasive Computing*. Austrian Computer Society, 2004.
- [GPZ04b] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A middleware for building context-aware mobile services. In *Proceedings of IEEE Vehicular Technology Conference*, 2004.
- [GPZ04c] Tao Gu, Hung Keng Pung, and Da Qing Zhang. Toward an osgi-based infrastructure for context-aware applications. *IEEE Pervasive Computing*, 3(4):66–74, 2004.
- [GPZ05] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005.
- [Gre06] Adam Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing*. Peachpit Press, 2006.

- [GS00] Yaron Y. Golan and Jeffrey C. Schlimmer. *Multicast and Unicast UDP HTTP Messages*, 2000. Internet Draft.
- [GSSS02] David Garlan, Dan Siewiorek, Asim Smailagic, and Peter Steenkiste. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, 1(2):22–31, 2002.
- [GTPZ05a] Tao Gu, Edmond Tan, Hung Keng Pung, and Daqing Zhang. Contextpeers: Scalable peer-to-peer search for context information. In *Proceedings of the International Workshop on Innovations in Web Infrastructure (IWI 2005), in conjunction with the 14th World Wide Web Conference (WWW 2005)*, 2005.
- [GTPZ05b] Tao Gu, Edmond Tan, Hung Keng Pung, and Daqing Zhang. A peer-to-peer architecture for context lookup. In *2nd Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2005)*, pages 333–341. IEEE Computer Society, 2005.
- [GWPZ04] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang. An ontology-based context model in intelligent environments. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, 2004.
- [HM01] Volker Haarslev and Ralf Mller. Racer system description. In *IJCAR '01: Proceedings of the First International Joint Conference on Automated Reasoning*, pages 701–706, London, UK, 2001. Springer-Verlag.
- [HP04] Jerry R. Hobbs and Feng Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(1):66–85, 2004.
- [HPSB<sup>+</sup>04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosz, and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. World Wide Web Consortium, May 2004. W3C Member Submission.
- [HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *LPAR '99: Proceedings of the 6th International Conference on Logic Programming and Automated Reasoning*, pages 161–180, London, UK, 1999. Springer-Verlag.
- [IAN02] IANA. *Special-Use IPv4 Addresses*, September 2002. IETF RFC 3330.

- [Int04] International Standards Organization. *ISO 8601:2004: Data elements and interchange formats - Information interchange - Representation of dates and times*. International Standards Organization, 2004. ISO 8601:2004.
- [IST<sup>+</sup>05] Valerie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Françoise Sailhan, Rafik Chibout, Nicole Levy, and Angel Talamona. Developing ambient intelligence systems: A solution based on web services. *Automated Software Engineering*, 12(1):101–137, 2005.
- [JW03] Michael Jeronimo and Jack Weast. *UPnP design by example: a software developer's guide to Universal Plug and Play*. Intel Press, Hillsboro, Oregon, USA, 2003.
- [KB01] Tim Kindberg and John Barton. A web-based nomadic computing system. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 35(4):443–456, 2001.
- [KBM<sup>+</sup>02] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic. People, places, things: web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002.
- [KC03] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
- [KF02] Tim Kindberg and Armando Fox. System software for ubiquitous computing. *IEEE Pervasive Computing*, 1(1):70–81, 2002.
- [KFJ03a] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 63, Washington, DC, USA, 2003. IEEE Computer Society.
- [KFJ03b] Lalana Kagal, Timothy W. Finin, and Anupam Joshi. A policy based approach to security for the semantic web. In *Proceedings of ISWC 200 - The Second International Semantic Web Conference*, pages 402–418, 2003.
- [KKS05] Reto Krummenacher, Jacek Kopecky, and Thomas Strang. Sharing context information in semantic spaces. In *OTM*

- Workshops: Proceeding of the Workshop on Context-Aware Mobile Systems (CAMS'05)*, volume 3762 of *Lecture Notes in Computer Science*, pages 229–232. Springer, 2005.
- [KLF04] Deepali Khushraj, Ora Lassila, and Tim Finin. *stuples: Semantic tuple spaces*. In *Proceedings of Mobiquitous 2004: The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 268–277, 2004.
- [KMRMR06] Reto Krummenacher, Francisco J. Martin-Recuerda, Martin Murth, and Johannes Riemer. *TSC Framework*, July 2006. D1.2 v1.1 Final draft.
- [KQ04] David R. Karger and Dennis Quan. *Haystack: a user interface for creating, browsing, and organizing arbitrary semistructured information*. In *CHI '04: Extended abstracts on Human factors in computing systems*, pages 777–778, New York, NY, USA, 2004. ACM Press.
- [KS05] Reto Krummenacher and Thomas Strang. *Ubiquitous semantic spaces*. In *Conference Supplement to the 7th International Conference on Ubiquitous Computing (UbiComp 2005)*, 2005.
- [KSF06] Reto Krummenacher, Thomas Strang, and Dieter Fensel. *Triple Spaces for an Ubiquitous Web of Services*. DERI - Digital Enterprise Research Institute, 2006. Position Paper of the W3C Workshop on the Ubiquitous Web.
- [LA03] Ora Lassila and Mark Adler. *Semantic gadgets: Ubiquitous computing meets the semantic web*. In *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, pages 363–376, 2003.
- [Las06] Ora Lassila. *Semantic Web, Quo Vadis?*, October 2006. Keynote of the Ninth Scandinavian Conference on Artificial Intelligence (SCAI 2006).
- [Lau98] Simon St Laurent. *Cookies*. McGraw-Hill, Inc., New York, NY, USA, 1998.
- [LH02] Choonhwa Lee and Sumi Helal. *Protocols for service discovery in dynamic and mobile networks*. *International Journal of Computer Research*, 11(1):1–12, 2002.

- [LK01] Diego López de Ipiña and Eleftheria Katsir. An ECA rule-matching service for simpler development of reactive applications. In *Supplement to the Proceedings of Middleware 2001 at IEEE Distributed Systems Online, Vol. 2, No. 7*, 2001.
- [LMH02] Diego López de Ipiña, Paulo Mendonça, and Andy Hopper. TRIP: a Low-Cost Vision-Based Location System for Ubiquitous Computing. *Personal and Ubiquitous Computing Journal*, 6(3):206–219, May 2002.
- [LMS05] Paul J. Leach, Michael Mealling, and Rich Salz. *A Universally Unique Identifier (UUID) URN Namespace*, July 2005. IETF RFC 4122.
- [LVG<sup>+</sup>06] Diego López de Ipiña, Juan Ignacio Vázquez, Daniel García, Javier Fernández, Iván García, David Sainz, and Aitor Almeida. EMI<sup>2</sup>lets: A Reflective Framework for Enabling Aml. *Journal of Universal Computer Science*, 12(3):297–314, 2006.
- [MAA<sup>+</sup>04] Martin Modahl, Bikash Agarwalla, Gregory Abowd, Umakishore Ramachandran, and T. Scott Saponas. Toward a standard ubiquitous computing framework. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, pages 135–139, New York, NY, USA, 2004. ACM Press.
- [McG00] Robert E. McGrath. *Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing*. University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2000. Technical Report UIUCDCS-R-2000-2154.
- [McG05] Robert E. McGrath. *Semantic Infrastructure for a Ubiquitous Computing Environment*. PhD thesis, School of Computer Science, University of Illinois at Urbana-Champaign, 2005.
- [MK07] Daniel A. Menascé and Jeffrey O. Kephart. Guest editors' introduction: Autonomic computing. *IEEE Internet Computing*, 11(1):18–21, 2007.
- [MLPS03] Ryusuke Masuoka, Yannis Labrou, Bijan Parsia, and Evren Sirin. Ontology-enabled pervasive computing applications. *IEEE Intelligent Systems*, 18(5):68–72, September-October 2003.
- [MMS<sup>+</sup>05] Ryusuke Masuoka, Mohinder Chopra, Zhexuan Song, Yannis Labrou, Lalana Kagal, and Tim Finin. Policy-based access control for task computing using rei. In *Proceedings of the Policy*

- Management for the Web Workshop*, WWW 2005, pages 37–43. W3C, May 2005.
- [MPL03] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. Task computing - the semantic web meets pervasive computing. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.
- [MRCM03a] Robert E. McGrath, Anand Ranganathan, Roy H. Campbell, and M. Dennis Mickunas. Incorporating "semantic discovery" into ubiquitous computing infrastructure. In *Proceedings of System Support for Ubiquitous Computing Workshop at the Fifth Annual Conference on Ubiquitous Computing (UbiComp 2003)*, 2003.
- [MRCM03b] Robert E. McGrath, Anand Ranganathan, Roy H. Campbell, and M. Dennis Mickunas. Use of ontologies in pervasive computing environments. Technical Report Technical Report UIUCDCS-R-2003-2332, University of Illinois at Urbana-Champaign, April 2003.
- [MRMC03] Robert E. McGrath, Anand Ranganathan, M. Dennis Mickunas, and Roy H. Campbell. Investigations of semantic interoperability in ubiquitous computing environments. In *Proceedings of the 15th International Conference Parallel and Distributed Computing and Systems (PDCS)*, 2003.
- [NZM<sup>+</sup>05] Lionel M. Ni, Yanmin Zhu, Jian Ma, Minglu Li, Qiong Luo, Yunhao Liuand, Shing-Chi Cheung, and Qiang Yang. Semantic sensor net: an extensible framework. In *Proceedings of the Third International Conference on Computer Networks and Mobile Computing*, 2005.
- [OAS04] OASIS. *UDDI Version 3.0.2*. OASIS, 2004. UDDI Spec Technical Committee Draft.
- [O'R05] Tim O'Reilly. *What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software*, September 2005. <http://www.oreilynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>. Retrieved on January 2007.
- [PB04] Davy Preuveneers and Yolande Berbers. Suitability of existing service discovery protocols for mobile users in an ambient intelligence environment. In *Proceedings of the International Conference on Pervasive Computing and Communications*, pages 760–764. CSREA Press, June 2004.

- [PCB00] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43, New York, NY, USA, 2000. ACM Press.
- [PH05] Feng Pan and Jerry R. Hobbs. Temporal aggregates in owl-time. In *Proceedings of the Workshop on Natural Language-based Knowledge Representations: New Perspectives*, Clearwater Beach, FL, USA, May 2005. At the Florida Artificial Intelligence Research Society International Conference (FLAIRS 2005).
- [PJC05] Filip Perich, Anupam Joshi, and Rada Chirkova. *Enabling Technologies for Wireless e-Business Applications*, chapter Data Management for Mobile Ad-Hoc Networks. Springer, July 2005.
- [PMBT01] Nissanka B. Priyantha, Allen K.L. Miu, Hari Balakrishnan, and Seth Teller. The cricket compass for context-aware mobile applications. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 1–14, New York, NY, USA, 2001. ACM Press.
- [QK04] Dennis Quan and David R. Karger. How to make a semantic web browser. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 255–265, New York, NY, USA, 2004. ACM Press.
- [RAMC04] Anand Ranganathan, Jalal Al-Muhtadi, and Roy H. Campbell. Reasoning about uncertain contexts in pervasive computing environments. *IEEE Pervasive Computing*, 03(2):62–70, 2004.
- [Ran05] Anand Ranganathan. *A Task Execution Framework for Autonomic Ubiquitous Computing*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005.
- [RC00] Manuel Román and Roy H. Campbell. Gaia: enabling active spaces. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop*, pages 229–234, New York, NY, USA, 2000. ACM Press.
- [RC03a] Anand Ranganathan and Roy H. Campbell. An infrastructure for context-awareness based on first order logic. *Personal and Ubiquitous Computing*, 7(6):353–364, 2003.

- [RC03b] Anand Ranganathan and Roy H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference*, 2003.
- [RCAM<sup>+</sup>05] Anand Ranganathan, Shiva Chetan, Jalal Al-Muhtadi, Roy H. Campbell, and M. Dennis Mickunas. Olympus: A high-level programming model for pervasive computing environments. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, pages 7–16. IEEE Computer Society, 2005.
- [Res00] Eric Rescorla. *HTTP Over TLS*, May 2000. IETF RFC 2818.
- [Rey01] Franklin Reynolds. An RDF Framework for Resource Discovery. In *Proceedings of SemWeb 2001: The Second International Workshop on the Semantic Web*, May 2001.
- [RHC<sup>+</sup>02a] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):65–67, 2002.
- [RHC<sup>+</sup>02b] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.
- [RMCM03] Anand Ranganathan, Robert E. McGrath, Roy H. Campbell, and M. Dennis Mickunas. Ontologies in a pervasive computing environment. In *Eighteenth International Joint Conference On Artificial Intelligence (IJCAI'03), Workshop on Information Integration on the Web (IIWeb'03)*. Academic University Press, August 2003.
- [RSCI<sup>+</sup>02] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. *SIP: Session Initiation Protocol*, June 2002. IETF RFC 3261.
- [SBM00] Steven Shafer, Barry Brummit, and Brian Meyers. The easyliving intelligent environment system. In *CHI Workshop on Research Directions in Situated Computing*, April 2000.
- [SD02] Michael Sintek and Stefan Decker. Triple - a query, inference, and transformation language for the semantic web. In *ISWC '02: Proceedings of the First International Semantic Web*

- Conference on The Semantic Web*, pages 364–378, London, UK, 2002. Springer-Verlag.
- [SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM Press.
- [SG02] Joo Pedro Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of WICAS3: the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, pages 29–43, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.
- [SG03] Joo Sousa and David Garlan. *The Aura Software Architecture: an Infrastructure for Ubiquitous Computing*, 2003. Carnegie Mellon Technical Report, CMU-CS-03-183.
- [SHP03] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web Services: Modeling, Architecture and Infrastructure workshop in ICEIS 2003*, Angers, France, April 2003.
- [SKB06] Brahmananda Sapkota, Edward Kilgarriff, and Christoph Bussler. Role of triple space computing in semantic web services. In *Frontiers of WWW Research and Development - APWeb 2006, 8th Asia-Pacific Web Conference, Harbin*, volume 3841 of *Lecture Notes in Computer Science*, pages 714–719. Springer, 2006.
- [SLM04] Zhexuan Song, Yannis Labrou, and Ryusuke Masuoka. Dynamic service discovery and management in task computing. In *First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04)*, pages 310–318, 2004.
- [SLP04] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, 2004.
- [SLPF03a] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Applications of a context ontology language.

- In *SoftCOM 2003: International Conference on Software, Telecommunications and Computer Networks*, 2003.
- [SLPF03b] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool: A context ontology language to enable contextual interoperability. In *Proceedings of DAIS 2003: 4th IFIP International Conference on Distributed Applications and Interoperable Systems*, volume LNCS 2893. Springer, 2003.
- [SLPF03c] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Integration issues of an ontology based context modelling approach. In *Proceedings of ICWI 2003: IADIS International Conference WWW/Internet*, 2003.
- [SMAL04] Zhexuan Song, Ryusuke Masuoka, Jonathan Agre, and Yannis Labrou. Task computing for ubiquitous multimedia services. In *MUM '04: Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia*, pages 257–262, New York, NY, USA, 2004. ACM Press.
- [SRC05] Chetan Shiva Shankar, Anand Ranganathan, and Roy Campbell. An eca-p policy-based framework for managing ubiquitous computing environments. In *MOBIQUITOUS '05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 33–44, Washington, DC, USA, 2005. IEEE Computer Society.
- [SRL98] Henning Schulzrinne, Anup Rao, and Robert Lanphier. *Real Time Streaming Protocol (RTSP)*, April 1998. IETF RFC 2326.
- [STK<sup>+</sup>06] Omair Shafiq, Ioan Toma, Reto Krummenacher, Thomas Strang, and Dieter Fensel. Using triple space computing for communication and coordination in semantic grid. In *Proceedings of the 3rd Semantic Grid Workshop colocated with the 16th Global Grid Forum*, 2006.
- [Sun99] Sun Microsystems. Jini architectural overview. Technical report, Sun Microsystems, 1999.
- [TZWC05] Joo Geok Tan, Daqing Zhang, Xiaohang Wang, and Heng Seng Cheng. Enhancing semantic spaces with event-driven context interpretation. In *Proceedings of Pervasive 2005: Third International Conference on Pervasive Computing*, volume 3468 of *Lecture Notes in Computer Science*, pages 80–97. Springer, 2005.

- [UPnP03] UPnP Forum. *UPnP Device Architecture v.1.0.1 Draft*, 2003.
- [VAL06] Juan Ignacio Vazquez, Joseba Abaitua, and Diego López de Ipiña. The Ubiquitous Web as a model to lead our environments to their full potential. In *Proceedings of the W3C Workshop on the Ubiquitous Web*. World Wide Web Consortium, 2006. Position paper.
- [vBYM05] Albert van Breemen, Xue Yan, and Bernt Meerbeek. icat: an animated user-interface robot with personality. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 143–144, New York, NY, USA, 2005. ACM Press.
- [VL04] Juan Ignacio Vazquez and Diego López de Ipiña. An interaction model for passively influencing the environment. In *Adjunct Proceedings of EUSAI 2004: the 2nd European Symposium on Ambient Intelligence*, 2004.
- [VL05] Juan Ignacio Vazquez and Diego López de Ipiña. A language for expressing user-context preferences in the Web. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 904–905, New York, NY, USA, 2005. ACM Press.
- [VLnS06] Juan Ignacio Vazquez, Diego López de Ipiña, and Iñigo Sedano. SoaM: An environment adaptation model for the Pervasive Semantic Web. In *Proceedings of the 2nd Ubiquitous Web Systems and Intelligence Workshop (UWSI 2006), collocated with ICCSA 2006. Lecture Notes in Computer Science - LNCS*, volume 3983, pages 108–117, May 2006.
- [WDC<sup>+</sup>04] Xiaohang Wang, Jin Song Dong, Chung Yau Chin, Sanka Ravipriya Hettiarachchi, and Daqing Zhang. Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, 3(3):32–39, 2004.
- [Wei96] Mark Weiser. *Computer Science Challenges for the Next Ten Years*. Xerox PARC, 1996. Presentation slides.
- [Wei99] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mobile Computing and Communications Review*, 3(3):3–11, 1999.
- [WJ99] Michael Wooldridge and Nicholas R. Jennings. The cooperative problem-solving process. *Journal of Logic and Computation*, 9(4):563–592, 1999.

- [Wor01a] World Wide Web Consortium. *Web Services Description Language (WSDL) 1.1*. World Wide Web Consortium, 2001. W3C Note.
- [Wor01b] World Wide Web Consortium. *XML Base*. World Wide Web Consortium, June 2001. W3C Recommendation.
- [Wor03] World Wide Web Consortium. *SOAP Version 1.2 Part 0: Primer*. World Wide Web Consortium, 2003. W3C Recommendation.
- [Wor04a] World Wide Web Consortium. *Architecture of the World Wide Web, Volume One*. World Wide Web Consortium, December 2004. W3C Recommendation.
- [Wor04b] World Wide Web Consortium. *OWL-S: Semantic Markup for Web Services*. World Wide Web Consortium, 2004. W3C Member Submission.
- [Wor04c] World Wide Web Consortium. *OWL Web Ontology Language Guide*. World Wide Web Consortium, February 2004. W3C Recommendation.
- [Wor04d] World Wide Web Consortium. *OWL Web Ontology Language Overview*. World Wide Web Consortium, February 2004. W3C Recommendation.
- [Wor04e] World Wide Web Consortium. *OWL Web Ontology Language Reference*. World Wide Web Consortium, February 2004. W3C Recommendation.
- [Wor04f] World Wide Web Consortium. *RDF Primer*. World Wide Web Consortium, February 2004. W3C Recommendation.
- [Wor04g] World Wide Web Consortium. *RDF Test Cases*. World Wide Web Consortium, February 2004. W3C Recommendation.
- [Wor04h] World Wide Web Consortium. *RDF Vocabulary Description Language 1.0: RDF Schema*. World Wide Web Consortium, February 2004. W3C Recommendation.
- [Wor04i] World Wide Web Consortium. *RDF/XML Syntax Specification (Revised)*. World Wide Web Consortium, February 2004. W3C Recommendation.
- [Wor04j] World Wide Web Consortium. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. World Wide Web Consortium, February 2004. W3C Recommendation.

- [Wor04k] World Wide Web Consortium. *XML Schema Part 2: Datatypes Second Edition*. World Wide Web Consortium, October 2004. W3C Recommendation.
- [Wor05a] World Wide Web Consortium. *Web Services Choreography Description Language Version 1.0*. World Wide Web Consortium, November 2005. W3C Candidate Recommendation.
- [Wor05b] World Wide Web Consortium. *Rule Interchange Format Working Group Charter*. World Wide Web Consortium, November 2005.
- [Wor05c] World Wide Web Consortium. *xml:id Version 1.0*. World Wide Web Consortium, September 2005. W3C Recommendation.
- [Wor06a] World Wide Web Consortium. *SPARQL Protocol for RDF*. World Wide Web Consortium, April 2006. W3C Candidate Recommendation.
- [Wor06b] World Wide Web Consortium. *SPARQL Query Language for RDF*. World Wide Web Consortium, April 2006. W3C Candidate Recommendation.
- [Wor06c] World Wide Web Consortium. *Time Ontology in OWL*. World Wide Web Consortium, September 2006. W3C Working Draft.
- [WZGP04] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *Proceedings of PERCOMW '04: Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, page 18, Washington, DC, USA, 2004. IEEE Computer Society.
- [ZMN05] Fen Zhu, Matt W. Mutka, and Lionel M. Ni. Service discovery in pervasive computing environments. *IEEE Pervasive Computing*, 04(4):81–90, 2005.
- [ZS05] Weibin Zhao and Henning Schulzrinne. Enhancing service location protocol for efficiency, scalability and advanced discovery. *Journal of Systems and Software*, 75(1-2):193–204, February 2005.



# Appendices



# Basic Semantic Web technologies

## A.1 Resource Description Framework

In the Semantic Web, RDF (Resource Description Framework) [Wor04j] [Wor04i] [Wor04h] is the mechanism to model the knowledge and relationships space: “*The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web*” [Wor04f].

RDF represents knowledge as humans do, using statements with subject, predicate and object. URIs are used to identify subjects and predicates, while objects can be represented via URIs, if they refer to other resources, or literal values.

**Example A.1.** Iñaki is currently staying in Bilbao (a town in the Basque Country), and his email address is ivazquez@eside.deusto.es. Iñaki’s car is a S24 of 1978, whose plate number is 886579. Currently the car is in the garage.

Table A.1 contains the URIs used to identify the resources, including the verbs. Thus, all the knowledge of Example A.1 can be represented in a set of triples (subject predicate object), using these URIs.

For example, “Iñaki is currently staying in Bilbao” can be represented as:

- **Subject:** <http://paginaspersonales.deusto.es/ivazquez>
- **Predicate:**  
<http://www.awareit.com/onto/2005/12/location#isLocatedIn>
- **Object:** <http://placesoftheworld.com/towns/Bilbao>

<b>Resource</b>	<b>URI</b>
<i>Iñaki (person)</i>	<a href="http://paginaspersonales.deusto.es/ivazquez">http://paginaspersonales.deusto.es/ivazquez</a>
<i>Has first name (predicate)</i>	<a href="http://xmlns.com/foaf/0.1/name">http://xmlns.com/foaf/0.1/name</a>
<i>Bilbao</i>	<a href="http://placesoftheworld.com/towns/Bilbao">http://placesoftheworld.com/towns/Bilbao</a>
<i>Located in (predicate)</i>	<a href="http://www.awareit.com/onto/2005/12/location#isLocatedIn">http://www.awareit.com/onto/2005/12/location#isLocatedIn</a>
<i>Basque Country</i>	<a href="http://placesoftheworld.com/regions/BasqueCountry">http://placesoftheworld.com/regions/BasqueCountry</a>
<i>Has Email (predicate)</i>	<a href="http://xmlns.com/foaf/0.1/mbox">http://xmlns.com/foaf/0.1/mbox</a>
<i>Iñaki's car</i>	<a href="urn:uuid:e20b8230-e9ad-11da-8ad9-0800200c9a66">urn:uuid:e20b8230-e9ad-11da-8ad9-0800200c9a66</a>
<i>Has owner (predicate)</i>	<a href="http://www.awareit.com/onto/2005/12/car#owner">http://www.awareit.com/onto/2005/12/car#owner</a>
<i>Has model (predicate)</i>	<a href="http://www.awareit.com/onto/2005/12/car#model">http://www.awareit.com/onto/2005/12/car#model</a>
<i>Has plate (predicate)</i>	<a href="http://www.awareit.com/onto/2005/12/car#plate">http://www.awareit.com/onto/2005/12/car#plate</a>
<i>Purchased in (predicate)</i>	<a href="http://www.awareit.com/onto/2005/12/car#purchasedIn">http://www.awareit.com/onto/2005/12/car#purchasedIn</a>
<i>Iñaki's garage</i>	<a href="urn:uuid:eff57010-e9ab-11da-8ad9-0800200c9a66">urn:uuid:eff57010-e9ab-11da-8ad9-0800200c9a66</a>

Table A.1: URIs assigned to identify the resources in Example A.1.

“The owner of the car is Iñaki”, as:

- **Subject:** <urn:uuid:e20b8230-e9ad-11da-8ad9-0800200c9a66>
- **Predicate:** <http://www.awareit.com/onto/2005/12/car#owner>
- **Object:** <http://paginaspersonales.deusto.es/ivazquez>

All the information can be visualised and navigated in a graph as illustrated in Figure A.1. Standardised notation shapes resources as ellipses, literal values as rectangles, while predicates are shaped as edges connecting both. Resources and edges are identified by URIs. For the sake of clarity, we have used the prefixes `foaf:` for <http://xmlns.com/foaf/0.1/>, `car:` for <http://www.awareit.com/onto/2005/12/car#> and `location:` as a short for <http://www.awareit.com/onto/2005/12/location#>.

RDF can be serialised over XML in what is known as RDF/XML [Wor04i]. Listing A.1 illustrates the RDF/XML serialisation of Example A.1.

Listing A.1: RDF/XML serialisation of Example A.1.

```

1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3   xmlns:dc="http://purl.org/dc/elements/1.1/"
4   xmlns:foaf="http://xmlns.com/foaf/0.1/"
5   xmlns:location="http://www.awareit.com/onto/2005/12/location#"
6   xmlns:car="http://www.awareit.com/onto/2005/12/car#">
7   <rdf:Description
8     rdf:about="http://paginaspersonales.deusto.es/ivazquez">
9     <foaf:name>Inaki</foaf:name>
10    <foaf:mbox>ivazquez@eside.deusto.es</foaf:mbox>
11    <location:isLocatedIn>

```

```

12 <rdf:Description
13     rdf:about="http://placesoftheworld.com/towns/Bilbao">
14     <location:isLocatedIn rdf:resource=
15         "http://placesoftheworld.com/regions/BasqueCountry"/>
16 </rdf:Description>
17 </location:isLocatedIn>
18 </rdf:Description>
19 <rdf:Description
20     rdf:about="urn:uuid:e20b8230-e9ad-11da-8ad9-0800200c9a66">
21     <car:owner
22         rdf:resource="http://paginaspersonales.deusto.es/ivazquez"/>
23     <car:model>S24</car:model>
24     <car:plate>886579</car:plate>
25     <car:purchasedIn>886579</car:purchasedIn>
26     <location:isLocatedIn
27         rdf:resource="urn:uuid:eff57010-e9ab-11da-8ad9-0800200c9a66"/>
28 </rdf:Description>
29 </rdf:RDF>

```

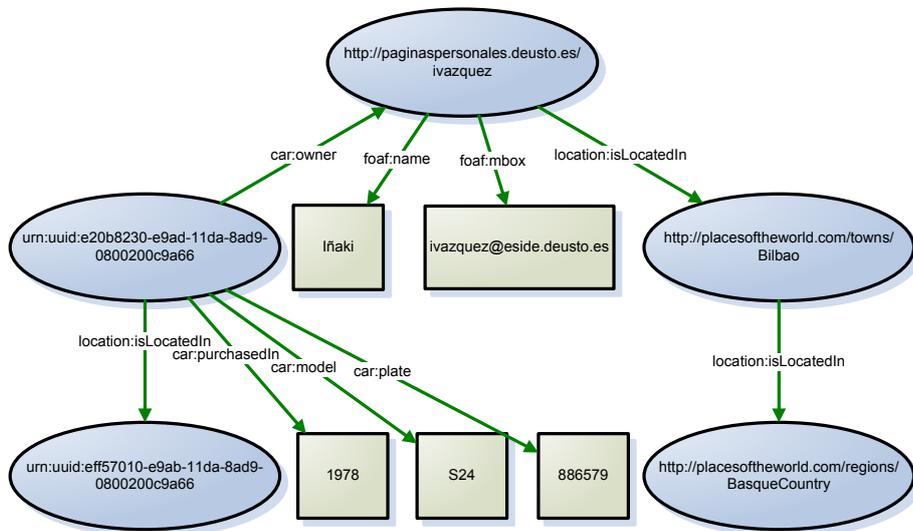


Figure A.1: RDF graph representing the Example A.1.

RDF graphs are navigable knowledge models where multiple vocabularies can be applied to represent concepts from disperse knowledge domains as in the previous example.

In order to help defining those vocabularies, a meta-vocabulary called RDF Schema [Wor04h] has been created. RDF Schema includes concepts such as `rdfs:Resource`, `rdfs:Class`, `rdfs:Literal` or `rdf:Property`, as well as predicates such as `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:range` or `rdfs:domain`.

RDF Schema provides a basic level of reasoning about categorisation. For example, considering the RDF triples:

```
person:ivazquez rdf:type job:Lecturer
job:Lecturer rdfs:subClassOf job:AcademicStaff
```

A RDF Schema enabled reasoner could add a new triple to the knowledge base:

```
person:ivazquez rdf:type job:AcademicStaff
```

However, RDF Schema still provides a limited reasoning capability.

## A.2 Ontologies

OWL (Ontology Web Language) [Wor04d] [Wor04c] [Wor04e] provides an additional vocabulary to represent meaning and semantics about resources and relationships, as well as a formal semantics based on description logics [BHS05].

OWL provides a framework for creating ontologies on different knowledge domains. An ontology is a “*a document or file that formally defines the relations among terms*” [BHL01]. Ontologies generally describe:

- Individuals: concrete instances of entities.
- Classes: groups of instances with similar features, similar type.
- Attributes: properties of entities.
- Relations: how entities are related.

OWL comes in three flavours, OWL Lite, OWL DL and OWL Full, depending on the expressiveness and reasoning capabilities required. OWL Full embraces OWL DL, which in turn embraces OWL Lite.

Some of the constructions provided by OWL are: `owl:sameAs`, `owl:inverseOf`, `owl:TransitiveProperty`, `owl:SymmetricProperty`, `owl:equivalentClass` or `owl:equivalentProperty`.

For example, considering the RDF triples:

```
person:ivazquez location:isLocatedIn places:room21
places:room21 location:isLocatedIn places:Bilbao
location:isLocatedIn rdf:type owl:TransitiveProperty
```

An OWL enabled reasoner could add a new triple to the knowledge base:

```
person:ivazquez location:isLocatedIn places:Bilbao
```

Since `location:locationIn` has been defined as a transitive property.

OWL is just one of the possible reasoning mechanisms (based on description logics in this case) the Semantic Web can be augmented with in order to increase intelligence. SWRL (Semantic Web Rules Languages) [HPSB<sup>+</sup>04] is other mechanism (based on first order logics) that is currently under development to create domain-specific rules that can embody heuristic knowledge.

Figure A.2 illustrates the different layers in the Semantic Web, whose ultimate goal is to achieve the Web of Trust.

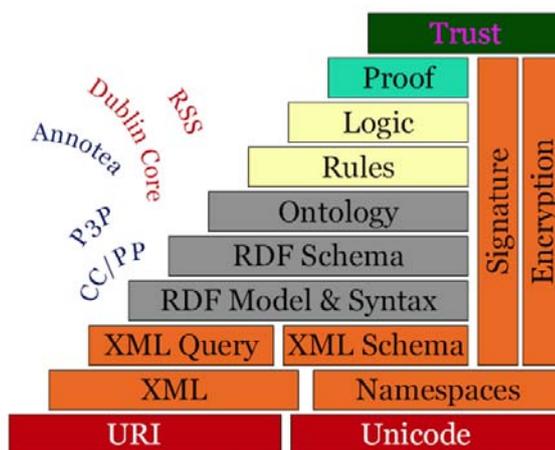


Figure A.2: The Semantic Web stack. Source: World Wide Web Consortium.

### A.3 SPARQL Protocol And RDF Query Language

SPARQL (SPARQL Protocol And RDF Query Language) [Wor06b] [Wor06a] is a SQL-like language for querying RDF graphs about resources, values or relations.

For instance, given the RDF graph of Figure A.1 a SPARQL query to retrieve the email of `http://paginaspersonales.deusto.es/ivazquez` could be:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?email
WHERE {
  <http://paginaspersonales.deusto.es/ivazquez> foaf:mbox ?email
}
```

An more complex SPARQL query to retrieve the name and current location of the owner whose car's plate is "886579":

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX car: <http://www.awareit.com/onto/2005/12/car#>
PREFIX location: <http://www.awareit.com/onto/2005/12/location#>
SELECT ?name ?location
WHERE {
    ?car car:plate "886579" .
    ?car car:owner ?owner .
    ?owner foaf:name ?name .
    ?owner location:isLocatedIn ?location .
}
```

Although the value of `?location` is initially "Bilbao", if an OWL ontology is applied and `location:isLocatedIn` is declared as a `owl:TransitiveProperty`, the SPARQL processor would have properly yielded two values for `?location`, "Bilbao" and "Basque Country".

SPARQL and OWL is a powerful combination for querying information, being able to produce results by reasoning over existing knowledge by applying ontologies.

There are four query constructions in SPARQL [Wor06b]:

- **SELECT**: performs a query by resolving concrete variables.
- **CONSTRUCT**: returns an RDF graph by performing substitutions on provided triple patterns.
- **DESCRIBE**: returns the RDF graph describing concrete resources that match a query.
- **ASK**: returns a boolean value about some provided conditions.

The SPARQL query language is complemented with the SPARQL Protocol: an HTTP interface for sending queries to RDF HTTP servers and retrieving results. The SPARQL Protocol has both bindings for HTTP and SOAP [Wor06a].

# SoaM Numbers, Ports and Namespaces

**mRDP IP Multicast Address**

224.0.24.1

**mRDP UDP Multicast Port**

2773

**Plant MIME Type**

application/com.awareit.plant

**ReDEL MIME Type**

application/com.awareit.redel+xml

**SoaM XML Datatypes MIME Type**

application/com.awareit.soamdt+xml

**SoaM XML Exchange Messages MIME Type**

application/com.awareit.soammsg+xml

**ReDEL XML Datatypes Namespace**

<http://www.awareit.com/soam/2006/04/redel>

**ReDEL Web Service Namespace**

<http://www.awareit.com/soam/2006/04/redelws>

**sRDF Web Service Namespace**

<http://www.awareit.com/soam/2006/04/srdfws>

**sRDF Web Service HTTP GET URI**

<http://www.awareit.com/soam/2006/04/srdfws#httpGet>

**SoaM XML Datatypes Namespace**

<http://www.awareit.com/soam/2006/02/soamdt>

**SoaM XML Exchange Messages Namespace**

<http://www.awareit.com/soam/2006/02/soammsg>

**SoaM Entities Management API – HTTP Binding WSDL Namespace**

<http://www.awareit.com/soam/2006/02/soamws>

**SoaMonto Namespace**

<http://www.awareit.com/soam/2005/12/soamonto>

# ReDEL: Resource Description Endpoints Language

## C.1 ReDEL XML Schema

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2
3 <!-- ReDEL XML Schema -->
4 <!-- Juan Ignacio Vazquez -->
5 <!-- Version 20060429.1 -->
6
7 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:redel="
  http://www.awareit.com/soam/2006/04/redel" targetNamespace="http
  ://www.awareit.com/soam/2006/04/redel" elementFormDefault="
  qualified">
8
9   <xs:element name="location">
10     <xs:complexType>
11       <xs:attribute name="url" type="xs:anyURI" use="required"/>
12       <xs:attribute name="ifaceBinding" type="xs:anyURI" use="
        optional" default="http://www.awareit.com/soam
        /2006/04/srdfws#httpGet"/>
13     </xs:complexType>
14   </xs:element>
15
16   <xs:element name="resource">
17     <xs:complexType>
18       <xs:sequence>
19         <xs:element ref="redel:location" minOccurs="0"
          maxOccurs="unbounded"/>
20       </xs:sequence>
```

```
21     <xs:attribute name="uri" type="xs:anyURI" use="required"/>
22   </xs:complexType>
23 </xs:element>
24
25 <xs:element name="redel">
26   <xs:complexType>
27     <xs:sequence>
28       <xs:element ref="redel:resource" maxOccurs="unbounded
29         "/>
30     </xs:sequence>
31   </xs:complexType>
32 </xs:element>
33 </xs:schema>
```

### C.2 ReDEL Web Service

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- ReDEL Web Service -->
4 <!-- Juan Ignacio Vazquez -->
5 <!-- Version 20060429.1 -->
6
7 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http
8   ://schemas.xmlsoap.org/wsdl/soap/" xmlns:http="http://schemas.
9   xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/
10  wsdl/mime/" xmlns:redelws="http://www.awareit.com/soam/2006/04/
11  redelws" xmlns:redel="http://www.awareit.com/soam/2006/04/redel"
12  targetNamespace="http://www.awareit.com/soam/2006/04/redelws">
13   <import namespace="http://www.awareit.com/soam/2006/04/redel"
14     location="http://www.awareit.com/soam/2006/04/redel.xsd"/>
15   <message name="redelContent">
16     <part name="content" element="redel:redel"/>
17   </message>
18   <portType name="redelSoapPort">
19     <operation name="SoapPostOp">
20       <input message="redelws:redelContent"/>
21     </operation>
22   </portType>
23   <portType name="redelHttpPostPort">
24     <operation name="HttpPostOp">
25       <input message="redelws:redelContent"/>
26     </operation>
27   </portType>
28   <binding name="HttpPost" type="redelws:redelHttpPostPort">
29     <http:binding verb="POST"/>
30     <operation name="HttpPostOp">
```

```

25         <http:operation location="" />
26         <input>
27             <mime:mimeXml part="content" />
28         </input>
29     </operation>
30 </binding>
31 <binding name="soapPost" type="redelws:redelSoapPort">
32     <soap:binding style="document" transport="http://schemas.
33         xmlsoap.org/soap/http" />
34 <operation name="SoapPostOp">
35     <soap:operation soapAction="#post" style="document" />
36     <input>
37         <soap:body use="literal" />
38     </input>
39 </operation>
40 </binding>
41 <!-- Example services declaration for IP address 169.254.0.3 -->
42 <service name="redelService">
43     <port name="redelServiceSoap" binding="redelws:soapPost">
44         <soap:address location="http://169.254.0.3/mrdpcallback" />
45     </port>
46     <port name="redelServiceHttpPost" binding="redelws:httpPost">
47         <http:address location="http://169.254.0.3/mrdpcallback" />
48     </port>
49 </service>
50 </definitions>

```

### C.3 Simple RDF Web Service

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- Simple RDF Web Service -->
4 <!-- Juan Ignacio Vazquez -->
5 <!-- Version 20060429.1 -->
6
7 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http
8     ://schemas.xmlsoap.org/wsdl/soap/" xmlns:http="http://schemas.
9     xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/
10    wsdl/mime/" xmlns:srdfws="http://www.awareit.com/soam/2006/04/
11    srfws" targetNamespace="http://www.awareit.com/soam/2006/04/
12    srfws">
13     <message name="empty" />
14     <message name="rdfContent">
15         <part name="content" />
16     </message>
17     <portType name="rdfHttpGetService">
18         <operation name="HttpGet">

```

```
14         <input message="srdfws:empty"/>
15         <output message="srdfws:rdfContent"/>
16     </operation>
17 </portType>
18 <binding name="httpGet" type="srdfws:rdfHttpGetService">
19     <http:binding verb="GET"/>
20     <operation name="httpGet">
21         <http:operation location=""/>
22         <input/>
23         <output>
24             <mime:mimeXml part="content"/>
25         </output>
26     </operation>
27 </binding>
28 <!-- Example services declaration for IP address 169.254.0.3 -->
29 <service name="srdfService">
30     <port name="srdfGetService" binding="srdfws:httpGet">
31         <http:address location="http://169.254.0.3/information"/>
32     </port>
33 </service>
34 </definitions>
```

# SoaM XML Datatypes and Exchange Messages

## D.1 SoaM XML Datatypes

```
1 <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2
3 <!-- SOAM XML Datatypes XML Schema -->
4 <!-- Juan Ignacio Vazquez -->
5 <!-- Version 20060211.1 -->
6
7 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soamdt="
  http://www.awareit.com/soam/2006/02/soamdt" targetNamespace="http
  ://www.awareit.com/soam/2006/02/soamdt" elementFormDefault="
  qualified" attributeFormDefault="unqualified">
8
9   <xs:import namespace="http://www.w3.org/XML/1998/namespace"
     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
10
11   <xs:complexType name="statementBaseType">
12     <xs:annotation>
13       <xs:documentation>Utility type</xs:documentation>
14     </xs:annotation>
15     <xs:attribute name="subject" type="xs:anyURI" use="required"/>
16     <xs:attribute name="predicate" type="xs:anyURI" use="required
17       "/>
18     <xs:attribute name="operator" type="xs:anyURI" use="optional
19       "/>
20     <xs:attribute name="id" type="xs:anyURI" use="required"/>
  </xs:complexType>
```

```

21 <xs:element name="constraint">
22   <xs:complexType>
23     <xs:complexContent>
24       <xs:extension base="soamdt:statementBaseType">
25         <xs:choice>
26           <xs:element ref="soamdt:objectLiteral"/>
27           <xs:element ref="soamdt:objectResource"/>
28         </xs:choice>
29         <xs:attribute name="requester" type="xs:anyURI"
30           use="required"/>
31         <xs:attribute name="expires" type="xs:duration"
32           use="optional"/>
33       </xs:extension>
34     </xs:complexContent>
35   </xs:complexType>
36 </xs:element>
37
38 <xs:element name="behavioralProfile">
39   <xs:complexType>
40     <xs:sequence>
41       <xs:element ref="soamdt:variable" minOccurs="0"
42         maxOccurs="unbounded"/>
43       <xs:element ref="soamdt:precondition" minOccurs="0"
44         maxOccurs="unbounded"/>
45       <xs:element ref="soamdt:postcondition" maxOccurs="
46         unbounded"/>
47     </xs:sequence>
48     <xs:attribute name="id" type="xs:anyURI" use="required"/>
49     <xs:attribute name="expires" type="xs:duration" use="
50       optional"/>
51     <xs:attribute name="requester" type="xs:anyURI" use="
52       required"/>
53   </xs:complexType>
54 </xs:element>
55
56 <xs:element name="variable">
57   <xs:complexType>
58     <xs:annotation>
59       <xs:documentation>optional?</xs:documentation>
60     </xs:annotation>
61     <xs:attribute ref="xml:id" use="required"/>
62   </xs:complexType>
63 </xs:element>
64
65 <xs:element name="precondition" type="soamdt:statementPatternType
66   "/>
67
68 <xs:element name="postcondition">
69   <xs:complexType>

```

```

62         <xs:complexContent>
63             <xs:extension base="soamdt:statementPatternType">
64                 <xs:attribute name="optional" type="xs:boolean"
65                     use="optional" default="false"/>
66             </xs:extension>
67         </xs:complexContent>
68     </xs:complexType>
69 </xs:element>
70
71 <xs:element name="objectLiteral">
72     <xs:complexType>
73         <xs:simpleContent>
74             <xs:extension base="xs:string">
75                 <xs:attribute name="datatype" type="xs:anyURI" use="optional"/>
76             </xs:extension>
77         </xs:simpleContent>
78     </xs:complexType>
79 </xs:element>
80
81 <xs:element name="objectResource">
82     <xs:complexType>
83         <xs:attribute name="resource" type="xs:anyURI" use="required"/>
84     </xs:complexType>
85 </xs:element>
86
87 <xs:element name="objectVariable">
88     <xs:complexType>
89         <xs:attribute name="ref" type="xs:IDREF" use="required"/>
90     </xs:complexType>
91 </xs:element>
92
93 <xs:complexType name="statementPatternType">
94     <xs:complexContent>
95         <xs:extension base="soamdt:statementBaseType">
96             <xs:choice>
97                 <xs:element ref="soamdt:objectLiteral"/>
98                 <xs:element ref="soamdt:objectResource"/>
99                 <xs:element ref="soamdt:objectVariable"/>
100             </xs:choice>
101         </xs:extension>
102     </xs:complexContent>
103 </xs:complexType>
104
105 <xs:complexType name="capabilityType">
106     <xs:sequence>
107         <xs:element ref="soamdt:subject" minOccurs="0" maxOccurs="unbounded"/>

```

```

107     <xs:element ref="soamdt:ontology" minOccurs="0" maxOccurs
        ="unbounded"/>
108     <xs:element ref="soamdt:predicate" minOccurs="0" maxOccurs
        ="unbounded"/>
109     <xs:element ref="soamdt:objectResource" minOccurs="0"
        maxOccurs="unbounded"/>
110     <xs:element ref="soamdt:objectLiteral" minOccurs="0"
        maxOccurs="unbounded"/>
111     </xs:sequence>
112     <xs:attribute name="id" type="xs:anyURI" use="required"/>
113 </xs:complexType>
114
115 <xs:element name="subject">
116     <xs:complexType>
117         <xs:attribute name="resource" type="xs:anyURI" use="
            required"/>
118     </xs:complexType>
119 </xs:element>
120
121 <xs:element name="ontology">
122     <xs:complexType>
123         <xs:attribute name="resource" type="xs:anyURI" use="
            required"/>
124     </xs:complexType>
125 </xs:element>
126
127 <xs:element name="predicate" nillable="false">
128     <xs:complexType>
129         <xs:attribute name="resource" type="xs:anyURI" use="
            required"/>
130     </xs:complexType>
131 </xs:element>
132
133 <xs:element name="capabilitiesCollection">
134     <xs:complexType>
135         <xs:choice maxOccurs="unbounded">
136             <xs:element ref="soamdt:perceptionCapability"/>
137             <xs:element ref="soamdt:operationCapability"/>
138         </xs:choice>
139         <xs:attribute name="owner" type="xs:anyURI" use="optional
            "/>
140     </xs:complexType>
141 </xs:element>
142
143 <xs:element name="perceptionCapability">
144     <xs:complexType>
145         <xs:complexContent>
146             <xs:extension base="soamdt:capabilityType"/>
147         </xs:complexContent>

```

```

148     </xs:complexType>
149 </xs:element>
150
151 <xs:element name="operationCapability">
152   <xs:complexType>
153     <xs:complexContent>
154       <xs:extension base="soamdt:capabilityType"/>
155     </xs:complexContent>
156   </xs:complexType>
157 </xs:element>
158
159 <xs:element name="constraintsCollection">
160   <xs:complexType>
161     <xs:sequence>
162       <xs:element ref="soamdt:constraint" maxOccurs="
163         unbounded"/>
164     </xs:sequence>
165     <xs:attribute name="owner" type="xs:anyURI" use="optional
166       "/>
167   </xs:complexType>
168 </xs:element>
169
170 <xs:element name="behavioralProfilesCollection">
171   <xs:complexType>
172     <xs:sequence>
173       <xs:element ref="soamdt:behavioralProfile" maxOccurs="
174         unbounded"/>
175     </xs:sequence>
176     <xs:attribute name="owner" type="xs:anyURI" use="optional
177       "/>
178   </xs:complexType>
179 </xs:element>
180 </xs:schema>

```

## D.2 SoaM XML Exchange Messages

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- SoaM XML Exchange Messages -->
4 <!-- Juan Ignacio Vazquez -->
5 <!-- Version 20060211.1 -->
6
7 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soamdt="
  http://www.awareit.com/soam/2006/02/soamdt" xmlns:soammmsg="http://
  www.awareit.com/soam/2006/02/soammmsg" targetNamespace="http://www.

```

```
awareit.com/soam/2006/02/soammsg" elementFormDefault="qualified"
attributeFormDefault="unqualified">
8 <xs:import namespace="http://www.awareit.com/soam/2006/02/soamdt"
  schemaLocation="http://www.awareit.com/soam/2006/02/soamdt.xsd
  "/>
9 <xs:element name="soamResult">
10   <xs:complexType>
11     <xs:simpleContent>
12       <xs:extension base="xs:string">
13         <xs:attribute name="code" type="xs:int" use="
          required"/>
14         <xs:attribute name="expires" type="xs:duration"
          use="optional"/>
15         <xs:attribute name="ref" type="xs:anyURI" use="
          required"/>
16       </xs:extension>
17     </xs:simpleContent>
18   </xs:complexType>
19 </xs:element>
20 <xs:element name="soamEntityId">
21   <xs:complexType>
22     <xs:attribute name="ref" type="xs:anyURI" use="required"/>
23   </xs:complexType>
24 </xs:element>
25 <xs:element name="soamResultsCollection">
26   <xs:complexType>
27     <xs:sequence>
28       <xs:element ref="soammsg:soamResult" maxOccurs="
          unbounded"/>
29     </xs:sequence>
30   </xs:complexType>
31 </xs:element>
32 <xs:element name="soamEntityIdsCollection">
33   <xs:complexType>
34     <xs:sequence>
35       <xs:element ref="soammsg:soamEntityId" maxOccurs="
          unbounded"/>
36     </xs:sequence>
37   </xs:complexType>
38 </xs:element>
39 </xs:schema>
```

## SoaM Entity Management API: SOAP and HTTP bindings

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!-- SoaM Entity Management API - SOAP & HTTP Bindings -->
4 <!-- Juan Ignacio Vazquez -->
5 <!-- Version 20060211.1 -->
6
7 <definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http
  ://schemas.xmlsoap.org/wsdl/soap/" xmlns:http="http://schemas.
  xmlsoap.org/wsdl/http/" xmlns:xs="http://www.w3.org/2001/XMLSchema
  " xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:
  mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soamws="http://
  www.awareit.com/soam/2006/02/soamws" xmlns:soamdt="http://www.
  awareit.com/soam/2006/02/soamdt" xmlns:soammsg="http://www.awareit
  .com/soam/2006/02/soammsg" targetNamespace="http://www.awareit.com
  /soam/2006/02/soamws">
8   <import namespace="http://www.awareit.com/soam/2006/02/soamdt"
     location="http://www.awareit.com/soam/2006/02/soamdt.xsd"/>
9   <import namespace="http://www.awareit.com/soam/2006/02/soammsg"
     location="http://www.awareit.com/soam/2006/02/soammsg.xsd"/>
10  <!-- Messages declaration-->
11  <message name="msgEmpty">
12  </message>
13  <message name="msgInformation">
14    <part name="data"/>
15  </message>
16  <message name="msgCapabilitiesCollection">
17    <part name="data" element="soamdt:capabilitiesCollection"/>
18  </message>
19  <message name="msgConstraintsCollection">
```

```
20     <part name="data" element="soamdt:constraintsCollection"/>
21 </message>
22 <message name="msgProfilesCollection">
23     <part name="data" element="soamdt:behavioralProfilesCollection
24         "/>
25 </message>
26 <message name="msgEntityIdsCollection">
27     <part name="data" element="soammsg:soamEntityIdsCollection"/>
28 </message>
29 <message name="msgResults">
30     <part name="results" element="soammsg:soamResultsCollection"/>
31 </message>
32 <message name="msgEntityId">
33     <part name="data" element="soammsg:soamEntityId"/>
34 </message>
35 <portType name="soamInformationService">
36     <operation name="retrieve">
37         <input message="soamws:msgEmpty"/>
38         <output message="soamws:msgInformation"/>
39     </operation>
40 </portType>
41 <portType name="soamCapabilitiesService">
42     <operation name="retrieve">
43         <input message="soamws:msgEmpty"/>
44         <output message="soamws:msgCapabilitiesCollection"/>
45     </operation>
46 </portType>
47 <portType name="soamProfilesService">
48     <operation name="retrieve">
49         <input message="soamws:msgEmpty"/>
50         <output message="soamws:msgProfilesCollection"/>
51     </operation>
52     <operation name="add">
53         <input message="soamws:msgProfilesCollection"/>
54         <output message="soamws:msgResults"/>
55     </operation>
56     <operation name="remove">
57         <input message="soamws:msgEntityIdsCollection"/>
58         <output message="soamws:msgResults"/>
59     </operation>
60     <operation name="renew">
61         <input message="soamws:msgEntityIdsCollection"/>
62         <output message="soamws:msgResults"/>
63     </operation>
64 </portType>
65 <portType name="soamConstraintsService">
66     <operation name="retrieve">
67         <input message="soamws:msgEmpty"/>
68         <output message="soamws:msgConstraintsCollection"/>
```

```

68     </operation>
69     <operation name="add">
70         <input message="soamws:msgConstraintsCollection"/>
71         <output message="soamws:msgResults"/>
72     </operation>
73     <operation name="remove">
74         <input message="soamws:msgEntityIdsCollection"/>
75         <output message="soamws:msgResults"/>
76     </operation>
77     <operation name="renew">
78         <input message="soamws:msgEntityIdsCollection"/>
79         <output message="soamws:msgResults"/>
80     </operation>
81 </portType>
82 <portType name="soamProfilesPostService">
83     <operation name="add">
84         <input message="soamws:msgProfilesCollection"/>
85         <output message="soamws:msgResults"/>
86     </operation>
87     <operation name="remove">
88         <input message="soamws:msgEntityIdsCollection"/>
89         <output message="soamws:msgResults"/>
90     </operation>
91     <operation name="renew">
92         <input message="soamws:msgEntityIdsCollection"/>
93         <output message="soamws:msgResults"/>
94     </operation>
95 </portType>
96 <portType name="soamConstraintsPostService">
97     <operation name="add">
98         <input message="soamws:msgConstraintsCollection"/>
99         <output message="soamws:msgResults"/>
100    </operation>
101    <operation name="remove">
102        <input message="soamws:msgEntityIdsCollection"/>
103        <output message="soamws:msgResults"/>
104    </operation>
105    <operation name="renew">
106        <input message="soamws:msgEntityIdsCollection"/>
107        <output message="soamws:msgResults"/>
108    </operation>
109 </portType>
110 <portType name="soamProfilesGetService">
111     <operation name="retrieve">
112         <input message="soamws:msgEmpty"/>
113         <output message="soamws:msgProfilesCollection"/>
114     </operation>
115     <operation name="remove">
116         <input message="soamws:msgEntityId"/>

```

```

117         <output message="soamws:msgResults"/>
118     </operation>
119     <operation name="renew">
120         <input message="soamws:msgEntityId"/>
121         <output message="soamws:msgResults"/>
122     </operation>
123 </portType>
124 <portType name="soamConstraintsGetService">
125     <operation name="retrieve">
126         <input message="soamws:msgEmpty"/>
127         <output message="soamws:msgConstraintsCollection"/>
128     </operation>
129     <operation name="remove">
130         <input message="soamws:msgEntityId"/>
131         <output message="soamws:msgResults"/>
132     </operation>
133     <operation name="renew">
134         <input message="soamws:msgEntityId"/>
135         <output message="soamws:msgResults"/>
136     </operation>
137 </portType>
138 <binding name="soamInformationSoap" type="soamws:
    soamInformationService">
139     <soap:binding style="document" transport="http://schemas.
        xmlsoap.org/soap/http"/>
140 - <operation name="retrieve">
141     <soap:operation soapAction="http://www.awareit.com/soam/
        information" style="document"/>
142 - <input>
143     <soap:body use="literal"/>
144 </input>
145 - <output>
146     <soap:body use="literal"/>
147 </output>
148 </operation>
149 </binding>
150 <binding name="soamCapabilitiesSoap" type="soamws:
    soamCapabilitiesService">
151     <soap:binding style="document" transport="http://schemas.
        xmlsoap.org/soap/http"/>
152 - <operation name="retrieve">
153     <soap:operation soapAction="http://www.awareit.com/soam/
        capabilities" style="document"/>
154 - <input>
155     <soap:body use="literal"/>
156 </input>
157 - <output>
158     <soap:body use="literal"/>
159 </output>

```

```

160     </operation>
161 </binding>
162 <binding name="soamProfilesSoap" type="soamws:soamProfilesService
163     ">
164     <soap:binding style="document" transport="http://schemas.
165     xmlsoap.org/soap/http"/>
166 - <operation name="retrieve">
167     <soap:operation soapAction="http://www.awareit.com/soam/
168     profiles" style="document"/>
169 - <input>
170     <soap:body use="literal"/>
171 </input>
172 <output>
173     <soap:body use="literal"/>
174 </output>
175 </operation>
176 - <operation name="add">
177     <soap:operation soapAction="http://www.awareit.com/soam/
178     profiles?add" style="document"/>
179 - <input>
180     <soap:body use="literal"/>
181 </input>
182 <output>
183     <soap:body use="literal"/>
184 </output>
185 </operation>
186 - <operation name="renew">
187     <soap:operation soapAction="http://www.awareit.com/soam/
188     profiles?renew" style="document"/>
189 - <input>
190     <soap:body use="literal"/>
191 </input>
192 <output>
193     <soap:body use="literal"/>
194 </output>
195 </operation>
196 - <operation name="remove">
197     <soap:operation soapAction="http://www.awareit.com/soam/
198     profiles?remove" style="document"/>
199 - <input>
200     <soap:body use="literal"/>
201 </input>
202 <output>
203     <soap:body use="literal"/>
204 </output>
205 </operation>
206 </binding>
207 <binding name="soamConstraintsSoap" type="soamws:
208     soamConstraintsService">

```

```

202     <soap:binding style="document" transport="http://schemas.
203         xmlsoap.org/soap/http"/>
204 - <operation name="retrieve">
205     <soap:operation soapAction="http://www.awareit.com/soam/
206         constraints" style="document"/>
207 - <input>
208     <soap:body use="literal"/>
209 </input>
210 - <output>
211     <soap:body use="literal"/>
212 </output>
213 </operation>
214 - <operation name="add">
215     <soap:operation soapAction="http://www.awareit.com/soam/
216         constraints?add" style="document"/>
217 - <input>
218     <soap:body use="literal"/>
219 </input>
220 - <output>
221     <soap:body use="literal"/>
222 </output>
223 </operation>
224 - <operation name="renew">
225     <soap:operation soapAction="http://www.awareit.com/soam/
226         constraints?renew" style="document"/>
227 - <input>
228     <soap:body use="literal"/>
229 </input>
230 - <output>
231     <soap:body use="literal"/>
232 </output>
233 </operation>
234 - <operation name="remove">
235     <soap:operation soapAction="http://www.awareit.com/soam/
236         constraints?remove" style="document"/>
237 - <input>
238     <soap:body use="literal"/>
239 </input>
240 - <output>
241     <soap:body use="literal"/>
242 </output>
243 </operation>
244 </binding>
<binding name="soamProfilesHttpPost" type="soamws:
    soamProfilesPostService">
    <http:binding verb="POST"/>
    <operation name="add">
    <http:operation location="?add"/>
    <input>

```

```

245         <mime:mimeXml part="data"/>
246     </input>
247     <output>
248         <mime:mimeXml part="results"/>
249     </output>
250 </operation>
251 <operation name="renew">
252     <http:operation location="?renew"/>
253     <input>
254         <mime:mimeXml part="data"/>
255     </input>
256     <output>
257         <mime:mimeXml part="results"/>
258     </output>
259 </operation>
260 <operation name="remove">
261     <http:operation location="?remove"/>
262     <input>
263         <mime:mimeXml part="data"/>
264     </input>
265     <output>
266         <mime:mimeXml part="results"/>
267     </output>
268 </operation>
269 </binding>
270 <binding name="soamConstraintsHttpPost" type="soamws:
    soamConstraintsPostService">
271     <http:binding verb="POST"/>
272     <operation name="add">
273         <http:operation location="?add"/>
274         <input>
275             <mime:mimeXml part="data"/>
276         </input>
277         <output>
278             <mime:mimeXml part="results"/>
279         </output>
280     </operation>
281     <operation name="renew">
282         <http:operation location="?renew"/>
283         <input>
284             <mime:mimeXml part="data"/>
285         </input>
286         <output>
287             <mime:mimeXml part="results"/>
288         </output>
289     </operation>
290     <operation name="remove">
291         <http:operation location="?remove"/>
292         <input>

```

```

293         <mime:mimeXml part="data"/>
294     </input>
295     <output>
296         <mime:mimeXml part="results"/>
297     </output>
298 </operation>
299 </binding>
300 <binding name="soamInformationHttpGet" type="soamws:
301     soamInformationService">
302     <http:binding verb="GET"/>
303     <operation name="retrieve">
304         <http:operation location=""/>
305         <input/>
306         <output>
307             <mime:mimeXml part="data"/>
308         </output>
309     </operation>
310 </binding>
311 <binding name="soamCapabilitiesHttpGet" type="soamws:
312     soamCapabilitiesService">
313     <http:binding verb="GET"/>
314     <operation name="retrieve">
315         <http:operation location=""/>
316         <input/>
317         <output>
318             <mime:mimeXml part="data"/>
319         </output>
320     </operation>
321 </binding>
322 <binding name="soamProfilesHttpGet" type="soamws:
323     soamProfilesGetService">
324     <http:binding verb="GET"/>
325     <operation name="retrieve">
326         <http:operation location=""/>
327         <input/>
328         <output>
329             <mime:mimeXml part="data"/>
330         </output>
331     </operation>
332 <operation name="renew">
333     <http:operation location="?renew"/>
334     <input>
335         <http:urlEncoded/>
336     </input>
337     <output>
338         <mime:mimeXml part="results"/>
339     </output>
340 </operation>
341 <operation name="remove">

```

```

339         <http:operation location="?remove"/>
340         <input>
341             <http:urlEncoded/>
342         </input>
343         <output>
344             <mime:mimeXml part="results"/>
345         </output>
346     </operation>
347 </binding>
348 <binding name="soamConstraintsHttpGet" type="soamws:
    soamConstraintsGetService">
349     <http:binding verb="GET"/>
350     <operation name="retrieve">
351         <http:operation location=""/>
352         <input/>
353         <output>
354             <mime:mimeXml part="data"/>
355         </output>
356     </operation>
357     <operation name="renew">
358         <http:operation location="?renew"/>
359         <input>
360             <http:urlEncoded/>
361         </input>
362         <output>
363             <mime:mimeXml part="results"/>
364         </output>
365     </operation>
366     <operation name="remove">
367         <http:operation location="?remove"/>
368         <input>
369             <http:urlEncoded/>
370         </input>
371         <output>
372             <mime:mimeXml part="results"/>
373         </output>
374     </operation>
375 </binding>
376 <!-- Example services declaration for IP address 169.254.0.3 -->
377 <!-- Information Retrieval Service-->
378 <service name="soamInformationService">
379     <port name="soamInformationServiceSoap" binding="soamws:
    soamInformationSoap">
380         <soap:address location="http://169.254.0.3/information"/>
381     </port>
382     <port name="soamInformationServiceHttpGet" binding="soamws:
    soamInformationHttpGet">
383         <http:address location="http://169.254.0.3/information"/>
384     </port>

```

```
385     </service>
386     <!-- Capabilities Retrieval Service-->
387     <service name="soamCapabilitiesService">
388         <port name="soamCapabilitiesServiceSoap" binding="soamws:
389             soamCapabilitiesSoap">
390             <soap:address location="http://169.254.0.3/capabilities"/>
391         </port>
392         <port name="soamCapabilitiesServiceHttpGet" binding="soamws:
393             soamCapabilitiesHttpGet">
394             <http:address location="http://169.254.0.3/capabilities"/>
395         </port>
396     </service>
397     <!-- Profiles Management Service-->
398     <service name="soamProfilesService">
399         <port name="soamProfilesServiceSoap" binding="soamws:
400             soamProfilesSoap">
401             <soap:address location="http://169.254.0.3/profiles"/>
402         </port>
403         <port name="soamProfilesServiceHttpPost" binding="soamws:
404             soamProfilesHttpPost">
405             <http:address location="http://169.254.0.3/profiles"/>
406         </port>
407         <port name="soamProfilesServiceHttpGet" binding="soamws:
408             soamProfilesHttpGet">
409             <http:address location="http://169.254.0.3/profiles"/>
410         </port>
411     </service>
412     <!-- Constraints Management Service-->
413     <service name="soamConstraintsService">
414         <port name="soamConstraintsServiceSoap" binding="soamws:
415             soamConstraintsSoap">
416             <soap:address location="http://169.254.0.3/constraints"/>
417         </port>
418         <port name="soamConstraintsServiceHttpPost" binding="soamws:
419             soamConstraintsHttpPost">
420             <http:address location="http://169.254.0.3/constraints"/>
421         </port>
422         <port name="soamConstraintsServiceHttpGet" binding="soamws:
423             soamConstraintsHttpGet">
424             <http:address location="http://169.254.0.3/constraints"/>
425         </port>
426     </service>
427 </definitions>
```

# SoaMonto specification

```
1 <?xml version="1.0"?>
2
3 <!-- SoaMonto: the SoaM ontology -->
4 <!-- Juan Ignacio Vazquez -->
5 <!-- Version 20051216.1 -->
6
7 <rdf:RDF
8   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
9   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
10  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
11  xmlns:owl="http://www.w3.org/2002/07/owl#"
12  xmlns="http://www.awareit.com/soam/2005/12/soamonto#"
13  xml:base="http://www.awareit.com/soam/2005/12/soamonto">
14
15   <owl:Ontology rdf:about=""/>
16
17   <owl:Class rdf:ID="Operator"/>
18   <owl:Class rdf:ID="Rule"/>
19
20   <owl:Class rdf:ID="Constraint">
21     <rdfs:subClassOf>
22       <owl:Class rdf:ID="Condition"/>
23     </rdfs:subClassOf>
24   </owl:Class>
25
26   <owl:Class rdf:ID="Sobject">
27     <rdfs:subClassOf>
28       <owl:Class rdf:ID="Entity"/>
29     </rdfs:subClassOf>
30   </owl:Class>
31
```

```
32 <owl:Class rdf:about="#Condition">
33   <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-
      syntax-ns#Statement"/>
34 </owl:Class>
35
36 <owl:Class rdf:ID="Orchestrator">
37   <rdfs:subClassOf rdf:resource="#Entity"/>
38 </owl:Class>
39
40 <owl:Class rdf:ID="Reasoner"/>
41 <owl:Class rdf:ID="Capability">
42   <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-
      syntax-ns#Statement"/>
43 </owl:Class>
44
45 <owl:Class rdf:ID="LocalReasoner">
46   <rdfs:subClassOf rdf:resource="#Reasoner"/>
47 </owl:Class>
48
49 <owl:Class rdf:ID="RemoteReasoner">
50   <rdfs:subClassOf rdf:resource="#Reasoner"/>
51 </owl:Class>
52
53 <owl:Class rdf:ID="Variable"/>
54 <owl:Class rdf:ID="BehaviouralProfile"/>
55 <owl:Class rdf:ID="ReasoningMechanism"/>
56
57 <owl:ObjectProperty rdf:ID="manages">
58   <rdfs:domain rdf:resource="#Orchestrator"/>
59   <rdfs:range rdf:resource="#Sobject"/>
60   <owl:inverseOf>
61     <owl:ObjectProperty rdf:ID="managedBy"/>
62   </owl:inverseOf>
63 </owl:ObjectProperty>
64
65 <owl:ObjectProperty rdf:ID="operator">
66   <rdfs:domain rdf:resource="#Condition"/>
67   <rdfs:range rdf:resource="#Operator"/>
68 </owl:ObjectProperty>
69
70 <owl:ObjectProperty rdf:ID="hasPostcondition">
71   <rdfs:domain rdf:resource="#BehaviouralProfile"/>
72   <rdfs:range rdf:resource="#Condition"/>
73   <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
      InverseFunctionalProperty"/>
74 </owl:ObjectProperty>
75
76 <owl:ObjectProperty rdf:about="#managedBy">
77   <owl:inverseOf rdf:resource="#manages"/>
```

```
78     <rdfs:domain rdf:resource="#Smobject"/>
79     <rdfs:range rdf:resource="#Orchestrator"/>
80 </owl:ObjectProperty>
81
82 <owl:ObjectProperty rdf:ID="hasOperationCapability">
83     <rdfs:range rdf:resource="#Capability"/>
84     <rdfs:domain rdf:resource="#Smobject"/>
85 </owl:ObjectProperty>
86
87 <owl:ObjectProperty rdf:ID="observesBehaviour">
88     <rdfs:domain rdf:resource="#Entity"/>
89     <rdfs:range rdf:resource="#BehaviouralProfile"/>
90 </owl:ObjectProperty>
91
92 <owl:ObjectProperty rdf:ID="usesReasoner">
93     <rdfs:domain rdf:resource="#Entity"/>
94     <rdfs:range rdf:resource="#Reasoner"/>
95 </owl:ObjectProperty>
96
97 <owl:ObjectProperty rdf:ID="isConstrainedBy">
98     <rdfs:range rdf:resource="#Constraint"/>
99     <rdfs:domain rdf:resource="#Smobject"/>
100 </owl:ObjectProperty>
101
102 <owl:ObjectProperty rdf:ID="involves">
103     <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Ontology
104     "/>
105     <rdfs:domain rdf:resource="#Rule"/>
106 </owl:ObjectProperty>
107
108 <owl:ObjectProperty rdf:ID="applies">
109     <rdfs:domain rdf:resource="#Reasoner"/>
110     <rdfs:range rdf:resource="#ReasoningMechanism"/>
111 </owl:ObjectProperty>
112
113 <owl:DatatypeProperty rdf:ID="profilesUri">
114     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI
115     "/>
116     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
117     FunctionalProperty"/>
118     <rdfs:domain rdf:resource="#Entity"/>
119 </owl:DatatypeProperty>
120
121 <owl:FunctionalProperty rdf:ID="capabilitiesUri">
122     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI
123     "/>
124     <rdfs:domain rdf:resource="#Smobject"/>
125     <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#
126     DatatypeProperty"/>
```

```
122 </owl:FunctionalProperty>
123
124 <owl:FunctionalProperty rdf:ID="constraintsUri">
125   <rdfs:domain rdf:resource="#Sobject"/>
126   <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#
      DatatypeProperty"/>
127   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI
      "/>
128 </owl:FunctionalProperty>
129
130 <owl:FunctionalProperty rdf:ID="informationUri">
131   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#anyURI
      "/>
132   <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#
      DatatypeProperty"/>
133   <rdfs:domain rdf:resource="#Entity"/>
134 </owl:FunctionalProperty>
135
136 <owl:InverseFunctionalProperty rdf:ID="hasPrecondition">
137   <rdfs:range rdf:resource="#Condition"/>
138   <rdfs:domain rdf:resource="#BehaviouralProfile"/>
139   <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#
      ObjectProperty"/>
140 </owl:InverseFunctionalProperty>
141
142 <owl:InverseFunctionalProperty rdf:ID="hasPerceptionCapability">
143   <rdfs:range rdf:resource="#Capability"/>
144   <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#
      ObjectProperty"/>
145   <rdfs:domain rdf:resource="#Sobject"/>
146 </owl:InverseFunctionalProperty>
147
148 <ReasoningMechanism rdf:ID="OwlFull"/>
149 <Operator rdf:ID="LessThan"/>
150 <Operator rdf:ID="NotEquals"/>
151 <ReasoningMechanism rdf:ID="OwlLite"/>
152 <ReasoningMechanism rdf:ID="OwlDL"/>
153 <ReasoningMechanism rdf:ID="GenericRules"/>
154 <ReasoningMechanism rdf:ID="Rdfs"/>
155 <owl:Thing rdf:ID="Any"/>
156 <Operator rdf:ID="GreaterThan"/>
157 <Operator rdf:ID="Equals"/>
158 <Operator rdf:ID="GreaterOrEqualsThan"/>
159 <Operator rdf:ID="LessOrEqualsThan"/>
160
161 </rdf:RDF>
```

## Example of smobject configuration file

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <smobjectConfiguration perceptionsPollingPeriod="1000" uuid="urn:uuid:
   plant1" constraintsCheckingPeriod="5000" defaultConstraintTimeout
   ="PT5M">
3
4   <mapping name="useReasoner" value="true"/>
5   <mapping name="ontologies" value="domain/temperature.owl domain/
   light.owl domain/location.owl"/>
6   <mapping name="nativeBehaviour" value="demo_plant/bp_plant.xml"/>
7   <mapping name="nativeDomainRules" value="domain/motenetwork_rules.
   txt"/>
8
9   <mapping name="perceptorFile" value="demo_plant/plant1_data.owl"/>
10  <mapping name="com.awareit.smobject.platforminterfaces.alerter.
   ttscommand" value="flite"/>
11
12  <perceptionCapabilityBinding bindingClass="com.awareit.smobject.
   platforminterfaces.PerceptorFile" id="urn:uuid:plant1_pcap1">
13    <subject resource="urn:uuid:plant1"/>
14    <ontology resource="http://www.awareit.com/soam/2005/12/
   soamonto#Any"/>
15  </perceptionCapabilityBinding>
16
17  <operationCapabilityBinding bindingClass="com.awareit.smobject.
   platforminterfaces.alerter.EffectorTTSAlerter" id="urn:uuid:
   plant1_ocap1">
18    <subject resource="http://www.awareit.com/soam/2005/12/
   soamonto#Any"/>
```

## A Reactive Behavioural Model for Context-Aware Semantic Devices

---

```
19     <ontology resource="http://www.awareit.com/onto/2005/12/  
20         alerting"/>  
21     </operationCapabilityBinding>  
22 </smobjectConfiguration>
```