Strategies for context reasoning in assistive livings for the elderly

Thibaut Tiberghien

THÈSE DE DOCTORAT CONJOINT
TÉLÉCOM SUDPARIS & UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité Informatique

École doctorale Informatique, Télécommunications et Électronique de Paris

Présentée par

Thibaut Tiberghien

Pour obtenir le grade de

DOCTEUR DE TELECOM SUDPARIS

Stratégies pour le raisonnement sur le contexte dans les environnements d'assistance pour les personnes âgées

Soutenue le 18 Novembre 2013

devant le jury composé de :

Pr. Jacques Demongeot	Université Joseph Fourier & CNRS AGIM	Rapporteur
Pr. Lawrence Wong	National University of Singapore	Rapporteur
Pr. François Pierrot	Université Montpellier 2 & CNRS LIRMM	Examinateur
Dr. Mohamed Ali Feki	Alcatel Lucent Bell Labs Belgium	Examinateur
Pr. Daniel Racoceanu	Université Pierre et Marie Curie & CNRS IPAL	Examinateur
Pr. Mounir Mokhtari	Institut Mines Télécom & CNRS IPAL	Directeur de Thèse

Thèse n° 2013TELE0026

Institut Mines-Telecom
Image & Pervasive Access Lab
CNRS (UMI 2955), France
I^2R / A*STAR, Singapore
1 Fusionopolis Way
#21-01 Connexis (South Tower)
Singapore 138632

DOCTOR OF PHILOSOPHY (PhD) THESIS
INSTITUT MINES-TELECOM & UNIVERSITY PIERRE AND MARIE CURIE

Specialization in Computer Sciences

Paris Doctoral School of Computing, Telecommunication and Electronics

Presented by

Thibaut Tiberghien

To obtain the degree of

DOCTOR OF PHILOSOPHY FROM TELECOM SUDPARIS

Strategies for Context Reasoning in Assistive Livings for the Elderly

Defended on 18th November 2013

in front of a doctoral comittee composed of:

Pr. Jacques Demongeot	Université Joseph Fourier & CNRS AGIM	Reviewer
Pr. Lawrence Wong	National University of Singapore	Reviewer
Pr. François Pierrot	Université Montpellier 2 & CNRS LIRMM	Jury Member
Dr. Mohamed Ali Feki	Alcatel Lucent Bell Labs Belgium	Jury Member
Pr. Daniel Racoceanu	Université Pierre et Marie Curie & CNRS IPAL	Jury Member
Pr. Mounir Mokhtari	Institut Mines Télécom & CNRS IPAL	Thesis Director

Thèse n° 2013TELE0026

Enthusiasm is one of the most powerful engines of success.
When you do a thing, do it with all your might.
Put your whole soul into it.
Stamp it with your own personality.
Be active, be energetic and faithful, and you will accomplish your object.
Nothing great was ever achieved without enthusiasm.

— Ralph Waldo Emerson, 1803–1882

Dedicated to my lovely wife Wanlyn.

Abstract

Research Challenge

One's interest in ambient intelligence may lie in the ability of an environment to respond in an appropriate manner to what is happening within it. It is the reaction of a computerised system to a *non-formalised* situation that is intriguing. Such systems are by nature able to instantiate a reaction, even complex, to a formalised and recognised situation, even complex. The true challenge is to provide a formalisation for machines to project situational data and make *sense* of it, i.e. build connections or bindings between it and the rest of the contextual knowledge. We call this challenge "context comprehension", and divide it into two main aspects: *(i)* formalising contextual knowledge to project situational data in it, and *(ii)* reasoning to connect such inter-correlated formalised knowledge or infer new one. The problem being studied in this doctoral work is: *What strategies can be put in place to provide context comprehension in assistive livings?*

Comprehension is defined in the Random House Kernerman Webster's College Dictionary as the "capacity of the mind to perceive and understand; [the] power to grasp ideas". We can see it as the process of simultaneously extracting and constructing meaning through the manipulation of sensed situational data. We use the words *extracting* and *constructing* to emphasize both the importance and the insufficiency of the sensed data as a determinant of comprehension. In the field of ambient intelligence, it consists in putting in place a translation mechanism between the sensed representation of a situation and its formalised, machine-readable version. This mechanism would probably be constituted of heterogeneous and complementary strategies, allowing the definition of a formalisation (or model), the naive projection of sensed data into the model, and the inference of knowledge into this model.

Outcome & Contributions of the Thesis

Leveraging our experience with the traditional approach to Ambient Assisted Living (AAL) which relies on a large spread of heterogeneous technologies in deployments, this thesis studies the possibility of a more "stripped down" and complementary approach, where only a reduced hardware subset is deployed, probing a transfer of complexity towards the software side, and enhancing the large scale deployability of the solution. Focused on the reasoning aspects in AAL systems, this work has allowed the finding of a suitable semantic inference engine for the peculiar use in these systems, responding to a need in this scientific community. Considering the coarse granularity of situational data available, dedicated rule-sets with adapted inference strategies are proposed, implemented, and validated using this engine. A novel semantic reasoning mechanism is proposed based on a cognitively inspired reasoning architecture. Finally, the whole reasoning system is integrated in a fully featured context-aware service framework, powering its context awareness by performing live event processing through complex ontological manipulation. The overall system is validated through in-situ deployments in a nursing home as well as private homes over a few months period, which itself is noticeable in a mainly laboratory-bound research domain.

Organisation of this Dissertation

This thesis concentrates its research efforts on the reasoning aspects in smart environments. The dissertation will focus mainly on the design of the reasoning engine which allowed us to develop an integrated system that has been deployed in real conditions. It is organised around five parts, each subdivided into one to three chapters. The first part provides the background and motivation behind this doctoral work, concluding with the positioning of the work. The second part consists in the conception part of the work. It is composed of an analysis of the related work in rule-based reasoning, a comparison of

semantic reasoning engines from an ambient intelligence point of view, the design of a novel rule-set for context comprehension, and its integration into a cognitively inspired reasoning architecture. The third part is more focused towards the implementation aspects of the work. After introducing the enabling technologies, it describes the evolution of the context-aware service framework designed and implemented, focusing on two notable milestones and their respective architectures. Details of the mechanisms implemented are provided, highlighting their contribution to the improvement of the overall process and performance. The fourth part provides results from the various validations that have been conducted; on one hand in a nursing home in Singapore and on the other hand in three individual homes in France, with the involvement in both countries of several partners from medical, research and engineering background. It also discusses the strategies which could enable the technological transfer into society of AAL solutions. The last part concludes on the work done and provides an overview of the further studies in perspective.

Keywords

Ambient Assisted Living, Ageing People with Dementia, Ambient Intelligence, Internet of Things, Context Awareness, Knowledge Modelling, Semantic Reasoning, Semantic Web, Inference Engine, Cognitive Model, Service Framework, Scalable Deployments.

Author's Publications

Journal Papers

H. Aloulou, M. Mokhtari, **T. Tiberghien**, J. Biswas, and P. Yap, "Real world deployment of assistive living technologies for cognitively impaired people in singapore: Demonstration guidelines," in *IEEE Journal of Biomedical and Health Informatics (J-BHI)*, IEEE, 2013 (In-Press).

H. Aloulou, M. Mokhtari, **T. Tiberghien**, J. Biswas, C. Phua, J. H. K. Lin, and P. Yap, "Deployment of assistive living technology in a nursing home environment: methods and lessons learned," in *BMC Medical Informatics and Decision Making*, vol. 13, p. 42, 2013.

M. Mokhtari, H. Aloulou, **T. Tiberghien**, J. Biswas, D. Racoceanu, and P. Yap, "New trends to support independence in persons with mild dementia – a mini-review," in *International Journal of Experimental, Clinical, Behavioural, Regenerative and Technological Gerontology*, vol. 58, pp. 554–563, Karger Publishers, 2012.

Conference Papers

R. Endelin, H. Aloulou, J. De Roo, S. Renouard, **T. Tiberghien**, and M. Mokhtari, "Implementation of allen's interval logic with the semantic web," in *International ACM Conference on Management of Emergent Digital EcoSystems (MEDES)*, ACM, 2013 (In-Press).

R. Endelin, S. Renouard, **T. Tiberghien**, H. Aloulou, and M. Mokhtari, "Behavior recognition for elderly people in large-scale deployment," in *Inclusive Society: Health and Wellbeing in the Community, and Care at Home (ICOST 2013)*, vol. 7910 of *Lecture Notes in Computer Science*, pp. 61–68, Springer, 2013.

A. A. Phyo Wai, J. H. K. Lin, V. Y. Lee, C. Phua, **T. Tiberghien**, H. Aloulou, Y. Liu, X. Zhang, J. Biswas, and P. Yap, "Challenges, experiences and lessons learned from deploying patient monitoring and assistance system at dementia care hostel," in *Inclusive Society: Health and Wellbeing in the Community, and Care at Home (ICOST 2013)*, vol. 7910 of *Lecture Notes in Computer Science*, pp. 292–297, Springer, 2013.

T. Tiberghien, M. Mokhtari, H. Aloulou, and J. Biswas, "Semantic reasoning in context-aware assistive environments to support ageing with dementia," in *Proceedings of the 11th International Semantic Web Conference (ISWC)*, vol. 7650 of *Lecture Notes in Computer Science*, pp. 212–227, Springer, 2012.

H. Aloulou, M. Mokhtari, **T. Tiberghien**, J. Biswas, and J. H. K. Lin, "A semantic plug&play based framework for ambient assisted living," in *Impact Analysis of solutions for chronic disease Prevention and Management (ICOST)*, vol. 7251 of *Lecture Notes in Computer Science*, pp. 165–172, Springer, 2012.

T. Tiberghien, M. Mokhtari, H. Aloulou, J. Biswas, J. Zhu, and V. Lee, "Handling user interface plasticity in assistive environment: Ubismart framework," in *Toward Useful Services for Elderly and People with Disabilities (ICOST)*, vol. 6719 of *Lecture Notes in Computer Science*, pp. 256–260, Springer, 2011.

J. Zhu, V. Y. Lee, J. Biswas, M. Mokhtari, **T. Tiberghien**, and H. Aloulou, "Context-aware reasoning engine with high level knowledge for smart home," in *Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*, pp. 292–297, SciTePress, 2011.

J. Biswas, A. A. Phyo Wai, A. Tolstikov, J. H. K. Lin, M. Jayachandran, S. F. V. Foo, V. Y. Lee, C. Phua, J. Zhu, T. H. Huynh, **T. Tiberghien**, H. Aloulou, and M. Mokhtari, "From context to micro-context-issues and challenges in sensorizing smart spaces for assistive living," in *Proceedings of the 2nd International Conference on Ambient Systems, Networks and Technologies (ANT)*, vol. 5, pp. 288–295, Elsevier, 2011.

Reports

T. Tiberghien, "Mechanisms of plasticity in the design of dynamic user interfaces for application in pervasive assistive environments," Master's thesis, Technische Universität München, Image & Pervasive Access Lab, 2010.

T. Tiberghien, "Tangible interaction to enhance users' experience," in *Sensor-Based User Interfaces – Science or Science-Fiction?*, pp. 11–18, Technische Universität München, 2010.

Acknowledgement

First and foremost I would like to thank Mounir for giving me this opportunity of course. I have benefited greatly from your vision, your friendship and your confidence. Thank you to Hamdi for your daily patience. It was great sharing the past three years with you, supporting each other and making things work better despite our differences. A shout out to Wan, my early partner in crime. I wish we had more of those crazy times. Next up are our chosen ones - the interns! Romain, designing all day with you was just great, even though you always had "one more little thing"... Jérémie, your maturity, perfectionism and open-mindedness made working with you such a breeze. I hope we'll have opportunities to work together again. Guillaume, you are a crazy guy, a visionary and an innovator! Come back whenever you want.

A great thank you to the Amupadh team: Kenneth, Alwyn, Andrei, Jaya, Aung Aung, Xian, Yan, Jin Song, Clifton, Weimin and Philip. We shared two great years, starting from scratch and building up a platform that we could deploy today. Those were some mad times. Thank you to Jit in particular for your advice and kindness, I do hope we can collaborate again soon.

Next up I would like to acknowledge the QoL team too. Thank you Stéphane. Although you were far and busy and we couldn't share so much time together, you have inspired me in many ways. It was good to have someone to look up to. Maybe we can work again together some day.

Thank you to all the other guys from IPAL. Antoine who, in three years went from being a colleague to being a groomsmen at my wedding. Stéphane for running this marathon with me from the very beginning to the end—it's good to see the light at the end of the tunnel together. Ludo, I have appreciated your quiet yet strong presence in IPAL all this while, and you have made it a better environment for me. Olivier, Michal, Joo Hwee, Renan, Patrick, Kim, Thibault, Solène, Suriana, Joreis, Pierre, Carole, Clément, Shue Ching, Shijian, Hanlin, Camille, Blaise, Nicolas, and the others I forget. We share many good memories that I cannot summarize here. Thank you to Coralie for your friendship, your "innovationism", for making us a bit more proud, and for your encouragement. Thank you as well to Daniel, for the great conversations we once had, for speaking about philosophy and reminding me that it is part of our sciences.

Thank you Jos for being such a motivating and inspiring person. For your support on EYE too and your optimism towards our work.

I wish to thank the scientific direction at the Institut Mines-Télécom for funding this work.

I would like to express my gratitude to the residents who have accepted to take part in the trials; in Peacehaven in Singapore and at home in France. This gratitude is shared by my teammates as well. You have been very accommodating in the past months.

Last but not least, I would like to thank my family—my parents for bearing with the concerns they had, and my parents-in-law for the pride they make me feel. And the biggest thank you of all goes to my wife Wanlyn. Thank you for your patience and your love during these three years. Thank you for your moral support and for sharing the dreams that help me go forward.

A special thought goes out to my grand-parents. I hope that my work will help families keep in touch in the future, despite life becoming crazier by the day, despite the distance among family-members. I wish we had the technologies I envision to grow a bit more together.

Contents

Part I

Introduction

The important thing is not how many years in your life but how much life in your years.

— Edward J. Stieglitz, 1899–1958

1 Towards Sustainable Ageing

1.1 Ageing in Place

1.1.1 Challenging Demographic Changes

The United Nations' World Population Ageing report [1] defines **population ageing** as the process by which older individuals become a proportionally larger share of the total population. With an unprecedented intensity, it is reported as one of the most distinctive demographic events of the twentieth century, and the twenty-first century will witness even more rapid ageing. Indeed, it is estimated in the World Alzheimer Report 2009 [2] that 1 in 5 people will be aged over 60 years old by 2050 compared to 1 in 10 today. Figure 1.1 provides an overview of the world's demographic changes in the second half of the twentieth century, together with an estimation for the first half of the twenty-first.

Population ageing is due to the **demographic transition**, which refers to the transition from high birth and death rates to low birth and death rates as a country develops from a pre-industrial to an industrialized economic system. Between 1950 and 2010, the global life expectancy has increased from 48 to 68 years old and it is expected to go up to 75 years old by 2050 [3]. There are still considerable disparities between the developed countries (82 years old) and the developing countries (74 years old), however this gap has narrowed greatly in the last decades. In parallel, the global fertility rate has fallen from 5 children per woman in 1950 to roughly 2.5 today, and is projected to drop to about 2 by

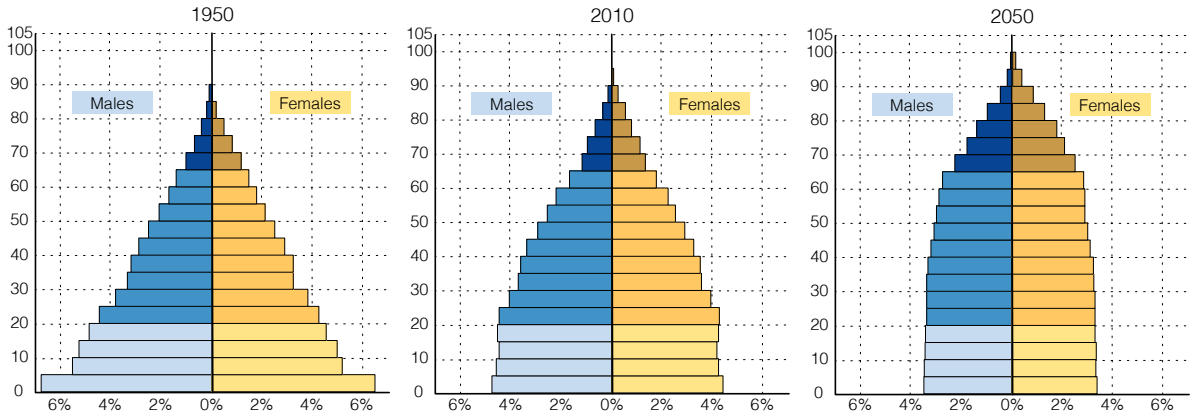


Figure 1.1: World Population by Age Groups and Sex (Ratio Over Total Population) [3]

2050 [3]. Most of this decline has occurred in the developing world, where the share of children in the population is expected to drop by half from the 1965 level by 2050.

If population ageing was initially experienced by the more developed countries, it is now a global phenomenon and all countries will face it at different levels of intensity and in different time frames. The evolution of societies' age structure caused by population ageing has a profound impact on a broad range of economic, political and social aspects. For instance, concerns are growing about the long-term viability of intergenerational social support systems, which are crucial for the well-being of both the older and younger generations [4]. This is especially true where provision of care within the family becomes more and more difficult as family size decreases and women are increasingly employed. In the Asian culture, and especially in Japan, it is customary for working adults to be the informal caregivers providing for the parents needs. As illustrated in Figure 1.2, this trait of society has however seen an important evolution since the percentage of elderly living with their kin in Japan has decreased from 85% in 1960 to 55% in 1995. In western countries, the viability of the socio-economical pension systems

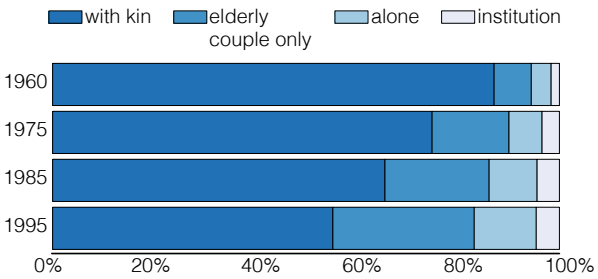


Figure 1.2: Living Arrangements of Japanese Elderly: 1960 to 1995 [5]

ensuring the financial support for the elderly is to be questioned. Such systems were indeed designed after the second world-war relying on a demographic balance that does not exist anymore. For example, Figure 1.3 highlights the evolution of the life course of a person between 1960 and 1995. It is given as an average over 15 western countries member of the Organisation for Economic Co-Operation and Development (OECD), an international organisation helping governments tackle economic, social and governance challenges. As more people live longer, retirement, pensions and other social benefits tend to extend over longer periods of time. This makes it necessary for social security systems to change substantially in order to remain effective [1]. Increasing longevity can also result in rising medical costs and increasing demands for health services, since older people are typically more vulnerable to chronic

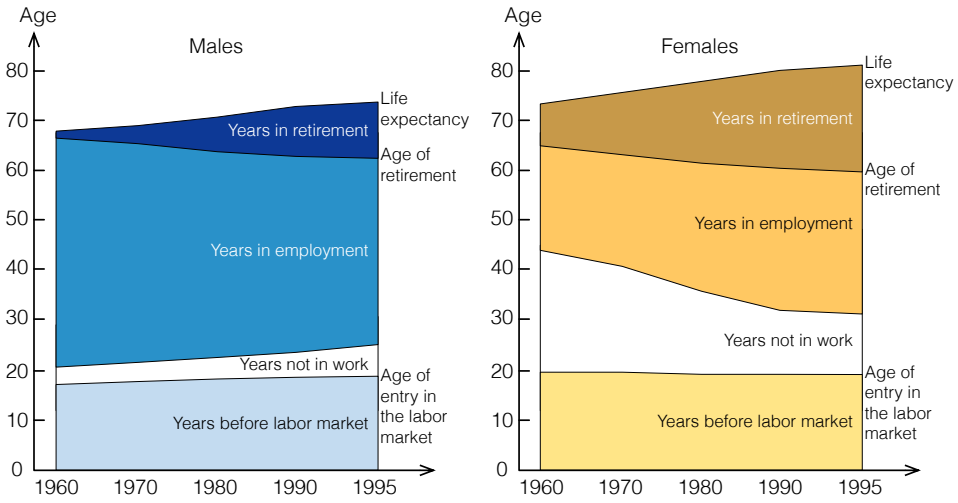


Figure 1.3: Decomposition of Average Life Course: 1960 to 1995 [5]

diseases. In Figure 1.3, one can observe that despite an increase in the numbers of years of employment for women, their retirement time has also increased from 8 years in 1960 to 21 years in 1995.

1.1.2 Normal Ageing

Ageing is a highly individualized, irreversible and inevitable process by which a person becomes more vulnerable and dependent on others [6]. It proceeds at different rates and within different functions depending on people. Changes, that can occur in cognitive, physiological and social conditions, are not necessarily related to a disease since they are, in a certain magnitude, a normal part of the ageing process.

Cognitive Changes

Cognitive changes related to normal ageing span across several aspects of the mind. If general knowledge is preserved, declines are normal in spatial orientation, numeral treatments, verbal comprehension and verbal fluidity [7]. The "mechanical intelligence" is in fact mainly affected due to a decrease in processing speed and sensory functioning related to normal ageing [8, 9]. Although changes in memory functioning with advancing age are also expected and feared, not all aspects of memory are affected. Figure 1.4 summarises the different types of memory with the impact of age under normal conditions. **Sensory memory** is the ability for each of the five senses to hold a large amount of sensory information for a very short period of time, and is independent of the attention to the stimulus. It suffers no major influence of ageing. **Short term memory** is a 20 to 30 seconds memory used to hold information for processing. It is a working memory that can hold 7 elements for direct manipulation. It is highly involved in Activities of Daily Living (ADL) as it enables multi-tasking and manipulation of information. The capacity to hold 7 elements is not affected by age, but the manipulation of this memory becomes difficult as elders have a limited capacity to divide their attention between two related tasks or inhibit unimportant information. **Long term memory** is a series of memory modules each responsible for holding different sorts of information. It is subject to three mechanisms (encoding, storage, retrieval) that are not affected by age in the same way. Encoding is usually subject to a less spontaneous organisation of information, so elders might need support on this. Storage is not affected. With age, retrieval becomes much more difficult in free recall (e.g. What were the 5 items?) than in recognition (e.g. Did I say apple? cucumber?) or with help (e.g. What were the 2 tools and 3 fruits?). Globally the

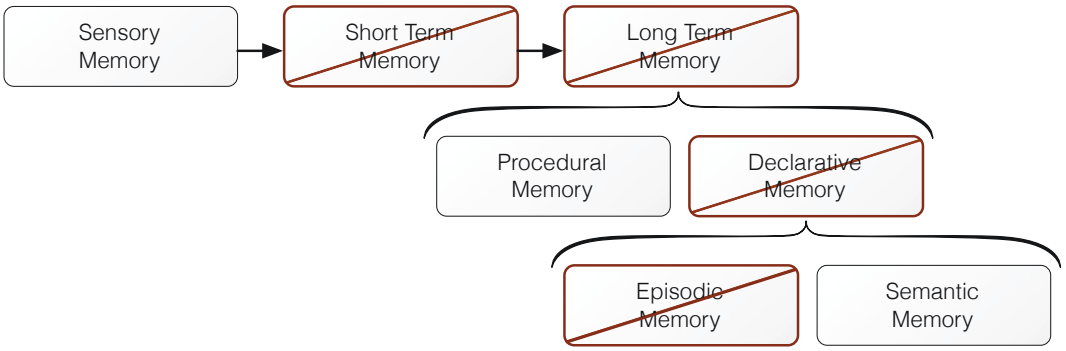


Figure 1.4: Preservation and Decline in the Normal Ageing Memory [6]

latency of retrieval increases with the advancing age. Among long term memory modules, **procedural memory** is the memory of "how to do" for cognitive and motor tasks and is well preserved under normal ageing conditions. **Declarative memory**, the memory of facts and events, is however partly affected. One usually experiences a decline of the **episodic memory** which is holding information such as what happened, when and where. This is due to a growing difficulty to remember the source and context in which information is learned. It is for example useful to remember where someone left his keys. **Semantic memory**, the memory of meanings and concept-based knowledge, is normally not influenced by age. Finally, cognitive changes related with normal ageing are also subjectively greater due to the difficulty of the overall language and communication experience due to a sharp decline in verbal fluidity and growing problems to handle complex sentence structures.

Physiological Changes

Similarly to the cognitive aspects, physiological difficulties associated with normal ageing span across several parts of the body. One of the greatest impacts on the functional abilities of elders is due to the declines of the sensory system. The sight of a 60 years old, for example, reaches only a third of the luminosity as compared to a 20 years old [10]. It becomes more difficult to adjust to darkness or distinguish colors. This has a strong impact on the quality of life of elders as they become anxious over performing their ADL safely. It also reduces their mobility while augmenting the risk of falls. Hearing loss is also part of the normal ageing process and is having a strong impact on elders' ease and quality of communication. Other senses are affected as well and participate to reduce the overall quality of life. It is noticeable that the sensory system is fully affected by normal ageing, which may reflect the deterioration of the nervous system rather than the sensory modalities themselves. Other physiological changes include a decreased muscle strength and power as well as a decline in cardiovascular and pulmonary functions affecting people's balance and functional abilities necessary to perform ADL.

Social Changes

Due to the changes exposed in the two paragraphs above,—let us remember these are considered normal ageing and do not incorporate declines or difficulties due to pathological ageing—people become dependent on others with advancing age. Indeed, cognitive declines make it growingly difficult to handle emergency situations and forgetfulness can sometimes threaten the safety of elders. Moreover, physiological changes make performing ADL difficult as activities with low metabolic demand are perceived as more and more demanding. 65 years old is commonly considered as the cut-off between adult age and elderly as it corresponds to the age where a significant percentage of the population needs assistance to perform their ADL (see Figure 1.5). It has also been observed that elders generally suffer from social isolation as well, mostly due to their communication difficulties and reduced mobility. This

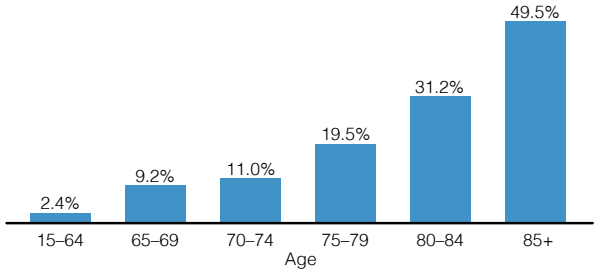


Figure 1.5: Percent of People Needing Assistance with Daily Activities by Age Groups [11]

has a greater impact than is usually recognised since their consequently lower quality of life increase their risk for depression.

1.1.3 Pathological Ageing

Pathological ageing is defined as an unexpected decline of one or several conditions (cognitive, physiological, social) of a person with advancing age. It can come forward as an unusual acceleration of changes under the scope of normal ageing, or as the apparition of changes unrelated to normal ageing. Although all conditions can be affected by pathological ageing, a particular focus is commonly put on the cognitive condition. Figure 1.6 illustrates the different stages of cognitive decline between normal ageing and dementia. **Mild cognitive impairment** characterises a cognitive impairment beyond that

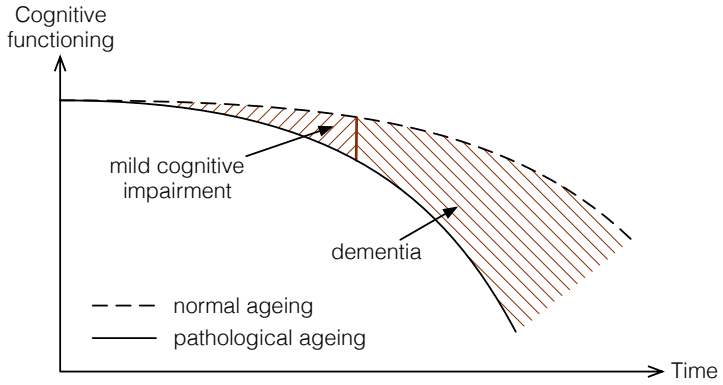


Figure 1.6: Pathological Ageing: Mild Cognitive Impairment and Dementia [6]

expected for the age and education of a person. It is a transitional stage between normal ageing and dementia, for which the impairment has no major effect on the performance of ADL. It is mainly seen as a risk factor towards dementia but at this stage, a person's condition can remain stable or even improve. **Dementia** is a syndrome that can be caused by a number of progressive illnesses that affect memory, thinking, behaviour and the ability to perform everyday activities [12]. **Alzheimer's disease** is the most common type of dementia, and other types include vascular dementia, dementia with Lewy bodies and frontotemporal dementia. The boundaries between the types are not clear, and a mixture of more than one type is common. Dementia mainly affects older people, although there is a growing awareness of cases that start before the age of 65. After age 65, the likelihood of developing dementia roughly doubles every five years.

Dementia and Alzheimer's Disease

Dementia is characterized by a progressive deterioration of intellectual and functional abilities, typically over a period of 7 to 10 years [13] and is classified into 5 stages (3–7) according to the Global Deterioration Scale (GDS) [14]. These stages are described in Table B.1 in Appendix B. At stage 3, the symptoms can be subtle and the patient can live independently without assistance. At stages 4 and 5, independent living becomes an issue, and in more advanced stages of the disease, the situation becomes critical, especially with verbal communication problems (aphasia), difficulty in identifying objects and persons (agnosia), and high-level disorder in performing familiar and learned tasks (apraxia). The elder, therefore, needs continuous support by a caregiver because he or she can no longer perform everyday tasks.

In epidemiology, the **prevalence** is the proportion of a population found to have a condition. The prevalence of dementia was estimated to 35.6 million people in 2010 and this number is nearly doubling every 20 years [2]. A study using a stochastic, multistate model was used in conjunction with United Nations' worldwide population forecasts and data from epidemiological studies of the risks of Alzheimer's disease in order to forecast the global burden of Alzheimer's disease by 2050 [15]. This study predicted that by the year 2050, the worldwide prevalence of Alzheimer's would grow fourfold to 106.8 million. Table 1.1, extracted from this study, shows the geographic distribution of the burden of the disease. It is estimated that 48% of cases worldwide are in Asia, and that the percentage in

Table 1.1: Projections of Alzheimer's Disease Prevalence (in Millions) in 2006 and 2050, by Regions and Stage of Disease* [15]

	Prevalence (in millions)					
	2006			**2050**		
	Overall	**Early-stage**	**Late-stage**	**Overall**	**Early-stage**	**Late-stage**
Africa	1.33	0.76	0.57	6.33	3.58	2.75
Asia	12.65	7.19	5.56	62.85	34.84	28.01
Europe	7.21	4.04	3.17	16.51	9.04	7.47
Latin America	2.03	1.14	0.89	10.85	5.99	4.86
North America	3.10	1.73	1.37	8.85	4.84	4.01
Oceania	0.23	0.13	0.10	0.84	0.46	0.38
Total	26.55	14.99	11.56	106.23	58.75	47.48

*Regions defined according to the United Nations Population Division [3]. Oceania includes Australia, New Zealand, Melanesia, Micronesia, and Polynesia.

Asia will grow to 59% by 2050. Figure 1.7 shows the growth in prevalence of Alzheimer's disease up to 2050 by stage of disease and by gender. It is estimated that about 62% of worldwide cases are female, reflecting the lower mortality rates among women.

The World Alzheimer Report 2010 [12] which focuses on the global economic impact of dementia estimates that the worldwide cost of dementia is USD 604 billions each year.

1.2 Gerontechnology

The sustainability of an ageing society depends upon our effectiveness in creating technological environments, including assistive technology and inclusive design, for innovative and independent living and social participation of older adults in good health, comfort and safety. **Gerontechnology** is an interdisciplinary academic and professional field combining gerontology and technology [16]. It concerns matching technological environments to health, housing, mobility, communication, leisure and work of older people [17]. It enables the design and development in relevant engineering disciplines of adapted solutions based on scientific knowledge about the ageing process [18]. Research outcomes

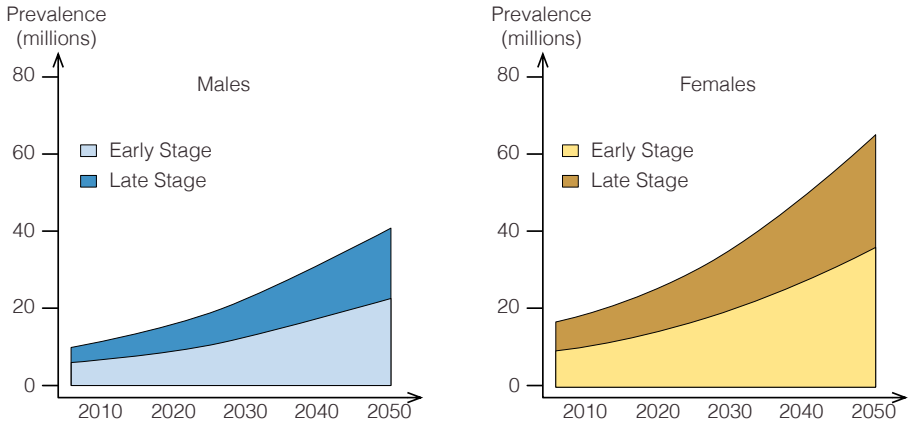


Figure 1.7: Worldwide Projections of Alzheimer's Prevalence (in Millions) for the Years 2006–2050, by Stage of Disease [15]

form the basis for designers, builders, engineers, manufacturers, and those in the health professions (nursing, medicine, gerontology, geriatrics, environmental psychology, developmental psychology, etc.), to provide an optimum living environment for the widest range of ages.

Gerontechnology research spans across a wide range of application domains where technology can help to handle age-related decline of abilities. It can for example help to compensate the decline of these capacities using solutions going from reading glasses to adapted motored vehicle, or from pinhole camera necklace building day diaries to smart homes assisting elders perform their ADL. Furthermore, specific research is targeting the improvement of caregiving conditions, for professional caregivers as much as for informal caregivers within the family. Hence, gerontechnology provides technical support like systems for lifting and transferring people with reduced mobility or monitoring solutions to ensure that an elder person is able to leave alone safely. Another important area of focus is the improvement of seniors' social life by enhancing their communication capabilities and providing them with opportunities in new roles and situations such as performing new leisure activities or introducing them to new social situations or communication channels.

For an efficient integration of these technologies in the environment and life of elders, it is crucial to look into the ease of use and acceptability of these technologies for an ageing use. The ageing population and its impact on economics, politics, education and lifestyle is already a global concern. In time, products and services relevant to the "silver industry" are expected to flood the marketplace. And with it, we expect an increase in the demand for designers and engineers who understand the needs of the ageing population using the knowledge of gerontological design processes.

1.3 Ambient Assisted Living

Making use of the transition towards pervasive, ubiquitous environments where embedded computing devices seamlessly integrate and cooperate to serve human needs, we can design systems specially fitted to provide care to the elderly. There is indeed no reason for the elderly to miss out on the benefits of Information & Communications Technology (ICT) which can help them remain independent, socially active and increase their mobility and safety. These technologies are particularly relevant to elderly people with dementia, who are usually accepting of their deployment and understand the impact on their safety and well-being in daily life [19]. **Ambient Assisted Living (AAL)** consists of a set of ubiquitous technologies embedded in a living space to provide pervasive access to context-aware assistive services. It can for example enhance ageing in place by helping elderly people with their ADL.

Through a multi-disciplinary approach involving engineers, researchers, designers and medical staff, and in close collaboration with stakeholders like elder citizens, families, professional caregivers and insurance companies, AAL research foster the emergence of innovative ICT-based products, services and systems for ageing well at home, in the community, and at work. It aims at increasing the quality of life, autonomy, participation in social life, skills and employability of elderly people, and reducing the costs of health and social care. It also provides support to carers, families and care organisations. Ultimately, AAL is a shared vision [20] to combine social, technological and business aspects to deliver:

- new models of service delivery and care that contribute to greater self reliance for older adults and greater support for informal carers;
- living spaces that can improve the quality of their everyday lives;
- ways for older people to remain active, including contributing as volunteers or providing mutual support.

The LeadingAge Center for Aging Services Technologies (CAST) is a non-profit organisation leading the charge to expedite the development, evaluation and adoption of emerging technologies that can improve the ageing experience. It has become an international coalition of more than 400 technology companies, ageing services organizations, research universities, and government representatives. In 2010, CAST produced a conceptual video giving a glimpse, through the eyes of one family, of what the future of ageing could look like with the help from developing technologies that are possible, practical, and affordable (`http://www.leadingage.org/Imagine-the-Future-of-Aging.aspx` [21]). The reader is invited to watch the video, as well as a second one available on the same webpage where industry experts describe how these technologies can potentially improve healthcare, preserve independence and ensure quality of life for seniors.

Use-Case: Carol, 79 Years Old

Carol is living alone in the same home that she has lived in for 30 years. She and her husband, who passed away two years ago, raised their three children in this home and although she is now living in it independently - it is home. Her children have long since moved out with the closest being Michelle who is a 45-minute drive away. Carol has some cardiovascular conditions and she also, like many people her age, is starting to experience a bit of cognitive decline. She just does not remember things as well as she used to.

Each day, Carol gets up around 7:00, uses the bathroom, weighs herself, goes to the kitchen to eat breakfast and take her various pills, makes a cup of tea and then settles into the den to watch the morning news. Around 9:00, a prompt appears across the TV screen reminding Carol to take her blood pressure, which she does with a wireless-enabled blood pressure cuff that is sitting next to her easy chair in the den.

Each morning around 10:00, Michelle, Carol's daughter, receives a text message on her cell phone that says "Mom's okay"—meaning that systems throughout her mother's home were able to determine that she got out of bed, she used the bathroom, her weight had not dramatically shifted, she took her

pills correctly, the gas on the stove is off, and her blood pressure is stable. Michelle uses her cell phone to call her mother and ask her how she's doing that morning, but she already knows that all is well and they talk about Michelle's kids. If Carol had forgotten to take her pills that morning or if something else about her daily living had been abnormal, Michelle would have been alerted and she could have called her mother to help coach her or preventively called for more specific professional health care support.

This use-case is extracted from the Continua Health Alliance website. Continua Health Alliance is a non-profit, open industry organization of healthcare and technology companies joining together in collaboration to improve the quality of personal healthcare. With more than 200 member companies around the world, Continua is dedicated to establishing a system of interoperable personal connected health solutions with the knowledge that extending those solutions into the home fosters independence, empowers individuals and provides the opportunity for truly personalized health and wellness management. http://www.continuaalliance.org/

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

— Mark Weiser, 1952–1999

2 Assistive Living Spaces

2.1 Whispering Things

As the Internet and the World Wide Web (Web) developed, more kinds of increasingly mobile computing devices became connected, and Web servers delivered ever-richer content with which these devices could interact. The next disruptive development consists in getting the majority of Internet traffic generated by "things" rather than by human-operated computers. Indeed, if the last 10 years of technology development were about making it easier for people to exchange information with one another—using Google, Facebook, Pinterest, Dropbox, and so on—the next 10 years will be about making it easy for the physical world to transmit data to the Internet [22]. This movement called **Internet of Things (IoT)**, or more prosaically, **Machine to Machine Communication (M2M)**, is well underway with the **things** being physical entities whose identity and state (or the state of whose surroundings) are capable of being relayed to an Internet-connected Information Technology (IT) infrastructure. Almost anything to which you can attach a sensor—a cow in a field, a container on a cargo vessel, the air-conditioning unit in your office, a lamppost in the street—can become a node in the Internet of Things. In 2008, the number of things connected to the Internet exceeded the number of people on Earth. Figure 2.1 illustrates using concentric disks the amount of things connected to the Internet until today and as estimated for the end of this decade.

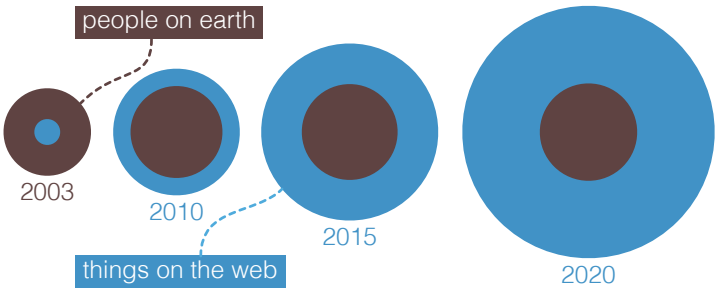


Figure 2.1: Proportion of People on Earth vs "Things" on the Web (Source: Cisco)

IoT belongs to a wider ICT revolution called **ubiquitous computing** (or sometimes **pervasive computing**). Mark Weiser coined the term around 1988, during his tenure as Chief Technologist of the Xerox Palo Alto Research Centre (PARC) [23]. It identifies a post-desktop model of human-computer interaction in which information processing has been thoroughly integrated into everyday objects and activities. It actually relates to a trend towards the deployment of ubiquitous, connected computing devices, unobtrusively embedded in the environment. Pervasive devices are meant to be very tiny or even invisible, mobile or embedded in almost any type of object imaginable. They should also make use of increasingly interconnected networks to enable communication [24]. One goal of IoT and ubiquitous

computing is to get deeper insights and real-time feedback that can help people make faster and better decisions. This obviously ties into big data and there are certain industries—such as manufacturing, healthcare, and public utilities—where this is going to have a huge impact in the immediate future.

2.2 Pervasive Interaction

Human Computer Interaction (HCI) in our daily life is mainly limited to the Graphical User Interface (GUI) paradigm but the increasing interest and research activity towards a more pervasive access to surrounding computing platforms open the door to new kinds of interaction. Complementing the IoT movement, interactivity is getting more natural as it gets embedded in the environment. Tangible user interfaces is a popular example of post-desktop interaction design using the physical world as interface [25]. It relies on concepts like ambient media taking advantage of people's background awareness or graspable media that gives a physical existence to computation using artefacts that can be grabbed, stretched, and so on. The goal is to create an intuitive, effortlessly portable, and constantly available way of interaction between human and machines [26].

While building smart environments, the heterogeneity of interaction paradigms is a rich opportunity but it remains challenging to make good use of it. Most of the time, applications are limited to one device but there is a well recognized need to go further, to enable the plurality of interaction modalities and pick the best of them in real-time. Depending on the environment, the user's activity and the type of service to be delivered, the interaction modality should be adapted. Should the interaction be graphical, ambient, tangible? Should the service be brought at the centre of focus or remain in the background until some attention is given to it? These questions are related to the newly existing pervasive access to information and are to be addressed in the development of assistive spaces. The interaction with pervasive computing platforms represents a difficult task especially when building systems for seniors with dementia. Indeed, the interface should be user-friendly in order to make it easy to enjoy the assistance provided by the smart environment. It should take into account not only users' preferences but also their cognitive capabilities. Moreover, different surrounding computing platforms and devices/sensors are considered as an additional source of heterogeneity.

When designing the interaction with pervasive services for elderly with dementia, there are several aspects to consider. **Polymorphism** characterizes the adaptation needed knowing that physical and intellectual limitations have to be taken into account, as well as preferences in order to help users engage in interaction [27]. The second important requirement is **multi-modality**, the possibility to provide a service through different modalities of interaction. Indeed, to guide the elderly throughout their activities, services should prompt them at their point of focus so as to decrease the trouble and enhance the seamlessness of interaction. We must find mechanisms to determine this point and optimize the interaction modality accordingly. These two aspects define my scope of the **user interface plasticity** challenge which characterizes *"the capacity of an interface to withstand variations of both the system physical characteristics and the environment while preserving usability"* [28].

2.3 Ambient Intelligence (AmI)

The integration of microprocessors into everyday objects like furniture, clothing, white goods, etc.—thus making these formerly silent objects able to communicate with each other and human users—has open the door to the field of Ambient Intelligence (AmI), i.e. "intelligent" pervasive computing. AmI aims at making use of the connected entities in order to provide users with an environment which offers services when and if needed.

One great challenge of such environments is how to adequately address the heterogeneity and dynamic nature of users, services and devices [29]. Key issues of the development of AmI are context-awareness and reasoning: How to identify the context of users, especially the ones with unpredictable

behaviours like elderly with dementia? How to identify and activate the appropriate service within a continuously changing multitude of services [30]? How to provide selected services in a relevant modality of interaction that will engage the user as much as possible? The ultimate goal of AmI is to make ambient services more intelligent and adaptive to the specific needs of their users. The core challenge lies in the mechanisms for the reasoning and decision making related to context-awareness.

There are numerous definitions for **context** and **context-awareness**, but one of the most referenced one is given by Anind Dey who defines context very widely as *"Any information which can be used to characterize the situation of an entity. An entity is a person, a place or an object which is considered relevant for the interaction between a user and an application, including the user and the application"* [31]. Chen and Kotz have classified context into four basic types of context [32]:

- computational context, e.g. the state of resources of the device and of the network,
- user context, e.g. people, places and objects,
- physical context, e.g. luminosity, noise, temperature,
- temporal context, e.g. hour, day, period of the year.

Abowd et al. proposed the notions of primary context such as localization, identity, activity and time, and of secondary context that can be deduced from the primary context and may be used for making decisions at a higher level of abstraction [33].

Conceptually, Gareth Jones sectioned context provisioning in three layers [34]: data acquisition and distribution, interpretation and utilization. Before raw contextual data gathered from sensors and devices can be used, it must be interpreted and evaluated with respect to its accuracy, stability and reliability. The interpretation layer may also combine contextual data from different sources to enhance its reliability or completeness. To make applications able to understand, describe and manage context information, it is necessary to have a context model, which can be defined at the application or the middleware layer. Strang and Linnhoff-Popien identified and compared six types of context models: attribute-value pairs, schema-based models, graphic models, logic-based models, object-oriented models and ontology-based models [35]. They concluded that the object-oriented and the ontology-based models are the most complete and expressive ones, and hence are the most suited for modelling context for ubiquitous computing.

The definition and taxonomy of the context considered in the scope of this thesis is based on the notions by Abowd et al. which have been extended based on the functional sectioning by Gareth Jones as described in chapter 5. The context has been modelled using ontologies as advised by Strang and Linnhoff-Popien (see chapter 4).

2.4 An AAL Round-up

In section 1.3, I defined the scope for the AAL research field and provided a use-case for it. Here I would like to emphasize how wide a domain it is, highlighting the numerous scientific bottlenecks it encompasses. In Appendix A, Figures A.1 and A.2 provide respectively a coarse-grain overview of research thematics within AAL and its subdivision into fine-grain research bottlenecks. AAL research is multidisciplinary and covers aspects from both technological, human and business perspectives. In order to provide relevant solutions, it is important to involve the stakeholders early in the design. This means involve seniors to ensure the acceptance of the proposed solutions, but also doctors and caregivers (professional and informal) who have better ideas about the needs, insurance companies who can provide insights into relevant business models and researchers from fields as various as human factors, HCI, embedded systems, wireless sensing, multi-agents systems, formal methods, knowledge modelling, etc.

Rougier et al., like Sixsmith and Johnson, Alwan et al. and many more, work on the detection of falls in a house using solutions as various as depth maps on video, infra-red arrays or floor vibration sensing setups [36, 37, 38]. It remains however an open challenge to find a generic solution covering a whole house for fall detection without using less-accepted wearable sensors or without driving the deployment cost to prohibitive amounts. Floeck et al. propose a system able to send alerts to a family member in case the movement detected in the house is abnormally low [39]. Stelios et al. perform indoor localisation of the residents to provide medical notifications in the environment [40]. Numerous other AAL solutions are available and, in most cases, robust. However, their scope of usability, mostly focused on security aspects, is generally very narrow. To help the generalization of AAL systems, it would be useful to integrate them in an interoperable way. This would decrease their cost by sharing hardware or even software resources. Leveraging the system in place, we could then provide other context-aware services like reminders or ADL assistance at a lower cost and start to tackle the Quality of Life (QoL) aspects [17]. For example, the Jawbone Up [41] is a fitness bracelet that, among others, provides sleep quality monitoring and could be used in AAL systems. However, there is a one-to-one mapping of the sleep data acquired and the sleep monitoring service. Thus, other services in the environment cannot make use of the bracelet's knowhow on people's sleep patterns and sleep quality. Similarly, the sleep monitoring service has no information about people's agenda, daytime activities or the local temperature, which could be provided by the smart home and are relevant factors with a potential impact on the resident's sleep quality.

The major contributors to AAL research are focused on developing solutions tailored for living laboratories—e.g. the CASAS project [42], the Gator Tech Smart House [43], the iDorm [44], and the Georgia Tech Aware Home [45]. However, many aspects of a smart assistive home are yet to be tackled and the choices made by researchers when developing in laboratories are not the ones they would have made if they had developed on-field [46]. I explain further in section 3.1.1 how AAL research needs to be brought outside of laboratories, outside of controlled environments, in order to address issues related to real-world settings in a relevant manner.

After two decades of research, the main problem faced by the AAL community is to deal with the reasoning that is fundamental to the decision on which activity is being performed. This is especially true when working on the integration of an ambient assistive system to be deployed in real settings, close to industrial requirements. In the field of dementia assistance, the proposed solutions are subject to an increasing reasoning complexity in order to provide specific and personalised services to the residents. Numerous approach have been or are currently investigated, among which rule-based systems with various syntaxes [47, 48, 49], fuzzy logic [50], neural networks [37], games theory [51], Dynamic Bayesian Network (DBN) [52], etc. In this area, an efficient and standardized context model and inference strategy are still lacking, many approaches are not scalable enough for the large-scale deployments necessary to the technology transfer to market. Finally little or no work has studied the combination of the numerous and complimentary reasoning techniques.

Summarising the points above AAL research is at a point where interoperability is a challenge to be tackled in order to increase the reach and impact of the proposed system. Reasoning, especially to perform context inference, remains the very first scientific bottleneck and is researched in numerous directions, sometimes lacking a framework to integrate and combine results from the different techniques. Finally, it is urgent to push deployments outside of laboratories in order to develop and test systems in an environment and in settings that are relevant to the targeted market. Moreover, going out of laboratories will also help increase the involvement of the various stakeholders.

The wise man questions the wisdom of others because he questions his own. The fool questions the wisdom of others because it differs from his.

— Leo Stein, 1872–1947

3 Positioning of this Doctoral Work

3.1 Easing AAL Technology Transfer into Society

3.1.1 A Need for Deployments in Real Settings

As explained in section 2.4, AAL research as it is performed today suffers a strong limitation due to the numerous "living lab" initiatives restraining the development and testing of AAL solutions to semi-controlled to controlled environments.

Related Work: Living Lab Initiatives

CASAS [42] This Washington State University project, under the School of Electrical Engineering and Computer Science, aims at creating an adaptive smart home that utilizes machine learning techniques to discover patterns in user behaviour and to automatically mimic these patterns. Its user interface, CASA-U, is designed as a simulation environment in which all previous and current activities can be visualized and residents are able to navigate through the map of the home, identify and modify automated events or their timings, and provide feedback to the smart home based on automation policies. Participants are actually recruited students or elder people who are performing/playing some tasks during trial periods of a few hours and which can be spread over a few days.

Gator Tech Smart House [43] This project from the University of Florida's Mobile and Pervasive Computing Laboratory aims at developing programmable pervasive spaces in which a smart space exists as both a runtime environment and a software library. It is mainly focused on improving smart homes from a technological point of view by developing software stacks to ensure the representation, communication, collaboration and control of smart technologies called "hot spots". Though its technological contribution is extremely valuable with concepts such as "smart home in a box", the Gator Tech Smart House is not focused on human trials and its very fine sensing granularity makes it less realistic when considering deployments in real homes.

iSpace [44] The intelligent Dormitory, iSpace (previously iDorm), from the Department of Computer Science at the University of Essex, United Kingdom is a real ubiquitous computing environment testbed. It is a two bedroom flat fitted with ad-hoc gadgets, as well as a myriad of networked sensors and effectors to enable intelligent agents to monitor and make changes to the environment conditions. It has been designed to allow research for a wide range of users including able bodied, disabled and elderly populations with a view to maximise the possible benefits of the new technology for all members of the community. Though the environment is interesting, it is still constrained and controlled for testing purpose and experiments are limited to specific sets of behaviours [50].

Georgia Tech Aware Home[45] The Aware Home is the first residential laboratory of its type. It is a 3-story, 5040 square foot facility designed to facilitate research, while providing an authentic home environment. It has two identical floors, each featuring a kitchen, dining room, living room, 3 bedrooms, 2 bathrooms, and a laundry room. The home was originally intended to allow a full-time resident/research participant to live on one floor, while enabling the prototyping of new technologies, sensing, and other research on the other floor. For a number of reasons (including legal), this intended use by long-term residents has not yet been realized. The facility is used for studies involving live-in participants for short periods only (e.g. 1–10 days).

Our Vision

Building prototyping environments is interesting as it helps to involve stakeholders in the design and testing of assistive solutions [53, 54]. However real life scenarios are unlimited and cannot be exhaustively tested in such environments where the number of users and the trial duration are limited. Moreover, the technical usability and reaction of stakeholders cannot be evaluated in these environments since technical problems (e.g. sensors pulled off by patients, bad network connectivity) or design problems (e.g. household routines, multiple users, adaptability to different users) are not predictable during the design process and may not appear in controlled settings. Still, these problems should be identified and resolved during the development phase.

We believe that there is an urgent need to deploy AAL systems in real settings at an early stage of development to ensure their matching the needs and constraints of stakeholders and users' environment. Without this, no meaningful evaluation of the proposed solutions is possible, hence leading to problems concerning technology transfer to the market. It is a primordial constraint to our field if we want to reach a societal impact under a decade. This is where the work described in this thesis differs critically from the state of the art which is mainly tailored for living laboratories. We are proceeding to the deployment in real settings at early stage of technological readiness. In a research domain where 84% of the systems are in their prototype stage [55], this thesis introduces both mechanisms to ease the deployment of complex systems and a stripped down vision of AAL, conceptualised in order to enhance the scalability of deployment of the proposed system. Our solution is targeted to be deployed for commercial operation in five hundred houses in France and a proof of concept has already been deployed in three houses for two months.

In order to deploy in real environments, we started by a semi-supervised environment where feedback can be gathered from professional caregivers able to assess the needs of and impact on elderly people with mild dementia. We conducted a pre-deployment analysis in a nursing home in Singapore, collaborating closely with end-users (dementia residents) and caregivers specialised in dementia care. This collaboration was precious in order to identify the needs, develop and deploy a technical system based on the collected requirements, then evaluate the performance and usability of the proposed solution in real settings. Our approach has involved healthcare specialists in the design process, as recommended by Orpwood et al. [56]. We also involved them in the evaluation of the performance and usability of the proposed solutions in real life conditions. We envision that such a multidisciplinary design approach, supporting a deployment in real life settings is crucial; and that a simple system developed and validated in this way is more relevant and valuable than a well-featured solution proven stable only in a laboratory.

After a 1-year deployment in the nursing home, we have gathered precious experience which highlighted the difficulty to deploy complex systems in large scale, even after putting in place multiple mechanisms to ease this deployment. Therefore we designed a new system with a thoroughly simplified hardware to be deployed in large scale in individual homes in France. Consequently, we had to make more complex and robust our reasoning algorithms to compensate the reduction in the hardware complexity. This transfer of complexity between hardware and software has highlighted new and interesting research bottlenecks which represent the core of this thesis.

3.1.2 Two Complementary Approaches

As explained in the previous section, we have tried to make AAL systems more scalable to deployments by simplifying the hardware complexity and by providing tools to help people deploy the different entities (sensing, interaction, etc.) into the environment. Moreover, stakeholders for such systems actually vary depending on the countries; which means that different adapted solutions should be available accordingly. Indeed, in European countries, the main financing stakeholders are insurance companies for whom placing elders who lose their autonomy in adapted homes is extremely costly. Thus the need is more towards a generic system, reliable, deployable into hundreds or thousands of homes by the technician of a private service provider, and allowing cost to go down with an increasing number of users. In this case, a control/call centre can even be considered. On the contrary, in most Asian countries, the main financing stakeholders are individuals like elders themselves or their family trying to improve the safety while leaving alone so as to preserve the elder's autonomy. In this case, the system should be more like adhoc modules, modularly customizable depending on one's needs, and deployable by the family. Therefore we have identified two complementary approaches.

Top-Down Approach

The **top-down approach** is designed for the Asian market or independent people willing to install their assistive home, with well specified and easy to understand system modules that people can relate to and want to buy. Each module has a specific function, and modules can be composed to tackle specific needs. In this approach, the scope of system is more narrow originally but several optional modules can be plugged to it in order to extend functionalities. The processing is dedicated towards a few scenarios backed by specific sensing devices, and users should be able to control the way the system behaves by programming it through a simple graphical interface. Services are provided in the environment through the most relevant device available depending on the context of the resident. Context inference is done by creating a binding between a set of sensor states and given scenarios. The device to be used for interaction is bound to the context of the user or a part of it (e.g. location). As many scenarios as wished can be added and bound to sets of sensor states and services. Here the key challenge is to make the installation of additional technological or service modules as easy as possible so as to provide an extremely adaptable system. This means that a hardware plug & play must be put in place, and backed by a programmable reasoning engine generating rules for the inference of the context based either on simple rules that the user can graphically provide, or on bindings created by the user between entities, their states and scenarios. Of course all the programmability left in the hands of the users can be instead performed by caregivers, technicians or autonomously by the system when software bundles containing scenarios are installed. This approach fits well with the "there is an app for that" model.

In the top-down approach, assisting services are primarily meant for the elder resident inside the home and his informal caregivers remotely. It can be reminders to perform some activities like taking medicine, or for safety purpose like turning off the gas stove. It can also help the resident physically with home control or cognitively with ADL assistance. On the caregiver side, and very similarly to the Continua Health Alliance use-case (see section 1.3), some level of remote monitoring is provided to ensure the safety of the resident. Daily messages can be sent to confirm a normal activity, and notifications can be used in case the likelihood of an emergency is detected. In this case, the system does not need a 100% accuracy of detection as it is preferred to leave some doubt to be lifted by the caregiver.

Bottom-Up Approach

The **bottom-up approach** is designed for the European market and their insurance companies or any large scale setup of assistive homes by a service provider offering a generic service to many households. It is based on a very robust and reliable hardware system, ensuring minimum complexity/cost of deployment and maintenance and thus increasing the scalability of the solution towards large-scale

deployments. Additionally, it relies on heavier data processing on servers in order to make the most out of available data. E.g. it can perform rule-based inference or employ more complex statistical and machine learning techniques to understand activities performed by residents, gather knowledge about their lifestyle, detect instant anomalies as well as long-term lifestyle evolutions.

The main constraint emerging from the scalability issue is to strip down the technological system to be deployed. Less sensors should be used and especially less types of sensors as each of them needs to be deployed in a specific way, with specific code and specific maintenance. Classical AAL state of the art solutions were found to be hard to deploy even by the engineers who created them [46], hence it seems unimaginable that their setup would be handled by third party service companies—as a marketable solution would require. We have also learned from the stakeholders that video analysis, even using on-camera image processing, is not a well-accepted option and that avoiding to rely on video cameras would increase the acceptance of such solutions. Similarly, reliance on wearable sensors was judged too constraining for a generic marketable solution. For all above reasons, the sensing system we have designed in the bottom-up approach is composed solely of motion sensors attached to the ceiling of each room in the house and a reed switch to detect the opening and closing of the house's main entrance door.

These imposed sensing constraints reduce drastically the contextual information available for the inference of activities. Indeed, it contains only data about the sensed motion in each room of the house as well as the changes of state of the entrance door. Thus the system's intelligence has to be able to cope with this, i.e. the intelligence has to be increased. Moreover, we set as requirement for our solution to provide in real-time an estimation of the activity based on sensor data, such that possibly dangerous situations would be addressed as quickly as possible. It has been judged, together with the stakeholders, that verifying if residents are all right, for example by a simple phone call, would be better than detecting a problem too late; and it might help as well to decrease the loneliness felt generally by elders at home. Beside the challenge of being reactive to the possibly abnormal behaviour of people with dementia, the key scientific novelty of this approach lies in:

1. the inference of activities from minimal contextual information available,
2. the real-time constraint on the activity inference.

In the bottom-up approach, the services deployed are not meant for direct assistance to the elder resident. However, it provides them with the assurance of some level of safety in their daily life. Indeed, behaviours are observed and classified into a high-level hierarchy of activities, other measurables are logged, and statistics are gathered about their lifestyle. Consequently notifications can be sent out in real-time to professional and informal caregivers, or even employees of a call centre, when activities detected are too different from the routine and might indicate that the resident is facing some problem. People are thus empowered to check on the elderly and react accordingly much faster. We have found out while interacting with senior citizens who started to experience some effects of mild dementia that such solutions were very welcomed when their effect on personal independence and safety was explained clearly. Another kind of service can be provided in this approach, namely the reporting of evolutions in the lifestyle of the resident. This can have significant impact on health assessment by doctors as, for example, it could help to foresee some degradation of one's condition and help prevent it. It can also help to detect the deterioration of one's social inclusion, which is nearly impossible for the moment. Here again, the system does not need a 100% accuracy of detection as interpretations are left to the people based on statistical representations and on interaction with the resident.

About the Configuration of Smart Homes

In term of configurability, end-user driven and system driven approaches can be seen as two opposite ends of a scale. An end-user driven approach empowers the users, giving them complete control in managing the system, whereas a system driven approach (or autonomous approach) hands complete

control over to the system itself. In most situations, empowering the user might seem the logical choice; an end-user driven approach not only encourages the creativity of the user but potentially makes the user feel more at ease with being immersed within a complex computer system. Nevertheless, some users may lack the ability or confidence to program their smart environment, even with a simple graphic interface. Furthermore, users with medical conditions may find it difficult or even impossible to interact with computer devices. In these situations an autonomous system is a better choice; it greatly reduces the cognitive and sometimes physical load placed on the user in managing the system. The **Callaghan-Clarke-Chin (3C) model** shown in Figure 3.1 is a socio-agent framework that illustrates this concept [57, 58]. Each quadrant represents one extreme type of usage that may be encountered as a system

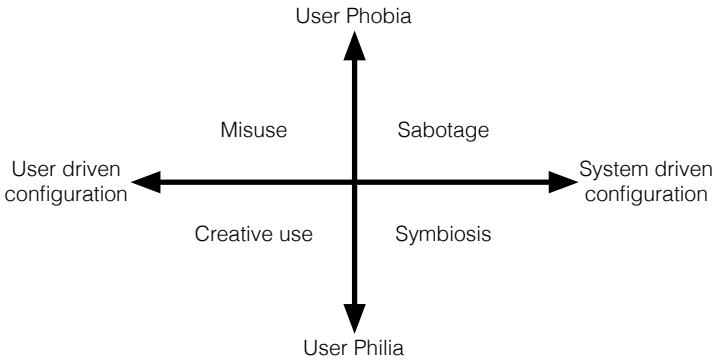


Figure 3.1: The Callaghan-Clarke-Chin (3C) Model

becomes exclusively autonomous or end-user driven, given the user has a phobia (fear) or philia (love) of the system. Ideally we wish to avoid misuse and sabotage of the system and maximise creative use and symbiosis (collaboration) between the user and system. Maximising creative use and empowering the user with an end-user driven system may increase a user's phobia causing the user to misuse the system, albeit perhaps unintentionally. On the contrary, trying to maximise symbiosis by providing an autonomous system may also trigger a user's phobia and cause the user to sabotage the operations within the system, again albeit perhaps inadvertently [59].

3.2 Specific Research Focus: Context Comprehension

3.2.1 Definition of the Research Challenge

One's interest in ambient intelligence lies in the ability of an environment to respond in an appropriate manner to what is happening within it. It is the reaction of a computerised system to a *non-formalised* situation that is intriguing. Such systems are by nature able to instantiate a reaction, even complex, to a formalised and recognised situation, even complex. The true challenge is to provide a formalisation for machines to project situational data and make *sense* of it, i.e. build connections or bindings between it and the rest of the contextual knowledge. This challenge will further be referred as, and hence provide my definition for, **context comprehension**. I divide it into two main aspects:

1. formalising contextual knowledge to project situational data in it,
2. reasoning to connect such inter-correlated formalised knowledge or infer new one.

The question I ask myself in this doctoral work is:

"What strategies can be put in place to provide context comprehension in assistive livings?"

Comprehension is defined in the Random House Kernerman Webster's College Dictionary as the "capacity of the mind to perceive and understand; [the] power to grasp ideas" [60]. We can see it as the process of simultaneously extracting and constructing meaning through the manipulation of sensed situational data. I use the words *extracting* and *constructing* to emphasize both the importance and the insufficiency of the sensed data as a determinant of comprehension. In the field of ambient intelligence, it consists in putting in place a translation mechanism between the sensed representation of a situation and its formalised, machine-readable version. This mechanism would probably be constituted of heterogeneous and complementary strategies, allowing the definition of a formalisation (or model), the naive projection of sensed data into the model, and the inference (explicit as well as implicit) of knowledge into this model. By explicit, I mean inference using deterministic techniques emerging from the formalisation of domain knowledge about human behaviours or sensing systems, e.g. rule-based techniques. In opposition, and complementarily, implicit inference refers to some more computational techniques deriving knowledge from the data itself and its intrinsic patterns, as with statistical or unsupervised machine learning techniques.

From the numerous research challenges presented in Figure A.2 at page 142, this thesis concentrate its research efforts on the reasoning aspects in smart environments. This document will focus mainly on the design of the reasoning engine which allowed us to develop an integrated system that has been deployed in real market conditions. The deployment has been performed in two different scenarios, on one hand in a nursing home in Singapore and on the other hand in three individual homes in France, with the involvement in both countries of several partners from medical, research and engineering background.

3.2.2 Related Work in Context Comprehension

Representing Contextual Information

One can find in the literature important information sources concerning the formalisation of contextual data. E.g. Hong & Landay explain in their position paper the necessary distinction between sensor data and contextual data [61]. Sensor data is not ambiguous but has specific characteristics such as granularity, precision or accuracy which impact their interpretation when inferring contextual data. Quality of Information (QoI) is primordial in the context representation and should be introduced in an atomic and uniform manner for all information. It is the idea defended by Henricksen et al. who propose to label semantically the QoI when known, and to compute or estimate the QoI when we infer [62]. Their proposed labeling mechanism however remains basic and the estimation mechanism is not described. Henricksen et al. also propose an excellent description of the peculiarities of contextual information; among others, they highlight:

- its imperfection, due to the existence of incorrect, inconsistent, incomplete or expired information; which shows the need for a QoI metric,
- its temporal characteristics, information being either static or dynamic, and eventually making up a need for memory,
- its multiple facets; it is indeed necessary to support several representations of a same context at various level of abstraction and capturing the relations between these several representations.

Their work is a good reference for us but remains more conceptual than functional.

Taking Uncertainty into Account

QoI seems to be an unavoidable aspect of the formalisation of contextual information. This leads to the field of uncertainty management, which remains a delicate aspect considering how contextual information is filled with ambiguity. If much work can be found in the literature on this aspect, none

address the issue in a general manner [63]. Hui Lei et al. enable the association of quality metrics such as freshness or confidence [64], but their model lacks formalism. Gray & Salber include quality as a meta-information and describe six attributes: coverage, resolution, accuracy, repeatability, frequency, and timeliness [65]. Their model seems relevant to characterising sensor data (for example), but they do not specify any mechanism to process this uncertainty and take it into account in the inference.

No Leading Reasoning Technique

Concerning the reasoning aspects, the numerous approaches to activity recognition present in the state of the art can mostly be brought down to two main classes: rule-based methods and pattern matching methods combined with preliminary learning.

Rule-based methods are mostly relying on some multi-modal sensor fusion and on the binding of emerging contextual data with activities. Various types of rules syntax have been used, spanning from Domain Specific Languages (DSL) [47] backed by purpose-build object-oriented models to the inference of ontologies using either query-based methods [48] with specific application code or rule-based methods [49]. Artikis et al. developed LTAR-EC (Event Calculus for Long-Term Activity Recognition), an activity recognition system consisting of an implementation of **Event Calculus** dialect, a logical language for representing and reasoning about actions and their effects, in Prolog [66]. Antoniou proposed a purpose-build algebra for reasoning about activities, but it lacks the reliance on a context model, thus rules cannot be generalised or abstracted and the reasoning is implemented in a very scenario-based manner [67]. Fuzzy logic has also been used in both rule-based and learning-based approaches [50, 68]. Rule-based activity inference has its great advantages since it makes it easy to incorporate in the inference domain knowledge gathered by experts. Such knowledge can then be tested in term of inference performance and due to its explicit formalisation, it is easy to update and test iteratively. It is also appreciated for its ability to infer activities concretely without requiring a heavy learning dataset to calibrate the algorithms beforehand. However, it is undeniable that such approach also has heavy limitations. Indeed, being based on the use of explicit prior knowledge, it is impossible to make knowledge emerge from patterns implicitly present in the contextual data; whereas this would be possible using statistical methods or machine learning techniques. It is also more prone to erroneous detection due to the noise present in situational data.

Pattern matching techniques are focusing on the recognition of patterns in sequences of eventually pre-processed sensor data; and require a machine learning step to extract frequent patterns hopefully corresponding to some activities of interest. Several families of pattern recognition techniques are commonly used in AAL systems. One can use statistical states machines like **Hidden Markov Models (HMM)** which are based on the observation of the transition probability between states [69, 70]. Probabilistic modelling methods are also possible, such as **Conditional Random Fields (CRF)** which are a kind of graphical model and classifier which is able to take neighbouring samples into account to predict labels for given data samples [71, 72]. Other learning models have been used, like **Support Vector Machines (SVM)** which is associated with non-probabilistic binary linear classifiers that map observation samples in either side of a hyperplane of the observation space [73]. The machine learning step is mostly performed in a supervised manner using labelled data obtained with various laboratory-bound processes [69, 72, 73]. These processes are however not applicable for systems to be deployed in market conditions [74]. Therefore, being unable to perform learning on deployment sites raises the issue of the validity of an average-resident profile learned in the laboratory. Indeed, some work has been published showing the low similarity between the lifestyle of elders with dementia throughout several homes [75]. Unsupervised learning has been the focus of very recent work and although it shows encouraging results [75, 76], it has strong limitations since the results were obtained with the fine-grain deployment of numerous sensors of different types throughout the house. Other issues are also mentioned, such as the need for a manual selection of the number of clusters to be detected having an impact on the quality of the result and the usability of the solutions. Beside the supervision of the learning, the deployment of learning-based assistive space in people's home neces-

sarily meet a cold-start issue, i.e. the system needs to learn the patterns of behaviour of the residents before it can provide relevant information or services. Finally, these techniques can only perform poor temporal reasoning without complexifying their internal models significantly since there are no explicit rules for computing the intervals.

Reasoning Limitations Due to Sensing Granularity

From our experience on the dataset gathered in the three homes in France, it appears that when the contextual information provided by sensors is limited, the amount of data is not sufficient to use traditional data mining techniques and other statistical methods like HMM. Indeed, they seem to not perform well due to the low granularity of the context information available. As for rule-based approaches, classical implementations need to be made more complex since activities can no longer be bound to the activation of a given set of sensors. Mining and statistical methods having a fundamental issue when using datasets with low granularity (bottom-up approach, see section 3.1.2), and considering the cold-start issue, I have decided to pursue with rule-based methods.

3.2.3 Presentation of the Method

This section summarises the scientific bottlenecks and challenges related to context comprehension. These challenges are mainly emerging from the peculiarities of situational data and from the behaviours of people, especially when suffering from mild dementia. Figure 3.2 provides a graphical overview of this thesis research contributions, structured around the challenges met when trying to formalise and make sense of sensed situational data. These challenges are arranged in the *ring of challenges* and classified according to their cause. E.g. we face difficulties due to situational data being highly dynamic since new sensor events are received every few seconds. Such data is also full of imperfection since sensors can fail to measure the right stimuli, communication can add noise in the data and falsify it, models cannot be exhaustively representing contexts, etc. When considering formal models to store and infer knowledge, the memory of past events is also an issue as it makes the models more complex and increases significantly the processing time needed to parse and infer the knowledge base. Situational data, as a representation of human behaviours is also highly interrelated, either explicitly or through implicit patterns, which must be leveraged in its formalisation. Similarly, it is a multi-faceted kind of information, encompassing multiple possible levels of abstraction, and in which an important domain knowledge from expert can eventually provide insight. To these challenges, one must add the specific issues of the bottom-up approach mentioned earlier, namely the inference of activities from minimal contextual information available, and the real-time constraint on the activity inference. Based on the identification of these challenges, several strategies have been investigated in the scope of this thesis. These strategies are represented in the *ring of contributions*, positioned right on the outside of the challenges that they address. I describe shortly each contribution below.

Context Modelling: Ontological Representation and Quality of Information Metrics

My contribution on the modelling of contextual knowledge is described in chapter 4. In this aspect, we may rely heavily on models available in the literature when it comes to the representation of human behaviours, even for people with dementia. However, the models available classify behaviours extensively into hierarchies but fail to define them in term of attributes that can be computed easily. I prefer a more naive and functional approach to gather technical requirements for further modelling. I have validated my naive and functional approach and an integration with extensive models from the literature remains in perspective work. I also provide a comparison of relevant modelling technologies and explain my preference for the semantic models. In term of QoI, I describe in section 6.4 an atomic way of labelling statements with related metrics through the enhanced representation of triples in ontologies using unnamed graphs.

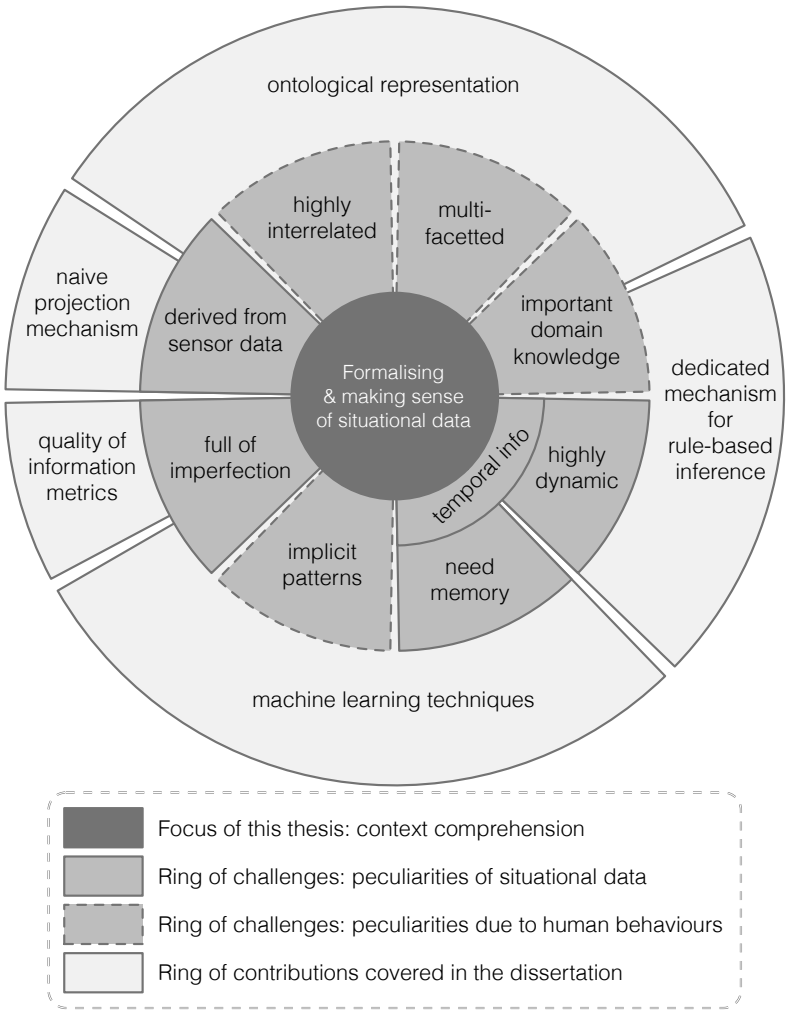


Figure 3.2: Structural Summary of this Doctoral Work

Naive Projection Mechanism

In order to derive contextual knowledge from sensed situational data, we have to project this data into the contextual model. I describe in section 4.4 a mechanism to gather and merge sources of data transparently, and then to automatically write into the model the data with minimal translation. The minimal translation is handled at the modelling level by providing a facet of contextual representation corresponding to the sensed data level.

Explicit Inference: Dedicated Mechanism for Rule-Based Inference

To enable the integration of expert knowledge into the activity inference, declarative methods and especially reasoning with semantic inference engines are well suited. Therefore, after starting with the development of a proof of concept using an imperative approach, I analysed the state of the art on decision support mechanisms with among others rule-based solutions (section 5.2). Consequently, I chose to develop our solution using semantic technologies and build a test-case to gather requirements and make technological choices concerning the specific use of such technologies into an AAL use-case.

This is described in section 5.3. Finally, I provide in section 5.4 the inference mechanisms that I have designed to ensure the context comprehension both in the top-down and bottom-up approaches. Naturally, rule-based methods have intrinsic limitations, for example due to the temporality of contextual knowledge making context models more complex for the designers and developers as well as for the processing by inference engines. I also note that the explicit connections which can be inferred by rules are only the tip of the iceberg: much more data is implicitly correlated and can be made sense of. Natural patterns should emerge from data and be noticed (a-posteriori) instead of being designed (a-priori) by humans. Finally, declarative methods do not take QoI metrics into consideration intrinsically and special mechanisms need to be put in place.

Implicit Inference: Data Driven Techniques

In order to compensate the limitations of declarative approaches and take into account the temporality, the implicit patterns and the QoI of contextual information, I propose in chapter 6 mechanisms to integrate data driven techniques into my rule-based solution. Moreover, one can bring out temporality of the context by doing observations over windowed or transitional representations of data (see section 6.3). Unsupervised machine learning techniques can be used on situational data or contextual information. I design in section 6.2 mechanisms to use these techniques on top on ontological representation of contextual knowledge in an attempt to get the best out of two worlds.

Part II

Semantic Reasoning for Context Comprehension

The most interesting facts are those which can be used several times, those which have a chance of recurring. [...] Which, then, are the facts that have a chance of recurring? In the first place, simple facts.

— Henri Poincaré, 1854–1912

4 Modelling of Contextual Knowledge

4.1 Motivation and Challenges

Smart spaces filled with sensors, devices and computational entities that generate unstructured data over time present a two-fold opportunity for system designers and integrators. The first by acting on the data and transforming it into structured data, so that the information can be extracted, semantically enabled and acted upon by intelligent system modules. The second by building intelligent modules that can process this semantic information and build upon it by inferring more information or making decisions. Semantic web technologies have the potential to provide syntaxes and mechanisms for the representation of structured data and explicit contexts, for the expressive query of knowledge bases as well as their flexible inference by reasoning engines [77].

Smart environments rely heavily on modelling. Architectural modelling is used to support better design and management of these environments. Device modelling is exploited to overcome the integration issues and to offer a shared knowledge on how the environment works and on how it can be controlled. Modelling does not limit itself to the environment's structure and components but it has an impact on the interaction level as well. Interaction modelling focuses on human activities to better interpret the interaction of people with pervasive systems. Human behaviours and their temporal evolution are analysed and linked with activities in order to learn frequent patterns, and possibly infer user needs. The context in which activities occur, e.g. the environment state, time, weather or even people's mood, is modelled to take into account all possible factors influencing human behaviours [78]. The AmI community has defined key research requirements for building intelligence in different environments, e.g. homes, offices, control centres, etc. They identify the need to develop new models with a higher level of abstraction to address heterogeneity, intelligence, innovative management techniques and human centric expressions of personal style [79]. Since architectural and device modelling deal with the components/artefacts that are described for each environment, they are identified as **Lower Level Modelling (LLM)**. By contrast, the abstract modelling employed to achieve intelligence and innovative management techniques is identified as **Higher Level Modelling (HLM)** as it deals with generic human centric expressions. Carreira et al. also proposed a two-layers approach to model environments, *(1)* at the requirement level which is abstract and nearer to the end-user point of view, *(2)* at the implementation level in order to ease the development of intelligent systems [80]. It can be observed that proper modelling can play a significant role in achieving the potential of smart environments. Proper modelling means to be formally correct and semantically valid, enabling applications to intelligently process models. Semantic web technologies can be used to describe smart environments by providing proper LLM and HLM.

I presented in section 3.2.2 the modelling bottlenecks emerging from the peculiarities of contextual knowledge, with among others its imperfection, temporal validity and multiple facets. I believe that a context model providing multiple layers for the representation of the same context would provide enough modularity to address each of these bottlenecks in a separate layer. This approach, which matches and extends the LLM and HLM proposed above, would thus leave to the modules consuming

context information the choice of the most suitable layers of representation. Additionally, I defend the idea that context models should provide a descriptive and parametric description of the context, instead of a declarative description as can be found in most ontological models in the literature. It is not about listing in the most exhaustive way the possible activities and behaviours of elderly people (for example), but rather to define semantic classes of contexts and parameters (attributes) for each of these classes so that applications can make sense of them without prior knowledge. For example, a behaviour or action should be described either as atomic (independent) or as a part of a sequence. Behaviours should also be characterised as safe or dangerous for the elder, or possibly as safe normally but dangerous in coexistence with a context from another specific class. Hence context modelling requires a strong competency towards the analysis and modelling of complex systems.

4.2 Related Work in Context Modelling

As described above, modelling is a key aspect for the provision of intelligence into an environment. It allows a computational understanding of low-level events coming from sensors and other devices, and of higher level events describing the context in the environment. Several modelling technologies can be used therefore. Among them, semantic and specifically ontological representations are seeing a growing adoption and their use is being spread from the web, which they were originally designed for, to various application domains. Several research publications describe how to use such representations to formalise various aspects of ambient intelligence and context reasoning.

HomeML, proposed by Nugent et al., is an XML-based solution defining a standard to represent the information transiting in ambient intelligence systems [81]. The description provided is focusing on the devices and spaces of the environment without taking into account the human factors. Thus, the rules used in HomeRuleML, HomeML's rules counterpart, are based on use-cases linking explicit sensor states to predefined human behaviours [82]. The introduction of a model for human behaviours would bring a deeper, richer semantic representation and enable a reasoning using more abstract rules in order to gain in versatility and accelerate the conception of context-aware applications. Indeed, ontological reasoning may be employed with two different approaches:

- inferring based on object properties (i.e. relations or links) existing between individuals (i.e. instances) in the ontology, as would be the case if HomeRuleML was transcribed into an ontological format,

- or inferring based on the matching between data properties (i.e. parameters or attributes) attached to the different individuals in the ontology, which requires a more complex modelling phase but offers more possibilities and superior flexibility at the application level.

Akdemir et al. formalise human activities as spatio-temporal interactions of primitives and contextual entities [83]. They present a semantic model of human activities based on an ontology populated with entities of the environment, interactions between them and sequences of events identifying an activity. The ELDeR ontology (Enabling Living inDependently of Risks) presented by Rodriguez et al. similarly relies on a sequential description of the different ADL to detect potential risks while elders are performing these activities [84]. These models are functional as they enable the inference of activities and risks based on fine-grain interactions detected in the environment. More particularly, they fit very well with the object-based activity recognition approach. Although, I consider this approach relevant and have deployed systems that uses it, it still requires a fine-grain sensing of the context and does not match the requirements described in section 3.1.2 concerning the bottom-up approach. Hence, I believe that a more abstract model should be proposed to distance developers from using predefined use-cases. Such a model should be extended to formalise human behaviours from a higher point of view as well, perhaps providing a hierarchical classification of activities based on lower level attributes.

The doctoral work of Farah Arab, focused on the modelling of human behaviours, provides an almost exhaustive semantic representation of these behaviours, classifying them hierarchically in term of coarseness of observation, importance towards independence, and statistical time or frequency of observation [85]. This model, written from the point of view of an ergonomist, is very strong on the human factors side. But even though it is an excellent reference of all the possible activities and behaviours, it is lacking in the level of *parametricity* needed to perform autonomous activity inference and risk assessment using algorithms. Indeed, this ontology do not provide any relation between the level of contextual information inferred from sensor data, and the behavioural level declared a-priori. This is a major obstacle to the integration of the proposed model into a system to be deployed for autonomous assistance. I believe that providing a parametric representation where activities are characterized along measurable and classifiable attributes would enable the estimation of the risk by inference instead of its declaration a-priori.

The representation of Paganelli and Giuli uses various semantic models for the monitoring and the assistance of people suffering from chronic diseases [86]. It is well parametrised, albeit more focused on measurable criteria of a person's condition without taking into account behaviours. Hence, the specificity of each person, as well as his preferences, cannot be taken into consideration at the decisional level. My approach is to focus on the parametricity in the modelling of human behaviours in order to ensure the straightforward integration and deployment in real systems. This means that my representation is at a level closer to the contextual information derived from sensor data. Concerning the completeness of the activities described and the consideration of human factors, a further integration between the models provided here and the model proposed by Farah Arab is planned.

4.3 Functional Approach to Context Representation

As part of this thesis work, the main contributions are in the reasoning aspect of the context comprehension. Yet, it is impossible to tackle reasoning issues without taking some time to look into the modelling aspect as well. Based on the observations given above about the related work, I have chosen to build a basic model that is complementary to the models published in the state of the art through its more functional aspect. For a more comprehensive description of activities, we rely heavily on models presented in section 4.2, especially the one from Farah Arab. For a more parametric description of activities, an application has been filed for the funding of a new doctoral thesis, in order to extend the contributions described in chapter 5.

4.3.1 Rapid Introduction to the Semantic Web

The semantic web and its ontological format offer state of the art capabilities for the modelling of domain knowledge or systems architecture. Hence it is an excellent syntactic candidate for the contextual model. Additionally, I expose in section 5.2 my choice to use such technologies from the reasoning point of view. More particularly, I explain why I use the Notation3 (N3) language to build the models and inference rules. I expose below some peculiarities of the ontological representation, and of N3 language, so that the reader can fully understand the following sections.

The mainstream language used by semantic technologies today is the **Resource Description Framework (RDF)**, a general-purpose language for representing information, together with its Extensible Markup Language (XML) serialization. **N3** is a more human-readable superset of RDF. The RDF data model is based upon the idea of making statements about resources (the atomic semantic entities) in the form of *subject-predicate-object* expressions called **triples** [87]. The terms **Terminological Box (TBox)** and **Assertional Box (ABox)** are used to describe two different types of statements. TBox statements provide a conceptual description of a system or domain in terms of classes and properties; hence defining a vocabulary. ABox statements are TBox-compliant statements

about **individuals** constituting this system or domain. TBox statements are sometimes associated with object-oriented classes and ABox statements associated with instances of these classes. Together these statements make up a **knowledge base** [88].

To be more specific, each entity of a triple is called a **resource**. Resources can have various roles, which are defined either in the RDF standard, or the **RDF Schema (RDF-S)**, a set of classes with given properties, providing basic elements for the description of ontologies, and intended to structure resources [89], or the **Web Ontology Language (OWL)**, a vocabulary extension for authoring ontologies [90]. `rdfs:Resource` is the class of everything. All things described in an ontology are resources. `rdfs:Class` declares a resource as a **class** for other resources. **Properties** are instances of the class `rdf:Property` and describe a relation between subject resources and object resources. When used as such a property is a **predicate**. One can divide further the `rdf:Property` class into the two subclasses `owl:ObjectProperty` and `owl:DatatypeProperty`. **Object properties** link individuals to individuals whereas **datatype properties** link individuals to resources from the `rdfs:Literal` class, the class of all **literal** values such as strings and integers.

For the AmI community, the TBox usually provides a vocabulary for the description of an environment through its space units such as rooms, its objects (connected or not), the people using the space and their habits, etc. The ABox of the ontology contains all statements describing the actual environment of deployment in a TBox-compliant manner, i.e. using the vocabulary described above. The ABox representation could be considered as a projection of the reality of the deployment into the semantic model provided by the TBox. It includes the actual house, sensing system, residents and their different activities of interest.

Syntactic Sugars for Notation3

N3 has some syntactic sugars that allow abbreviations and factorizations; for instance the predicate `rdf:type` which is used to state that a resource is an instance of a class can be abbreviated as the letter "`a`" (as in "is a"). Moreover, if several statements are written for the same subject, then predicate-object pairs can be separated by semicolons (";"). Similarly, if a predicate applies several times to the same subject with different objects, then one can separate the objects by a comma (","). Resources of an ontology are always defined in a **namespace** and given a Uniform Resource Identifier (URI) in this namespace. In N3, the `@prefix` keyword can be used at the top of a document to define aliases for the namespaces used in the document. In this thesis, each namespace will be introduced once, when needed, and considered as known for the subsequent sources code extracts (called Sources) where only new namespaces will be explicited.

I provide in the next sections a graphical representation of the TBox, together with a minimal yet functional extract of the ABox for the models dedicated to service delivery and activity recognition. The actual ontologies naturally contain more classes and individuals but are simplified here for more clarity. The full ontologies are available in Appendix E.

4.3.2 Functional Model for Service Delivery

Since this thesis focuses mainly on the reasoning mechanisms and not on the modelling itself, the approach has been to define specific submodels, sometimes overlapping, for the different mechanisms employed in the reasoning. These submodels can later on be integrated together as the full reasoning process is implemented. The first submodel is presented in this section and supports the inference of the services to be provided in the environment based on the detected context of the resident. The context itself is not inferred here. It also supports the choice of the most relevant device for the interaction between the resident and the service provided. The reasoning mechanisms corresponding to this model are described in section 5.4.2.

Terminological Box

Figure 4.1 illustrates the design of the first submodel with its graph representation. For more readability and clarity, only a part of the whole ontological submodel in use is represented, thus containing only high-level concepts related to the application and not to the internal reasoning process. The full version is provided in Source E.1 in its N3 syntax. For this graph and the following semantic models, the legend is given in Figure 4.2: each box represents a class, datatype properties which individuals of a class might have are listed in the dotted extension below the box of the class, and object properties are represented by the arrows between boxes. An arrow between the class A and the class B means that the property has for `rdfs:domain` the class A and `rdfs:range` the class B, i.e. it can link a subject individual from the class A to an object individual from the class B. The arrows with empty white heads are representing a `rdfs:subClassOf` relation between two classes. This submodel will then be populated with knowledge about the actual environment of deployment, users and their profile, services activated, hardware deployed and real-time knowledge derived from the sensor data concerning the activities of the residents.

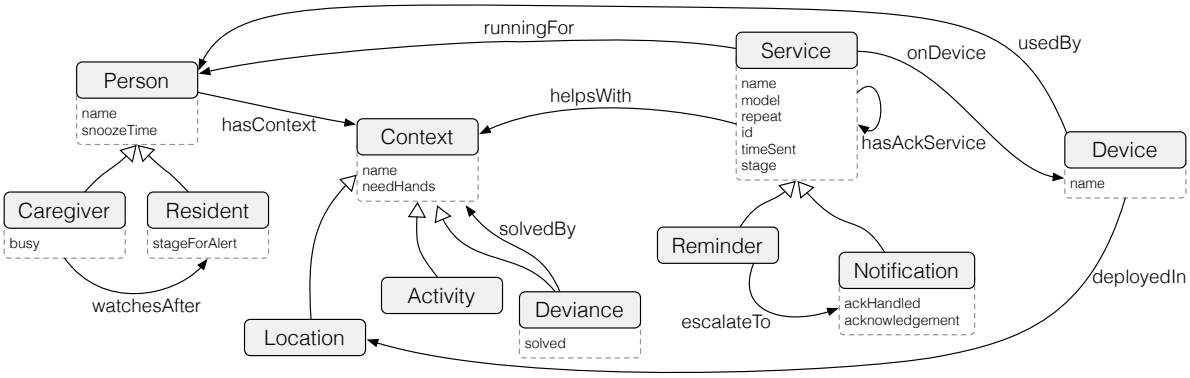


Figure 4.1: Semantic Model (TBox) for the Service Delivery to an Elder in a Smart Space

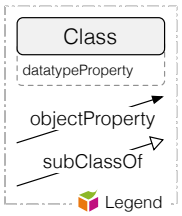


Figure 4.2: Legend for the Graphical Representation of Semantic Model (TBox) in this Thesis

I consider in this model an environment where residents can be helped by caregivers, such as a nursing home or a family house when elders live with kin. Sensors are embedded in the environment, albeit no represented yet, and provide through processing modules some context information such as the location of a resident, his current activity or problematic behaviour (deviance). In this submodel, I bring down the context-aware services provided by the framework to two kinds: **reminders** (sent to residents) or **notifications** (sent to caregivers). For instance, if a person has finished showering but has left the water running, a reminder will be sent to him, which can be escalated to a notification to a caregiver if he ignores it.

The model I designed contains four main classes: `Person`, `Device`, `Service` and `Context`. Individuals (instances) of each of them have specific attributes (data properties) and can be linked to other individuals (object properties). Some classes are refined into sub-classes. For example, an individual of the `Person` class can be a `Resident` or a `Caregiver` to differentiate the role of caregivers. Residents are related to their caregivers by the `watchesAfter` relation and caregivers have the specific attribute `busy` to keep track of their availability. The model can be extended so that residents would have properties to describe their habits, limitations or special needs in order to select the most relevant services. Devices are not characterized much yet, they can be related to their owner/user or location respectively by `usedBy` and `deployedIn` relations. Of course the user and device profiles will be developed further when merging submodels to integrate the presentation engine, which infers how services should be displayed [27].

The `Context` class has three sub-classes: `Location`, `Activity` and `Deviance` used to differentiate semantically the kinds of information that can be collected about residents. A **deviance** refer to the context of a resident going away from the "track" of regular activities and denotes a possibly dangerous or more generally problematic situation that the system should help to resolve by providing relevant assistive service to the resident. Deviances are considered solved when a change of context is observed that corresponds to their `hasSolvingContext` relation. In the same way, contexts can initiate the provision of services that are linked to them through the `helpsWith` relation.

For this submodel, the two classes of services considered are `Reminder` and `Notification`, both have common attributes like `timeSent`, `repeat` or `stage` that are used to handle service cycles. A reminder can become an alert if the situation becomes critic; this is implemented based on the `escalateTo` relation. Alerts have some acknowledgement system enabling the forwarding of notifications among caregivers, as well as the use of greetings messages when deviances have been solved as a confirmation to the resident. Services are delivered based on the `onDevice` relation to the person pointed by the `runningFor` relation.

Assertional Box

The ABox of the ontology contains all the individuals specific to a deployment. These individuals should be **populated** at the deployment site using relevant protocols. For instance, we have designed a semantic plug & play mechanism ensuring a software hot plug of instance, complementary to the existing hardware hot plug provided by various protocols. This mechanism is especially useful in the case of sensors or devices. For the description of the users, dedicated graphical tools can be provided in order to gather detailed knowledge in a comfortable way. Contexts and services should be added in "bundles" by selecting the features of interest from a catalogue or store. As this thesis does not treat these aspects, we usually write the ABox manually and sometimes develop tools for automating this. At runtime, the ABox is automatically modified and populated some more in order to reflect the changes in the environment. The main modifications lie in the addition of object properties between existing individuals of the ontology, and of datatype properties to capture the evolution of information about people, devices or even software instances like services. I provide in Source 4.1 a simplified example of a possible ABox for the present submodel. This example only has one resident, one caregiver and one context of interest with the associated service. A more complete example is given in Source E.2.

Source 4.1: Example of a Simplified ABox for Service Delivery, in Notation3

```
@prefix log:  <http://www.w3.org/2000/10/swap/log#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix env:  <environment#>.
@prefix ske:  <skeleton#>.
```

```
## INITIAL DATA ##
## People
env:patient a ske:Resident;
  ske:name """John"""@en.
env:caregiver a ske:Caregiver;
  ske:name """Tom"""@en;
  ske:watchesAfter env:patient.

## Location or environments
env:bathroom a ske:Location.
env:bedroom a ske:Location.

## Devices
env:speaker1 a ske:Device;
  ske:deployedIn env:bathroom.
env:iphone a ske:Device;
  ske:usedBy env:caregiver.


## SERVICES ##
## Shower too long
env:showerEmpty a ske:Context.
env:showerTooLong a ske:Deviance;
  ske:solvedBy env:showerEmpty.
env:showerTooLongReminder a ske:Reminder;
  ske:helpsWith env:showerTooLong;
  ske:escalateTo env:showerTooLongNotif.
env:showerTooLongNotif a ske:Notification;
  ske:helpsWith env:showerTooLong.
```

4.3.3 Functional Model for Activity Recognition

As explained, the model described in section 4.3.2 is catered for the service selection and provision part of the framework. It will be later on (section 5.4) associated with rules inferring the services to be provided and their most relevant interaction device based on some already-fused-and-enhanced contextual information. The second submodel, which is the focus of this section, is build in order to perform scoped activity recognition from coarse grain sensed contextual data. Hence, it is complementary to the first submodel and it is well adapted to the strong constraints imposed on the system in the bottom-up approach (see section 3.1.2).

Terminological Box

I present in Figure 4.3 the model dedicated to the contribution described here. Note that only the TBox is illustrated here. The graphical conventions are the same as the ones described for Figure 4.1 and given in Figure 4.2. An example of a complementary ABox is given further in N3 syntax.

For this submodel, I consider a person living independently at home and am interested in the coarse grain inference of a few activities from situational data. Hence, the classes `Person` and `Resident` are still used but `Caregiver` is omitted here. The vocabulary for the description of the environment of deployment is extended with the rooms in which residents can be detected (`Room` class and `detectedIn` property) and the objects—only doors at this stage—they can use (`Object` class and `useNow` property). Rooms are part of a `House` where the resident lives (`liveIn`) and are sub-classified according to their function. Spatial organisation is mainly described through the `partOf` property, but this can easily be extended. Objects can optionally be associated with a specific space using the `locatedIn` property. The description of the sensors deployment into the house is also enabled. A `Sensor` has its own `SensorType`, and should either be attached to an `Object` (`attachedTo`) or deployed into an `Environment` like a room

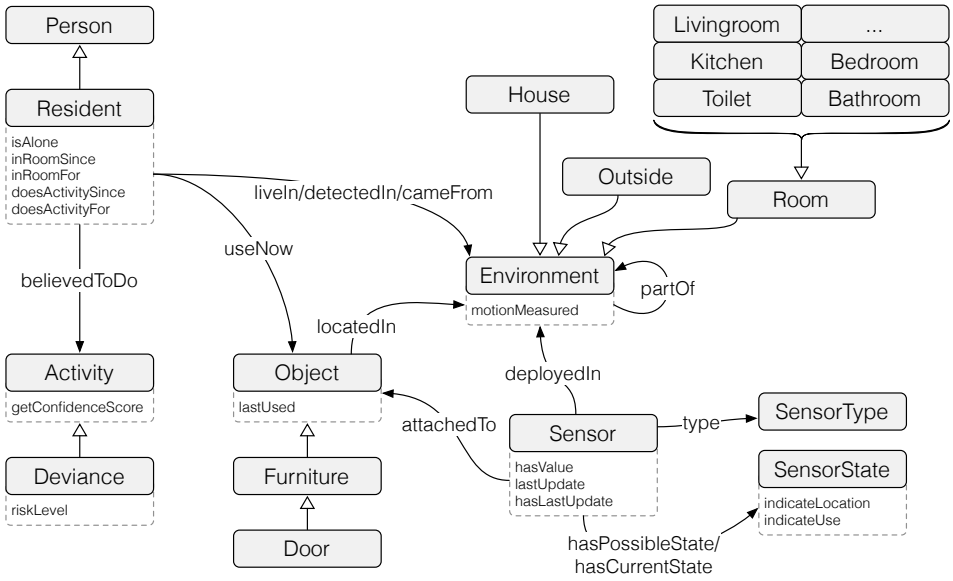


Figure 4.3: Semantic Model (TBox) for a Stripped-Down Activity Inference
Note: Legend in Figure 4.2

(`deployedIn`). All sensor states are declared (`SensorState` class and `hasPossible/CurrentState` properties); the active states providing information about the location of a resident or the use of an object are characterised as such using boolean literals (datatype properties `indicateLocation/Use`). Additionally, a class is provided to describe the activities of interest for which we want to track the residents (`Activity` class and `believedToDo` property). Similarly to the first submodel, potentially dangerous activities are defined into the subclass `Deviance`. Datatype properties are also defined to keep track of timestamps such as the latest activation of a sensor (`lastUpdate`), the latest use of an object (`lastUsed`), the time when a resident was first detected in a room (`inRoomSince`) or believed to perform an activity (`doesActivitySince`). Such properties are also used to store numerical variables, such as sensors values (`hasValue`) or the amount of motion measured in rooms (`motionMeasured`), as well as parameters, like the risk level of a deviance (`riskLevel`). Finally datatype properties are used to store inference results like a boolean to flag the detection of multiple people (`isAlone`), or a decimal confidence score given to each activity in the reasoning process (`getConfidenceScore`).

Assertional Box

As explained in the previous section, the ABox of the ontology contains all statements describing the actual environment of deployment in a TBox-compliant manner, i.e. using the vocabulary described above. In this section, it includes the house, the sensing system, the residents and the different activities of interest. I provide a minimal yet functional extract of the ABox in Source 4.2. The actual ontology naturally contains more rooms, sensors and activities, and is given in Source E.6. Here, similarly to the previous ontology, the population of the ABox is ensured through graphical tools for the description of the environment, the same semantic plug & play mechanism for the sensors, and the choice of software "bundles" for the activities of interest.

Source 4.2: N3 Description of a Home

```
@prefix qol:  <model#>.
@prefix hom:  <home#>.
```

```
## Home (with example rooms)
hom:france a qol:Environment.
hom:notAtHome a qol:Outside;
  qol:partOf hom:france.
hom:homeN a qol:House;
  qol:partOf hom:france.
hom:johndoe a qol:Resident;
  qol:liveIn hom:homeN.

hom:room1 a qol:Livingroom;
  qol:partOf hom:homeN.
hom:room2 a qol:Bedroom;
  qol:partOf hom:homeN.
hom:object1 a qol:Door;
  qol:locatedIn hom:homeN.

## Sensors (examples)
hom:reed a qol:SensorType.
hom:pir a qol:SensorType.

hom:r1_on a qol:SensorState.
hom:r1_off a qol:SensorState;
  qol:indicateUse true.
hom:r1 a qol:Sensor;
  qol:type hom:reed;
  qol:attachedTo hom:object1;
  qol:hasPossibleState hom:r1_on;
  qol:hasPossibleState hom:r1_off.

hom:p2_on a qol:SensorState;
  qol:indicateLocation true.
hom:p2_off a qol:SensorState.
hom:p2 a qol:Sensor;
  qol:type hom:pir;
  qol:deployedIn hom:room1;
  qol:hasPossibleState hom:p2_on;
  qol:hasPossibleState hom:p2_off.

hom:p3_on a qol:SensorState;
  qol:indicateLocation true.
hom:p3_off a qol:SensorState.
hom:p3 a qol:Sensor;
  qol:type hom:pir;
  qol:deployedIn hom:room2;
  qol:hasPossibleState hom:p3_on;
  qol:hasPossibleState hom:p3_off.

## Activities (examples)
hom:getUp a qol:Activity.
hom:goToilet a qol:Activity.
hom:fall a qol:Deviance.
```

4.3.4 No Memory: a Strategic Choice

When observing the models described above, one can realise that no memory has been introduced in the representation, at the exception of the `cameFrom` property in Figure 4.3 that provides a single step memory for the `detectedIn` property. Actually, the model designed should be seen as a contextual snapshot of a house, providing the latest known state of each sensor, the current location and activity of the resident, the current motion measured in each room, and the current state of service activated

in the framework. Although this might be seen as a limitation of the model, I consider it as a feature that was chosen for a strategic purpose. Indeed, disabling the memory in the model allows for a more understandable and less complex ontology, which means that rules are simpler to design and the processing time is decreased since there are less triples in the ontology and no filters are needed to extract the latest state of the house.

The use of semantic models and their corresponding semantically labelled data is also known as **Linked Data**. Sir Tim Berners-Lee, director of the World Wide Web Consortium (W3C), coined the term in 2006 in a design note discussing issues around the Semantic Web project. In the literature, the memory issue and other issues related to processing data generated by stream sources (e.g. sensors) and enriched with semantic descriptions—i.e. following the standards proposed for Linked Data—have been identified and referred to as **Linked Stream Data**. Linked Stream Data enables the integration of stream data with Linked Data collections and facilitates a wide range of novel applications. I can refer here to the work by Anicic et al. about Event Processing SPARQL (EP-SPARQL) [91], by Barbieri et al. about Continuous SPARQL (C-SPARQL) [92], or by Le-Phuoc et al. about Continuous Query Evaluation over Linked Streams (CQELS) [93]. However, this work only addresses memory and other Linked Stream Data issues from the query point of view. Indeed, each team proposes its own extension of the **SPARQL Protocol And RDF Query Language (SPARQL)**, a query language for databases able to retrieve and manipulate data stored in RDF format. Since my approach consists more in using inference mechanisms, i.e. rules, rather than query mechanisms, the work cited does not provide much help. Additionally, no relevant publication was found about inference of Linked Stream Data using rules. In section 6.3, I explain how to compensate the memory-less design by providing modules that are able to build a dedicated memory for specific applications that may need it.

Naturally, a memory-less model has strong limitations. Indeed, historical information could be leveraged in order to infer activities in many use-cases. With the proposed approach, numerous application-specific modules may be integrated to handle historical observations. This would however contribute to increase the complexity of the platform. A trade-off is therefore necessary in this aspect. It is crucial in my opinion to separate open research challenges into independent "bundles" that can be treated separately. Although this doctoral work is closely related to the semantic processing of temporal information (Linked Stream Data), the focus and efforts are placed elsewhere and I try to keep my work independent from this open research challenge. The idea is to perform scoped temporal observations based on specific needs, while keeping for further work the integration of advances in Linked Stream Data techniques.

4.4 Naive Mechanism for Data Projection

As explained previously in section 3.2, the first aspect of context comprehension consists in formalising contextual knowledge in order to project situational data in it. This projection incorporates the need to translate situational data into the computing-ready syntax of the contextual knowledge, i.e. the context model. One must be aware of the shift intrinsically existing between situational data and the actual situation it represents, since situational data is only a measurable facet of the situation that depends on the actual sensing system deployed, and taking into account the uncertainty introduced by sensors which can be faulty. It is therefore primordial not to add any additional shift during the projection of the situational data into the model, for example due to an implicit projection error when the situational data and its domain of projection are too different. I choose to provide in the ontology a level of representation that is corresponding exactly to the situational data so that its translation only consists in adding proper syntax and metadata without requiring any extrapolation. Therefore, when sensor data is received, it may be written "as such" into the ontology based on the identification number of the emitting sensor and its known states, information added in the ontology at the time of deployment.

Although data projection can sometimes be challenging, I provide a naive projection mechanism of

low-level data in the model which is basically handled through the provision of a facet of the contextual model corresponding to the sensed data level. Based on this, mechanisms are provided by specific software modules to gather and fuse data sources transparently, and then to automatically write into the model the data with minimal translation. For example, following the representation of the second submodel presented in section 4.3.3, if the sensor with identification number "pir23az5" sends a "on" state, we simply replace the current triple with pattern {`hom:pir23az5 qol:hasCurrentState *`} by {`hom:pir23az5 qol:hasCurrentState hom:pir23az5_on`}. With such a representation, if any further translation is needed, e.g. between situational data and a higher level contextual knowledge, it is made explicitly using inference rules; thus it is possible to trace semantically the translation errors that might arise. Finally, with this approach and when properly capturing in the ontology the expected behaviours coming from sensors, we can also make a first level of verification of the data received from sensors such that a received value that would not be in the range of possible values may be ignored and the sensor marked as possibly faulty.

4.5 Perspective Work: a More Parametric Context Model

This doctoral work focuses mainly on the inference mechanisms for context comprehension in assistive livings, thus the context modelling aspect is not deeply researched on. However, it seems difficult to present work on ontological reasoning without presenting the models used first. As a matter of fact, the main doctoral contribution is yet to come and I propose here some perspectives for a work that has been submitted as an independent doctoral thematic.

In section 4.2, I exposed the scientific bottleneck constituted by the absence of models formalising human behaviours in smart spaces in an *equilibrate* manner, halfway between human aspects and system aspects, and with a functional parametricity. Such an ontology would provide a vocabulary accessible to human readers as much as to algorithms, allowing the definition and inference of contextual information relying much less on use-cases. This could be leveraged at the reasoning level, enabling a better abstraction of the inference and decision rules ensuring the system's intelligence. It would indeed allow an enhanced parametricity of the inference engine in the short term and ease the design of innovative auto-adaptive context-aware systems in the longer term.

There are two kinds of truths, those of reasoning and those of fact.

— Gottfried W. Leibniz, 1646–1716

5 Designing a Semantic Context Comprehension Engine

5.1 Introduction

5.1.1 A Taxonomy for Context Comprehension

Nurmi and Floréen have categorized the reasoning for context comprehension into four main perspectives [94]:

- The low-level perspective: it includes basic tasks such as data pre-processing, data fusion and basic context inference.
- The application-oriented perspective: where the application can use a wide variety of reasoning methods to process the context data.
- The context monitoring perspective: the main concern at this level is a correct and efficient update of the knowledge base as the context changes.
- The model monitoring perspective: the main task is to continuously evaluate and update learned context, taking into account user feedback.

This chapter is positioned over the context monitoring and application-oriented perspectives, the low-level perspective being handled by the framework described in chapter 8. The model monitoring perspective has not been explored much in this work but an extension on the techniques involved is provided in chapter 6.

5.1.2 Explicit Reasoning

As described in section 3.2.2, there are numerous approaches to reasoning for context comprehension, and no technique has been proven fundamentally superior. Overall, each approach has its own set of advantages and drawbacks, but all seem to be affected when the sensing granularity decreases. I believe, however, that rule-based techniques can cope better in coarse sensing cases by leveraging domain knowledge to compensate the decrease of situational data. It is therefore my approach of predilection as coarse sensing is a major challenge to tackle in the bottom-up approach.

Rule-based (or explicit) reasoning can be used to perform some matching between phenomenons of various natures provided that their description is available in a model. Considering the models described in chapter 4, it is suitable for the recognition of situations (or context) based on the representation of sensed data. It can also match residents' context with relevant services and interaction modalities. It is however an open challenge to provide an efficient, scalable and extensible mechanism to perform such matching.

In this chapter, I analyse the requirements for such a mechanism, provide a comparison of the technological alternatives, and propose mechanisms for the different levels of matching mentioned above based on the technological choice made.

5.1.3 Heterogeneous Needs for the Context Granularity

In the following, context comprehension is widely covered. One must however remember that contextual knowledge is only inferred in order to be used, either by a person or a machine. Contextual knowledge is not the end itself, but rather a mean. Consequently, the level and granularity of contextual knowledge to be reached depends primarily on its targeted use. For instance, a resident or a professional caregiver in a nursing home would probably like to receive reminders or notification when needed—e.g. if medication needs to be taken or if the resident is having issues to perform ADL like taking a shower. In this case, a well scoped albeit fine grain activity recognition is needed, to be used for automatic retrievals. In the case of a resident's doctor, the interest would probably lie in coarse grain context reports containing information like sleep time and quality, amount of movement, shower duration, etc. Such information would provide an approximate lifestyle report, which evolutions could help in assessing the condition of the resident. Finally, a family member might just be contented with a very coarse grain activity report indicating whether the activity of the day is similar to other days or too different, thus letting them know whether things are fine or not.

5.2 Related Work in Rule-based Reasoning Techniques

5.2.1 Imperative and Declarative Paradigms

In order to perform rule-based reasoning, a number of technologies are available, from imperative programming languages such as Java or Prolog, to declarative programming languages permitting a more efficient separation between application logic and underlying models [95]. **Imperative programming** is a programming paradigm that describes computation in terms of statements that change a program state. In this paradigm, the program is composed of one or more procedures, also known as subroutines or functions. Each function creates, modifies or deletes variables that constitute the state of the system represented and processed by the program. Imperative programs usually handle systems modelling and processing altogether, which means the logic specific to an application and the system model being processed are implemented together and little reusability is possible, whether for the model or the logic. On the contrary, **declarative programming** is a programming paradigm that expresses the logic of a computation without describing its control flow [96]. In other word, a declarative program is a program that describes what computation should be performed and not how to compute it. Such program usually defines separately the model to be processed and the application logic, each having their own language of description. The processing itself is then left to the language's implementation. Declarative programming includes a number of better-known programming paradigms, such as constraint programming, DSL, functional programming and many modelling languages.

In an application domain like AmI where relying on a context model is important, I bring down interesting declarative languages to the numerous DSL and modelling languages. Most modelling languages do not allow any inference to be performed and are thus not usable in our case. DSL usually provide non-standard modelling and rules languages. In some cases, as for Structured Query Language (SQL), the language might be standardised, but the vision behind it and hence the practice in the community of users are not articulated around sharing models and data stored. We consequently reach a "silo-based" network of systems where each system defines its own model, stores its own data and protocols are defined to exchange part of this data without ever merging the models, but instead by defining manual mappings between models.

For a pervasive computing application where numerous tiny modules, agents, algorithms are contributing around a shared language and model in a decentralised manner; and hoping to be able to leverage domain knowledge models when available without defining mappings and transferring data between silos; a more open and linked modelling and rules language is needed. Semantic Web technologies offer such a language, in a standardized syntax, and with state of the art modelling and reasoning

capabilities based on Description Logic (DL). The semantic web enables knowledge representation and management. Although computers were originally designed for numerical calculation, it enables them to compute knowledge and comprehend semantic documents or data. Ontologies are seen as a shared understanding of a domain, overcoming specific terminologies by giving more importance to the meaning of information [97]. They not only have the advantage of enabling the reuse and sharing of domain knowledge among several applications [98], but also of allowing the use of logic reasoning mechanisms to deduce high-level contextual information [99].

Adopting an imperative approach to implement context sensitive applications is very robust and requires only a short design phase. However, it brings deep constraints in term of reusability in personalized environments and adaptability in dynamic environments. A declarative approach allows for a more efficient separation between application logic and underlying models describing the use-case and peculiarities of the environment. Although this choice represents an important trade-off on the effort to be put at the design phase, I have decided to use a declarative approach, and more specifically semantic technologies as it enhances the reusability and adaptability of the system. This choice seemed unavoidable when targeting a deployment of more than just a dozen of homes. Although reasoners are the heart of AAL solutions, they do not need to be extremely powerful or complex. Their true requirement is to reach a consistent result in a limited time; and this can be implemented using semantic reasoners. The selection of relevant services and interaction modalities can be performed using semantic matching between the knowledge about users' context derived from sensor events and formalized into an ontology, and respectively services' and devices' semantic profiles. Finally, and as detailed in chapter 4, semantic technologies are perfectly adapted to model contextual information, along with its peculiarities.

5.2.2 Semantic Technologies

A Metadata Data Graph Model

As introduced in section 4.3.1, the mainstream language for semantic technologies is RDF, a general-purpose language for representing information build around a metadata data model [87]. A collection of RDF statements intrinsically represents a labelled, directed multi-graph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model underlying the SQL language among others. One can build additional ontology languages upon RDF, as was demonstrated by RDF-S which basically is a set of RDF classes with given properties specifically designed to provide a more functional and structured vocabulary for the description of ontologies [89].

Similarly OWL extends RDF with a entire family of vocabularies for authoring ontologies [90]. The OWL family contains many species, serializations, syntaxes and specifications with similar names. OWL and OWL 2 are used to refer respectively to the 2004 and 2009 specifications. The W3C-endorsed OWL specification includes the definition of three variants of OWL, with different levels of expressiveness. In order of increasing expressiveness, these variants are OWL Lite, OWL DL and OWL Full, each being a syntactic extension of its predecessor [100]. In the case of OWL 2, profiles have been introduced and are defined by placing restrictions on the structure of OWL 2 ontologies [101]. They are a trimmed down version of OWL 2 that trades some expressive power for the efficiency of reasoning. Many profiles are possible but three are considered more standard.

OWL Sublanguages as Defined by the W3C

OWL Lite OWL Lite supports users primarily needing a classification hierarchy with simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. Since it has a lower formal complexity, it is simpler to provide tool support for OWL Lite than its more expressive relatives.

OWL DL OWL DL supports users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. For example, while a class may be a subclass of other classes, it cannot be an instance of another class. OWL DL gets its name due to the correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.

OWL Full OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. Hence, it is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

OWL 2 EL OWL 2 EL is particularly useful in applications employing ontologies that contain a very large number of properties and/or classes. This profile captures the expressive power mostly used by such ontologies and is a subset of OWL 2 for which the basic reasoning problems can be performed in time that is polynomial with respect to the size of the ontology. Dedicated reasoning algorithms for this profile are available and have been demonstrated to be implementable in a highly scalable way. The EL acronym reflects the profile's basis in the EL family of description logics (EL++), a level that provide only Existential quantification.

OWL 2 QL OWL 2 QL is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems. Using a suitable reasoning technique, sound and complete conjunctive query answering can be performed in logarithmic time with respect to the size of the data (assertions). As in OWL 2 EL, polynomial time algorithms can be used to implement the ontology consistency and class expression subsumption reasoning problems. The QL acronym reflects the fact that query answering in this profile can be implemented by rewriting queries into a standard relational Query Language.

OWL 2 RL OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines. The ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in time that is polynomial with respect to the size of the ontology. The RL acronym reflects the fact that reasoning in this profile can be implemented using a standard Rule Language.

Notation3 (N3): an Assertion and Logic Language for Humans

RDF is used with a variety of data serializations, among which the most common is certainly the XML format. This format is often called simply RDF because it was introduced among the other W3C specifications defining RDF. However, it is important to distinguish the XML format from the abstract RDF model itself. In addition to serializing RDF as XML, the W3C introduced N3 as a serialization of RDF models designed with human-readability in mind. N3 is much more compact and readable than XML RDF notation because it is based on a tabular notation which makes the underlying triples encoded in the documents more easily recognizable. The format, developed by Tim Berners-Lee and others from the Semantic Web community, was motivated by the frustrations resulting from their use of RDF over the years. N3 has several features that go beyond the serialization of RDF models, such as the support of RDF-based rules. It is in fact a full assertion and logic language, a superset of

RDF which extends the latest's datamodel by adding **formulae** (literals which are graphs themselves), variables, logical implication, and functional predicates. The aims of the language are:

- to optimize expression of data and logic in the same language,
- to allow RDF to be expressed,
- to allow rules to be integrated smoothly with RDF,
- to allow quoting so that statements about statements can be made, and
- to be as readable, natural, and symmetrical as possible.

A full description of the syntax extracted from the W3C submissions website [102] is available in section C.2. Beside the extension of RDF by the N3 syntax, a vocabulary of new predicates is also introduced, which lets us write about the provenance of information, contents of documents on the Web, and provide a variety of useful functionality such as string manipulation, cryptographic, and mathematic functions. These properties are defined by the **Notation3 Logic (N3Logic)** specification; they are not part of the N3 language, but are properties that allow N3 to be used to express rules, and express statements about the concepts defined above. Just as OWL is expressed in RDF by defining properties, so rules, queries, differences, and so on can be expressed in RDF with the N3 extension to formulae. N3Logic uses the N3 syntax and also includes a set of predicates. Its vocabulary is the union of the N3 syntax and the set of URI references defined in the `log:`, `crypto:`, `list:`, `math:`, `os:`, `string:`, and `time:` namespaces. While N3Logic properties can be used simply as ground facts in a knowledge base, they are more powerful when taking advantage of the fact that they can actually be calculated. N3Logic includes axiom schemas for each of these terms; reasoners can use these axioms to evaluate formulae and bind variables. These are called built-in functions and they can be used to provide a variety of functionality. The fact that the rule language and the data language are the same gives a certain simplicity (there is only one syntax) and completeness (rules can operate on themselves, anything written in N3 can be queried in N3). This would be broken if a special syntax were added for built-in functions and operators. Instead, these are simply represented as RDF properties. Rules may have full N3, even with nested graphs, on both sides of the implication. This gives a form of completeness where rules can generate rules [103].

Rule Syntaxes and Inference Engines

As described above, the N3 syntax can be used in conjunction with N3Logic predicates used as built-in functions in order to write rules that infer ontologies. In this case, the implementation of the N3Logic specifications is left to the inference engine. In the field of knowledge engineering, an **inference engine** is a program that tries to derive conclusions from a knowledge base, usually by using a combination of **deductive logic** (in which specific examples are derived from general propositions) and **inductive logic** (in which general propositions are derived from specific examples). Inference engines are considered as a special case of **reasoning engines**, which can use more general methods of reasoning. An inference engine is given the ontology to infer and the rules to be applied. Optionally, it may hold internally standard formulae and/or rules that will be applied in the inference. It then provides as output all reached conclusions, eventually with a generated proof of inference. The inference engine commonly used for N3Logic is *closed-world machine (cwm)*, a general-purpose data processor written in Python by Tim Berners-Lee. However, I recommend using the more efficient engine developed by Jos De Roo from AGFA Healthcare called **Euler YAP Engine (EYE)**. EYE is based on **Euler**, an inference engine supporting logic based proofs, and is a backward-chaining reasoner enhanced with Euler path detection. It is interoperable with cwm via N3 and offers much higher performance due to its Prolog optimised implementation and compilation via Yet Another Prolog (YAP), a high-performance Prolog compiler developed at the University of Porto.

Naturally, numerous other rule syntaxes and inference engines are available. Among others, Jena is a Java framework for building semantic web applications on the basis of RDF ontologies. Jena can perform ontology inference using its internal inference engine and a dedicated rule language described fully in section C.1. One can also use the Jena framework with the more standard reasoner Pellet. Pellet is one of the top-tier OWL 2 reasoners, it performs OWL DL reasoning and supports OWL 2 EL profile by default. It notably supports rules in the **Semantic Web Rule Language (SWRL)** syntax [104], a proposed language for the Semantic Web that can be used to express rules as well as logic, combining OWL DL or OWL Lite with a subset of the **Rule Markup Language (RuleML)**, a markup language developed to express both forward (inductive) and backward (deductive) rules in XML for deduction, rewriting, and further inferential tasks [105]. Other independent inference engines are also available, supporting various (sometimes dedicated) rule languages and with uneven levels of expressivity, thus impacting the decidability of the engine. To compare inference engines, other features can be taken into account as well. E.g. RacerPro is a commercial semantic web reasoning system which provides an integrated knowledge repository. RacerPro performs OWL DL reasoning using its own rule format.

5.2.3 Usage in the AmI and AAL Communities

There is much research work on reasoning systems to build context-aware service frameworks. However the notions put behind the word "context" are different for each work, as are the way models or ontologies are used and the approach to their inference [106]. Zhu et al. propose an approach based on DSL to their reasoning engine [47]. They use Drools, a business rule management system to incorporate high level domain knowledge and common sense in their inference system for smart homes. This kind of decision support system is however not optimised for the knowledge maintenance tasks needed in context-aware spaces. Indeed, as Goldberg et al. noted, "decision support rules are not easily accessible to knowledge engineers for maintenance; new rules require programming resources to implement rules and custom data fetches; deployments are not reusable across multiple systems" [107].

Broadened to the AmI community, numerous ontological syntaxes and reasoning engines have been used with specific goals and use scenarios. To name but a few systems, Gaia provides a generic computational environment mixing ubiquitous computing with its physical space of embodiment [108]. It is based on ontologies written in **DAML+OIL**, an early ontological language that was superseded by OWL. Gaia can perform rule-based inference as well as employ machine learning techniques, the rule-based part being implemented with first-order logic rules given priorities such that only one rule can be applied at a time. The Context Broker Architecture (CoBrA) is an infrastructure supporting the development of context-aware agents or applications by providing raw and inferred context information via its "context broker" which is handling the context inference in a centralised manner [109, 110]. The context broker uses several layers for the reasoning, the first being implemented using Jena as a rule-based engine inferring OWL ontologies. Semantic Space is yet another infrastructure for building context-aware pervasive applications around a shared ontological knowledge base [77, 99]. Several layers of ontological representations are used in the framework and links are made with existing and well-accepted ontologies such as friend-of-a-friend (FOAF). Semantic Space uses two complementary context reasoners based respectively on description logic and first-order logic, both implemented using Jena with forward chaining rules. SAMOA is a framework to support the creation of semantic context-aware social networks around mobile users, based on physical proximity and sharing of common interests [111]. SAMOA provides context-aware social matchmaking services which are based on a semantic matching of the profiles of places, people and their discovery wishes. It relies on the Pellet DL reasoner to implement the matching algorithms involved. OWL-SF is another semantic service framework dedicated to ubiquitous and distributed services [112]. It is build around an OWL representation of context information and uses both an RDF inference mechanism implemented with Jena and OWL-DL reasoning responsible for the classification and answering of OWL-DL queries implemented with RacerPro.

More focused on AAL, Meditskos et al. studied the combination of Jena and Pellet into a practical reasoner for the OWL 2 RL profile that combines the forward-chaining rule engine of Jena and the Pellet DL reasoner [113]. More recently, the same authors proposed for the Dem@Care European project an ontology of activity patterns where activities are defined in a composite manner, partly as models and partly as rules expressed in SPARQL based on the SPIN extension [114]. SPIN is an RDF representation of the semantic web query language SPARQL that enables rules using SPARQL `CONSTRUCT` queries [115]. Chen et al. used Euler to perform rule-based inference of activities based on a knowledge driven approach [116]. Foo et al. proposed an ontology-based system to monitor and react to agitation for people with dementia where an OWL ontology is held into the Sesame framework for querying and analysing RDF data [117]. Though Sesame does not provide any inference mechanism, context reasoning is proposed through the use of DL Implementation Group (DIG) classifiers and specifically RacerPro. DIG specifies a common interface for DL reasoners and allows a variety of tools such as ontology editors to make use of these reasoners [118].

5.2.4 Conclusion

The main observation emerging from the analysis of the literature concerning reasoning systems for context aware applications is the numerous and heterogeneous technologies involved. I have explained why knowledge-based approaches are more suited for the coarse sensing of situations, I also noted the better maintainability and extensibility of semantic technologies compared to other knowledge based approaches such as DSL. Still, many languages, rule syntaxes and reasoning engines are possible and have been used in the AmI and AAL communities, each offering different levels of expressivity, decidability and in a more general manner practicality of use.

The numerous semantic web tools available have very disparate characteristics and performance. Moreover, benchmarks for such tools have limitations and a more qualitative observation on the requirements is needed to give useful hints to developers [119]. As explained by Luther et al., "choosing the appropriate combination of a reasoning engine, a communication interface and expressivity of the utilized ontology is an underestimated complex and time consuming task" [95]. I provide in the next section a detailed study of the requirements on semantic reasoning engines in order to be used efficiently in pervasive context-aware frameworks. I also propose a comparison of selected engines with regard to the requirements gathered. Responding to the complexity in choosing a suitable reasoning engine for our use, I contribute here to the community by putting in place an appropriate test-case in order to give useful hints to AAL researchers and developers.

5.3 Which Inference Engine for AAL?

As explained above, it is a challenging and important task to build an appropriate test-case in order to compare quantitatively and qualitatively inference engines to make informed technological choices and provide useful insight to the community. I spent a year putting in place an appropriate test-case in order to give useful hints to researchers and developers, curious about the requirements and recommendations on inference engines that may be used in AAL use-cases. This particular study was done in a nursing home in Singapore, at the early stages of deployment described in details in section 9.3. I detail below the results of the study and the choices that emerged from it. Section 5.3.1 describes the requirements gathered for reasoners and ontology/rules syntaxes to be efficiently integrated in AAL systems. Section 5.3.2 provides a comparison of some reasoners with regard to the suggested requirements and my final choices and recommendations.

5.3.1 Requirements Gathering

In this part, I try to highlight the "must have" features of a semantic reasoner in order to be used efficiently into an AAL system.

Negation As Failure (NAF) and Retractability of Knowledge

This requirement emerges from a fundamental pre-requisite of Semantic Web. Indeed, Semantic Web is based on DL which is monotonic and adhere to the **Open World Assumption (OWA)**, i.e. conclusions drawn from a Knowledge Base (KB) must be based on information explicitly present in the KB. This differs from classical reasoning formalisms, usually non-monotonic, that apply the **Closed World Assumption (CWA)**. Under this assumption, all non-provable expressions are assumed to be false, which is known as **Negation As Failure (NAF)**. NAF is a non-monotonic inference rule in logic programming, used to derive *not* p (i.e. that p is assumed not to hold) from failure to derive p. Note that *not* p can be different from the statement $\neg p$ of the logical negation of p, depending on the completeness of the inference algorithm and thus also on the formal logic system. NAF has been an important feature of logic programming since the earliest days of Prolog.

The choice to rely on the OWA is natural considering the envisioned applications of the Semantic Web. Indeed, when scaling to the Web, the absence of a piece of knowledge should not generally be taken as an indication that this piece of knowledge is false. However, there are also application scenarios where the CWA, or at least the partial closure of the KB, is a more natural choice. Such scenarios can for example occur if ontology-based reasoning is done in conjunction with data stored in a database. Databases are often considered to be complete, thus statements not in the database should be taken as false. Knorr et al. provide the following example where a combination of OWA and CWA is desired [120]. Consider the large case study described in [121], containing millions of assertions about matching patient records with clinical trials criteria. In this clinical domain, open world reasoning is needed in radiology and laboratory data. For example, unless a lab test asserts a negative finding, no arbitrary assumptions about the results of the test can be made. That is, we can only be certain that some patient does not have a specific kind of cancer if the corresponding test has a negative result. However, the closed world assumption can and should be used with data about medical treatment to infer that a patient is not on a medication unless otherwise stated. The work described applies only open world reasoning but claims that the usage of closed world reasoning in data about medical treatment would be highly desirable and that the combination of OWA and CWA is an open problem in their work [121].

In assisted living spaces, contextual information is evolving and a detected situation is valid for a short period of time only. The most needed feature for a reasoner to be used in AAL is the possibility and ease to retract information, and derive its negation, both for asserted and inferred information. It has not been ignored that removing pieces of knowledge from an ontology is traditionally not a good practise. However, there are several reasons to support this choice in the targeted use-case. Most importantly, we do not want to overload the triplestore with deprecated triples having an older timestamp. We would also prefer to avoid dedicating processing time to select triples with the newest timestamp. To support this choice, I propose a mechanism such that the existence of a "thing" is never removed from the ontology. In other words, triples defining a new class, property or individual will never be removed. In an ontological graph, nodes are therefore anchored, while branches can be changed freely to represent the current contextual information available. E.g. if a resident walks to another room, the triple `ns:resident aal:locatedIn ns:kitchen` is replaced by `ns:resident aal:locatedIn ns:bedroom`, whereas the "existential" triples `ns:kitchen rdf:type aal:Room` and `ns:bedroom rdf:type aal:Room` remain untouched. This approach leverages the relatively short life-cycle of statements in the KB as compared to resources. Indeed, if statements are dynamically added and removed according to the evolution of the situational data, resources are more of a static entity in our use-case. For instance, classes and properties define the model of the KB and are created at the design level; while individuals characterise the environment of deployment (e.g. spacial organisa-

tion, sensing hardware, people) and are created at the deployment level or when the environment is structurally modified.

We would like as well to retract inferred triples easily, when the conditions necessary to their inference are not fulfilled anymore. Using a graphical analogy, let us consider an asserted piece of knowledge as a node, and the knowledge inferred partly from this node as new nodes branching downwards (unidirectional relation) from it. The expected behaviour is that if a node is removed, which means the represented piece of knowledge is withdrawn, all nodes branching downwards from it should be removed as well. Although it is easy to use SPARQL queries, among others, to update the asserted triples in the ontology, the automatic removal of inferred triples as described above is more complex. Due to the OWA of RDF, and SWRL being built on top of RDF, SWRL rules can be written to add new triples into an ontology but not to retract triples from it [122]. If one tries to assert a new value for a property, two values will then be coexisting. Optionally, the property can be characterised as *functional* to indicate that only one value is possible. However, this does not mean that the property will be updated but rather that the KB will become inconsistent when the two values are coexisting.

Some reasoners—e.g. Pellet as mentioned above—have a rule syntax that is not expressive enough to allow the retraction of knowledge. Others—like EYE or Prolog—provide a syntax for partial closure and negation (**Scoped Negation As Failure (SNAF)**) which can however be used only in rule antecedents. It is thus possible to verify the negation of a statement but not to infer it. In both cases, one must annotate a part of the knowledge as deprecated and write external queries (e.g. with SPARQL) to filter it out. Finally, some reasoners—e.g. Jena, RacerPro—use rules that can remove triples directly. In the three cases, it is needed to manually retract knowledge inferred from the asserted-then-retracted "nodes" (same graphical analogy). I did implement some inference rules dedicated to cleaning the ontology after a retraction happened. Although it is working well, this increases the complexity at design level and naturally decreases the performance of the system. I finally realized when experimenting with EYE that even though its expressiveness did not allow the retraction of knowledge, the reasoner having *no live state*, the knowledge previously inferred from now-deprecated data is simply not inferred anymore. The live state of a reasoner is the state in which the reasoner remains in between two inferences. It is used to keep in memory the inferred state of an ontology, thus inferred knowledge does not need to be inferred again. In our use-case however, I'd rather use a rule engine with no live state (i.e. no memory), as it is then only needed to care about information being asserted or retracted, and the rest is handled automatically, similarly to the "downwards branching nodes" approach described above. I observed that reasoners often implement complex mechanisms to infer knowledge with incremental updates; but I found more suitable, in the AAL use-case, to use a *naive-only inference* (i.e. with no live state) like what is provided by EYE. To summarise the life-cycle of statements in the KB when adopting EYE, asserted statements live as long as they are not retracted, they are not affected by other retractions than their own; inferred statements' lifespan is conditioned by the life of their inference conditions, if the conditions have not changed, the statement will be inferred again, whereas if one of the conditions was retracted the statement does not hold anymore.

In conclusion, although combining the OWA and CWA is desirable, it has fundamental implications as well as an impact on the decidability of the inference. I find that the limited and more conservative use of SNAF in EYE may be a good middle ground. Moreover, EYE's peculiarity of not having a live state, although counter-productive in some use-cases, seems helpful with maintaining a reasonably sized ontology as needed in semantically driven context-aware systems.

Processing Efficiency

Taking into account more common applications of the Semantic Web in the cloud, one can easily imagine having reasonable resources to process knowledge. However, in the AAL use-case, it may be necessary to embed the reasoner into a low processing power and low power consumption device, so that this device can be easily integrated anywhere in a house. E.g. the reasoner used for our deployment in Singapore runs on a tiny debian machine whose CPU turns at 500MHz with 500Mb of RAM, and

consuming only 5W. Moreover, the data inferred is highly dynamic; unlike traditional web data which is updated by human users with a low frequency, context information is derived from ambient sensors' activations representing people's behaviour in real-time, therefore the update rate is below the minute. Finally, the inference is used to compute which services should be provided in the environment and with which interaction modality depending on users' action. Users should have the feeling of an almost instant response time, therefore the inference period has to be set very low (i.e. high frequency). These three peculiarities of the AAL use-case form the requirement on the processing time, thus on processing efficiency.

Scalability of Inference

Our service framework for assistive living is usually tested in a few rooms, or at most a few houses. But it is difficult to estimate the extent of the monitored/serviced space in which it will be deployed once AAL technologies get a larger adoption. Let us consider that we are deploying at the scale of a whole building. A standard public housing block holds over 90 apartments in Singapore. Should we plan to have one reasoner per room, per apartment or even per building? With regard to the Linked Data philosophy [123] and due to the interconnection of events inside the building, it makes sense to think of a reasoner per building to be able to draw relations between the data from all apartments. Considering the numerous smart cities initiatives that suggest the extension of smart spaces to the city level [124], the number of triples to be considered at the reasoning step might suffer a genuine explosion. Thus we have to include a requirement on the scalability of the system, e.g. reasoners inferring with quadratic cost should be prioritized compared to those running with exponential cost. Although I expressed in the beginning of this section some reservations towards semantic reasoners' benchmarks, I give in section 5.3.2 some figures to compare selected reasoners in this perspective.

5.3.2 Comparison on Inference Engines

The previous section introduced the main requirements I believe are to be fulfilled by a semantic reasoner in order to be used efficiently in the AAL use-case. Some are immediate necessities like the retractability of knowledge or the processing efficiency, others are key challenges enabling larger scale deployments like the scalability. Below, in a bit of a narrative manner to preserve the thought process I had during the study, I analyse the suitability of four available reasoners that I have selected and tried as part of this doctoral work.

Jena: The Predominant Semantic Framework

In the AAL community, the **Jena** framework [125] is predominantly used. This might partly be explained by a certain unawareness about the possible alternatives, as well as by the apparent ease of use of Jena compared to other reasoning engines. Indeed, Jena has a few advantages compared to its rivals. For instance, it probably provides the most complete Java Application Programming Interface (API) for building semantic applications. Unlike most of the other alternatives, Jena was designed to be used in Java and its principles of programming are therefore more natural for a programmer getting a first hand on semantic web technologies. Actually, taking into account the possibility to implement Java built-ins that can be called directly from an inference rule, one might not even realize the differences induced by the declarative reasoning paradigm. Moreover, Jena comes fully featured with, among others, an API to build, populate and modify ontologies, an inference engine using its own rule format, and an integrated SPARQL query point.

Despite all the above, I am having mixed feelings about my experience developing using Jena and would like to express some reservations about it. In fact, without having to load the ontology much, I could observe some inconsistencies in the reasoning when trying to use several rules to collaborate on one decision. I could indeed infer the same ontology with the same rules consecutively and obtain different

results. This is of course not the general case but could be easily observed when introducing some dependencies between the antecedent of a rule and the conclusion of another, which is something quite basic. While trying to explain this flimsy behaviour, I found out that Jena's integrated inference engine, although more expressive than OWL-DL, was actually providing an incomplete OWL-DL entailment [126]. Using Pellet as an external reasoner through the DIG interface was advised. Consequently I started to compare the features of available semantic reasoners and their ease of use in our peculiar use-case. My motivation towards this particular part of the doctoral work grew as I met researchers in the community interested in finding an appropriate reasoner.

Pellet: The Famous Alternative

Since Jena can be used through DIG with **Pellet** [127] as an external reasoner, it lets developers switch reasoners while keeping the system infrastructure in place. They do not need to implement again the non-inference-related modules, such as the module updating the ontology depending on sensors inputs. Thus, Pellet is a popular inference engine that is often adopted to replace Jena while being combined with it since Jena remains the central component of the semantic framework although it does not handle the inference anymore. Through DIG, Pellet can also be used directly from Protégé, a popular ontology editor, making it a valuable tool at the ontology/rules design level. Moreover, Pellet's rules are based on SWRL which makes it interchangeable with some other inference engines at the rule level. I logically decided to try out Pellet, however as explained in section 5.3.1, SWRL does not support the retraction of triples and makes it difficult to be used in the AAL use-case. I consequently abandoned Pellet due to basic expressivity issues.

RacerPro: The Fully-Featured Commercial Option

I then searched a reasoner able to tackle the knowledge retractability issue (see section 5.3.1) and found **RacerPro**[*] [128], a commercial reasoner with add-ons to the W3C recommendations. RacerPro provides an integrated knowledge repository with custom optimizations to enhance performance and, more essentially for us, to increase the expressiveness of inference as it enables the retraction of triples from the repository. Hence, although it is necessary to write rules dedicated to retract triples in order to clear the ontology, the system is at least functional. Other than being a closed-source shareware, RacerPro has its own downsides emerging from its expressive rule/query language. Indeed, this language is actually the most complex that I have used, which increases significantly the workload at the implementation phase.

EYE: The Lightweight "Naive" Reasoner

While facing implementation difficulties with RacerPro, I stumbled upon another alternative with **Euler** [129], and more specifically the EYE implementation by Jos De Roo from AGFA Healthcare who is also involved in the W3C Semantic Web Activity Working Group. EYE is notably using N3, the most human-readable RDF syntax. It has the advantage to be among the fastest reasoners I found that had a full OWL-DL entailment, and it is also the most lightweight of the reasoners I selected. However, if EYE provides a syntax for SNAF, it can only be used in rules antecedent and I therefore faced again an expressivity issue related to statements retraction from the ontology. Despite this limitation, I realized that EYE providing a *naive* inference, the issue could in fact be bypassed to some level as I explained in section 5.3.1. My current choice remains EYE for the reasons stated above, and the remainder of this chapter has been designed and implemented using it.

[*] I would like to express my gratitude to Racer Systems for according me a free educational license of RacerPro following their effort to support education and research.

Synthetic Comparison

I have described above the selection process I went through and highlighted the advantages and drawbacks for each selected reasoner. Table 5.1 summarizes the aspects taken into consideration with some key specifications for each of the four reasoners. The first half of the table provides a good represen-

Table 5.1: A Comparative Table of Semantic Reasoners

	Jena	**Pellet**	**RacerPro**	**EYE**
OWL-DL entailment	incomplete	full	full	full
Rule format	own, basic & built-ins	SWRL	own, powerful	N3 & restricted built-ins
Retractability	yes	can emulate stateless	yes	stateless
Ease of use	average	easy	complex	easy
Response time for 100 triples*	783ms	442ms	∼503ms	4ms
Response time for 1,000 triples*	29,330ms	38,836ms	∼44,166ms	40ms
Response time for 10,000 triples*	out of memory	out of memory	out of memory	436ms
Scalability	Very limited	Average	Limited	Good
Size (download)	22.3Mb	24.3Mb	60.3Mb	12.9Mb
Licensing	freeware, open source	freeware, open source	shareware, closed source	freeware, open source

*Figures extracted from the linear relationship benchmark [130] for Jena, Pellet and EYE (cwm also available). Cross-integration of RacerPro through a comparison with Pellet [95].

tation of the engines' expressiveness, the ease of use being purely qualitative, subjective, and based on my hands-on experience. It is interesting to note that the languages closest to the W3C specifications are the ones I found the most straightforward to use. Based on this first half, Pellet and EYE appear to be the two best options (when considering that the naive reasoning style of EYE can be emulated for Pellet). I refined then my analysis with more quantitative specifications (second half of the table), addressing in a way the concerns about processing efficiency and scalability of inference described in section 5.3.1. Despite the reservations I hold about such benchmarks, we observe in the response time the superiority of EYE. Finally, the qualitative classification of engines' scalability is given subjectively, taking into account the response time profile, the ease of use and the inference completeness (OWL-DL entailment). These multiple aspects and requirements taken into consideration, I have argumented and explained my choice of EYE as the premier inference engine used in our context-aware service framework. To be specific about the scope of this choice and recommendations, I would like to highlight that EYE has two advantages applicable to any use-case: its processing efficiency due to its optimized implementation and compilation based on YAP, and its human readable formalization language using N3. However, EYE is a naive (memoryless) reasoner, which is crucial from our perspective but might be counter-productive in many applications. Here lies the main trade-off in my choice.

At this point, one might also wonder about the level of reasoning chosen in my implementation. Using EYE for the inference, developers are able to use any subset of rules catering to their specific needs. The chosen subset simply needs to be written in an N3 file and fed to EYE when launching the inference. My implementation is so far using a subset of OWL 2 RL, but I might choose to use rules from a higher level of reasoning if needed in the future.

5.4 Rule Design for Context Comprehension

5.4.1 General Concepts of The Rule Design

Following my design principle, the rules described in this section can be interpreted according to the **"Data-Information-Knowledge-Wisdom" (DIKW) hierarchy**. This hierarchy, which refers loosely to a class of models for representing structural and/or functional relationships between levels of formalisation of a domain [131], has been defined as follows by D. P. Wallace [132]: *Data* is a nonsensical stimuli providing a subjective observation that is meaningless and mostly useless on its own; *Information* is data endowed with meaning and purpose, obtained by the fusion of multiple sources of data inferred with common sense, making it valuable; *Knowledge* is a fluid mix of framed experience, values, contextual information, expert insight and grounded intuition. There is less reference to *Wisdom* which has a more debatable meaning, hence I choose to downplay its role in the hierarchy. To summarize, I consider that the rule-based inference of residents' context from sensor data should be seen as the structuration and linkage of *situational* data into *contextual* knowledge. In our peculiar domain, *Data* is a meaningless (no semantics) and mostly useless (redundancy) signal coming from sensors. *Information* is non-redundant and has attached semantics since it is available in an ontological form; thus it is more useful and calculable. *Knowledge* is inferred from *Information* and formalises the contextual information at a higher level, closer to how a human being would formulate it. The main difference between my consideration and the definition by D. P. Wallace is that I choose to fuse sources and incorporate common sense one stage later, i.e. at the inference of information into knowledge.

The inference itself, based on DL, is composed of a set of first-order rules like "*antecedent (i.e. conditions)* $\Rightarrow$ *consequent (i.e. conclusions)*". Such rules can be written in many ways and at disparate levels of abstraction. Here lies one of the challenges of such systems: we seek to design rules that are simple enough to enable fast deployment, iterative design and collaboration between rules; but complex enough to be generic, handle complex situations and keep the number of rules limited. The direction I have been taking is to characterize individuals in the model with parameters that capture key information and can be used in rule antecedents to perform the inference. The rules' conditions are then set on the parameters of instances rather than on the instances themselves, which abstracts the reasoning and tends to replace implementation of rules by their *implementability*. In summary, I try to write abstracted rules that rely on a better model parametricity.

In some cases, such as for the selection of an interaction device based on the context of the user, a good level of parametricity can be achieved and abstract rules can therefore be written. In other cases, the parametrisation of the model is more challenging and other techniques must be used. Let us consider the simplified example below:

$$\forall\ Service\ s,\ Resident\ r,\ Location\ l,\ Device\ dc,\ Activity\ a,\ Deviance\ da$$
$$(r\ hasContext\ da) \wedge (s\ helpsWith\ da) \Rightarrow (s\ runningFor\ r)$$
$$(s\ runningFor\ r) \wedge (r\ locatedIn\ l) \wedge (dc\ deployedIn\ l) \wedge \neg(dc\ fitted\ false) \Rightarrow (s\ onDevice\ dc)$$
$$(r\ hasContext\ a) \wedge (a\ needHands\ true) \wedge (dc\ handheld\ true) \Rightarrow (dc\ fitted\ false)$$

In this example, three rules are used. The second and the third one, handling the selection of a device of interaction, are well parametrised. Indeed, the device is selected based on its location in the second rule, and on its ability to be used hands free in the third rule. For the first rule however, the model parametrisation is more challenging as it concerns the comprehensive representation of the residents' context. As explained in section 4.5, the context model designed as part of this doctoral work is kept simple and its parametrisation of the context is insufficient to write abstract rules inferring the services to start based on the residents' context. In this specific case, services can be directly linked to a given context since they are designed to respond to a given context. This is how we can still, in the first rule, write an abstract rule which, although less parametrised and thus less adaptive, performs the selection of services to provide to the residents.

When no obvious parametricity can be incorporated in the model and no abstraction of the rules can be achieved, we must find mechanisms to ease the design of rules and keep it iterative, while preserving the collaboration between rules. This is for example the case when we want to infer the lowly parametrised context of residents based on situational data gathered from sensors. It is even truer when we place ourselves in a bottom-up use-case where only drastically limited situational data is available. First, I believe that a scope of inference must be defined in order to preserve a good control on the number of rules while ensuring that all the necessary situations are being addressed. We must then find a way to leverage as much of what can be abstracted in the rules as possible, and, in parallel, simplify the way in which the domain knowledge and common sense are formalized. I describe in the next section the approach I propose to infer activities in a bottom-up use-case.

5.4.2 Activity Inference: Balancing Rationalism and Empiricism

Scope of the Activity Inference

Taking into account the constraints described above, I had to define properly the scope of the activity recognition. I decided to reduce the possible outcomes of the inference to a selected set of primordial activities thought to be high-level enough such that their detection would be possible with the limited situational data available and with minimal assumptions made about the resident's lifestyle. The activities chosen are listed and described shortly in Table 5.2.

Table 5.2: Scope of Activities to be Detected by the System

Activity	Description
Sleep	Present with little motion in a bedroom
Get up	Fixed duration after sleep, if no other notable activity
Go toilet	Go to the toilet (limited duration)
Hygiene	Go to the bathroom (limited duration)
Cook	Prepare a meal in the kitchen
Eat	Eat the meal in any room
Clear table	Clear table and clean dishes back in the kitchen
Take a nap	Similar to sleep out of the bedroom
Occupied	Encompass several finer grain activities with motion
Go outside	Go out of home
Come home	Fixed duration after go outside
Socialize	Meet other people at home
Run away	Go out in the middle of the night or for too long
Fall	Unusual lack of movement in the home

Naturally, each resident has his own lifestyle where activities could be differentiated based on time or duration factors. E.g. we could take into account that a resident usually takes a nap of approximately 80 minutes (spanning between 45 and 110 minutes) around 14:00 in order to differentiate it from a possible fall. However, I present here a method with no cold-start, i.e. which does not require any learning in order to perform the inference. The only data available for the inference is therefore the motion of residents in their home. This method could therefore be used to provide estimated labels for the supervised learning of other methods.

Rule-Based Activity Inference

As introduced above, in order to ensure the recognition of activities in real-time and with no cold-start, i.e. without requiring any learning of the lifestyle of the resident, I define an explicit rule-based

inference of the activities performed by the residents with no assumption made about their habits. This implies that I cannot use any time-based condition for the inference since elderly people, especially when facing some level of dementia, are prone to have their lifestyle lightly to heavily shifted from the social standards. For example, they could be sleeping late at night and for a short period of time, while compensating by naps during the day. The time at which they take their meals might be unpredictable as well. After consulting with doctors, we learned that this was not an issue, whereas observing changes overtime in their own habits was more likely to indicate some clinical issue. Taking into account these observations, I propose for the activity inference a mechanism based on three complementary types of rules: rules of reasoning, rules of fact and arbitration rules.

As explained by Dargie et al., knowledge in context-aware systems can be acquired through pure reasoning alone or via experiences as perceived through the senses and stored in the memory [106]. This is highly related to the philosophical opposition between rationalists and empiricists, who for centuries debated the epistemological question of how knowledge is acquired by humans. Dargie however tends to agree with both sides. Philosophically, my position would be the closest to the ideas defended by Gottfried W. Leibniz, a German mathematician and philosopher. Leibniz was known, among others, as one of the great seventeenth century advocates of rationalism. He notably claims in his famous work, the *Monadology* [133], that "*there are two kinds of truths, those of reasoning and those of fact. The truths of reasoning are necessary and their opposite is impossible; the truths of fact are contingent and their opposites are possible*". This recognises the role played by empiricism in epistemology, but highlights the superiority of rationalism. He defends in this work that in principle, all knowledge can be accessed by rational reflection (**truths of reasoning**). However, due to the shortcomings in their rational faculties, human beings must also rely on experience as a mean of acquiring knowledge (**truths of fact**). This is to provide an alternative in formulating knowledge when we face a too high complexity in rational reflection. Leibniz calls this the **infinite analysis**.

Similarly and applied to the AmI field, rules use to infer contextual knowledge should ideally be **rules of reasoning**, i.e. abstract rules that translate some irrevocable knowledge or common sense and can be applied with no doubt about the existence of a contradiction. However, due to limitations emerging from both the situational data available and our modelling faculties, such rules cannot always be written. Hence, we need to introduce **rules of fact** to translate products of experience and intuition, which however may find holding contradictions. These types of rules are described in details below.

Type I – Rules of Reasoning These rules perform the fusion and augmentation of contextual data received from sensors into higher-level contextual information such as residents' location or their use of objects. They also compute durations from timestamps. The relation to rationalism is due to the rules being translated from known scientific models or common sense. In Source 5.1, the first rule is tracking the location of residents in their house. This rule is formalized in DL below for reference and comparison with its N3 syntax (Source 5.1).

$$\begin{aligned}\forall\ &\textit{Sensor se; SensorState st; Room r, r2; House h; Resident u; Timestamp t;}\\ &\textit{(se hasCurrentState st)} \wedge \textit{(se hasLastUpdate true)}\\ &\wedge \textit{(st indicateLocation true)} \wedge \textit{(se deployedIn r)}\\ &\wedge \textit{(r partOf h)} \wedge \textit{(u liveIn h)}\\ &\wedge \textit{(u detectedIn r2)} \wedge \textit{(r2} \neq \textit{r)} \wedge \textit{(se lastUpdate t)}\\ \Rightarrow\ &\textit{(u detectedIn r)} \wedge \textit{(u cameFrom r2)} \wedge \textit{(u inRoomSince t)}\end{aligned}$$

Source 5.1: Rules of Reasoning (N3, Type I)

```
@prefix log:  <http://www.w3.org/2000/10/swap/log#>.
@prefix math: <http://www.w3.org/2000/10/swap/math#>.
@prefix ts:   <triplestore#>.

## track resident location [live + persistent]
```

```
{?se qol:hasCurrentState ?st. ?se qol:hasLastUpdate true. ?st qol:indicateLocation true
    . ?se qol:deployedIn ?r. ?r qol:partOf ?h. ?u qol:liveIn ?h. ?u qol:detectedIn ?r2.
     ?r log:notEqualTo ?r2. ?se qol:lastUpdate ?t} => {?u qol:detectedIn ?r. ?u qol:
    cameFrom ?r2. ?u qol:inRoomSince ?t. ts:n3store ts:update {?u qol:detectedIn ?r. ?u
     qol:cameFrom ?r2. ?u qol:inRoomSince ?t}}.

## infer durations [live only]
{?since qol:hasDurationEquivalent ?for. ?x ?since ?start. hom:clock qol:hasValue ?now.
    (?now ?start) math:difference ?duration} => {?x ?for ?duration}.
```

In the first rule, `lastUpdate` marks for each sensor the timestamp of the last event received, while the predicate `hasLastUpdate` is a boolean flagging the sensor which sent the last event that started the reasoning cycle. We also note that rules can have a live and/or persistent range, i.e. their consequent can be inferred temporarily and hold until the end of the reasoning cycle or written in the triplestore (using `ts:n3store ts:update`) so that it still holds in the next reasoning cycle. The second rule in Source 5.1 infers the value of all duration properties "`?for`" for which a binding exists with their timestamp equivalent "`?since`". This binding, if any, should be declared in the model using the property `hasDurationEquivalent`.

Type II – Rules of Fact The rules from this second type correspond to the evolution from information to knowledge in the DIKW hierarchy. Their relation to empiricism is due to their incorporation of domain knowledge emerging from experts' experience, intuition and insight. Each rule votes independently in favour or against an activity by giving it a score ranging from -10 to 10. The score given by a rule should increase when either the activity or the rule is more meaningful; for example the "sleep" activity in Source 5.2 is considered specific and can be distinguished easily from others so it receives a high score, whereas the "occupied" activity is more like a default activity when movement is detected and nothing more specific can be inferred about the context of the resident, thus it receives a lower score. Dangerous activities that cannot be missed are also given a high score as can be seen for "fall" and "run away". Rules can also give a negative score to vote against activities in order to balance positive scores given by other rules. E.g. we can see that the second rule for "occupied" and "sleep" decreases the score obtained by the activity when it has been inferred continuously for a long time. This is useful to take into account the maximum duration for which it is safe to accept an activity—we do not want to detect showering for three hours. Of course the score is decreased in one step in this example but some fuzzy scoring is also possible.

Source 5.2: Rules of Fact (N3, Type II)

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix func: <http://www.w3.org/2007/rif-builtin-function#>.
@prefix prof: <profile#>.
@prefix :     <rules#>.

## definition of localy useful property
:getScore a owl:DatatypeProperty;
  rdfs:comment "Indicates that a rule is in favor of recognizing an activity with a
      score."@en;
  rdfs:comment "the score should be from -10 to 10."@en;
  rdfs:domain qol:Activity;
  rdfs:range xsd:int.

## go to the toilet (+ max duration)
{?u qol:detectedIn ?r. ?r a qol:Toilet. ?u qol:inRoomFor ?d. ?d math:lessThan ?u!prof:
    maxToiletDuration} => {hom:goToilet :getScore 7}.

## occupied (+ max duration)
```

```
{?u qol:liveIn ?h. ?h qol:motionMeasured ?m. ?m math:notLessThan ?u!prof:
    minOccupiedMotion} => {hom:occupied :getScore 2}.
{?u qol:believedToDo hom:occupied. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!
    prof:maxOccupiedDuration} => {hom:occupied :getScore -1}.

## sleep (+ max duration)
{?u qol:detectedIn ?r. ?r a qol:Bedroom. ?u qol:inRoomFor ?d. ?d math:notLessThan ?u!
    prof:minSleepInitiation. ?r qol:motionMeasured ?m. ?m math:lessThan ?u!prof:
    maxSleepMotion} => {hom:sleep :getScore 6}.
{?u qol:believedToDo hom:sleep. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!prof:
    maxSleepDuration} => {hom:sleep :getScore -5}.

## run away
{?u qol:useNow ?d. ?d a qol:Door. hom:clock qol:hasValue ?t. ?t func:hours-from-
    dateTime ?h. ?h math:notLessThan ?u!prof:outTooLate. ?h math:lessThan ?u!prof:
    outTooEarly} => {hom:runAway :getScore 9}.
{?u qol:detectedIn ?o. ?o a qol:Outside. ?u qol:inRoomFor ?d. ?d math:notLessThan ?u!
    prof:outTooLong} => {hom:runAway :getScore 9}.

## fall
{?u qol:believedToDo hom:nothing. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!
    prof:maxInactiveDuration} => {hom:fall :getScore 8}.

## meet people at home
{?u qol:isAlone false} => {hom:socialize :getScore 8}.
```

Generally, scores should be set by a system expert who understands the impact that high or low scores might have, as well as the relative importance of scores between the rules of potentially overlapping activities. In the rules of Source 5.2, one can also note the use of the `prof:` namespace to query information stored in the resident's profile, as well as another syntactic sugar of N3 with the "`!`". "`!`" is a syntactic sugar specific to EYE which would actually be replaced by "`.`" according to the N3 specifications. It allows the creation of forward traversal paths of the graph that would be read as " 's" in English. For instance:

```
?d math:lessThan ?u!prof:maxSleepDuration.
```

would be recomposed as

```
?u prof:maxSleepDuration ?x.
?d math:lessThan ?x.
```

in full N3. The `prof:` namespace provides a number of datatype properties that can be used to describe the habits of residents. E.g. properties like `maxSleepDuration` or `maxSleepMotion` are profiling details about residents that can be learned statistically overtime. However, to ensure the proper cold-start performance of the framework, I have pre-filled each resident's profile with default values set purposely very loosely such that conditions using them would not be affected. For example, `maxSleepDuration` and `outTooLate` were respectively set to 10 hours and 23:00, which is in both cases so much that it gets alarming for any elderly person. Thus, I do as little assumptions as possible about the residents' lifestyle, yet I leave opportunities for performance improvements when meaningful customized behaviour analytics will be available. Another avenue of improvement would be to add to all activities timely conditions as I did for the "run away" rules. This also requires an historic to be built to mine the statistical time of the day for each activity, if any relevant value can be found.

Type III – Arbitration Rules The rules from the third type arbitrate the scoring system by computing the **Rule-Based Confidence Score (RBCS)** of each activity and decide on the most probable activity. The RBCS of an activity represents the percentage of all distributed points given to the activity. It requires to compute the total score `getFinalScore` for each activity (first rule in

Source 5.3). In the second rule, if the total amount of points distributed is not over a given threshold, we consider that no activity is really standing out and a score of 10 is given to the activity "nothing", which might result in a fall being detected. The threshold used is fixed according to the score distribution profile of the type II rules such that the "nothing" activity comes up in case no activity gets sufficient confidence to be inferred. For instance, in the example provided "occupied" and "sleep" both go down to 1 point in case they have been inferred for a too long time; the threshold is hence set to 2 point such that "nothing" takes over in such case. The threshold must be fixed by the system expert who has given the score of the different type II rules. The third rule computes the RBCS for each activity and the fourth infers that the resident performs the activity with the highest RBCS. The arbitration is kept simple for now as we take the maximum RBCS to infer the activity but more complex mechanism could be tested and a global arbitration with results coming from other activity recognition algorithms may be introduced here.

Source 5.3: Arbitration Rules (N3, Type III)

```
@prefix e:  <http://eulersharp.sourceforge.net/2003/03swap/log-rules#>.

## definition of localy useful property
:getFinalScore a owl:DatatypeProperty;
  rdfs:comment "Computed total score over all single-rule scores."@en;
  rdfs:domain qol:Activity;
  rdfs:range xsd:int.

## sum scores and compute confidence [live only]
## if no activity stand out then give score to hom:nothing [live only]
{?ac :getScore ?x. ?SCOPE e:findall (?sc {?ac :getScore ?sc} ?list). ?list math:sum ?
    total} => {?ac :getFinalScore ?total}.
{?SCOPE e:findall (?sc {?ac :getScore ?sc} ?list). ?list math:sum ?grandtotal. ?
    grandtotal math:lessThan 2} => {hom:nothing :getScore 10}.
{?SCOPE e:findall (?sc {?ac :getFinalScore ?sc} ?list). ?list math:sum ?grandtotal. ?
    grandtotal math:notLessThan 0.1. ?ac0 :getFinalScore ?fs. (?fs ?grandtotal) math:
    quotient ?cs} => {?ac0 qol:getRBConfidenceScore ?cs}.

## infer most probable activity [persistent only]
{?SCOPE e:findall (?rbc {?ac qol:getRBConfidenceScore ?rbc} ?list). ?list e:max ?maxrbc
    . ?ac qol:getRBConfidenceScore ?maxrbc. ?u qol:liveIn ?h. hom:clock qol:hasValue ?
    now} => {ts:n3store ts:update {?u qol:believedToDo ?ac. ?u qol:doesActivitySince ?
    now}}.
```

We note here the use of `e:findall` which needs to be further detailed. In order to make semantic technologies more usable and practical to use in applications, the W3C semantic workgroup provides non-W3C-endorsed extensions under the Semantic Web Application Platform (SWAP) initiative [134]. One extension is the possibility to use built-in functions, often specific to an inference engine, which are properties that the engine can infer, or use to infer statements. The most commonly used built-ins are from the logic framework defined by the W3C under the namespace prefixed `log:` in Source 5.1. For instance, we use the `log:` namespace in all the rules as the inference capability is provided by the `log:implies` built-in hidden behind the syntactic sugar "`=>`". EYE also has its own dedicated built-ins declared in the namespace prefixed `e:` in Source 5.3, among which `e:findall` can be found. Used as `?SCOPE e:findall (?SELECT ?WHERE ?ANSWER)`, it unifies `?ANSWER` with a list that contains all the instantiations of `?SELECT` satisfying the `?WHERE` clause within the `?SCOPE` of all asserted and inferred N3 formulae.

5.4.3 Rules Verification Using Formal Methods

The heterogeneity of technology and the reliance on ad-hoc communication networks make pervasive systems highly complex [135]. Furthermore, various environmental inputs and unpredictable user behaviours might drive the system beyond control, especially when multiple users are interacting with it

simultaneously. Therefore, it is a challenging task to guarantee the correctness and even the safety of such systems. Traditional validation methods such as simulation and testing have their limitations in performing this task since they can only cover partial system behaviours based on selected scenarios. Our system, and especially my rules have been validated using formal methods to analyse all reachable states of the system in order to overcome these limitations. This work was realised in collaboration with a specialised team of the National University of Singapore, and notably through the corresponding doctoral work of Ms. Zhang Xian & Ms. Liu Yan [136].

To begin, we identify critical properties of the system and provide their specification patterns in corresponding logics. According to the stakeholders (designers, engineers and users), safety requirements are essential to pervasive computing systems. Arapinis et al. proposed some critical requirements of a homecare system [137]. For instance, sensors should never be offline when a patient is in danger; or if a patient is in danger, assistive services should be provided within a given time. In our work, we classify the important requirements into safety properties (i.e. nothing bad should happen) and liveness properties (i.e. something good should eventually happen). Furthermore, formal specification patterns of these properties are proposed; and consequently we can verify the critical properties against the system model by using automatic verification techniques like model checking [138]. Hence, design flaws can be detected at the early design stage.

Properties to Check

Liveliness Properties I provide here two examples of liveliness properties that have been checked to certify the system's behaviour.

1. Deadlock freeness: This is one of the important requirements. A deadlock happens when the system reaches a state where no more action can be performed. It can lead to serious consequences such as undetected falls of the resident. Deadlock checking is supported in most model checking tools.

2. Guaranteed services: A well-designed service framework should impose a fundamental responsiveness requirement on the service provision. For example, if a resident falls, it should be guaranteed that a notification is sent within a reasonable amount of time. Effectiveness of these services is an important characteristic of the system for the users. To specify this requirement, we propose patterns of liveness properties using Linear Temporal Logic (LTL) such as $\triangle\ (ResidentFall \rightarrow \diamondsuit\ FallAlertSent)$. In this example, $\triangle$ and $\diamondsuit$ are LTL operators read as "always" and "eventually" respectively. The services are usually required to be delivered in bounded time. To specify the bounded liveness properties, one can use Timed Computational Tree Logic that integrates clock constraints in model checking.

Safety Properties Like in the development and prototyping of numerous systems, cost can be reduced by the early detection of issues, which is desirable considering the re-engineering workload. Fortunately, unwanted scenarios can be specified in properties and checked using reachability verification algorithms. I give as example three safety properties that can be checked in this manner.

1. System inconsistency: Sensor failures or wireless networks may cause situational data in the system to be out-dated. The KB may consequently be inconsistent with the actual environment. By defining and formalising such states, one can test against the system model to see if they are reachable and how. Hence these states may be avoided by fine tuning the rule design.

2. Conflicting services: Guaranteeing that services are eventually delivered is not enough. It is also important to check if these services are sent properly. Some problems have been reported by domain experts, such as conflicts of reminders [139]. These problems are especially common in multi-user settings. They can however be specified as states and checked using reachability tests.

3. Rules properties: The correctness of rules is essential to the correct behaviours of systems. Problems include duplicate rules, conflicting rules and unreachable rules. This is also easy to specify. For example, to check whether a rule is unreachable, the condition of the rule can be defined as a state and a property can be expressed to test if the state is reachable.

Verification of the Properties

To summarise, a property that we want to check must first be defined as a state. We then use reachability verification algorithms to exhaustively search the system state space to see if such a state is reachable. The certification is obtained when liveliness properties are reachable and safety properties are not. The system state space is defined as a representation of the whole system (human, hardware and software) into a giant state machine where properties can be defined. This state machine is composed of individual state machines that describe all possible behaviours of each component of the system. This is illustrated in Figures 5.1 and 5.2 which represent the state machines implemented to describe the behaviour of a resident as well as the bed pressure sensor respectively.

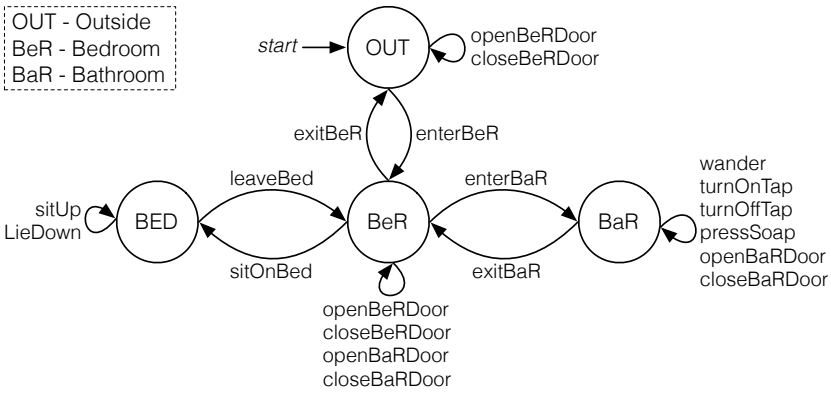


Figure 5.1: State Machine Describing Residents Behaviour

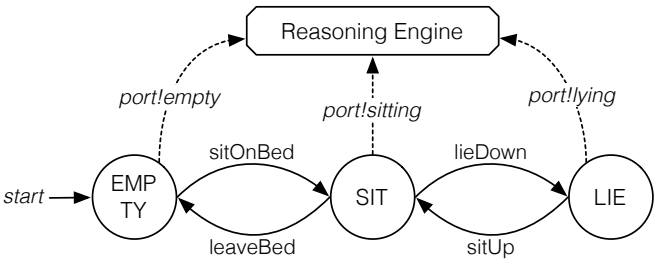


Figure 5.2: State Machine Describing the Bed Pressure Sensor Behaviour

The remaining components of the system, including the various rules in use, are also described in the same manner. Rules and their properties can therefore be checked with the rest of the system to detect design issues before the deployment. All states of the machine are checked exhaustively to know whether the state of given properties are ever possibly reached. In case a state is reached, a example path to that state is provided, in the other case it is certify non reachable. Table 5.3 provides a sample of the results obtained during the model checking process that is described in more details in [136].

We can conclude a few things from these results. First, model checking for systems as complex as smart homes is very computationally challenging. The number of states and transitions that the

Table 5.3: Results of the Model Checking Verification of the System

Property	Result	Nb of states	Nb of transitions	Time (s)
Global deadlock freeness	-	-	-	out of memory
Bathroom deadlock freeness	True	10.8M	15.8M	7045
Guaranteed wrong bed reminder	True	1.6M	2.43M	1945
Guaranteed tap not off reminder	False	70K	131K	39

algorithms have to scan to provide an exhaustive test is of the order of millions. For instance, we were not able to certify a global deadlock freeness for the system (test going out of memory) and had to divide it into deadlock freeness per room. For information, the experiment was run on an Intel Xeon CPU running at 2.13GHz with 32GB RAM. This computational issue aside, model checking techniques were found to be a very good tool for the rule design. From our experience, it provides valuable feedback concerning eventual issues present in the reasoning, and which would probably take hours of deployment to be detected. Hence we feel that it is a primordial way to improve the system safety, especially for early adopters of AmI technologies. Indeed, it is important to find unexpected bugs based on the stakeholders requirements before the deployment of the whole system. The model checking test revealed, among others, an insufficiency in the rules antecedent for the selection of the "tap not off" service. For instance, this service should be provided to a resident that has left the bathroom without turning off the water tap. It was however sent to the wrong person when someone would enter the bathroom at the same moment as someone else would leave it. Thanks to model checking tools, we were able to add conditions in the rules to correct this mistake, thus reducing the risk of troubling residents.

Machine learning is the science of getting computers to learn without being explicitly programmed.

— Andrew Ng, Stanford

6 Incorporating Data Driven Techniques and Quality of Information

6.1 Limitations of a Purely Rule-Based Approach

I argued in the first part of this document my choice of a knowledge driven approach to context comprehension with rule-based inference techniques. Indeed, such approach makes it easy to leverage the important domain knowledge and common sense inherent to understanding people's behaviour. It also performs better than other techniques in situations where a very coarse granularity of situational data is available. Finally, it solves the cold-start issue that we would face if we were using data driven statistical techniques.

This being said, using a purely rule-based and knowledge driven approach has limitations as well. Since this approach relies on the availability of structured and formalized data and prior knowledge, it does not let information emerge from frequently observed patterns implicitly present in the data. I believe that this could be compensated by combining data driven techniques and semantic reasoning. It would indeed make a powerful coupling of rule-based techniques capturing domain knowledge, expert knowledge and common sense; and statistical techniques extracting patterns from data itself in a less "human bound" manner (i.e. without relying on prior knowledge and rules written by experts), which would perhaps be more scalable. A data driven approach would also enable longer term observations of the behaviours without having to make the semantic model used more complex and augment the size of the ontology processed. Hence, shifts in the residents' lifestyle could be observed and translated into health assessment markers. The extensions proposed in this regard are described in section 6.2.

Another limitation of the semantic approach as I designed it, is the unavailability of previous ontological states, and thus previous sensor states or activities for example. This makes it impossible to write rule antecedents with conditions on the past activities of a resident. This limitation and the related extensions are discussed in section 6.3.

Finally, although Quality of Information (QoI) is an important aspect of contextual information, it has not been addressed yet as the reasoning engine proposed treats each piece of information as an absolute truth. I describe in section 6.4 the challenges related to representing uncertainty in ontologies, and propose an extension of the model to attach QoI metrics to ontological statements, together with the corresponding rule syntax to integrate and propagate such metrics at the reasoning level.

6.2 Data Driven Analysis of Ontological Knowledge

6.2.1 Traditional Machine Learning Techniques on Ontologies

As a format for the storage of structured data, ontologies represent a convenient media on which to perform machine learning. Although the triple-based representation is not the most conductive,

mechanisms can be introduced to *lift* the data into a more suitable format and *lower* it back into its triple-based form. Indeed, data can be extracted from ontologies using deterministic queries and serialized to a suitable format, say vectorised, in order to be fed to machine learning algorithms. Moreover, the lowering of the results back into their semantic representation makes it easier to translate results into meaningful observations. Using semantically labelled input data and being able to project eventual patterns found in the data into a semantic space enables the semi-automatic translation of these patterns in the vocabulary defined for the observation domain. This bridges the gap between extracting interesting patterns and extracting meaning from the data.

In order to enable such connections, I propose the introduction in our framework of a module in charge of the vectorisation of selected data or information at the end of each reasoning cycle. This module queries the triplestore to extract a subset of the knowledge that will be analysed offline using machine learning techniques. It then computes a vector version of this knowledge, together with a label vector enabling the reverted translation of the vector into its semantic representation. Vectors are finally logged in a database used by offline analysis modules. The implementation of this feature helped us to start a research collaboration with colleagues from the data mining and machine learning fields, since a common vocabulary and data format was defined as comparison ground for the heterogeneous contributions. Some techniques, coupled with relevant visualisations, have already shown interesting insight concerning the habits of residents (see Figure 6.1) and are studied further in order to inject the extracted knowledge back into the ontology. E.g. through enhanced user profiling information which would enable a better rule-based activity inference. In conclusion, I currently rely mainly on explicit rule-based inference but am prepared to leverage any useful knowledge coming from back-end analytics.

6.2.2 Rule-Based Clustering

In the previous section, I give an example of how unsupervised learning could be performed on ontological data through the vectorisation of a subset of the ontology. Machine learning techniques, and especially *unsupervised* approaches, are powerful in extracting useful information from data; yet they remain extremely difficult to parameterize and the selection of features is also a challenge. Although rule-based inference is not as autonomous in term of knowledge extraction, it is much easier to setup and results are more "expectable". Hence, I hope to make the best out of both paradigms by introducing the **Rule-Based Clustering (RBC)**. RBC consists in using the results of the rule-based inference as an approximate label for the corresponding data during the vectorisation. The labelled data can then be clustered using *supervised* learning, where setup and parameterisation is easier compared to unsupervised techniques. Finally, new vectorised data can be classified by the supervised learning algorithm and the result may be compared and/or combined with the rule-based inference.

Let us develop the concept of an example RBC engine. To begin, we must consider each ontological state as a graph-based *map* where nodes and branches correspond to statements of the ontology. My idea is to log each map with its corresponding inferred activity. We can then retrieve a stack of maps corresponding to a single activity and compute the average map for the activity: I call it the **Fuzzy Activation Map (FAM)** of the activity. It can be seen as a graph where nodes and branches appear in a gradient of colour where the temperature of the colour corresponds to the statistical activation of the statement for the activity. When a new ontological state is available, its map can then be compared to the FAM of each activity by calculating the temperature distance. The state is then labelled with the activity of the closest FAM. The same operation can be computed on two versions of the maps, an instant Fuzzy Activation Map (iFAM) and a time-windowed Fuzzy Activation Map (wFAM). I call the overall process Fuzzy Activation Map Engine (FAME) and its implementation, based on a vector representation of the maps, is under way.

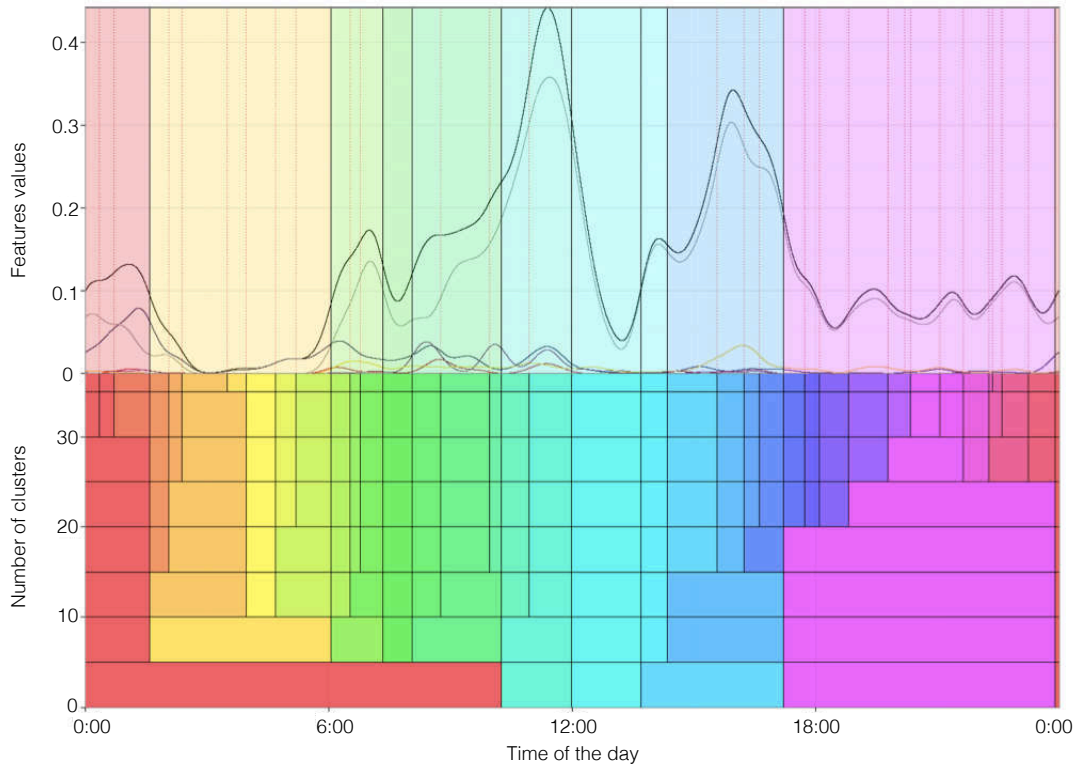


Figure 6.1: Unsupervised Segmentation and Hierarchical Clustering of Activities

Note: This is the result of a work by Guillaume Dufour and Antoine Veillard at IPAL on data vectorised by our framework. The dataset used is described in section 9.4. The method has yet to be published and consists in a pre-segmentation of the data on the inflection points of the main feature, followed by the tree-based hierarchical clustering of the segments.

6.2.3 Combining Different Techniques

To conclude on the dilemma between knowledge driven and data driven context comprehension, I believe that both approaches have specific advantages and drawbacks. Both can also be implemented using numerous techniques that involve trade-offs as well. We observe an interesting parallel between the limitations of data driven approaches and the advantages of knowledge driven approaches, and vice-versa. Hence, I strongly believe that combining them would probably improve the results in context comprehension. I proposed above a system where different algorithms can be used in parallel, extracting from the ontology the knowledge they can analyse, and providing their result back in an ontological format. A final decision may finally be made by dedicated semantic rules, taking into account all available results and employing any arbitration mechanism judged relevant. Such arbitration mechanisms were not studied as part of this doctoral work and are proposed as perspective work, together with the proposition of tighter interfaces between the heterogeneous inference modules. The contribution of this doctoral work in this aspect is limited to a more technical side of designing and implementing the interface to enable the plugging of data-driven inference modules on a semantic platform, and highlighting the possibilities that it creates.

6.3 Introducing Memory in the Reasoning

As introduced in section 4.3.4, my semantic model does not provide any kind of memory. In other words, it does not support the storage in the ontology of previous sensor events, previous activities performed by the residents, or more generally previous states of the ontology. It should be seen as a contextual snapshot, providing the latest known state of each sensor, the current location and activity of the residents and the current motion measured in each room. Naturally, concepts like motion do not have any meaning without taking duration into account and an alternative must be found. This limitation of the model is a trade-off that lets us keep a better control on the size and understandability of the ontology, on the complexity of the rules design, and on the processing time required for the inference. To compensate this limitation, I introduce in the design of the framework alternative reasoning modules to compute memory-bound concepts or variables (such as the motion). For instance, memory is introduced as a projection in the ontology of a time-windowed observation.

To illustrate the mechanism involved, let us detail how we incorporate the motion measurement in the ontology. To begin, we add for each room the datatype property `motionMeasured`, which holds an integer indicating the number of activations received from sensors in the room during a given time window (e.g. five minutes). Semantic reasoning modules do not have access to previous sensor events since they are not available in the triplestore, so they cannot compute the motion values. We therefore implement a specific module called MotionEstimator which handles the calculation of the motion independently. This module does not use semantic inference and can build its own dedicated model and/or memory. Referring to the vocabulary introduced in Figure 7.2, MotionEstimator is a Cerebration module. This will be detailed later on. On start-up, MotionEstimator queries the triplestore to get information such as the different rooms and the sensors deployed in them. It builds its own Java model to store such information and initializes some arrays (the memory) to store all the events grouped by room. At runtime, after each event is received, MotionEstimator queries the triplestore to get the latest event, i.e. the only one not yet in its memory, and stores it in memory. This event remains in memory for as long as it is within the five minutes time window. Finally the number of events for each room is counted and the motion values updated in the triplestore. This mechanism is an example of how to build time-windowed observations that are projected as a single value in the ontology. In this case we simply count the events on the time-window, but other kinds of analysis, such as pattern matching in sequences (e.g. HMM), can naturally be implemented in Cerebration modules.

6.4 Quality of Semantic Information

6.4.1 Representing Uncertainty in N3

QoI is crucial when handling context information in AAL systems. Imprecise or incorrect information can indeed lead to decisions that may affect the safety or serenity of the assisted person. On one hand, ontological knowledge is naturally processed as an absolute truth if no notion of uncertainty or QoI is introduced in the semantic model or if the reasoner is not conceived to consider these notions during the inference. On the other hand, it is common for sensor events to carry a measurable level of uncertainty. Thus, we propose to introduce in the semantic model an alternative context representation with classes of information and associations corresponding to the observable aspects of QoI. As noted by Kokar et al., this becomes particularly important if the system performs data fusion or higher-order reasoning [140].

Some work related to uncertainty integration into semantic models can be found in the literature. Hybrid semantic models combining fuzzy logic [141, 142, 143], Bayesian networks [144], probabilistic representations [145] or Dempster-Shafer theory [146] have been proposed. However, these models suggest a complete re-design of the used ontologies. We propose instead a representation that is retro-compatible with a no-QoI model, hence allowing the simultaneous representation and inference

of knowledge with or without QoI metric. We also make sure that the representation scales to any kind of QoI, present in the data or introduced by reasoning, by bringing it down to a core labelling of the various QoI metrics on the ontology's triples. The contribution described below is detailed in the doctoral work of Hamdi Aloulou [147] and was realised thanks to my contribution in semantic models and rule-based reasoning using N3.

Due to the triple-based representation of knowledge in ontologies, it is complex to represent uncertainty attached to a piece of knowledge. It is indeed necessary to introduce a fourth element in each triple that is carrying the QoI for the triple, which goes against the most basic requirement for RDF-based representations. In more details, a subject may be related to various objects through various predicates, thus the subject cannot carry the QoI information alone; it has to be attached to each specific relation. The problem is exactly symmetrical on the object and the predicate sides; these cannot carry the QoI information either. Since this is similar to the issue related to *timestamping* triples, we compensated the limited literature in uncertainty representation for ontologies by drawing a parallel with the work done in the linked stream data community. **Linked Stream Data** is the RDF data model extended for representing stream data generated from sensors and social network applications [148]. Indeed, in the same way we are unable to attach QoI metrics to triples, this community has faced the problem of attaching a timestamp to triples for years. The recent work by Anissa Rula et al. summarises and compares the various approaches to representing temporal information on Linked Stream Data [149]. Our representation of uncertainty is based on their recommendation of the **N-ary relationship** design patterns [150]. These patterns model an N-ary relation with a set of RDF statements by *(1)* introducing a specific resource to identify the relation, and *(2)* creating links between this resource and the constituents of the relation. For instance, let $\langle s, p, o\rangle$ be an RDF statement, r a new resource, $p1$ and $p2$ two properties, q_r a property holding QoI information, and l_u a level of uncertainty; the N-ary-relationship-based representation is defined as follows:

$$\langle s, p1, r\rangle\langle r, p2, o\rangle\langle r, q_r, l_u\rangle$$

Although $p1$ and $p2$ can be two new properties, one of the two is usually equal to p as in the example presented in Figure 6.2.

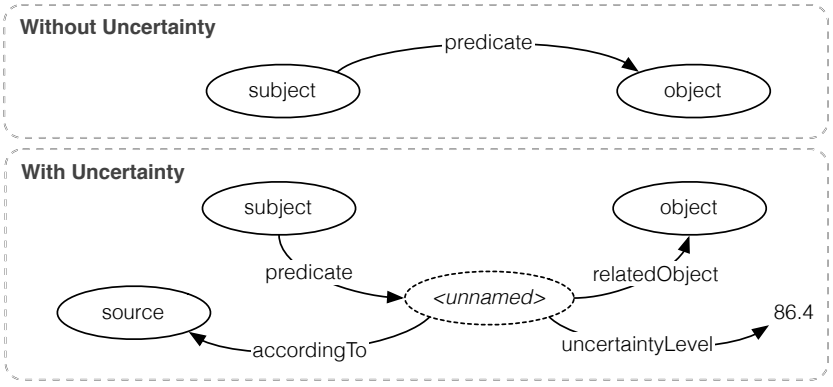


Figure 6.2: Uncertainty Representation in N3

In our use-case, we have combined the N-ary-relationship-based representation with the *blank nodes*, also known as *unnamed resources*, which have been introduced in N3 [151]. Our representation is illustrated in Figure 6.2 and the corresponding TBox extension is given in Source 6.1.

Source 6.1: Uncertainty modelling in N3

```
@prefix unc:  <uncertainty#>.
```

```
unc:Uncertainty a rdfs:Class.
unc:relatedObject a owl:ObjectProperty;
  rdfs:comment "Nominal object of the property under uncertainty."@en;
  rdfs:domain unc:Uncertainty.
unc:accordingTo a owl:ObjectProperty;
  rdfs:comment "Source of the information under uncertainty."@en;
  rdfs:domain unc:Uncertainty.
unc:uncertaintyLevel a owl:DatatypeProperty;
  rdfs:comment "Measure of the uncertainty, equivalent to quality of information."@en;
  rdfs:comment "Between 0 and 100, higher means more uncertain"@en;
  rdfs:domain unc:Uncertainty;
  rdfs:range xsd:double.
```

This extension incorporates a new class `Uncertainty` with two object properties and one data property. An unnamed resource of the class `Uncertainty` is instantiated each time we need to express the notion of uncertainty over a triple. This unnamed resource is related to the subject through the nominal predicate of the triple and to the object through the property `relatedObject`. The data property `uncertaintyLevel` represents the level of uncertainty for the whole triple, based on the information received from the source linked via `accordingTo`. The N3 syntax representing the evolution from a classic triple to a triple including uncertainty is given in Source 6.2.

Source 6.2: Uncertainty Representation in N3

```
## Classical Representation
_:subject a _:Class.
_:object a _:AnotherClass.
_:predicate a rdf:ObjectProperty;
  rdfs:domain _:Class;
  rdfs:range _:AnotherClass.

_:subject _:predicate _:object.


## Representation including Uncertainty
_:subject a _:Class.
_:object a _:AnotherClass.
_:source a _:SourceOfInformation. # e.g. sensor, algorithm, etc.
_:predicate a rdf:ObjectProperty;
  rdfs:domain _:Class;
  rdfs:range _:AnotherClass.

_:subject _:predicate [ a unc:Uncertainty;
  unc:uncertaintyLevel 86.4;
  unc:relatedObject _:object;
  unc:accordingTo _:source ].
```

6.4.2 Reasoning under Uncertainty in N3

The QoI level measured from the sensor stream should be propagated through the reasoning in order to achieve the representation of high-level-context uncertainty. In other words, when a rule is applied and infers higher-level context information based on low-level situational data that incorporates some QoI metric, the quality of context resulting from the inference should be estimated and incorporated within the rule conclusion [152]. This estimation can be based on the quality of the information in the rule antecedent, as well as on the average quality of inference of the given rule. The first approach has been adopted in our framework, where the uncertainty is propagated from sensors level to high-level context information using semantic rules. To illustrate this, I provide in Source 6.3 a simplified rule tracking

the resident's location and its evolution when incorporating QoI metrics to propagate uncertainty from the sensing level to the context level.

Source 6.3: Rule Modification to Include Uncertainty in N3

```
## Classical Representation of Example Rule
{ ?se qol:hasCurrentState ?st. ?se qol:hasLastUpdate true. ?se qol:deployedIn ?r. ?r
    qol:partOf ?h. ?u qol:liveIn ?h } => { ?u detectedIn ?r }.

## Example Rule With Uncertainty
{ ?se qol:hasCurrentState [ a unc:Uncertainty; unc:uncertaintyLevel ?ul; unc:
    relatedObject ?st ]. ?se qol:hasLastUpdate true. ?se qol:deployedIn ?r. ?r qol:
    partOf ?h. ?u qol:liveIn ?h } => { ?u detectedIn [ a unc:Uncertainty; unc:
    uncertaintyLevel ?ul; unc:relatedObject ?r; unc:accordingTo ?se ] }.
```

In conclusion, we believe that the uncertainty aspect will not be tackled by the engine itself, but it is rather the way the engine is used and coupled with other techniques that can ever address this aspect.

The unconscious is the real psychic; its inner nature is just as unknown to us as the reality of the external world, and it is just as imperfectly reported to us through the data of consciousness as is the external world through the indications of our sensory organs.

— Sigmund Freud, 1856–1939

7 A Cognitively Inspired Reasoning Architecture

7.1 Conscious and Unconscious Minds

In the design of the reasoning architecture, two main challenges are to be tackled. On one hand, a balance must be found between purely rule-based reasoning on the ontology and more computational methods that can perform better for some kinds of operations. For instance, when we count the number of sensor activations per room over a time window, it is much easier and efficient to build a dedicated module keeping in memory all the events for a certain amount of time and counting them, rather than trying to find a way around the constraining rule syntax to perform an equivalent operation. Rule-based reasoning comes with great potential but also great constraints, thus it needs to be used with precaution and only when optimal. On the other hand, a mechanism must be proposed to avoid conflicts or deadlocks that might arise from using rule-based reasoning with various modules performing independent yet correlated inference, i.e. inferring the KB with no cooperative strategy though their results might affect one another.

For the first challenge, I introduce in the reasoning architecture two families of reasoning modules. Of course, there are the semantic modules that infer the KB based on rule-based methods using EYE reasoning engine. Additionally, I propose a second family encompassing a variety of other techniques which are individually defined with a specific scope of inference and a dedicated reasoning process. This second family lets us use more computational inference processes, as described above. In these modules, any computation or any additional modelling layer can be designed and implemented.

For the second challenge, I make use of the capability of EYE to handle interdependent rules by leaving it to the engine to solve the conflicts emerging from such rules. Indeed, when the conditions for some rule have changed due to the conclusions of another rule, EYE automatically proceed to a new inference iteration, and such until a stable ontological state is reached. The strategy here is to centralise the inference of all semantic modules into a single reasoning pass, i.e. one call to the inference service provided by EYE. Therefore, I propose a registration mechanism whereby all modules register their rules to a central "intelligence" module.

In summary, I propose a hybrid reasoning architecture composed of a central "intelligence" module that handles all of the rule-based inference, and a family of heterogeneously designed and implemented modules performing independent and well scoped reasoning actions. A parallel must be drawn here with the cognitive model for human thinking proposed in the beginning of the twentieth century by Sigmund Freud, Austrian neurologist and founding father of psychoanalysis [153]. Indeed, this model was in great part the inspiration for the design described above. I illustrate in Figure 7.1 the actual cognitive reference that I consider as the basis for this design. The reader must note that although I find great inspiration in Freud's model, I have distanced myself from it freely in order to propose the best possible design for my digital context-awareness mechanism. I must also highlight that this design is "cognitively" inspired, and not "biologically" inspired, since the representation in Figure 7.1 is biologically wrong. Such work in cognitive science is in fact closer to philosophy, than it is to biology.

Freud's cognitive model introduces the concepts of **conscious and unconscious mind**, where the

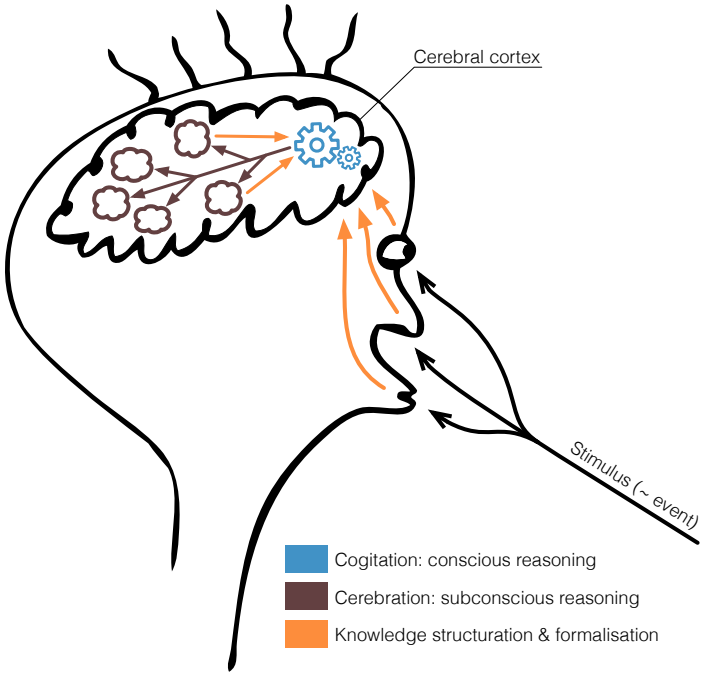


Figure 7.1: "La Pensée": UbiSMART's Hybrid Reasoning Cognitive Inspiration

unconscious mind consists of the processes in the mind that occur automatically and are not available to introspection, including thought processes, memory, affect, intuition, and motivation [154]. Back to the design—and to Figure 7.1 which define "La Pensée", my free interpretation of the cognitive model—I introduce the centralised semantic and rule-based inference mechanism as an equivalent to the conscious mind. I call it **Cogitation**. It is the part of the reasoning that translates domain and expert knowledge in an understandable manner. Its model, as well as its logic, are implemented with attached semantics. It can therefore be expressed in natural language and a proof for its conclusions can be generated to express the transitions of reasoning leading from known facts to conclusions. Complementarily, I define an equivalent to the unconscious mind with the family of heterogeneous reasoning modules called **Cerebration**. Such reasoning processes can hardly be expressed in natural language. They automatically provide information based on complex models capturing data in an implicit way, storing it in semantically unavailable memory, and processing it in a very computational manner. In other words, Cogitation is the parallel to conscious thinking about a problem in term of known facts and logic and what can be deduced from it. It is very similar to the way we would solve a physics problem in school for example. In opposition, Cerebration is the parallel to the unconscious data processing that happens in the brain and is needed for our grasp of situations but stays beyond understanding. It is similar to the processes that help us recognise objects, dodge obstacles, estimate weights, etc.

Additionally to Cogitation and Cerebration, some mechanisms are necessary to structure the knowledge into a formal syntax that is calculable by Cogitation. This is true for stimuli (let us consider events coming from sensors) and is implemented in a module called **Stimulistener**. It is also valid for the internal stimuli that result from the Cerebration processes. Thus, although Cerebration modules can be implemented using any technique, they must provide their conclusion in a semantically valid syntax, in order to enable the collaboration between Cogitation and Cerebration modules.

Finally, a supervision and arbitration is necessary between these modules. The reasoning cycles must be handled to let Cogitation and Cerebration alternatively infer the KB. Conditions must be

determined to decide when to stop the reasoning, and eventual conflicts in the decisions need to be handled. These tasks are implemented in a module called **Cortex** in reference to the cerebral cortex where human thinking happens.

In the following, Cerebration is represented as a single module standing for all of the independent Cerebration modules running in the framework. For instance, a call to the `think()` method of Cerebration means that all of the modules' inference service are actually invoked in an arbitrary order. Eventual conflicts due to interdependent inference actions are handled by Cortex which ensures that a stable decision is reached before the reasoning stops. The following section describes how the different modules composing the reasoning were integrated.

7.2 Live Event Processing Using EYE Through the NTriplestore

In our context-aware service framework, called Ubiquitous Service MAnagement & Reasoning architecture (UbiSMART), EYE was integrated and is used by Cogitation in order to perform rule-based inference of activities (and related decision making) based on the projection of the live datastream of sensor events into a semantic model. Cerebration modules are using any reasoning technique to infer semantically labelled knowledge from data and information that they query from the KB shared with Cogitation. Figure 7.2 provides an architectural and functional overview of the framework, highlighting the integration of EYE through our triplestore.

UbiSMART being based on the Open Service Gateway initiative (OSGi) specification [155], it is implemented in a modular manner where the various features of the framework are handled by independent modules called *bundles*. Figure 7.2 illustrates the service invocations and data exchange between the different bundles. The numbering indicates the sequence of the processing of an event and is used in the detailed description below. Firstly, static N3 files are loaded in the triplestore on framework start-up (0). This loads the semantic model as well as the file containing a static description of the peculiarities of the environment. Other environment related information is added in the triplestore by the environment discovery module, however this is not represented in the graph. Relying on a publish & subscribe event bus, sensor events are pushed in real time to Stimulistener (1). For each event, Stimulistener queries the related statements in the triplestore to find relevant bindings and translates the event into its semantic expression to update the triplestore (2). It then invokes Cortex to handle the reasoning cycles (3). Actions within a reasoning cycle are numbered in Roman in Figure 7.2 so that they can be differentiated. For each cycle, Cortex alternately invokes the inference methods provided by Cerebration (i) and Cogitation (iv) until stability is observed in the decision made by Cogitation (xi). A variety of stop conditions can be defined to rule the behaviour of Cortex when it comes to handling the number of reasoning cycles. Taking the previous example of solving a physics problem, one will stop reasoning about the problem once he reaches a solution that is contradicted neither by related physics laws, nor by his intuition. Similarly, Cortex will observe the decisions made by Cogitation over the cycles of reasoning and stop when the decisions are judged stable (e.g. when they came out three times in a raw). This is ensuring that a decision made is not contradicted in the next reasoning cycle, neither by Cogitation, nor by Cerebration. Additionally, we add a condition on the maximum number of cycles that Cortex can perform. This ensures that a result, even approximate, is found in finite time, and can be compared to a human passing a question in the physics test when he cannot find the exact result. After a decision is made, Cortex sends it to be processed for further actions (5). In the further actions, persistent updates to the triplestore can be made, services can be started or stopped, information can be logged for offline analysis, etc.

Within a reasoning cycle, Cerebration modules are performing a scoped inference of the KB using a variety of reasoning techniques. In order to do this, they start by querying the KB to extract the information to analyse (ii). I consider each Cerebration module as a black box at this level, and each

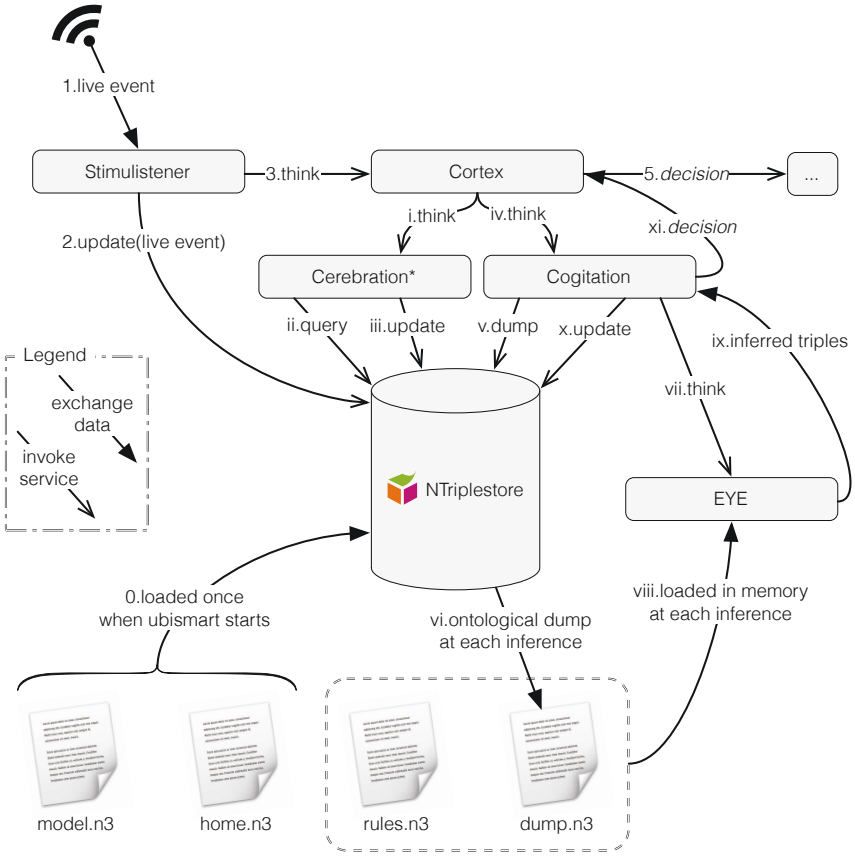


Figure 7.2: Functional Integration of EYE Through NTriplestore, a Purpose-Build N3 Triplestore for Live Events Processing

box provides semantically labelled inferred knowledge as output. This knowledge is updated in the KB by the module itself (iii) and becomes available for other Cerebration modules and for Cogitation. Cogitation is the single module handling the semantic rule-based inference of higher-level contextual knowledge and activities. This inference, for which details are described in section 5.4.2, is further "subcontracted" to EYE. EYE has been wrapped as an OSGi bundle to be integrated as a service in the UbiSMART framework. This bundle provides EYE's inference as a service through the `think()` method registered publicly in the framework. This method performs the inference in a completely *stateless* way, which means that no state of the ontology is kept in memory by EYE. At each invocation of `think()`, EYE parses the ontology, infers it, returns the inferred statements, and free the memory altogether. To start the inference, Cogitation dumps the content of the triplestore in a N3 file (v, vi) and then invokes EYE (vii) who parses the dumped file and the N3 file containing rules to infer (viii) to perform the inference. A short sample of a dump file is provide in 7.1 for reference but a full dump file usually contains several hundreds of such statements. Selected inferred statements, i.e. statements queried as output in the rule file, are then returned to Cogitation (ix) which extracts the decisions and triplestore updates. Updates are made to the triplestore (x) and the statements (including the decisions) are returned to Cortex (xi) to be evaluated, and either start a new cycle of inference or proceed with these decisions.

Source 7.1: Sample from a Triplestore Dump File

```
hom:johndoe qol:inRoomSince "2012-06-05T11:15:34+01:00"^^xsd:dateTime.
qol:indicateLocation a rdf:DatatypeProperty.
qol:detectedIn rdfs:range qol:Environment.
hom:house1203001 qol:motionMeasured 9.
qol:believedToDo rdfs:range qol:Activity.
hom:a7 qol:hasCurrentState hom:a7_on.
hom:a7 qol:lastUpdate "2012-06-05T12:12:43+01:00"^^xsd:dateTime.
hom:b4_on a qol:SensorState.
```

7.3 Complex Ontological Manipulation in the Inference Mechanism

7.3.1 Ontological States

Since my inference mechanism essentially consists in complex ontological manipulation, I believe that it would be useful to provide an overview of the different states the ontology (a.k.a. the KB) goes through during a reasoning pass. I illustrate this in Figure 7.3 where the states are represented with the transitions between them. From the final state $O_f(n-1)$ of the previous reasoning cycle, the

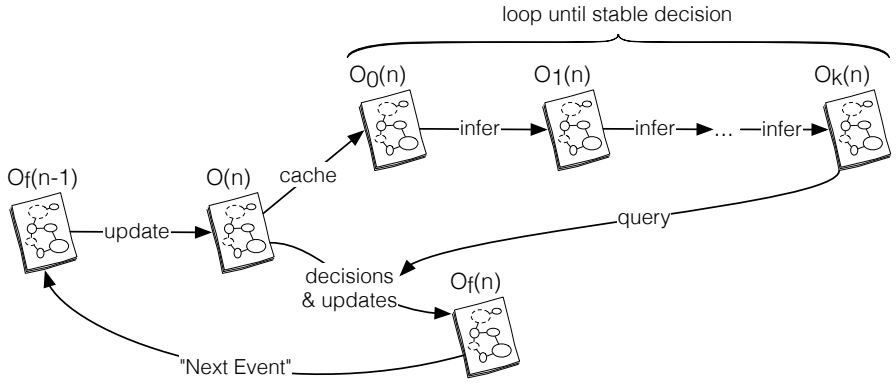


Figure 7.3: Inference Mechanism: Ontological States Transitions

ontology is updated following the reception of a new event by Stimulistener. It is in state $O(n)$, the initial state of a reasoning pass. A copy of $O(n)$ is cached into $O_0(n)$ which becomes the operational state from which the ontology will be inferred. Cortex then proceeds to invoke the inference methods of Cogitation and Cerebration and the ontology is inferred iteratively. Each inference by a Cerebration module or Cogitation brings the ontology from state $O_i(n)$ to $O_{i+1}(n)$, and such until Cortex reaches a stop condition, which leaves the ontology in state $O_k(n)$. At the end of each inference by Cogitation, non-updating queries are performed on the ontology to extract the decisions and updates to be passed to Cortex. In the last Cogitation inference queries are performed on $O_k(n)$ and the updates extracted are applied by Cortex on $O(n)$, which results in the final ontological state of this reasoning pass: $O_f(n)$.

7.3.2 Semantic I/O

For a greater understanding of the ontological manipulation in the inference mechanism, I provide in Figure 7.4 a bird's eye view of all the inputs/outputs (I/O) of the KB, before, during and after the inference. One can actually consider the KB as a sea of semantic information where modules can "pick" information by query or "dump" it by update. I call this vision of the KB **semantic sea (semsea)**.

Originally, the semsea is loaded only with the content of the N3 files illustrated in Figure 7.2, *model.n3*

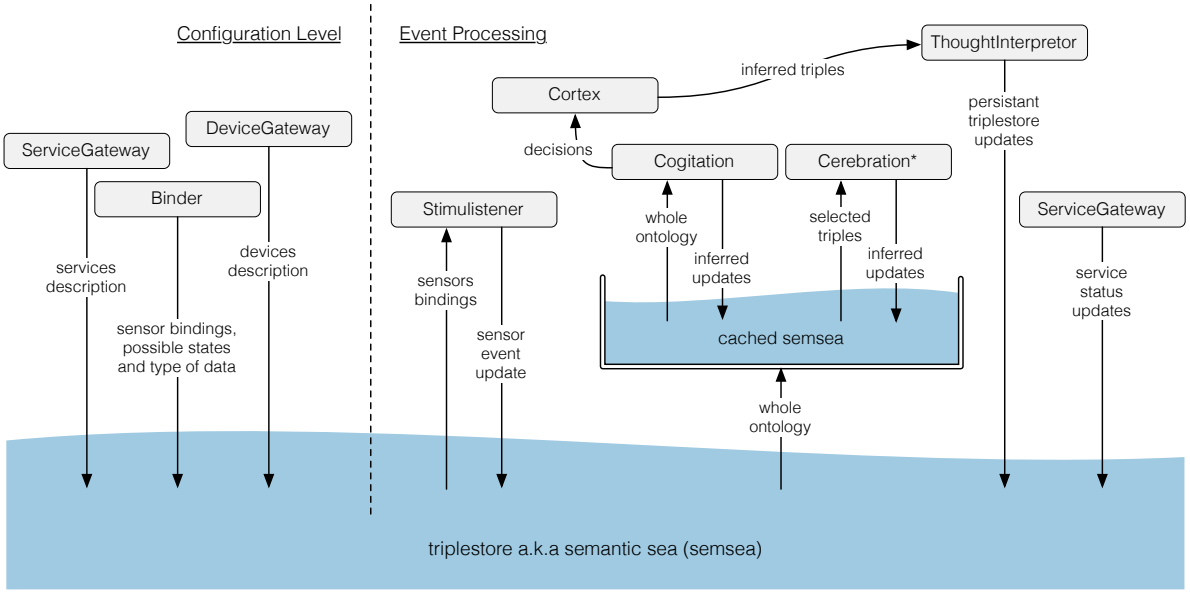


Figure 7.4: Bird's Eye View on the Inputs and Outputs to the Triplestore

and *home.n3*, which respectively contain the semantic model (TBox) that defines the vocabulary used by all modules, and the description of the peculiarities of the home in which the framework is deployed. At the configuration level (left in Figure 7.4), i.e. as a one-time process when the framework is deployed, initial updates are made to the semsea to represent the services installed in the framework, the devices detected in the environment, and the various descriptions concerning the sensors deployed and their configuration. At the event processing level (right in Figure 7.4), one can split the semantic I/O into before, during and after the actual inference, respectively positioned on the left, on top, and on the right of the cached semsea. Before the inference, Stimulistener receives an event, queries the semsea to get the corresponding sensor bindings in order to translate the event into its semantic expression and update it into the semsea. As was presented in Figure 7.3, the whole ontology is copied into a cached version on which the actual inference is performed. This allows temporary updates to be made on the semsea for the sake of the inference mechanism and to let Cerebration and Cogitation modules exchange information seamlessly. Using a cached semsea avoids having to label and remove temporary updates from the final state of the semsea ($O_f(n)$ in Figure 7.3). For the inference, Cerebration modules query some triples from the semsea according to their scope of inference, whereas Cogitation dumps the whole ontology into a file and provide it to EYE for the inference. In both cases, inferred triples are dumped back into the cached semsea. Cogitation also provides the permanent updates and decisions to Cortex. After the last cycle of reasoning, Cortex invokes a module called **ThoughInterpretor** to take care of the further processing of the decisions and permanent updates. ThoughtInterpretor applies the permanent updates in the original semsea (not the cached version) and starts/stops services as needed using the service gateway that updates the status of services in the semsea when they change.

7.4 Integration Into a Context-Aware Service Framework

In order to be tested and used in deployments for AAL, the reasoning system proposed above has been integrated in a context-aware service framework for AAL. We call this framework Ubiquitous Service MAnagement & Reasoning archiTecture (UbiSMART).

UbiSMART's situational data is provided by a Wireless Sensor Network (WSN) for which dedicated applications are implemented based on use-cases to process sensor signals and extract useful states. This is handled in the *sensing modules* block illustrated in Figure 7.5. The states are then processed by the *reasoning modules* where a higher level of contextual knowledge is inferred, enabling the context-aware selection of services, together with relevant interaction modalities (central block in Figure 7.5). Finally, services are instantiated and their interface is adapted to the constraints of the interaction modality in use. The service delivery is handled by specific modules illustrated in the block to the right in Figure 7.5.

In parallel, specific modules catering to the configuration needs of such a framework are developed in order to ease the deployment of the system. These modules, which compose the Smart Space Composer (S2C), are illustrated in an independent block in Figure 7.5.

More details about the implementation of UbiSMART are provided in chapter 8.

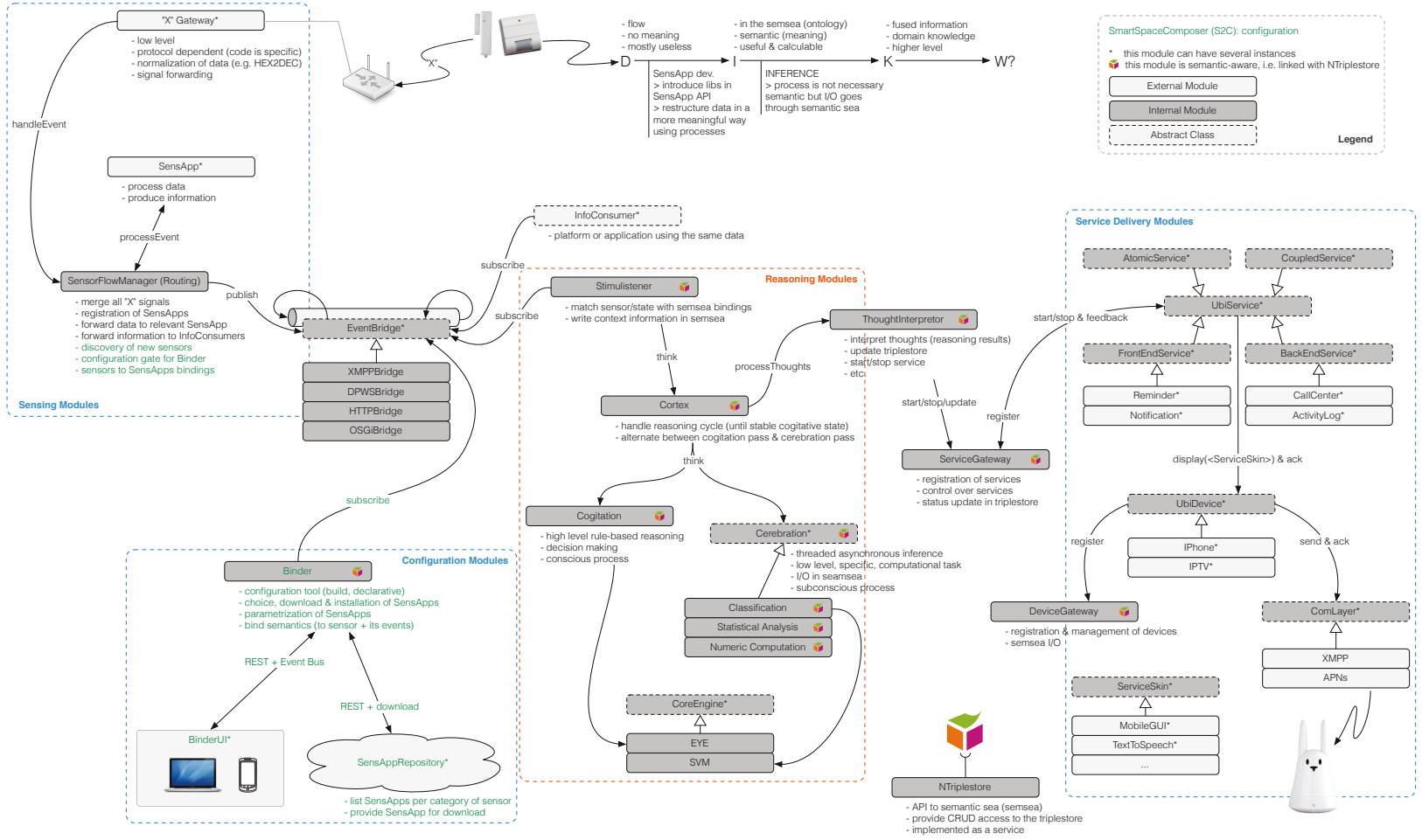


Figure 7.5: Integration of the Cognitively Inspired Reasoning Architecture Into a Fully Featured Service Framework

Part III

UbiSMART Framework: Ubiquitous Service MAnagement and Reasoning archiTecture

There's many a slip between a potential brave new technological world and a reality that could improve the quality of life of a significant proportion of humankind.

— Peter Lucas, Carnegie Mellon

8 Detailed Description of UbiSMART Framework

UbiSMART is a context-aware service framework that was designed over time to match the requirement of both the top-down and the bottom-up approach (see section 3.1.2). Its design has naturally evolved with the various research focuses and I present here two milestones of its development. The version 1 described in section 8.2 is better suited for the top-down approach and focused on the interaction between numerous modules around a shared KB. The version 2 presented in section 8.3 leverages the first design and modifies it to take into account the peculiarities of the bottom-up approach by making it more server-side ready. It introduces a novel hybrid reasoning architecture and includes unique features such as its N3 triplestore and the plug & play mechanism designed to ease the customization of the system deployed in each house. The UbiSMART framework is the major tangible contribution of this doctoral work; it is based on and encompasses the implementation of the contributions conceived in Part II.

8.1 Enabling Technologies

8.1.1 Service Oriented Architecture (SOA)

The **Service Oriented Architecture (SOA)** is a top-down design approach that emphasizes the separation of system functionalities into independent, loosely coupled and interchangeable modules. Each module contains everything that is necessary to provide a defined function or set of functions. The coordination between the modules of a system is ensured via their interfaces. These interfaces are the signature of the modules as they specify the functionalities provided by them to the others. There is no need to be aware of the working code (implementation) corresponding to the functionalities declared in the interfaces. This design approach is very beneficial to AAL solutions since, as complex systems, SOA enforces their logical structure by breaking their complexity into simpler tasks making them more efficient and easier to understand and modify. Moreover, designing and developing AAL systems involve different disciplines (sensing, networking, reasoning, data mining, human-machine interaction), thus adopting such an approach allows multidisciplinary players to work on several parts of the system at the same time with little need for communication, thus making the development of the system more straight forward. It allows making drastic changes to one module without requiring any change to other modules. In fact, modules are substitutable and reusable. A module can replace another at design time or even at run-time without reassembling the whole system. It can also be integrated into another system to provide its functionalities.

More concretely, SOA represents a set of principles and methodologies for designing and developing systems in the form of interoperable services delivered and used on demand [156]. Its architecture is composed of the three main components shown in Figure 8.1:

– The **service provider** is the entity which implements one or more services and publishes them for others to invoke.

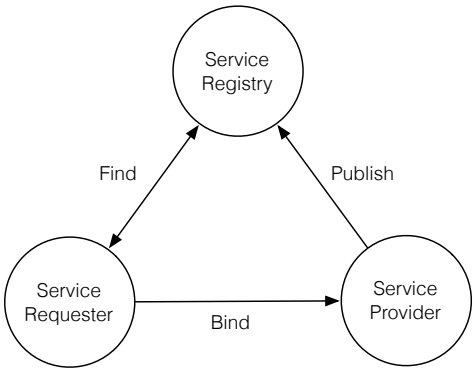


Figure 8.1: The Key Components of the Service Oriented Approach

- The **service requester**, also called consumer, is like the "client" that invokes a service. It can be a front-end application or another back-end service.
- The **service registry** is a software entity that acts as a service locator where new services are published. It implements the discovery mechanism and suggests service providers for the requester of a specific service.

These elements play different roles which define the contracts between them as follow:

- **Publish** is an operation that acts as service registration or advertisement. It operates between the service registry and service provider. The service provider publishes a service in the service registry.
- **Find (or Discover)** is the contract between a service requester and a service registry. This operation is executed on the registry according to a list of search criteria specified by the requester. Search criteria may be the type of service, Quality of Service (QoS), etc... The service requester queries for a specific service in the service registry. The service registry replies with the identification of service providers that provide the requested service.
- **Bind** is the operation that binds both the service provider and requester in a client/server-like relationship. The service requester invokes the service from the service provider.

As Forrester analyst Jeffrey Hammond was explaining at the recent EclipseCon 2013 conference in Boston, "by breaking down applications and systems into loosely coupled services, service oriented architecture has paved the way for enterprise architects to support smaller, more numerous, and even more *experimental* projects within their organizations". One of the advantages SOA brings to organizations is the ability to abstract important parts of applications as reusable, standardized services that can be run in any and all connecting systems. The emergence of these flexible service layers means architects, developers, and even business users can more readily put together new business workflows and processes without the need to rewire or rewrite underlying applications.

8.1.2 Open Service Gateway initiative (OSGi)

The **OSGi** framework is a standardized module system for networked services that is the foundation of enhanced SOA. OSGi is a specification for Java of the principles defined by SOA. Its standards are defined by the OSGi Alliance and published in the OSGi specification documents. The scope of this service framework is as follows [157]:

- Providing a standard, non-proprietary, software component framework for manufacturers, service providers, and developers. The fact that the OSGi specifications are an open standard enables a fair playing field for all participants.
- A cooperative model where applications can dynamically discover and use services provided by other applications running inside the same OSGi service platform. This cooperative service model is considered as a key element for service dependencies.
- A flexible remote management architecture that allows platform operators (the organization that manages the platform) and enterprises to manage thousands of service platforms from a single management domain.

An OSGi platform is basically a container running functional components called bundles. Life cycle management is one of the most prominent features of the OSGi framework. It provides the necessary mechanisms to allow remote management of bundles and also allows bundles to manage other bundles life cycles. Using these mechanisms and based on the OSGi dynamic component model, bundles can be remotely installed, started, stopped, updated and uninstalled at runtime without acting on other bundles or restarting the platform. The OSGi dynamic service registry allows bundles to register, listen and detect the addition or removal of services, and thus adapt accordingly [158]. The different layers of the OSGi framework are represented in Figure 8.2.

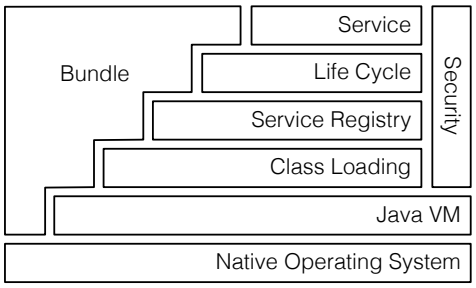


Figure 8.2: OSGi Framework Stack

Any framework that implements the OSGi standard provides an environment for the modularization of applications into smaller bundles. Each bundle is a tightly coupled, dynamically loadable collection of classes, jars, and configuration files that explicitly declare their external dependencies (if any). The framework is conceptually divided into the following areas:

- Bundles: Bundles are normal jar components with extra manifest headers.
- Services: The services layer connects bundles in a dynamic way by offering a publish-find-bind model for Plain Old Java Interfaces (POJI) or Plain Old Java Objects (POJO).
- Services Registry: The API for management services, namely service registration, service tracking and service reference.
- Life Cycle: The API for life cycle management, namely install, start, stop, update and uninstall bundles.
- Modules: The layer that defines encapsulation and declaration of dependencies, i.e. how a bundle can import and export code.
- Security: The layer that handles the security aspects by limiting bundle functionality to predefined capabilities.

– Execution Environment: Defines what methods and classes are available in a specific framework.

To conclude, OSGi provides several useful APIs to manage systems composed of services and has been used in numerous projects for smart home development [159, 160, 161]. It is however build around the assumption of a single container, hosting the different bundles composing the system. Therefore it defines no inter-container communication protocol and needs to be extended in order to be used in a distributed environment. As such and in the AmI field, it is only relevant if we think of a service framework as running on a local home set-top box.

8.1.3 Representational State Transfer (REST)

Considering the bottom-up approach described in section 3.1.2 where cost can be shared among numerous houses through a server-based deployment, it is useful to look into server-side architectures for SOA. In this aspect, several technologies, architectures and approaches are possible, among which **Representational State Transfer (REST)** is well accepted. REST is a style of software architecture for distributed systems that has emerged as a predominant web API design model [162]. REST-style architectures conventionally consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document that captures the current or intended state of a resource. Key goals of REST include the scalability of component interactions, the generality of interfaces and the independent deployment of components.

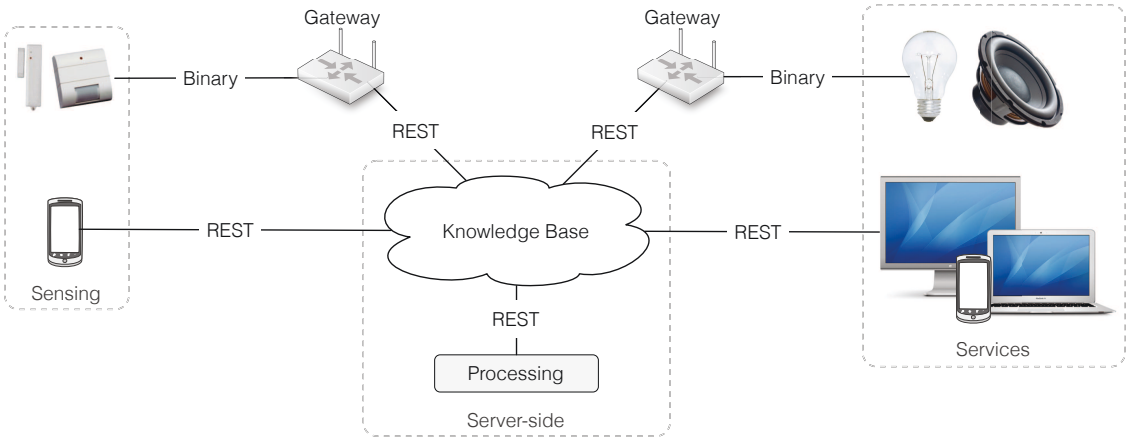


Figure 8.3: REST Architecture for Ambient Intelligence

In Figure 8.3, I illustrate the components and protocols that would make a REST-compliant system for AmI. Sensors (whether intrinsic hardware or embedded in more complex devices such as smartphones) and actuators are connected to the network using a REST or binary protocol. The gateway is a bridge between the binary-bound entities and the cloud data network, i.e. essentially the KB. It can aggregate or dispatch events between entities and the network. The minimal implementation of a gateway would provide simple relaying of binary to REST protocol. Gateways may also have custom event processing logic to provide more sophisticated event processing actions. The KB on the server is the core of the event processing. It can be located either in the cloud or on local network. The main tasks of the server are to provide event fusion, processing into higher-level knowledge and transmission to other modules or entities on the network (e.g. context-aware applications). Services can be implemented as web applications, eventually with attached mobile clients, that provide information about

the situation in the home or the result of trends analysis. They can also control actuators in the home in real-time.

Binary protocols should be used for small and simple devices that do not have enough processing capabilities or have no Transmission Control Protocol - Internet Protocol (TCP/IP) support and are connected to other devices via simple protocols (RS-232, ZigBee, etc.). REST protocol is more adapted for devices or computers that support TCP/IP connections and have enough hardware capabilities to serialize/deserialize the state to be transferred into the chosen data interchange format, e.g. JavaScript Object Notation (JSON).

8.2 Fully Distributed Reasoning Architecture: UbiSMART v1

8.2.1 UbiSMART's Service Architecture

UbiSMART's first version is based on the SOA paradigm and provides a fully distributed reasoning architecture where numerous independent reasoners are deployed around a shared KB to collaborate on the inference of the context, services and devices. In this approach, each module queries the KB to extract the part of the knowledge to be processed; then process it to update the KB and/or react accordingly. This implementation is naturally adapted for the top-down approach and makes it easy to plug new modules to the framework. This version was implemented at a time when we were focused on building a service framework that is context-aware and provides services with an adapted user interface. As illustrated in Figure 8.4 (full page, page 86), the framework addresses three main aspects (context awareness, service management and service delivery) composed of several collaborative modules. All modules share a semantic KB on a producer-consumer basis where each producer provides a semantic description of its contribution, thus all consumers can make sense of it. The KB can be inferred by producers using description logic rules (among others) and then queried by consumers.

Context Producers

Context producers are basically the modules linked to specific sensors. They provide application specific code that extracts low-level sensor data from the sensor signal received on a binary channel (ZigBee at the time of implementation). The sensor data is then communicated to the framework through an event bus acting as middleware communication layer (1.2 in Figure 8.4). This event bus has been implemented using ActiveMQ and the Extensible Messaging and Presence Protocol (XMPP) a.k.a. Jabber for our first deployment. The context acquisition module is handling the reception of the data from the numerous sensor specific modules. It then publishes each event into the context stream (2) and the KB (KB.1). The ubiquitous component registry provides a registration scheme (1.1) that enables the plug & play mechanism described further in section 8.3.5.

Context Synthesisers

The context synthesisers are a class of modules for which the purpose is to provide a higher level of context information to other modules. The reasoning paradigm used by these modules can be anything from imperative algorithms to machine learning, passing of course by semantic inference engines. Each module extracts the information it needs from the KB or its non-semantic counterparts. An up-to-date semantic snapshot of the context is available from the KB (KB.2), a listener can alternatively be set on the context stream (KB.4) and the historical events of the stream are to be queried from the persistent storage (KB.5). The context synthesiser modules then process the information to possibly infer more contextual knowledge. The inferred knowledge is then added or updated in the KB (KB.2).

I have implemented the context understanding module using semantic inference as described earlier in the thesis. Other context synthesisers such as the grammar based activity recognition or the multiple hypothesis (MHT) over DBN modules were implemented by other researchers participating to the

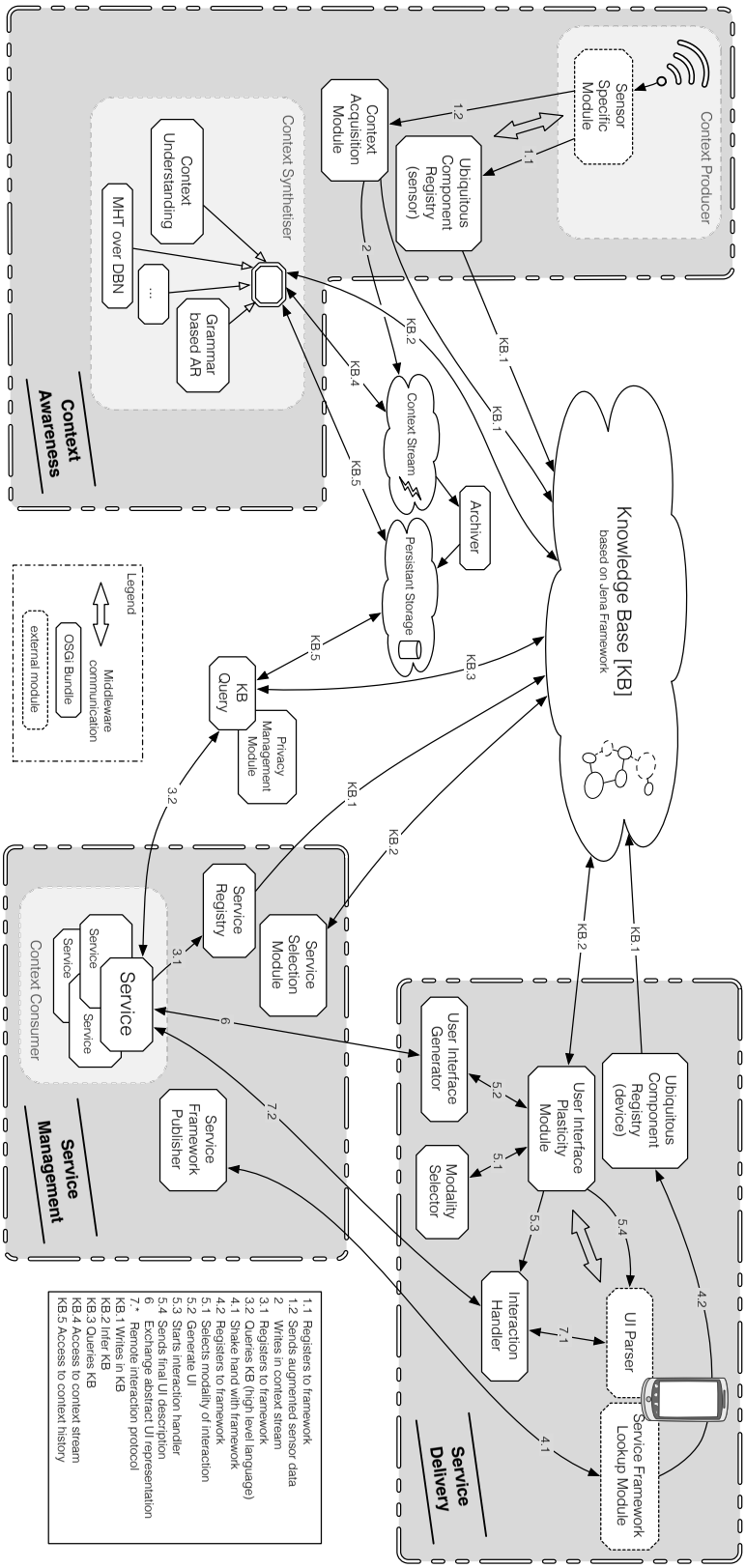


Figure 8.4: Service Architecture of UbiSMART (first version)

AMUPADH project and have been integrated in the framework for the shared deployment in a nursing home in Singapore (see section 9.3).

Knowledge Base

In this version, the KB is implemented using the Jena Framework used as a triplestore for RDF triples. Knowledge producers (which are almost all modules considering the fully distributed reasoning paradigm) provide Jena with triples to be added or updated in the KB (KB.1 and KB.2 in Figure 8.4) whereas knowledge consumers such as the context understanding module or the service selection module query the KB through Jena to extract the knowledge of their interest (KB.2 and KB.3). A non-semantic version of the context is also available to non-semantic context synthesisers through the live context stream (implemented using an event bus) or its persistently stored version. A dedicated archiver module ensures the storage of the context stream.

Service Management

The framework administrates a list of services that can be provided to the residents and are described semantically. Each installed service automatically registers to the service registry module. This module is an extension of the native OSGi registry that adds support for the semantic description of services. A reasoning engine performs semantic matching to select services that are useful to residents in real-time based on the contextual information present in the KB. This is supported by the service selection module which implements the rules described in section 5.4.1. Information about the (de)activation of services is updated in the KB (KB.2) so that the service delivery modules can handle the actual provision to the resident.

It has also been designed but not implemented that services could query the KB using a high level language to obtain some contextual information in order to adapt their content. The theoretical "KB Query" module and its associated privacy management module are dedicated to this (3.2).

Service Delivery

Finally, the interaction must be instantiated between residents and services. The User Interface Plasticity (UIP) module adapts the interaction to residents' profile and their context of use in order to build the most natural and seamless interaction possible. First the modality of interaction is chosen using semantic inference by matching the context, the selected services and the available devices for interaction into the KB (KB.2 and 5.1, see section 5.4.1). An adapted User Interface (UI) must then be generated (5.2) based on the abstract UI provided by the service (6) and the parameters of the device available in the KB. The generated UI is then sent to the device (5.4) where a parser instantiates the service. A module is finally handling the feedbacks and interaction between the device (7.1), the service (7.2) and the UIP module (5.3).

It has also been designed that devices might have to discover a service framework before it registers and communicates with it. The theoretical service framework publisher (which is like a broadcaster of the framework's identity) and service framework lookup module are planned for the discovery of the framework (4.1). A registration to the framework is then needed in the same way as for sensors (4.2).

8.2.2 Communication

One can observe from the reasoning architecture presented above that independent reasoning engines do not communicate results to one another. Instead, all inferred knowledge is updated in the shared KB, thus made available to other modules. This has been designed to avoid low-level dependencies between modules in the framework, thus increasing the modularity and easing the hot plugging of new modules. Consequently, there is no service invocation used to start the inference of each module and a mechanism must be found to handle the reasoning cycles. Naturally, the naive approach would be

to set all reasoning engines to perform their inference at a given frequency. However, as the inference is the most time consuming process in the framework, I wish to avoid the unnecessary processing that would result from such approach. Hence we must find a way for modules to receive a "token" when it is their turn to perform the inference.

The system designed is based on a **publish and subscribe** mechanism around topics, which modules can publish on, or subscribe to. When a knowledge producer updates the KB, it publishes a token on the topic corresponding to the update, then knowledge consumers that have subscribed to this topic are forwarded the token and can perform their inference. Topics are actually linked with the type of knowledge that producers are updating and consumers are interested in. For instance, there could be a topic "location" on which modules tracking the location of residents would publish and modules using such information (e.g. activity recognition modules) would subscribe to. The token sent on the topic can be anything from a simple "flag" to inform that an update was made, to the triple that was updated itself.

We assume that the communication protocol chosen should allow reasoners distributed over a network to communicate and should preserve the modularity of the system. Each module should indeed be able to (un)register freely without having any impact on other modules. The flexibility of registration for the modules reminds of course of the capabilities offered by OSGi and its packaged event bus. However, it is not a valid option since OSGi is meant as a single framework system, whereas we seek to provide a pervasive event bus system.

Our choice has been to create the pervasive event bus based on an XMPP server and Multi-User Chat (MUC) rooms to implement the topic system. **XMPP** is an open source technology for real-time communication, which powers a wide range of applications including Instant Messaging (IM), MUC, voice and video calls, collaboration, lightweight server-based middleware and generalized routing of XML data [163, 164]. We have used it as an open-standard communication protocol for event-based communication between modules. The system runs an XMPP server using **openfire**, a cross-platform open source XMPP server [165]. For each topic, we open on the server a MUC room that modules can connect to and share messages through. Sending a message on a room is equivalent to publishing on the corresponding topic, whereas logging in to a room and setting a listener to receive its messages is equivalent to subscribing to the topic.

XMPP provides a platform independent communication protocol. Indeed, since it is used by many IM services like Google Talk (up to May 2013) or Skype, libraries are available in numerous programming languages running on all kinds of connected media-enabled devices like computers, smartphones or tablets. It enables lightweight IM services but is also used in many industrial projects as middleware for pervasive computing systems. It is known to be scalable as it can be used for small applications as well as deployed on a worldwide scale to build services such as Google Talk.

8.2.3 Sequence Diagram

For a better understanding of this "distributed" reasoning architecture, and to ease the comparison with the "hybrid" reasoning architecture described in section 8.3, Figure 8.5 provides an overview of the sequence of service invocations between the different modules. One can see that several engines are contributing to the inference in each of the three aspects of the framework (Activity Recognition, Service Selection and User Interface Plasticity). The context understanding module is a special case as it does not updates the KB (Jena) directly, but instead does it through the service selection module. The `update()` calls between engines actually corresponds to the token being passed over XMPP as described in section 8.2.2. It is not implemented as an actual service invocation but this representation is used for improved clarity.

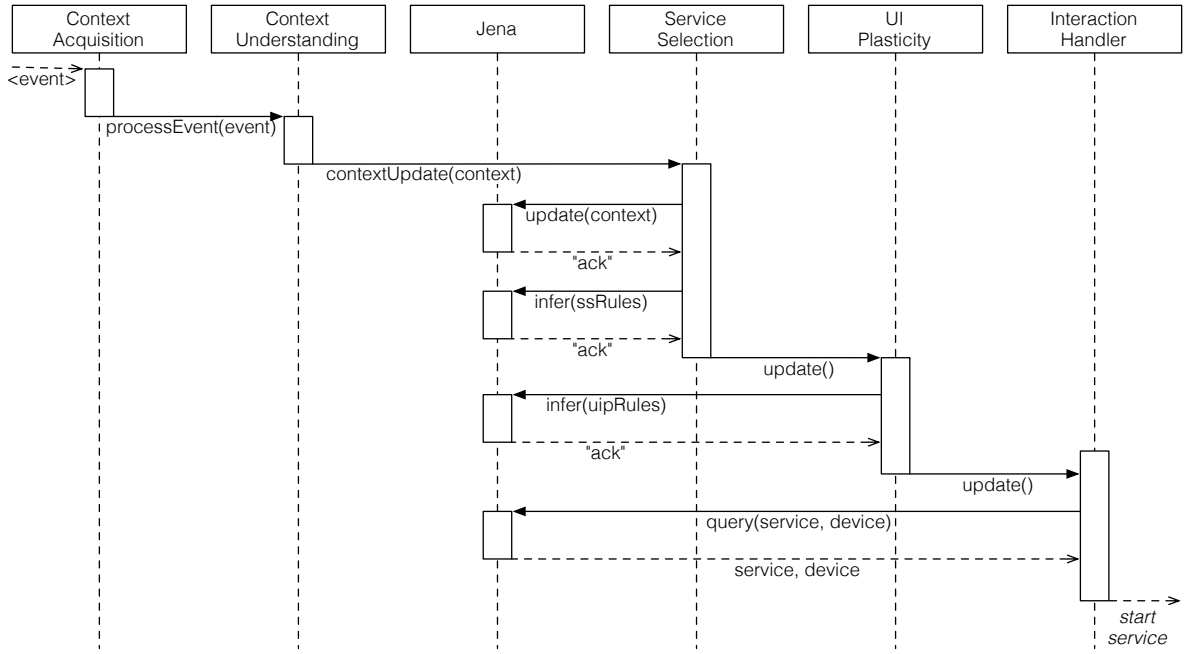


Figure 8.5: Sequence Diagram for the Reasoning in UbiSMART v1

8.2.4 Detailed Implementation Using Jena Inference Engine

As explained in section 8.2.1, the first version of UbiSMART is implemented around a shared KB developed with Jena, and relies on the inference capability of Jena to perform the semantic selection of services and devices depending on the contextual information available. Jena is a Java framework for building semantic web applications that provides a programmatic environment for RDF, OWL and SPARQL and includes a rule-based inference engine. The implementation following the SOA paradigm, the framework is developed as an OSGi container hosting the different modules. Jena has been bundled as required and publishes a handful of services constituting the API to query and infer the KB. This API is used by the reasoning modules of the framework, such as the modules for service selection or UI plasticity. This implementation is illustrated in Figure 8.6. The figure also includes a second OSGi container that hosts all the low-level modules that gather and process sensor signals to provide contextual information to the service selection module. The inter-container communication, which is not defined by the OSGi specification, relies here on the same XMPP and MUC protocol as the one used in intra-container communication for the tokenization of the inference between the different reasoning modules (3 and 5 in Figure 8.6). I describe in this section how the KB API provided by Jena is used, e.g., for the invocations numbered 1, 2 and 4 in Figure 8.6.

Syntactic Introduction

Jena has its own syntax for rule writing, which can be parsed by the integrated engine. A full description of the rule grammar is available in annex C.1 and some elements necessary to the comprehension of this section are given hereafter. Jena rules allow the use of prefixes to shorten the URIs of the models where classes and properties are defined. Prefixes like `rdf:` or `owl:` are used to refer to standard RDF or OWL properties. The prefix `pre:` is used to shorten the URI of our own model. Rules can call some built-in primitives which are predefined procedural functions like `lessThan(?x, ?y)` (return true if the value of x is less than the value of y) or `notEqual(?x, ?y)` (return true if both variables are

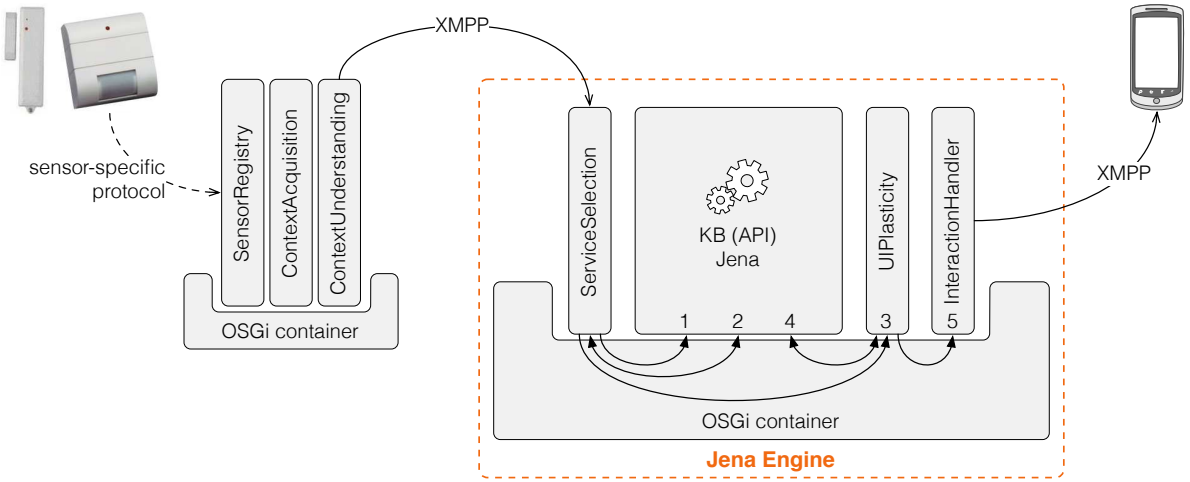


Figure 8.6: OSGi Bundling of the Jena Engine

different). As Jena rule expressivity is quite limited (some operators are missing like logical negation or the existential qualifier), I implement specific built-in methods to fit our needs. Built-in methods can be defined and registered to the reasoner in Java using the following Java line of code:

```
BuiltinRegistry.theRegistry.register();
```

A very basic rule, corresponding to the transitivity associated to the subclass relation between the classes `Resident` and `People`, is given below for example:

```
[Transitivity:  (?X rdf:type pre:Resident) -> (?X rdf:type pre:People)]
```

Implemented Ruleset

In this section, I describe the rules used and their implementation. The work described here is inspired by the doctoral work of Mossaab Hariz [27]. I differentiate three types of rules: population, propagation and access rules. These have nothing to do with, and must not be mixed up or compared with the types of rules described in section 5.4.2. *Population* rules are combined with model queries into what I call the *population mechanism*, which uses information received from the event bus to instantiate or remove new individuals or relations in the model. *Propagation* rules are in charge of building more knowledge based on the data already in the model, they are mainly finding the changes made to the model by population mechanisms and propagating these changes further in the model. *Access* rules play the "output" role; they can for example search for undelivered services for which a device of interaction has been selected and send the services to the corresponding devices.

Population mechanism The population mechanism instantiates new individuals or relations in the model depending on the messages received through the event bus. For example, when an update of the contextual information is received, we must link in the KB the resident to his context with the `hasContext` relation. The first thing is to check the model to know whether this relation already exists or not. This is done using the Java querying code in Source 8.1 where `user` and `context` are arguments of the corresponding API method.

Source 8.1: Existantial Query with Jena

```
String sQuery = "PREFIX pre: <"+modelURI+"> " +
```

```
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
    "SELECT DISTINCT ?U ?C WHERE { " +
    "?U rdf:type pre:Resident . " +
    "?C rdf:type pre:Context . " +
    "?U pre:hasContext ?C ." +
    "?U pre:name \""+user+"\" ." +
    "?C pre:name \""+context+"\" ." +
    "}";

Query query = QueryFactory.create(sQuery);
QueryExecution qe = QueryExecutionFactory.create(query, model);
ResultSet result = qe.execSelect();
```

When the query is executed, results are bound to the `ResultSet` object `result` which can be checked to know about the prior existence a such a relation in the model. In case the relation does not exist, rules are applied to infer the model and create it. The rules used are given in Source 8.2 with the Java code starting the inference. In this case, the ruleset is defined as a string in Java since variables like `user` and `context` are required. However, when no variables are required, it is preferable to define the rules in a separate file.

Source 8.2: Population Rules in Jena

```
// prepare built-ins
BuiltinRegistry.theRegistry.register(new stopService());
BuiltinRegistry.theRegistry.register(new clearContext());

// call rule to clear similar context and instantiate new one
PrintUtil.registerPrefix("pre", modelURI);
PrintUtil.registerPrefix("rdf", rdfURI);
String rules =
  "[StopService:  (?U rdf:type pre:Resident) (?C rdf:type pre:Context) (?S rdf:type pre
      :Service) (?U pre:name \"" + user + "\")" + "(?C pre:name ?contextName) notEqual
      (?contextName, \"" + context + "\")" + "(?U pre:hasContext ?C) (?S pre:helpsWith
      ?C) (?S pre:runningFor ?U) -> stopService(?S)]" +
  "[ClearContext: (?U rdf:type pre:Resident) (?C rdf:type pre:Context) (?U pre:name \""
       + user + "\") (?C pre:name ?contextName) notEqual(?contextName, \"" + context +
      "\") (?U pre:hasContext ?C) -> clearContext(?C)]" +
  "[NewContext:   (?U rdf:type pre:Resident) (?C rdf:type pre:Context) (?U pre:name \""
       + user + "\") (?C pre:name \"" + context + "\") -> (?U pre:hasContext ?C)]";

Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));
InfModel infModel = ModelFactory.createInfModel(reasoner, model);
infModel.prepare();
```

Here again we define prefixes first. The rule `ClearContext` checks for an older context of the resident to remove it from the model, it uses the primitive built-in `notEqual` and calls the built-in `clearContext`. The rule `StopService` works similarly to the previous rule but stop running services linked to an eventual older context of the resident. The rule `NewContext` creates the relation between the resident and his new context.

Built-ins are registered to the reasoner as shown in Source 8.2; they can then be used in the rules as long as they comply with some requirements emerging from their extension of the class `BaseBuiltin`. I provide in Source 8.3 the skeleton for the implementation of a built-in method.

Source 8.3: Built-in Skeleton for Jena

```
static class BuiltinSkeleton extends BaseBuiltin {
  // VARIABLES
  final private String NAME = "BuiltinSkeleton"; // name of the builtin
  final private int NBOFARGS = 2; // number of arguments of the builtin

```

```
  public String getName() {
    return NAME;
  }

  public int getArgLength() {
    return NBOFARGS;
  }

  public void headAction(Node[] args, int length, RuleContext context) {
    //TODO implement here action to perform when builtin called from a rule consequent
  }

  public boolean bodyCall(Node[] args, int length, RuleContext context) {
    boolean result;
    //TODO implement here test to perform when builtin called from a rule antecedent
    return result;
  }
}
```

Four methods must be implemented, `getName` has an obvious role, `getArgLength` returns the number of argument the built-in call should have in the rule, the two last methods are the one called by the rule. `bodyCall` is called when the built-in is used in the rule antecedent. It should test some conditions and return a boolean. `headAction` is called when the built-in is used in the rule consequent and implements the desired processing.

Propagation rules As explained shortly, propagation rules detect changes made by the population mechanism and infer more knowledge emerging from these modifications. For example, when a new context is detected which should trigger the provision of a service, propagation rules can be used to infer the service to start and link it to the relevant resident. The corresponding simplified rule is given in Source 8.4. The rule used in this example is generic, and thus does not use any variables; it can be defined in a file that is parsed by Jena using the code in Source 8.5. Here, as earlier, the rules are parsed into a `Reasoner` object which is applied on the ontological model to create an inferred model.

Source 8.4: Example of a Propagation Rule in Jena

```
@prefix pre: <example.owl#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

# Start service for new context
[StartService:  (?U rdf:type pre:Resident) (?C rdf:type pre:Context) (?S rdf:type pre:
    Service) (?U pre:hasContext ?C) (?S pre:helpsWith ?C) notExist(?S, pre:runningFor,
    ?U) -> (?S pre:runningFor ?U) (?S pre:sent "false")]
```

Source 8.5: Parsing of a Rule File in Jena

```
List<Rule> rules = Rule.rulesFromURL("localPathToRuleFile");
Reasoner reasoner = new GenericRuleReasoner(rules);
InfModel infModel = ModelFactory.createInfModel(reasoner, model);
infModel.prepare();
```

Access rules I take here the example mentioned above where access rules are used to start services that need to be started. The implementation is similar to the propagation rules, only the rule file used changes as described in Source 8.6.

Source 8.6: Example of an Access Rule in Jena

```
@prefix pre: <example.owl#>
```

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

# send service when it has not been sent yet
[CheckService:  (?U rdf:type pre:Resident) (?S rdf:type pre:Service) (?S pre:runningFor
     ?U) (?S pre:sent "false") (?U pre:name ?user) (?S pre:name ?service) ->
    sendService(?service, ?user)]
```

Here we search for services that have not been sent by verifying the value of the `sent` property. If there is a candidate, the name of the service is held in the `?service` variable and the name of the corresponding resident in the `?user` variable. The service is then sent to the user using the built-in `sendService`.

Summary The ruleset described here is similar to the real implementation, the differences lies in the number of rules deployed and in the complexity of the related queries or built-ins. Queries are always similar to the one given here in the way they are written. Built-ins also remain similar to the example explained above: the call gives in argument URIs or names of resources to use, the ontological model can then be navigated from these resources to eventually find other resources or statements (triples) to remove or to bind new ones.

8.2.5 Performance Validation and Discussion

In order to validate the performance of this first version of UbiSMART, we implemented and integrated it into a context-aware reminder and notification system, which we deployed in a nursing home in Singapore (details in section 9.3). The first aspect in which I wish to judge the system's performance is regarding its uptime. In this aspect, we learnt a lot from our deployment in the nursing home. Due to malfunctions coming from different parts of the system, we started with an uptime of three days in December 2011. After identifying and solving or improving the issues one by one, we managed to increase the system's uptime to eleven days in May 2012. Over this six months period, we observed and gathered data about the different reasons that would cause the system to malfunction. In order of frequency, the main reasons observed were the failure of sensors' batteries, packets being lost over the WSN, failures in the reasoning, sensors being removed, and WiFi disconnections. Figure 8.7 represents the relative frequency of these issues. The sensors' battery issue was reduced by changing

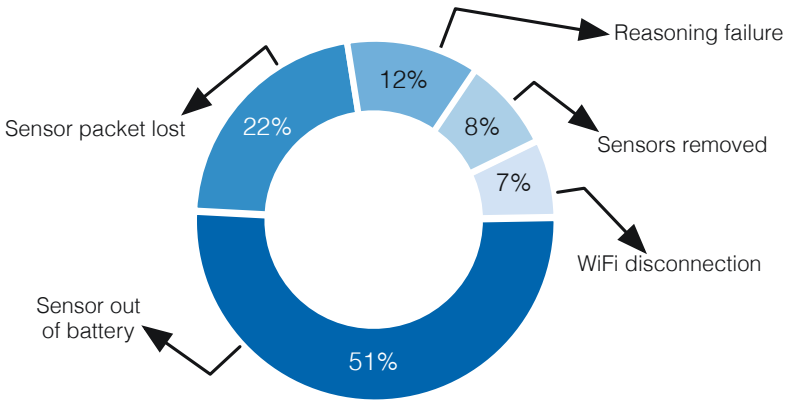


Figure 8.7: Relative Frequency of the System's Malfunctions in Peacehaven

the class of the batteries used, shortening the maintenance cycle, and reducing the frequency of wireless communication as this is the most power consuming process. The loss of sensor packets was reduced in two ways: by using a mesh topology for the WSN but this increased power consumption, and

subsequently by calculating Cyclic Redundancy Check (CRC) and sending an acknowledgement on the messages received. WiFi disconnections have been limited by upgrading the hardware used for the WiFi network. Sensors removed could not be helped much, except by explaining again the reasons of the deployment to the residents. Reasoning failures are detailed hereafter.

We observe that reasoning failures are responsible for 12% of the system's malfunctions. Further debugging of the engine demonstrated that flaws of reasoning could be observed in simple cases when increasing the complexity of rules in a way that several rules would collaborate on a single decision. As explained in section 5.3.2, I believe that this reasoning issue is due to Jena's partial OWL-DL entailment. Without requiring any further profiling or load test to estimate the performance of my reasoning engine, I decided to compare Jena with other inference engines to make a more informed technological choice. Consequently, I stopped using Jena in favour of EYE as argued in section 5.3.2.

In parallel, we gathered information about other areas in which the framework could be improved. I describe below the main points that were addressed in the second design of UbiSMART.

1. I believe it should be simpler to implement inference modules. Developers should only need to focus on rules while relying completely on the declarative approach to separate application logic and underlying models. The convenience of use of Jena is limited because is it a bit too low-level and its implementation-style remains close to the imperative paradigm. Moreover, the inference sequence should not have to be fixed a-priori, since this is an obstacle to the deployment of more heterogeneous inference modules. These issues are addressed by the centralized semantic inference proposed in section 8.3.

2. It should be possible to use non-semantic processes for more computational operations of inference, or even introduce data-driven techniques in the reasoning. This concern is addressed by the hybrid reasoning architecture proposed in section 8.3, and more specifically by the introduction of Cerebration modules in the engine.

3. We want to introduce a communication layer with interchangeable protocols to move part of the framework to a remote server when and if needed. Such a layer would also allow adapting the communication protocols used to the peculiar settings of each deployment. This communication layer is proposed and described in section 8.3.4.

4. It would be useful to reuse sensor signal processing algorithms based on use-cases. In order to do so, we introduce the SensorFlowManager and SensApps in section 8.3.

5. We must make it easier to deploy and configure the framework and its related hardware (sensors and devices). A semantic plug & play mechanism is proposed in section 8.3.5 and enable our configuration tool called Smart Space Composer (S2C).

8.3 Hybrid Reasoning Architecture: UbiSMART v2

8.3.1 UbiSMART's RESTful Architecture

The second version of UbiSMART is based on the first one and improves it in two main aspects: its cloud-readiness and its semantic inference as a service. Indeed, the architecture of the framework was modified towards two distinct purposes. On one hand, emerging from the bottom-up approach described in section 3.1.2, there is a need to go towards a "cloud-proof" design in order to provide services over hundreds of homes with health assessment features. Modules have been added to handle the heterogeneity in the communication protocols used for the sensor networks deployed in various homes, as well as between the gateways and the server. The new design allows for the pre-processing of the sensor data to be performed on a local server or in the cloud (see the "Sensor Flow Manager & SensApps" section below). Therefore flexibility has been added in the communication layer as detailed

in section 8.3.4. On the other hand, the architecture has been reviewed from the ground up in order to provide not only a share KB to the various modules but also a shared semantic inference service, which modules can register their rules to be processed in a centralised manner. This centralisation allows for rules of the same level (i.e. rules that feed each other or provide similar information) to be used without generating conflicts or deadlocks, as it would have been with the first architecture of UbiSMART due to the predefined order in which rules would be applied. In other words, where UbiSMART v1 would have had modules infer the KB one by one in a predefined order, thus giving a sort of priority to some rules over others, UbiSMART v2 actually gathers the rules of all modules using semantic inference and run them at once leaving the inference engine (i.e. EYE) to handle the multiple pass of inference needed to reach a stable inferred ontological state. The second architecture, called "RESTful" since it relies on a REST design to enable the communication between the deployed hardware and the server-based framework, is illustrated in Figure 8.8.

As explained in section 5.4.1 and illustrated in Figure 8.8, the inference mechanism follows the "Data-Information-Knowledge-Wisdom" (DIKW) paradigm. *Data* is the meaningless (i.e. no semantics attached) and mostly useless (due to its redundancy) signal coming from the sensors. *Information* is non-redundant and has attached semantics; it is available in an ontological form; thus it is more useful and calculable. Among the modules described below, *Information* is derived from *Data* by the SensApps and its semantics is attached by the Stimulistener. *Knowledge* is inferred from *Information* by fusing different sources and by incorporating domain knowledge; it formalises the contextual information at a higher level, closer to how a human being would formulate. The transformation of *Information* into *Knowledge* happens at the reasoning stage, especially in the Cogitation module and the various Cerebration modules as will be described further in section 8.4.

NTriplestore (KB API)

In the first design, all modules were sharing a central KB implemented using the Jena framework for Java. In this design, we keep a shared KB with an attached API to update the ontology following the Create Read Update Delete (CRUD) paradigm. Since we are using the N3 syntax in this new version of UbiSMART, nor Jena neither any triplestore can be used to implement the KB. Thus we implement our own triplestore, focusing mainly on the features required for our specific use-case and keeping for further development the full standard implementation. More details are provided about this triplestore in section 8.3.3.

The tricolour cube represented on top of the NTriplestore module in Figure 8.8 is the Semantic Web logo. Over the rest of graph, each module containing the logo represents a "semantic-aware" module, i.e. a module able to query the triplestore following the CRUD paradigm through the NTriplestore services.

"X" Gateway

An "X" Gateway is a sensor gateway written specifically for the "X" communication protocol. For instance, we have implemented a ZigBee Gateway for our sensor deployment and a DB Gateway in order to use historical data stored in relational databases. Other gateways can be implemented according to the needs and choices made to deploy wireless sensor networks. A gateway is a low level, protocol specific implementation providing features like the normalisation of data coming from sensors, the forwarding of the data over a REST link to the server-based framework, etc. It is dependent on the specific link to the hardware gateway (the wireless communication module receiving the signals from all sensors) which can for example use RS232 or USB connectivity.

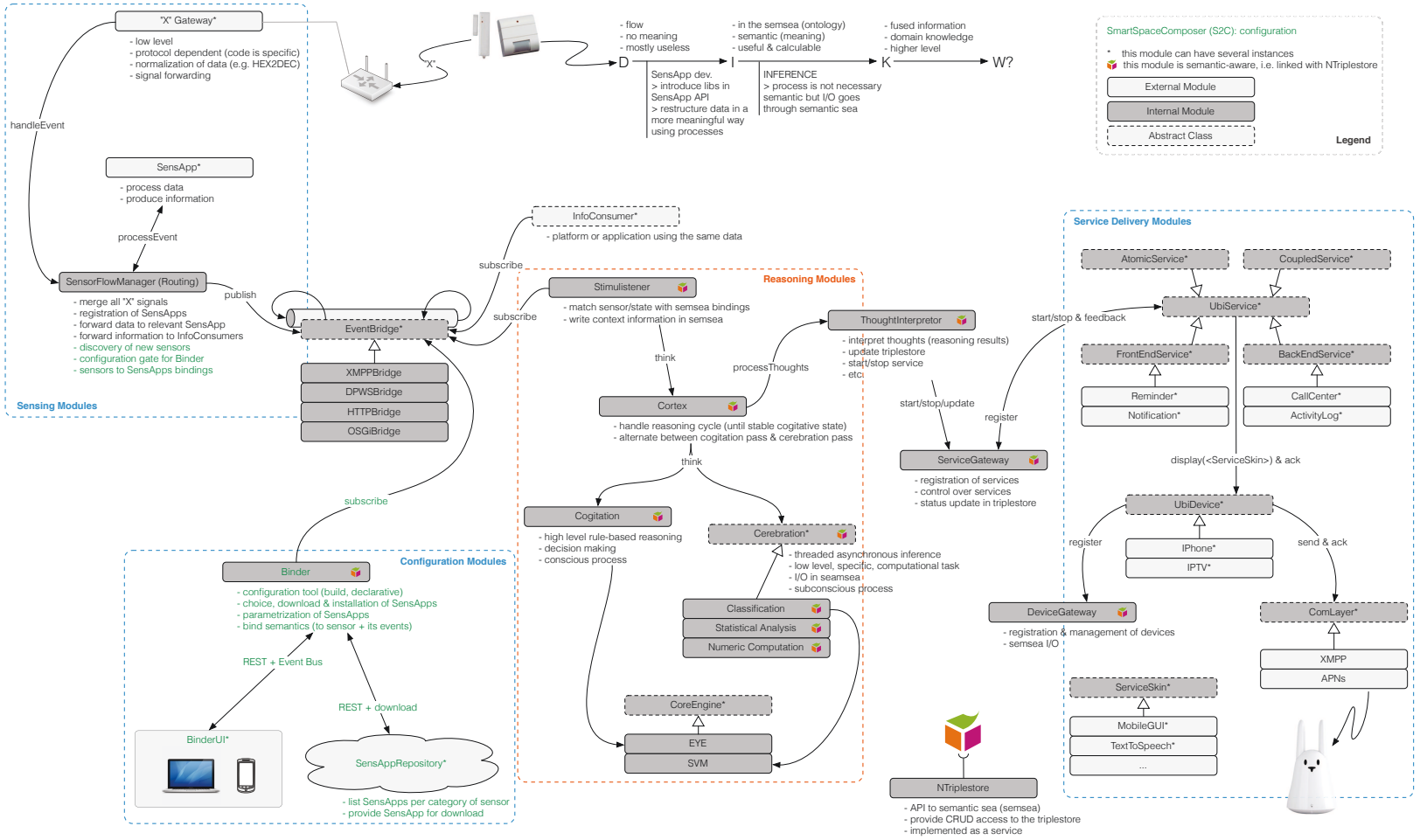


Figure 8.8: RESTful Architecture of UbiSMART (Second Version)

Note: The reasoning modules are detailed in Figure 8.14

Sensor Flow Manager and SensApps

The SensApps are independent applications that process sensor data in order to produce information related to a specific use-case. For example, a SensApp could detect the usage of a water jug based on the signal coming from a accelerometer/gyroscope sensor attached to it. Such application can be written specifically for a deployment, shared into a repository on the web and downloaded on demand.

Sensor Flow Manager is making the link between incoming sensor signals and SensApps. Therefore it matches the sensor event received with the different SensApps that can process this event based on the configuration of each sensor. The `processEvent` method uses a publish & subscribe event bus with SensApps based on semantic topics of communication that are generated according to the sensors' configurations. Sensor Flow Manager is an entry point to the framework for the signals coming from the various "X" Gateways in use. It can be running on a local or cloud server. If it runs locally, `handleEvent` makes a local REST query while the event bus to Stimulistener connects to the cloud server with an XMPP or Hypertext Transfer Protocol (HTTP) bus for example. If it runs in the cloud, `handleEvent` makes a internet REST query while the event bus to Stimulistener can be local, for example using OSGi's EventAdmin service. More details is given about the event bus communication in section 8.3.4.

Sensor Flow Manager is also handling a part of the discovery and configuration of new sensors. Indeed, when it receives the signal of an unknown sensor, it forwards it to the Binder module described below which provides configuration features for sensors. Binder will then send the relevant parameters back to Sensor Flow Manager for the binding between the new sensor and available SensApps.

Configuration with Binder

The configuration aspect of the framework is pretty much at the design level for the time being as the full mechanism has not been implemented. It is however extending the plug & play mechanism described in section 8.3.5 which was implemented. Binder receives the signals from new sensors that lets it extract information such as sensor type, identification number, etc. Binder is also connected to the SensApp repository from which he obtains all available SensApps compatible with the new sensor (based on its type). It then combines all information needed for the configuration of the new sensor. This information is provided as a web service to the BinderUI module which is a front-end client for the configuration of sensors. BinderUI is dynamically updated using AJAX in web browsers and XMPP or push notifications on native mobile applications.

The configuration of a sensor is partly saved in the triplestore, partly sent to the Sensor Flow Manager. The semantic description of the bindings between the sensor and its deployment environment, its possible states and the type of data it may provide are written in the triplestore in order to enable the inference of contextual information. The information needed for the binding of the sensor's signal to the relevant SensApps is sent back to the Sensor Flow Manager. Once a sensor is configured successfully, its signal is automatically forwarded to the relevant SensApps and then to Stimulistener, it disappears from the Binder's duty and client.

Stimulistener

Stimulistener is the entry point to the framework from a semantic point of view. It is the module updating the triplestore with each event received after its translation into the semantic syntax. It subscribes to the topics where the information coming from sensors (augmented by SensApps or not) is published by Sensor Flow Manager. When an event is received, it translates it based on the sensors' semantic description available in the triplestore and updates the triplestore accordingly. Once the update done, it starts a reasoning cycle by invoking Cortex's `think` method.

Reasoning Modules

The reasoning mechanism in UbiSMART v2 is a hybrid mechanism that incorporates semantic inference centralised as a service in the Cogitation module, and a distributed non-semantic inference through various ad-hoc approaches implemented under the Cerebration family of modules. The Cortex module is handling the reasoning cycles between Cogitation and Cerebration. It makes sure a stable decision is reached before the result is sent for further processing. The designed mechanism is described in details in the dedicated section 8.4.

Persistent Triplestore Updates and Service Control

Once a stable inferred state of the ontology is reached, the inferred triples are sent to be interpreted by the Thought Interpretor module which performs the relevant actions. For instance, persistent updates to the triplestore are made and services are started, stopped or updated as required using the corresponding services provided by the Service Gateway module. The Service Gateway is handling the registration of the end user services started in the framework and provides control methods as services. It also updates the status of the services in the triplestore when it changes. In this aspect, the second design is very similar to the first one. Similarly, the services and devices modules are used in the same manner as in the first design.

8.3.2 Sequence Diagram

As introduced above in the "Reasoning Modules" paragraph, UbiSMART v2's reasoning has evolved into a centralised semantic inference, coupled with a distributed network of ad-hoc non-semantic reasoning engines. For the semantic inference, each independent reasoning module provides its rules to the Cogitation module through a registration step at the framework start-up. The Cogitation module then single-handedly infer the ontology, applying all registered rules and thus leaving to EYE engine itself the handling of potential conflicts or deadlocks that might emerge from the independently designed rules. In the case of the non-semantic engines, factorisations have yet to be needed, thus a distributed mechanism is satisfactory. The sequence diagram resulting from this evolution is given in Figure 8.9 and is meant for comparison with Figure 8.5.

We observe among others how the serial organisation of the semantic engines from Figure 8.5 has evolved into a parallel organisation in the new design. Indeed, the modules for context understanding, service selection and UI plasticity now register their rules at the framework start-up, and all rules are inferred at once by Cogitation. One can also note the combination between the Cogitation module handling the semantic inference, and the various Cerebration modules introducing other reasoning techniques. These modules are put at the same level in the design and feed each other with ontological updates until a stable decision can be made by Cogitation. The whole hybrid reasoning mechanism is further detailed in the coming section 8.4.

8.3.3 Extra: N3 Triplestore

Motivation

The context-awareness aspect in UbiSMART is ensured by the "live" processing of events gathered from sensors on field. These events are fetched in real-time and added in the KB to be processed by the reasoning engine. The premier engine used in the hybrid architecture is EYE which uses primarily the N3 language, hence the KB is implemented as a N3 triplestore. N3 is a superset of RDF, the mainstream semantic language, which extends it in many ways in order to enable the expression of statements describing logic, and their inference. Despite its fundamental superiority on RDF, N3 has not yet received the adoption it deserves from the Semantic Web community. Therefore some tools are still not available off-the-shelf, such as triplestores which are dedicated databases for the storage,

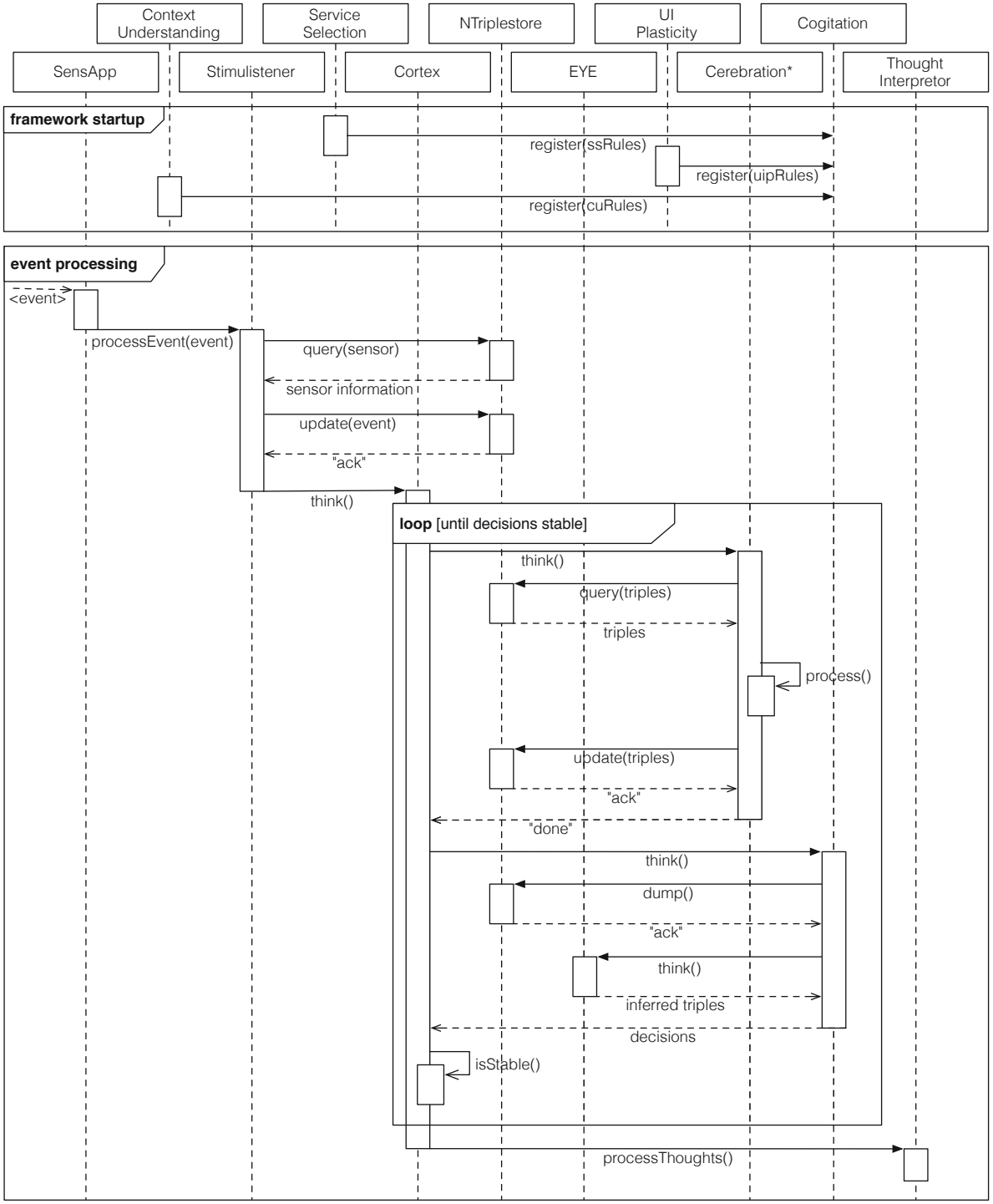


Figure 8.9: Sequence Diagram for the Reasoning in UbiSMART v2

update and retrieval of triples. Consequently, and in order to use EYE reasoning engine and N3, we have several options:

- store the triples into N3 files,
- store the triples into a RDF store and feed RDF to EYE which can handle the translation,
- store the triples into a relational database and use some existing but yet to be completed approach to handle the translation (SPARQL CONSTRUCT e.g. via D2RQ, a database to RDF mapper),
- or implement our own N3-compliant triplestore.

The first option is our legacy approach but is rather tedious when it comes to implementing the file parsing and modification. I observed that it is also very inefficient in term of processing time. Not relying on N3 file parsing is actually the main motivation behind using a triplestore. Due to the fundamental loss of expressivity incurred by the second option and the apparent technical complications of the third, we choose to design and implement our own purpose-build triplestore, i.e. a triplestore designed with as much modularity and abstraction as possible to ease further developments but for which only specifically required features are implemented. We call it **NTriplestore**.

As far as I am aware of, there has been only one prior initiative to provide a triplestore for N3. This initiative was supported by Ruben Verborgh, a doctoral researcher at the Ghent University's Multimedia Lab, Belgium [166]. However, the store in its current state does not yet support the use of namespaces, which is essential in our case. Thus we hope to switch to it if it reaches completion as it aims to be fully compliant with the standards, but we choose to implement our purpose-built version for the time being.

Implementation

The class diagram for NTriplestore is given in Figure 8.10. It is built around four main functional requirements. First, it is implemented as a service that any other module can invoke based on its interface published through the OSGi architecture. Second, it is able to read N3 files and parse them, including the various syntactic sugars and prefixes. Third, it serves as a cache for the quick manipulation of triples. Thus, queries are implemented to search, add, remove and update triples. Finally, it can dump the whole content of the triplestore or a part of it into a file, in order e.g. to launch EYE reasoner.

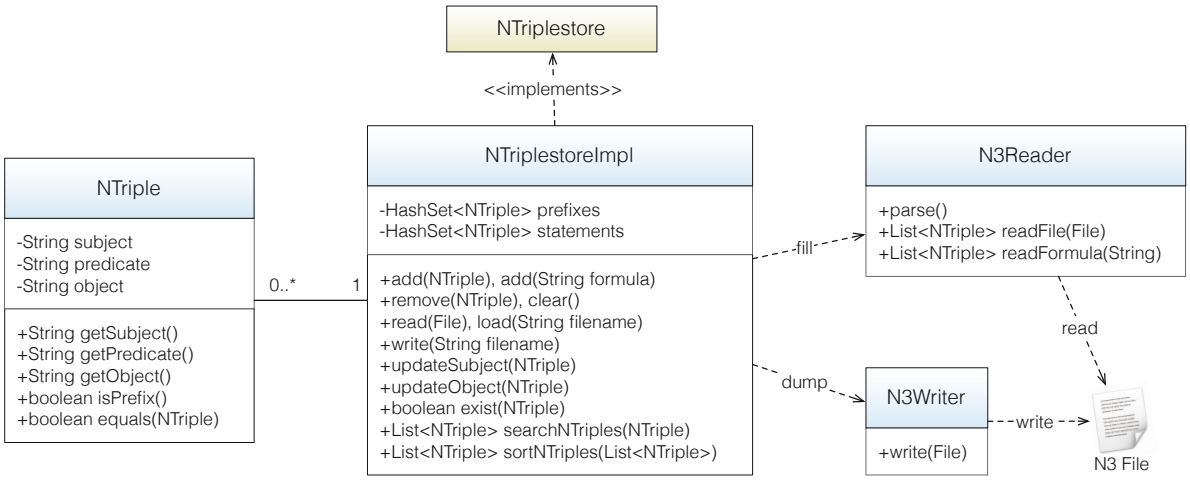


Figure 8.10: Triplestore Simplified Class Diagram

To ease the manipulation of N3 triples, the `NTriple` class is available providing the basic representation of $\{subject, predicate, object\}$ triples, together with accessors and usability methods for the comparison of triples for example. In NTriplestore, the collection of NTriples are stored in a **HashSet**—a collection that uses a hash table for storage—in order to ensure the uniqueness of each NTriple. The `searchNTriples` method requires as argument a partial description of the triples to be found where unknown fields (e.g. the object) should be replaced by "$*$". Searching for the triple $\{*, *, *\}$ would thus return the whole content of the triplestore, whereas searching for $\{Bob, hasDaughter, *\}$ will return the triples binding Bob with each of his daughters. Update methods are dedicated to update one field of a triple, for example `updateObject` will remove all triples with similar subject and predicate from the triplestore and replace it by the triple given in argument. `exist(NTriple)` performs a search for the triple in argument and returns `true` if a match is found, `false` otherwise. The N3Reader provides methods to parse either files or formulae, i.e. String variables that contain one or several triples. Thus `readFile` is invoked by the methods `read(File)` and `load(String Filename)`, while `readFormula` is invoked by `add(String formula)`.

Performance Gain

In order to evaluate the gain in performance due to the triplestore, we have conducted an experience over 10,000 events gathered in our deployment in France (about a week of data for a single occupant in a private home). For each event, the corresponding triples are added or updated in the ontology; then the inference is performed by EYE reasoning engine. During the experience, we measure the overall reasoning time, which we refine into the time taken for the inference by EYE itself, and the time taken for the various updates to the KB. There are about 10 to 20 updates for each event: a few when the event is received before the reasoning, and more after the inference to perform the persistent updates in the KB. All updates are implemented using file parsing and rewriting in one case, and by invoking triplestore methods in the other. The size of the ontology inferred varied only a little and was of 350 triples in average. The results of the experience are presented in Table 8.1.

Table 8.1: Performance Comparison: N3 File Parsing vs. NTriplestore

Processing time for:	File parsing	NTriplestore	Improvement factor
Whole reasoning	1,226ms	150ms	8
EYE inference only	148ms (12%)	147ms (98%)	-
KB updates only	1,078ms (88%)	3ms (2%)	359

Nota: Average measurement over 10,000 iterations corresponding to a single house with an average of 350 triples. Computed on a Linux server (Ubuntu 12.10 32-bit) powered by an Intel Core 2 Duo CPU E6550 at 2 x 2.33GHz and 3.8GB RAM.

We observe that using a triplestore is much more efficient in term of processing time as compared to parsing and rewriting N3 files as we used to do. The overall reasoning is performed 8 times faster than it used to be. Moreover, when discarding the time taken by EYE itself, which is independent from my implementation, the KB updates are performed 359 times faster in average. In conclusion, the main issue met when using N3 due the unavailability of off-the-shelf tools is solvable and my choice of this language consists less of a trade-off.

8.3.4 Communication

Following the "cloud-proofing" of UbiSMART's design, it is necessary to rethink the communication layer since the protocols used in the first design are more suited for a home set-top box kind of deployment. We particularly wish to gain modularity in the choice of the protocols used for the deployments in order to adapt to the peculiarities of each deployment, for example making it possible to shift either the SensApp processing, or the whole reasoning from the cloud to a local server and back.

Taking into account the peculiarities of each deployment would also let us make an appropriate choice of the communication protocols used depending on the volume of data, or the use of mobile devices, etc. We introduce in this second design an abstract event bus called Event Bridge, which has numerous protocol-specific implementations. For instance, and as illustrated on Figure 8.11, interchangeable bridges can be implemented based on XMPP, Devices Profile for Web Services (DPWS), HTTP or simply OSGi's EventAdmin service. Each bridge provides access to a publish and subscribe event bus by wrapping the underlying communication protocol. E.g. OSGi already is based on the publish and subscribe paradigm so the wrapping is trivial, however an equivalence needs to be made between XMPP MUC rooms and the publish and subscribe topics. This can be done as described in section 8.2.2. In each of the bridges, events are actually dictionaries of *(label, value)* pairs.

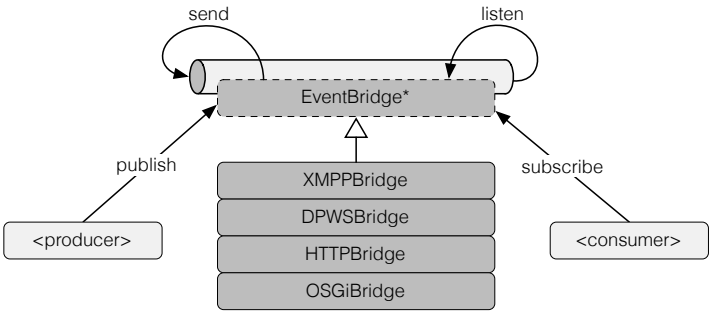


Figure 8.11: Abstracted Event Bridge for In-Place Replacement of Communication Layer

The main idea here is to enable a standard in-place exchange between interchangeable event buses in order to easily adapt to the requirements of deployments. For instance, we rely in the UbiSMART v1 on XMPP MUC for the communication. This is not a high-speed eventing channel, thus it may be inadequate if rapid inferencing is required. The abstracted event bridge allows to select another eventing protocol and replace XMPP MUC as and if needed. In Figure 8.8, I have represented the Event Bridge only for the communication between the Sensor Flow Manager and the Stimulistener due to space constraints, but this mechanism is actually used for the `processEvent` call between the Sensor Flow Manager and the SensApps (see details in the dedicated section above), as well as for the `handleEvent` call from the Gateways to the Sensor Flow Manager. For instance, in the `handleEvent` call, the HTTP bridge can be used by the sensor gateway gathering binary signals from a wireless sensor network, thus decreasing the processing power needed and enabling a minimal gateway that could be running on an Arduino platform. On the contrary, to receive signals from a smartphone where a two-way communication may be desirable, an XMPP bridge might be a more suitable choice.

8.3.5 Extra: Semantic Plug'n'Play

This section describes a work realised in close collaboration with Hamdi Aloulou, for which the mechanical part of the mechanism and most of the implementation was part of his doctoral work defended successfully in June 2013. More details can therefore be found in his doctoral dissertation.

Introduction

To build AAL spaces or smart spaces in general, one must integrate a line-up of entities: a network of sensors, a reasoning engine, environment actuators, interactive devices and services. By enhancing the modularity and flexibility of our UbiSMART framework, we could go towards a larger scale of deployment without decreasing the customizability of the proposed solution. SOA is beneficial since it provides mechanisms for the deployment and maintenance of entities as well as for the communication

between them [61]. However, these mechanisms only apply to software entities that are packaged into OSGi bundles. Hence, in order to extend the OSGi-based modularity we enjoy with services to hardware entities as well, we propose an OSGi-based discovery protocol of hardware entities enabled by the automatic generation of software bundles representing them into the framework [167]. This mechanism acts as plug & play support for the sensors, actuators and devices deployed in the environment. Although adding and removing entities is made much faster and simpler, we only provide here a **mechanical plug & play** where entities can discover each other and start exchanging data. They actually ignore each other's bindings with the environment, which makes the full understanding of the data exchanged impossible. E.g. it could be useful to know where is one particular motion detector deployed, or who is carrying a given handphone. Being able to parse data received from a new unknown entity is not enough; you need to be aware of its semantics. We therefore designed a **semantic plug & play** where entities (services, sensors, actuators or devices) provide their semantic profile when "shaking hands" with the framework. This profile can be edited during the development, the deployment, or updated at run-time by users or even other entities. Hence, a real plug & play behaviour is created where new entities are able to genuinely *understand* each other to collaborate.

Discovery and Bundle Automatic Generation

In the literature, pervasive systems often utilize a layer providing a level of abstraction common to all entities, helping communication, discovery and collaboration using protocols and data formats [61]. Our alternative approach is to use semantic web technologies to bring down to each entity the possibility to understand newly discovered other entities, thus decreasing the overhead on this layer, which is then solely in charge of a higher level system coordination.

I illustrate in Figure 8.12 the communication steps between a newly added entity and the service framework. Steps are divided in (numbered according to Figure 8.12):

1. discovery and registration on a entity-specific communication protocol (e.g. ZigBee for sensors, IP over 3G or Bluetooth for devices),
2. bundle automatic generation by the module in charge of the registration,
3. system-level registration to the environment discovery module which updates the KB accordingly.

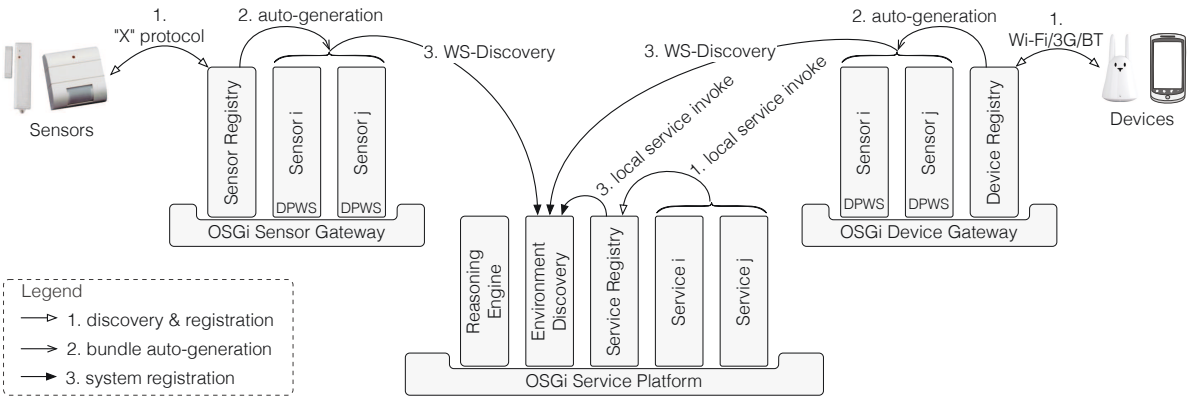


Figure 8.12: Discovery, Registration and Communication Protocols for the Plug & Play

A sensor registry bundle has been implemented to handle the ZigBee communication between sensors and the framework. When a sensor is turned on in the environment, this bundle receives the new signal (1 in Figure 8.12) and automatically generates a bundle representing and describing the sensor in the

framework (2 in the Figure). A similar mechanism ensuring hot plugging for devices is partially implemented and supports heterogeneous communication layers (e.g. WiFi, Bluetooth, 3G). To handle the discovery and events exchange between the different bundles in the framework, we are using the DPWS protocol. DPWS uses several standards from the web services specification—namely the Web Services Description Language (WSDL), Web Services (WS)-Discovery, WS-Eventing and Simple Object Access Protocol (SOAP)—in order to advertise and discover bundles, as well as for events exchange. Once a bundle representing an entity in the environment is generated, it uses the WS-Discovery protocol to advertise itself and send a description of its capabilities (3 in the Figure). A DPWS client (the environment discovery bundle) is handling the discovery of this bundle on the framework side and updates the KB with a semantic description of the entity. Other modules can then obtain the entity's description from the KB and start exchanging data.

Knowledge Base Update

Let us consider that the ontology is build from a set of files written in N3 and containing different kinds of information; its update (e.g. by the environment discovery module) is reduced to files parsing and modification. As illustrated in Figure 8.13, there is a file (skeleton.n3) constituted of the classes and properties that can be instantiated in the whole system to represent the current contextual information. It is the TBox of the ontological model. Another file (environment.n3) contains the knowledge coming

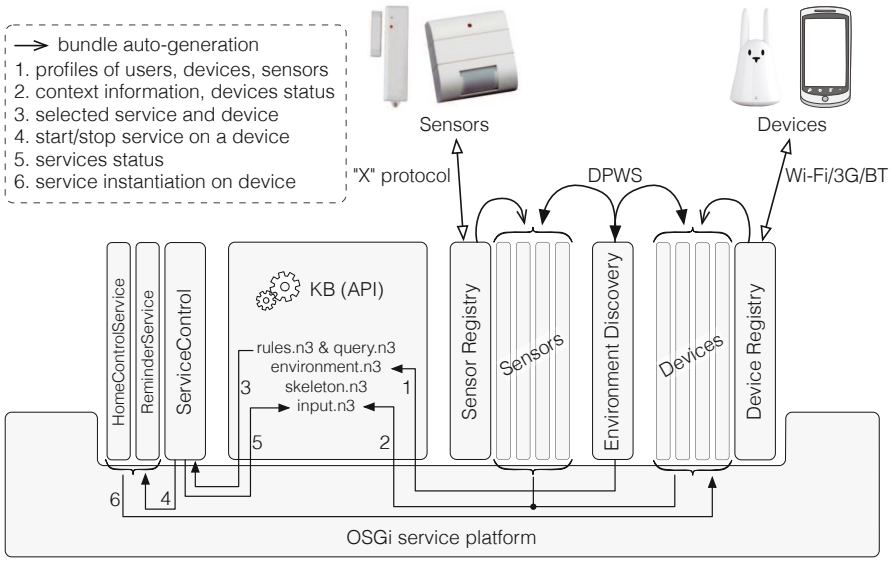


Figure 8.13: Detailed Semantic Process Supporting the Plug & Play Mechanism

from the environment discovery phase: e.g. actual users and their profile, or sensors, devices and services along with their semantic profile. Two files (rules.n3 and query.n3) contain the rules and queries necessary for the inference process, thus centralize the application logic, i.e. is the system context-aware decision-making. Finally, a file named input.n3 is updated at run-time through a dedicated interface to reflect the changes in the environment: real-time context information, services or devices status, etc. On Figure 8.13—a single container version of Figure 8.12 that removes the pervasive aspect of the mechanism but helps to understand the different updates and queries of the KB by the different modules—one can see that the environment discovery module updates the environment file (1), thus making it possible for the events coming from the sensors or devices to be translated, added into the KB (2) and understood by other modules and used, among others, for the context inference (3).

8.4 Detailed Implementation of the Hybrid Architecture

In this section, I describe in details the implementation of the reasoning part of UbiSMART v2. I provide as main support the detailed class diagram of the reasoning modules in Figure 8.14. This diagram covers only the part of Figure 8.8 related to reasoning; the modules concerned are Stimulistener, Cortex, Cogitation, Cerebration and EYE. As Cerebration is an abstract class, I consider as example MotionEstimator, an implementation of Cerebration that computes an estimate of the motion in each room of the house. In order to keep the diagram readable, I have omitted the multiplicity information when it was equal to 1. The reader should thus take 1 as the default multiplicity throughout the diagram. The NTriplestore module is not described in details here as it was already done in Figure 8.10. Considering a replacement of the interface in Figure 8.14 by the full diagram of Figure 8.10 is however correct and advised; one must then add the Activator for the NTriplestore bundle, which was not represented in Figure 8.10.

In the class diagram, each bundle is delimited by a dotted line. One can note that each module has an `Activator` implementing the `BundleActivator` class provided by OSGi API; this is a requirement under the OSGi specification. The `Activator` provides generic methods for starting and stopping a bundle, it is an equivalent of the `Main` class in traditional Java, since it is the class creating the instances of other classes, linking them together, starting them, and more specifically here registering their services to the OSGi container's service registry.

In the following, I will also describe how each module updates the ontology in order to clarify the mechanism and highlight the complementary roles of all modules. Therefore, I take one simple example, used as a guiding thread, and illustrate partly the ontology and its updates around the example use-case. Figure 8.15 is the initial state of the ontology to be considered. In this example, a person named John Doe lives in a house and is currently detected moving in the livingroom; there is no movement in any other room of the house, and the activity inferred previously is "occupied", i.e. John Doe is moving without alarming behaviour but we do not know exactly what he is doing. The ontology is represented here from a purely ABox point of view, i.e. the individuals of the ontology are represented without being attached explicitly to their classes. The objects' and properties' names are a simplified and more readable version of their URI.

8.4.1 Stimulistener

Stimulistener is the module receiving all events coming from sensors and updating the ontology accordingly. As represented on Figure 8.14, the `Stimulistener` class extends `EventHandler`, which is the class for the reception of events under OSGi API. `Activator` creates a `StimulistenerImpl` object and registers it as the listener on the relevant topic of the event bridge in use. Thus, the `handleEvent()` method of this object is called by the event bridge each time a new event is published. `handleEvent()` adds the event in the `LinkedBlockingQueue` publicly provided by the `Activator`, where it is fetched by `StimuliDecoder`. `StimuliDecoder` is a thread started by `Activator` that retrieves events from the queue and performs the ontological updates illustrated in Figure 8.16.

In the example, an event is received from sensor "A2" with state "on". The decoder queries the triplestore to get the URI of the sensor with id "A2"; answer is `pir2`. The sensor state's URI is then obtained by appending the state to the URI of the sensor: `pir2_on`. And the new state is updated in the ontology replacing the triple `(pir2, hasCurrentState, pir2_off)` by `(pir2, hasCurrentState, pir2_on)`. The timestamp of the event is also used to update the `hasLastUpdate` datatype property. A simplified extract of the corresponding source code is given in Source 8.7.

Source 8.7: Triplestore Query and Update by Stimulistener

```
// extract values from event
sensor = (String) event.getProperty("sensor");
value = (String) event.getProperty("value");
```

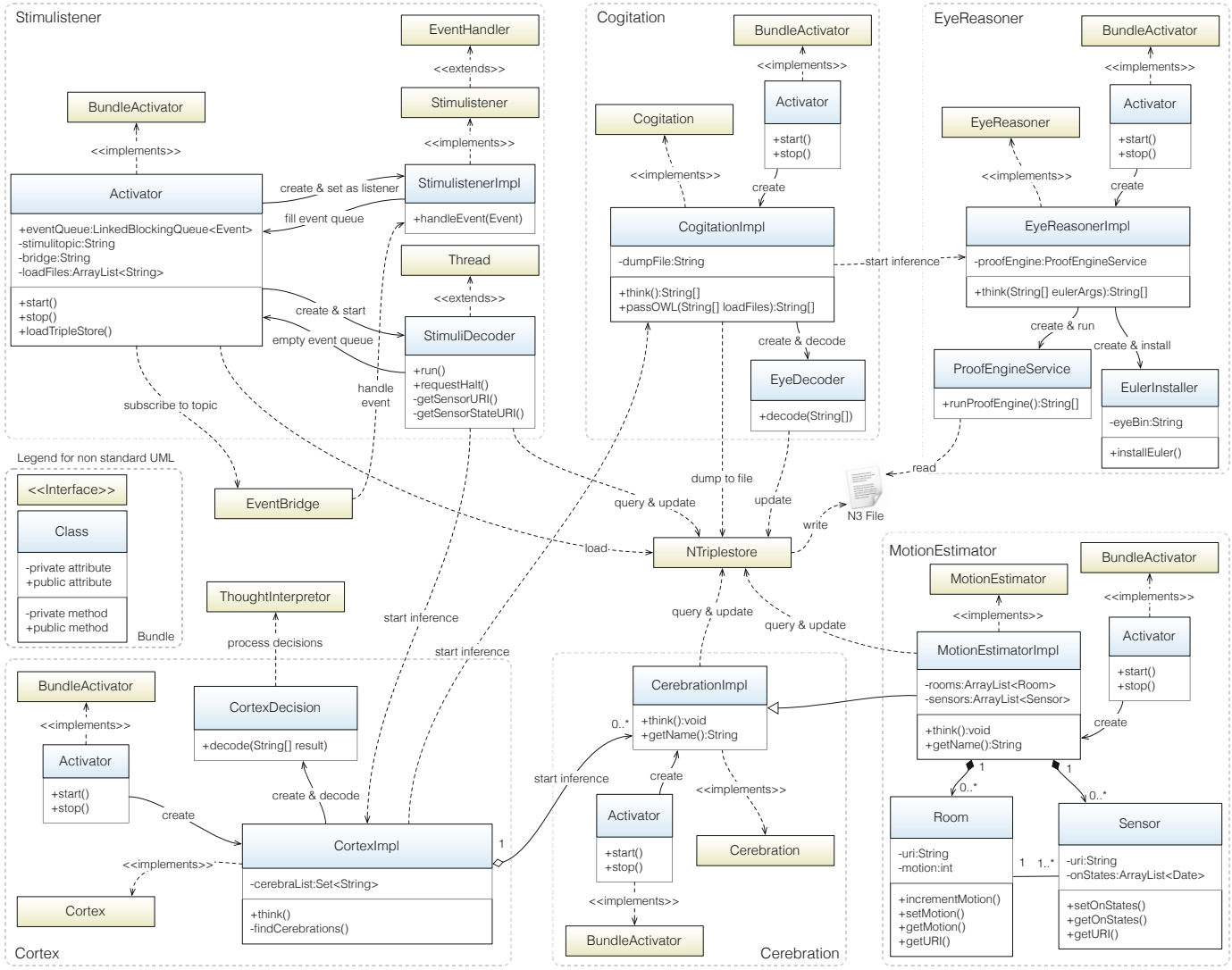


Figure 8.14: Detailed Class Diagram of the Reasoning Modules in UbiSMART v2

Note: Default link multiplicity is 1.

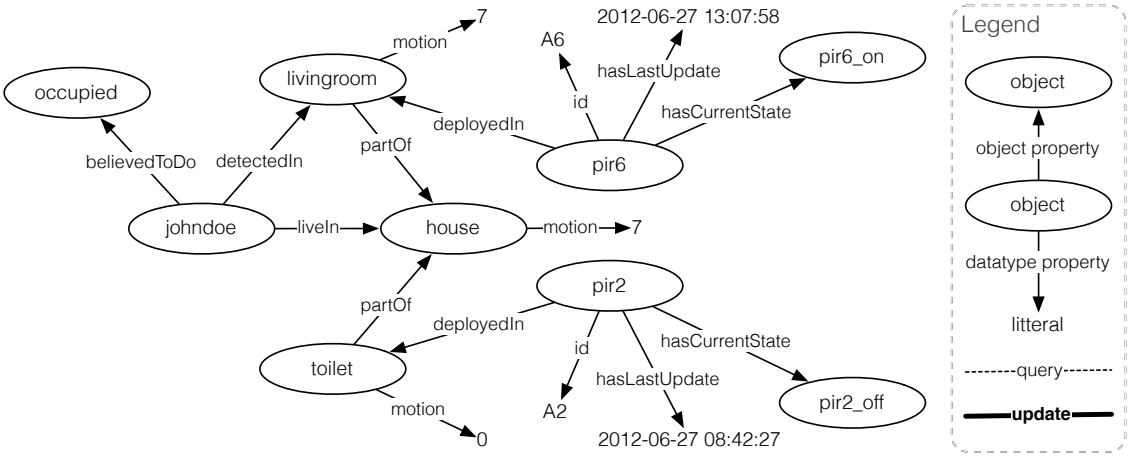


Figure 8.15: Ontology Evolution Example: Initial Ontology

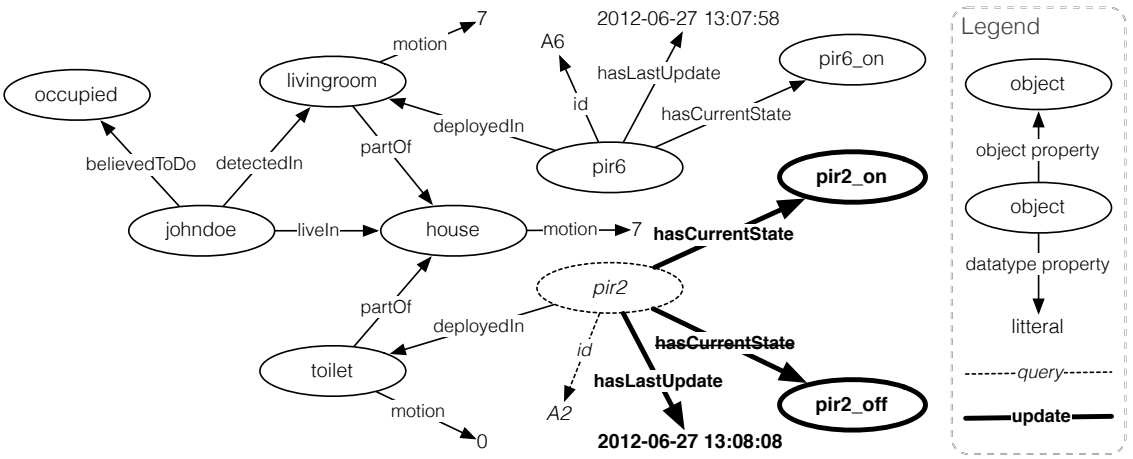


Figure 8.16: Ontology Evolution Example: New Event Update

```
time = (String) event.getProperty("time");

// update sensor current state
n3Store.updateObject(getSensorURI(sensor), MODEL_NS+"hasCurrentState",
    getSensorStateURI(sensor, value));
n3Store.updateSubject(getSensorURI(sensor), MODEL_NS+"hasLastUpdate", "true");

// update last update time
n3Store.updateObject(getSensorURI(sensor), MODEL_NS+"lastUpdate", dater.getN3Time(time)
    );
```

8.4.2 Cortex

After updating the ontology, `StimuliDecoder` invokes Cortex to start the inference using the `think()` method. This is possible because Cortex's `Activator` has created a `CortexImpl` object and published its interface into the service registry. Cortex is the module managing the reasoning cycles, calling alternatively Cerebration and Cogitation to perform their inference on the ontology, and observing

the stability of the resulting decisions to decide whether to stop and proceed with the results, or launch a new reasoning cycle. Cogitation is a single bundle, which Cortex discovers in the same way `StimuliDecoder` has discovered Cortex: through the service registry. Cerebrations, however, are numerous and a-priori unknown to Cortex. Cortex here makes use of the dictionary-based discovery mechanism provided by OSGi's service registry and finds all modules that have published their service with the type "cerebration". This is how each Cerebration implementation's `Activator` publishes the bundle's services. The Cerebration modules are cached in Cortex for better performance.

Essentially, Cortex's `think()` method consists in a loop where each Cerebration module is invoked in an arbitrary order, then the Cogitation module is invoked and returns its result. The result is decoded and analysed by `CortexDecision` which keeps an history of the decisions and analyses their stability. When a stable decision is reached, the loop is terminated and the decision is given to `ThoughtInterpretor` for further processing.

8.4.3 Cerebration and MotionEstimator

As indicated previously, Cerebration was introduced to perform more complex or processor-hungry inference. For instance my semantic model is a stateless, memoryless model that can be seen as a snapshot of the contextual knowledge; so if memory is needed to infer some kind of information, a Cerebration module can be implemented with an imperative approach to handle the memory locally (i.e. out of the ontology), perform the inference and dump the result back in the ontology. This allows not to complicate the semantic model too much, in order to preserve the sanity of the knowledge base designer and the low processing time of the overall inference. Similarly, when inference operations are too computational, using semantic rules might not be optimized so Cerebration modules offer the possibility to perform the computational part in imperative, with all necessary optimizations, and dump the result back into the ontology. Input data for Cerebration modules are always queried by the modules themselves, and output data updated by them as well. Hence, the Cerebration modules are aware of the semantics of their I/O but are not bound to any processing technique for the inference.

For instance, we see in Figure 8.14 that MotionEstimator uses a Java model for the rooms and sensors in order to keep the history of the sensor events (the `onStates` ArrayList in the `Sensor` class). When the bundle's `think()` method is called, the current states of the sensors are queried from the ontology (see Figure 8.17), filtered and stored in the `onStates` ArrayList. `onStates` is then filtered according to the time-window of observation and event are counted as an estimate of the motion in each room. The values obtained are stored in objects of the `Room` class and later translated to be updated in the ontology as can be seen in Figure 8.17. A simplified version of the corresponding source code is given in Source 8.8. One can note that in order to switch the knowledge between the ontological representation and the Java representation, the URI for each Java object is preserved in its attributes.

Source 8.8: An Example Cerebration Process for Windowed Motion Estimation

```
// get clock time
String sClock = (String) n3Store.searchURIs("hom:clock", "qol:hasValue", "?").toArray()
    [0];
Date clock = dater.getDate(sClock);

// remove on states out of time window and compute motion in each room (sensor by
    sensor)
for (int i=0; i<sensors.size(); i++) {
  ArrayList<Date> on = sensors.get(i).getOnStates();
  for (int j=0; j<on.size(); j++) {
    // remove if too old
    if(clock.getTime() - on.get(j).getTime() > TIME_WINDOW) {
      on.remove(j);
    }
```

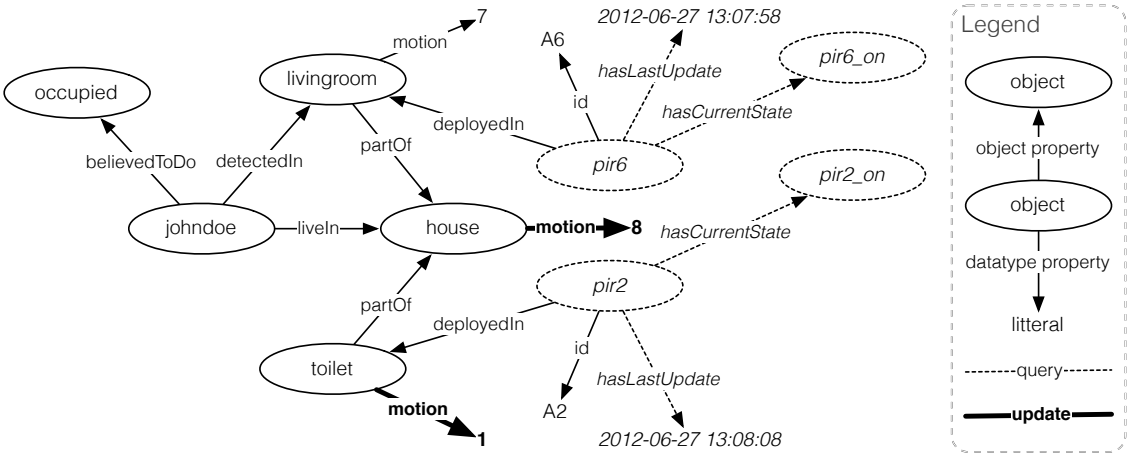


Figure 8.17: Ontology Evolution Example: Motion Estimation by Cerebration

```
    }
14    sensors.get(i).setOnStates(on);
      int roomID = sensors.get(i).getRoom();
      rooms.get(roomID).incrementMotion(sensors.get(i).getOnStates().size());
    }

19  // check latest sensor update and update motion estimation (sensor by sensor)
    for (int i=0; i<sensors.size(); i++) {
      // get last update from triple store
      String slU = (String) n3Store.searchURIs(sensors.get(i).getURI(), "qol:lastUpdate", "
          ?").toArray()[0];
      Date lU = dater.getDate(slU);
24    // get arraylist of previous on states
      ArrayList<Date> pon = sensors.get(i).getOnStates();
      // check if last update different from the one in arraylist
      if(pon.size() != 0) {
        if(lU.getTime() - pon.get(pon.size()-1).getTime() != 0) {
29        String state = (String) n3Store.searchURIs(sensors.get(i).getURI(), "qol:
              hasCurrentState", "?").toArray()[0];
          // check if state is on
          if(state.endsWith("on")) {
            // add to arraylist
            pon.add(lU);
34          sensors.get(i).setOnStates(pon);
            rooms.get(sensors.get(i).getRoom()).incrementMotion(1);
          }
        }
      } else {
39      String state = (String) n3Store.searchURIs(sensors.get(i).getURI(), "qol:
            hasCurrentState", "?").toArray()[0];
        // check if state is on
        if(state.endsWith("on")) {
          // add to arraylist
          pon.add(lU);
44        sensors.get(i).setOnStates(pon);
          rooms.get(sensors.get(i).getRoom()).incrementMotion(1);
        }
      }
    }
49
    // sum motion of whole house and update triple store
```

```
int homeMotion = 0;
for (int i=0; i<rooms.size(); i++) {
  int roomMotion = rooms.get(i).getMotion();
54  n3Store.updateObject(rooms.get(i).getURI(), "qol:motionMeasured", Integer.toString(
      roomMotion));
  homeMotion = homeMotion + roomMotion;
}
n3Store.updateObject(house, "qol:motionMeasured", Integer.toString(homeMotion));
```

8.4.4 Cogitation and EyeReasoner

Cogitation is the module where the semantic inference described in chapter 5 is performed. Its implementation is quite straight forward as it relies mainly on the inference service provided by EyeReasoner. Indeed, as illustrated on Figure 8.14, the process consists in invoking NTriplestore to dump the whole ontology to a given file, feed EyeReasoner with this file as well as the files containing the rules and queries for the centralised inference, and call `EyeDecoder` to decode the results provided by EyeReasoner.

Cogitation and EyeReasoner have been deliberately separated in order to ease an eventual change of reasoning engine in the future. EyeReasoner is a bundled version of EYE that contains several classes:

- `EulerInstaller` in charge of performing a one-time installation of EYE when the framework starts,
- `ProofEngineService` which is a native class of EYE's Java API, which I modified to fit with the peculiarities of OSGi, notably the dynamic class loading feature,
- and `EyeReasonerImpl`, the higher level class which creates an instance of `ProofEngineService` and provides the inference service.

The ontology updates resulting from the inference using the rules described in section 5.4.2 are illustrated in Figure 8.18. The rules that have fired resulting in these updates are given in Source 8.9.

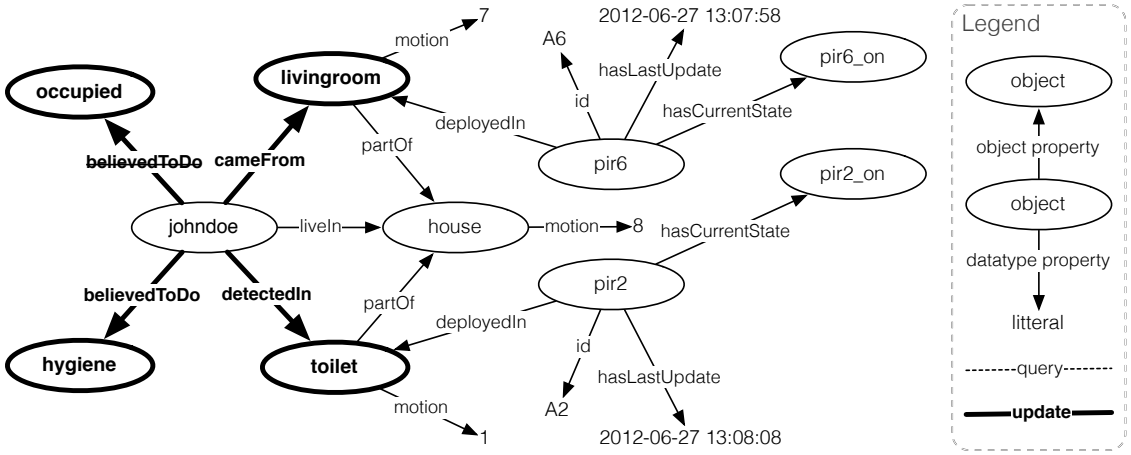


Figure 8.18: Ontology Evolution Example: Cogitation Inference

Source 8.9: Fired Rules Leading to Update in Figure 8.18

```
## tracks resident location in house [persistent]
```

```
{?se qol:hasCurrentState ?st. ?se qol:hasLastUpdate true. ?st qol:indicateLocation true
    . ?se qol:deployedIn ?r. ?r qol:partOf ?h. ?u qol:liveIn ?h. ?u qol:detectedIn ?r2.
     ?r log:notEqualTo ?r2. ?se qol:lastUpdate ?t} => {?u qol:detectedIn ?r. ?u qol:
    cameFrom ?r2. ?u qol:inRoomSince ?t. ts:n3store ts:update {?u qol:detectedIn ?r. ?u
     qol:cameFrom ?r2. ?u qol:inRoomSince ?t}}.

## hygiene activities (limited to 30 minutes)
{?u qol:detectedIn ?r. ?r a qol:Bathroom. ?u qol:inRoomFor ?d. ?d math:lessThan 1800}
    => {hom:hygiene :getScore 9}.
```

8.4.5 Performance Validation

I performed some load test on my reasoning engine by simulating an increasing number of houses (with residents, sensors, rooms, etc.) in the ontology. I run the test for a number of houses ranging from 1 to 250, corresponding to a number of triples in the ontology ranging from 344 to more than 40,000. The result obtained is provided in Figure 8.19. We observe a quadratic evolution of the processing time when increasing the number of houses, while keeping EYE constantly responsible for 98% of the processing duration. I also calculated on one month data the average frequency of events for each one-hour period of the day and found a maximum frequency during the 11:00 to 12:00 period corresponding to one sensor event received every 34 seconds (see Figure 8.20).

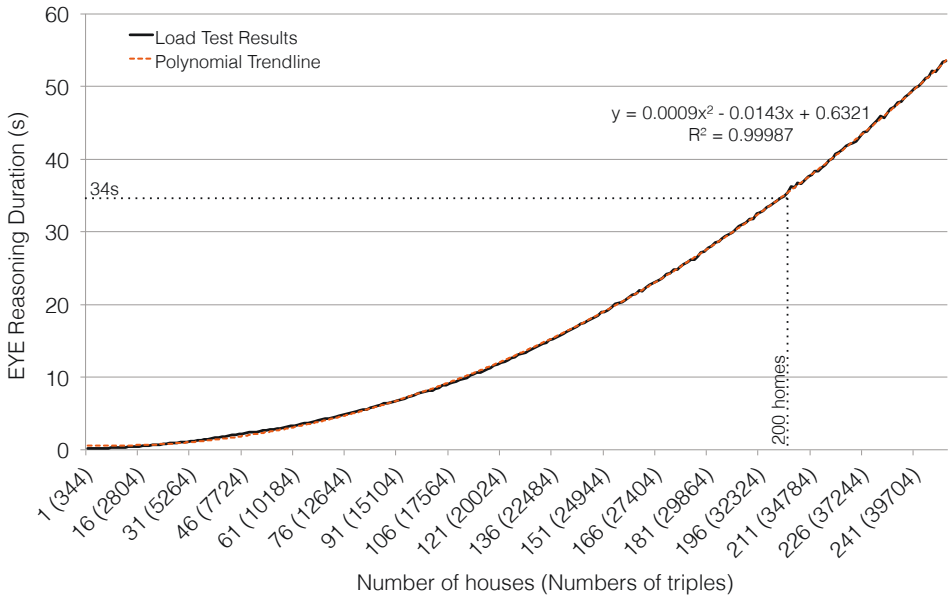


Figure 8.19: Reasoning Load Test up to 250 Houses (40K+ Triples)

Based on the activity observed in this particular house, I estimate that one processor core of the desktop could be processing 200 houses in parallel while preserving just-in-time processing of events. The correspondence between 34 seconds and 200 houses is highlighted in Figure 8.19. The test was performed on a Linux server (Ubuntu 12.10 32-bit) powered by an Intel Core 2 Duo CPU E6550 at 2 x 2.33GHz and 3.8GB RAM. Since a normal server has a much higher processing power, I consider this result as a solid proof of concept of the commercial feasibility.

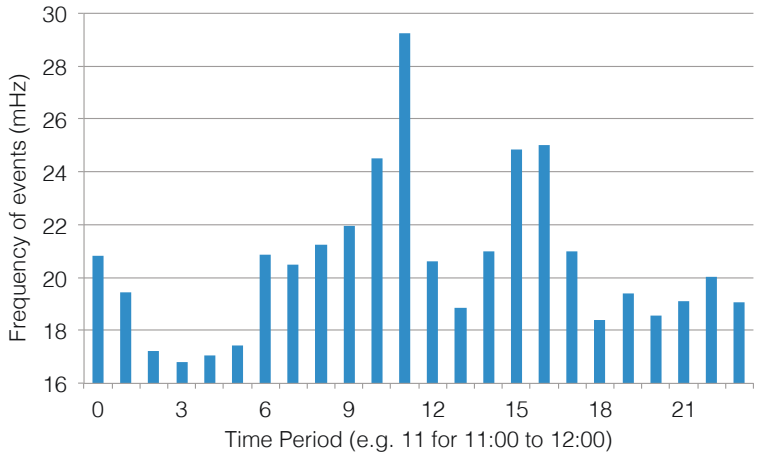


Figure 8.20: Average Frequency of Events for Each 1-Hour Period

8.5 Discussion: Arbitration Between Reasoning Techniques

I have discussed in chapter 6 the potential of combining my semantic approach with more computational methods such as machine learning and data mining techniques. This short section is dedicated to highlight how such a combination can be realised from an implementation and integration point of view. Indeed, we have designed UbiSMART v2 with this integration in mind and did our best to enable it straight from the framework's architecture.

Using data mining techniques requires a two-fold approach. First, a learning phase is required in order to build the relevant model based on knowledge implicitly present in the data. This should be done offline and on a dedicated machine since it is usually a processor-hungry operation and since it requires several steps following a trial and error process in order to be fine-tuned by a data scientist. The requirement here is to be able to log a chosen subset of the KB that will be analysed by various algorithms. In the framework, we enable this by adding dedicated logging features. In the queries performed by EYE to extract decisions and KB updates, I add a predicate to extract selected triples of interest and log them on a specific topic. For instance, `ts:n3store ts:update {formula}` is the rule conclusion used to extract triplestore updates; so I define in the same manner `lgr:a_topic lgr:append {formula}` as the rule conclusion to log triples on the topic "a_topic" given in the subject. The logging is then performed by a dedicated method in `ThoughtInterpretor` and I have also implemented a module to vectorise such triples to prepare for the learning step. The vectorisor module can be used to translate triples into vectors, as well as vectors back into a triple form or a barycentric representation of triples. Hence it allows the representation of the learning results in the ontological syntax.

Then, the actual classification of the context can be performed online based on the model built during the learning phase. This step can be seen as a projection of a subset of the KB into a data-driven space obtained by learning and enabling the classification of the context. The context classification result is finally written in the KB to be fused with the results of other algorithms. Classification algorithms query a part of the KB, vectorise it, classify it using specific techniques, and write the result back into the KB. Hence they fit with the definition of Cerebration modules, so they can actually be implemented and integrated in the framework as such. Numerous techniques can be used for the classification, and each technique might be used in several manners. Thus we propose for core tools (e.g. SVM) used by several modules of data mining to be implemented as a service and shared between these modules, in the same way EYE is available as a service. This can be seen in Figure 8.8.

Part IV

Validation

Nothing is as empowering as real-world validation, even if it's for failure.

— Steven Pressfield, 1943–

9 Deployments and Validation

9.1 Validation Approach

The validation of this doctoral work has been done in a three-fold manner. Firstly, we validated the rule design from a purely logical point of view by performing rules verification using formal methods. This serves as a proof that under defined circumstances, certain properties of the system are verified and certified. The verification process was presented in section 5.4.3. Then we deployed our framework in real settings with users recruited among genuine stakeholders. This in-situ "ecological" testing serves as a verification that the system works properly in the environment of use, i.e. without being permanently monitored and maintained, providing sustained functionality in harsh network conditions, and considering the unpredictable behaviour of real users which might act erratically due to their dementia. Finally, as the rules tested are defined only according the use-cases related to the actual deployments, we ensure that adding complexity in the reasoning by including more triples does not affect the system's behaviour through a load test that has been performed and which result is given in section 8.4.5.

Concerning the in-situ testing of the framework, we have had three complementary deployment phases in different settings. First, we focused on a technical validation of the integrated framework in a controlled yet realistic environment. We deployed UbiSMART in A*STAR's StarHome, a designer 2 bedroom flat with fully equipped kitchen, bathroom, and livingroom. This is described in section 9.2. Then we deployed the system in a nursing home in Singapore where three rooms were set up for activity recognition, enabling real-time reminder and notification services, hence corresponding to the top-down approach (see section 3.1.2). This is described in section 9.3. Finally, we moved to individual private homes in France, with a stripped-down hardware deployment corresponding to the bottom-up approach (see section 3.1.2). This is described in section 9.4. In each deployment, UbiSMART is recomposed from its building blocks according to the peculiarities of the environment and the needs of the residents.

9.2 Technical Validation: STARhome Showcase

9.2.1 Context of the Deployment

The Activity Monitoring and UI Plasticity for supporting Ageing with mild Dementia at Home (AMUPADH) project is a two-year research project (2010–2012) involving the Image & Pervasive Access Laboratory (IPAL), Singapore's Institute for Infocomm Research (I2R) and the School of Computing at the National University of Singapore (NUS). AMUPADH is one of the eleven A*STAR Science & Engineering Research Council (SERC) Home2015 projects. Home2015 is a national research program of Singapore promoting cross-disciplinary research enabling technologies, foundations, or frameworks for future home systems. AMUPADH is focused on the automated recognition of behaviours in smart homes for people with dementia, and aims at proposing a framework for the context-aware provision

of assistive services.

The first deployment of the UbiSMART framework was done in STARhome. An innovative programme initiated and funded by SERC, STARhome provides a home equipped with state of the art technology that facilitate household affairs and build environments catering to the lifestyles of modern families. It is a fully furnished and functional 180 square metres apartment located in the Fusionopolis building where I2R is hosted. STARhome is a technology showcase and a realistic platform featuring leading-edge technologies and sophisticated home concepts that enhance everyday living with ease and convenience. With cameras and one-way mirrors enabling usability studies, researchers are able to gain insights into the needs of future users, thus connecting technology and people cohesively. STARhome has for objectives to be a technology showcase for smart home projects and to provide a technical test-bed in a realistic environment.

The main objective of this experimentation was to perform a technical validation of the system. This phase was not user driven but mandatory before deploying in real conditions with the involvement of end users. My role was to provide a suitable reasoning engine to manage the delivery of services in a context-aware manner. I have collaborated with engineers and other researchers to achieve this goal.

9.2.2 System Description

In STARhome, we deployed our system in each possible space, namely the livingroom, kitchen, bedroom and bathroom. Servers and sensor gateways were setup in the technical part of the apartment, hidden behind one-way mirrors. The hardware architecture of the system deployed is summarised and illustrated in Figure 9.1.

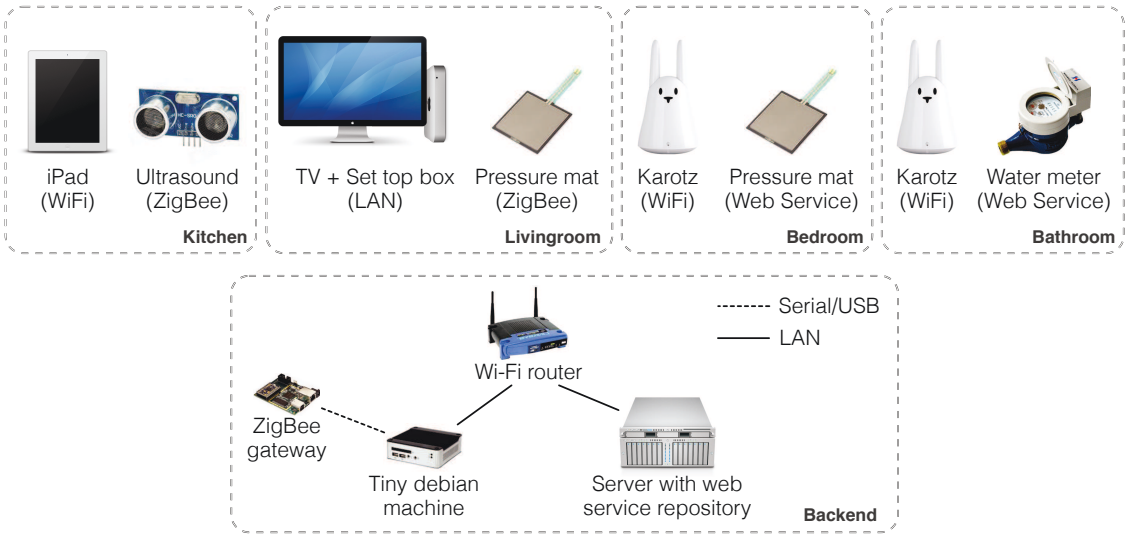


Figure 9.1: Hardware Architecture of the Deployment in STARhome

As presented in Figure 9.1, the system deployed in STARhome incorporates a variety of sensors and devices. We rely partly in this deployment on sensors already deployed in the environment and which data or enhanced states can be consumed via web services available on a local server. This is for example the case for the bed pressure profile detector located in the bedroom (A in Figure 9.2) and the water meter in the bathroom (C in Figure 9.2). The bed pressure profile detector is the product of a previous project deployed in STARhome, which interprets the data received from the fibre optic cable spread under the mattress and provides over the network the occupancy state of the bed (empty, sitting or lying). We also deploy dedicated sensors communicating events to the ZigBee gateway attached to the

debian machine centralising the event processing and reasoning. Over ZigBee, we setup the proximity sensor (ultrasound) represented as B in Figure 9.2 which provides information about the presence of people in the kitchen, as well as a pressure mat (force sensing resistor) placed under the sofa in the livingroom.

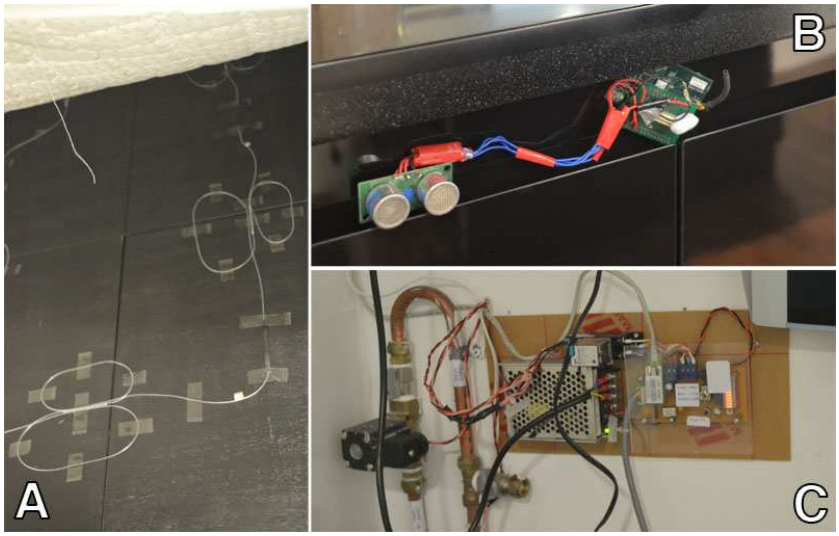


Figure 9.2: Photos of the Sensors Deployed in STARhome
A. Bed pressure profile detector accessed via web service
B. Proximity sensor to detect presence in kitchen
C. Water meter in bathroom accessed via web service

Concerning the interaction for the service delivery, we use in the livingroom an IPTV constituted of an HD TV and our tailor-made set top box. The set top box interface is based on a full screen web browser that fetches a video stream and displays reminders on an upper layer using AJAX and XMPP. Air is also used to provide highly interactive features such as cognitive games, home control or Skype calls. The IPTV is represented as B in Figure 9.3. Additionally, we use in the kitchen an iPad with a dedicated app developed to display reminders received over XMPP (C in Figure 9.3). Finally, we installed in the bedroom and bathroom two Karotz (A in Figure 9.3), which are smart internet-connected rabbits which can move their ears, display some colours through LED illuminated bellies, and play audio files. We used the Karotz mainly as an audio interface with an avatar. The rabbit also embeds a Radio-Frequency Identification (RFID) reader, a small camera and a microphone, which we did not use.

9.2.3 Results

Beside the technical validation, we seize the opportunity of this deployment to estimate the time needed to setup the system into a new environment. Indeed, our goal was to build a flexible system that can adapt to different environments and needs. Therefore, we have analysed the time needed to adapt the operational framework to a new use-case, counting on a team of two engineer-researchers. With the imperative approach used before our adoption of semantic technologies, the first reasoner was written in five days and its subsequent adaptation took three days. We then needed several months to build the first semantic version of the framework. As we were not experienced, we had to discover the existing tools, as well as design the required models/rules. The subsequent adaptation to a new deployment with its peculiarities took us only two to three hours, mostly to adapt the ontological description of the environment. In our semantic framework, the model, the rules, and thus the system logic are kept

Figure 9.3: Photos of the Devices Deployed in STARhome
A. Karotz, smart rabbit for sound, light and avatar movements in the bathroom
B. IPTV with dedicated set-top box in the living room
C. iPad with dedicated app in the kitchen

unchanged. We conclude that the flexibility of the platform has been greatly improved due to our reliance on semantic descriptions enabling the separation of application logic from underlying models.

9.3 Top-Down Approach: Nursing Home in Singapore

9.3.1 Context of the Deployment

Within the AMUPADH project, and in order to validate our framework in real conditions, we also worked in close collaboration with Dr. Philip Yap, senior consultant geriatrician, director of the geriatric centre and clinician leader of the Memory and Dementia Care Service in Khoo Teck Puat Hospital (KTPH), Singapore. In partnership with Dr. Yap, we could initiate discussions with the staff of Peacehaven, a nursing home for people with dementia in Singapore, in order to gather stakeholders' requirements and opinions. These discussions led to the staff of Peacehaven accepting to be our pilot site.

The deployment in Peacehaven adopts a top-down approach where user needs are analysed extensively and formalised ahead of the technical requirements. This is the observation stage. User needs are translated in assistive services whose technical feasibility is then analysed, leading to an eventual technical design of the solution to be deployed. E.g., in order to guide a resident who is wandering during the night (which is a common problem for people with dementia), motion sensors can be installed on the bedroom ceiling to detect abnormal movements, pressure sensors can be placed under the mattress to detect a prolonged absence from bed, and speakers can be used to interact with the resident. During the observation period, I visited Peacehaven regularly to understand the user needs and translate them into technical requirements. This was done in an iterative way with constant feedback from the caregivers involved. During the observation stage, an initial prototype of the framework was developed. We integrated the prototype for a proof of concept demonstration to the nursing home staff and management in May 2011. Good feedback was received about the features and apparent performance. Shortly after this, an ethics approval was submitted for a real-world deployment with genuine residents. In August 2011, we received the approval from the Institutional Review Board of NUS under the number 11-222, and deployed a part of the system for technological real-world experimentation. This system trial period started in August 2011 and lasted until October, the peculiarity being that interaction was instantiated only with caregivers so as to test the system without affecting residents with eventual false alarms. The rest of the deployment was organised in three phases, alternating between deployment and analysis/improvements. The timeline of our work in Peacehaven is detailed in Table 9.1. Beside the analysis of the user needs, I was in charge in this experimentation of the design and implementation of the reasoning engine handling the context-aware service provision.

Table 9.1: Timeline for the Development and Deployment of our Solution in the Nursing Home (14 Months Trial)

Timeline	Description	Activities
Mar. 2010 – Mar. 2011	Observations, discussions and prototyping	Pre-deployment
Apr. – June 2011	Prototyping and demo	
Jul 2011	Application for ethics approval	
Aug. 2011	Ethics approval obtained	
Aug. – Oct. 2011	Initial trial setup and field testing of system	
Oct. – Jan 2012	First phase (1 room)	Deployment + Ground truth + Data analysis
Jan. – Feb. 2012	Analysis, features update and performance tuning	
Feb. – May 2012	Second phase (1 room)	
May – June 2012	Analysis and questionnaire survey	
June – Nov. 2012	Third phase (3 rooms)	
Nov. – Dec. 2012	Analysis and questionnaire survey	

9.3.2 Description of the Use-Case

Peacehaven is a three-storey nursing home depending from Singapore's Salvation Army. It hosts around 400 residents with dementia ranging from stage 4 to 5 according to the GDS (see Appendix B). Residents living on the second floor usually have a dementia of stage 5 (moderate), while residents on the third floor have a mild dementia evaluated at stage 4. Each floor is managed by eight professional caregivers who assist residents day and night, although residents from the second floor require more assistance and help. We decided to conduct our study on residents with moderate dementia, on the second floor, where caregivers have a greater need for a solution to reduce their workload. The deployment of our framework in the nursing home assists the residents with reminders to increase their independence and autonomy, as well as the caregivers by raising targeted notifications when an abnormal situation is detected and cannot be solved independently by the resident.

Results of the Observation Stage

During the observation stage, we conducted weekly visits to the nursing home and participated in focused group discussions with caregivers. We also followed the daily common schedule of the residents, accompanying two or three of them, or participating in group activities organized by the caregivers for 10 to 15 residents. From our observations and discussions, we could understand better the living conditions in the nursing home and identify the different problems that residents and caregivers are facing. We noticed that although residents are free to move around in the common areas, they spend most of their "unsupervised" time in their bedroom and visit the attached bathroom alone frequently. Hence, we decided to focus our action around these two environments.

Apart from pure assistance in performing ADLs, most of the caregiving work consists in a few tasks, namely encouraging residents to initiate some activities by showing them the first steps, serving medication, or reminding them to drink water. It is however challenging for them to follow and assist all the residents' activities. This is especially true during the night when there are fewer caregivers on duty, and when residents are prone to more critical situations. Therefore, caregivers have considered that notifications could be helpful to remain informed of how residents are performing their ADLs, and to provide support as and when appropriate. In order to avoid increasing the attention needed from caregivers, we decided that the system should stay silent in the background when no issue is detected. Moreover, in order to help residents overcome their problems, caregivers have emphasized that reminders should first be sent to the residents to encourage them to think and retain some level of independence. Caregivers should interfere only when the residents lose their way and are not able

to solve their problems independently.

Services Deployed

Residents face a variety of problems due to their dementia. Some may shower several times because they forget the activities that they have already done. Others may remain in the shower very long and let the water run due to an initiation problem, i.e. they do not remember how to start an activity. A lot of wandering is observed where residents walk around aimlessly, which is more dangerous at night due to the ambient darkness. Sometimes, they might use other residents' belongings, e.g. sleep on someone else's bed or wear someone else's clothes. In Peacehaven, the services deployed have been designed in collaboration with the caregivers according to the specific needs of the residents who agreed to participate in the trial. These services are monitoring deviances (i.e. problematic behaviours) that are the most likely to lead to a fall. On one hand, there are bathroom activities where the space is narrow and ground wet, with notifications being raised when a resident has been showering for an unusual time or when he forgets to turn off the tap of the basin. And on the other hand, we raise a notification when a resident is detected to be wandering during the night.

Profile of the Participating Residents

During phases 1 and 2 of the deployment, two residents living in the same room (room 9) and two caregivers were first involved in the study. Later on in phase 3, six other residents from two other rooms (three in room 8 and three in room 11) have granted their approval to participate in the trial. The eight residents are women living on the second floor of the nursing home and aged between 78 and 92 years old. Two of them need minimal assistance, e.g. to walk or to lie on the bed, while the six others need moderate assistance, i.e. they require help to take their shower or to eat. Table 9.2 provides an overview of the different residents' profile.

Table 9.2: Profile of the Residents Involved in the Peacehave Trial

Resident	Age	Level of assistance requied	Room number
Resident 1	90	minimal*	8
Resident 2	92	moderate**	8
Resident 3	85	moderate	8
Resident 4	79	minimal	9
Resident 5	87	moderate	9
Resident 6	92	moderate	11
Resident 7	82	moderate	11
Resident 8	78	moderate	11

* Minimal assistance: consists only in elementary activities such as walking, lying, etc.
** Moderate assistance: includes more critical activities such as eating, toileting, etc.

9.3.3 System Description

I provide in Figure 9.4 a partial floor plan of the nursing home in which we deployed our framework. It includes the room number 9 with the beds of the residents 4 and 5, and the attached bathroom; as well as the common room (not to scale) where all residents can meet, watch TV, and where the caregivers have their central desk. On the plan, I also represent the various sensors and devices deployed, and the central computer that hosts the processing of the signals and the inference of the context. For this deployment, UbiSMART is running on a tiny (115 x 115 x 35 mm, 505g) fanless debian machine, mounted with a 500MB RAM/500Hz CPU, a 4GB Compact Flash drive, and having a power consumption of only 5W.

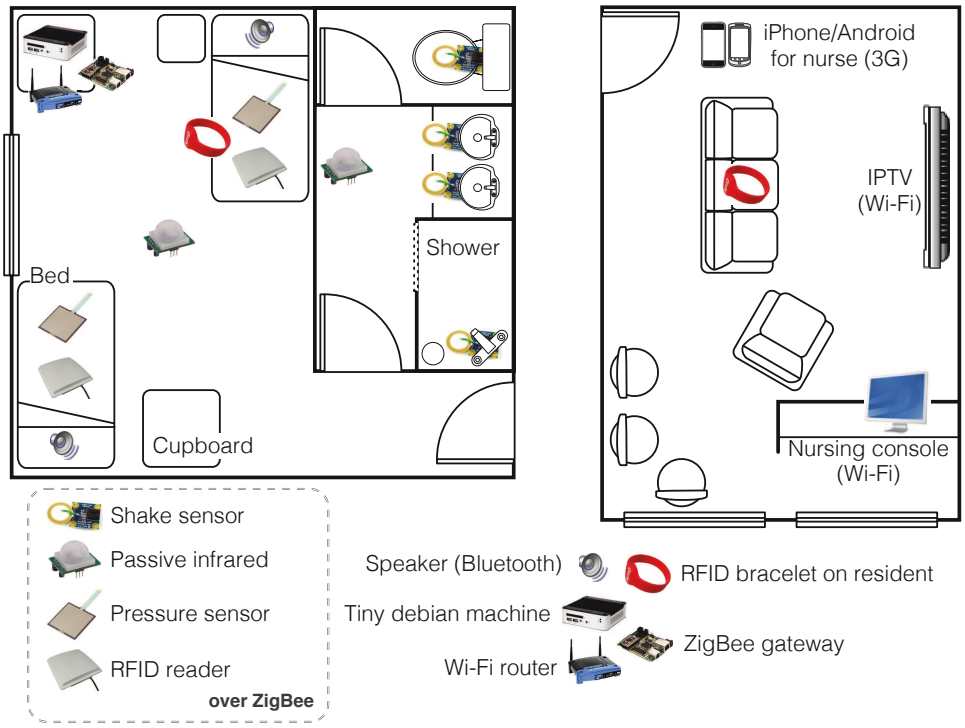


Figure 9.4: Partial Floor Plan of the Deployment in Peacehaven

Sensors are using the ZigBee communication protocol on a wireless sensor network based on Crossbow's IRIS mote platform. A Crossbow node is connected via serial port to the debian machine, serving as gateway. The communication with other devices in the environment uses bluetooth for the speakers, XMPP over WiFi for the IPTV and the nursing console (a touchscreen Windows 7 machine), and XMPP over 3G for the caregivers' smartphones (Samsung Galaxy S2 with Android 2.3 and Apple iPhone 4 with iOS 5).

Figure 9.5 is a photomontage of some of the sensors deployed in Peacehaven. Motion sensors (passive infra-red, B in Figure 9.5) are positioned on the ceiling of both the bedroom and the bathroom to detect the presence of people and measure the amount of activity by estimating the motion in the rooms. The bedroom also has pressure mats (force sensing resistors, A in Figure 9.5) installed under each mattress and RFID readers attached to the bedsides (D in Figure 9.5), allowing respectively the detection of residents and their identification. The RFID readers were paired with RFID bracelets worn by the residents around their wrist to enable the detection of a resident using the wrong bed or also to infer by deductive rules who is inside the toilet so as to adapt the services. RFID readers are off by default and are turned on only for a short period of time when the corresponding pressure mat changes state. The activities in the bathroom are monitored using shake sensors (accelerometers, C in Figure 9.5) placed on the different pipes to detect the usage of the taps or the shower. A shake sensor is also embedded in the soap dispenser of the shower to determine whether residents are using soap, as this allows the detection of initiation problems.

Similarly, I provide in Figure 9.6 a photomontage of the interaction devices deployed in Peacehaven and used for the provision of the assistive services described above. Bluetooth speakers were thought of as an efficient and easy to deploy modality of interaction for the residents. Being small and cost effective, they can be deployed in many places, making the interaction more pervasive. They do not

Figure 9.5: Photos of the Sensors Deployed in the Nursing Home
A. Bed pressure mat under the bed
B. Motion sensors on the ceiling of the bedroom and bathroom
C. Shake sensors on the bathroom pipes
D. RFID reader on the side of the bed

require learning from the residents who simply need to be informed about the presence of such devices. For instance, the residents were already used to speakers being used for management announcements in the nursing home. Finally, speakers have a greater reach than screen-based modalities as they do not require the resident to be close or able to see properly. We deployed such speakers in both the bedroom and the bathroom (respectively B and A in Figure 9.6). The communication with caregivers is ensured through dedicated apps on smartphones which can be carried around (C in Figure 9.6) or through a central nursing console shared by all caregivers and that centralises all notifications (D in Figure 9.6). When a notification is received, the phones vibrate and beep until the notification is acknowledged. On iPhone, Apple Push Notifications service is also implemented to ensure that all notifications go through independently of the app's sleep/connection status. The nursing console remains connected 24/7 and play an alert sound when a notification is received. Alerts automatically disappear when they are detected as solved. Moreover, the console also displays continuously the context status related to deployed services. E.g. if a resident is taking his shower, this information would be silently available to the caregivers.

In order to evaluate the performance of our system, and as we are committed not to use video recording to preserve the privacy of the residents and caregivers, we have chosen to rely on log-sheets filled by the caregivers. During the trial, we collected system logs related to numerous aspects of the framework and extracted information concerning sensors states, or residents' context, among others. Such information was then cross-referenced with the content of the caregivers' log-sheets. As explained in the comparison of reasoning approaches (see section 3.2.2), we believe that it is close to impossible to gather genuine ground truth in a real AAL deployment. Therefore, the caregivers' log-sheets are considered as an approximate ground-truth for our validation. Caregivers have been asked to fill the log-sheets with hourly information about the location, the abnormal behaviours and other possible remarks concerning the residents. I provide below a sample of the log-sheet used for our ground-truth collection.

1. Where is the resident right now?

    (a) bedroom

Figure 9.6: Photos of the Devices Deployed in the Nursing Home
A. Bluetooth speaker in the bathroom
B. Bluetooth speaker in the bedroom
C. iPhone with dedicated app for the caregivers
D. Central nursing console for the caregivers

(b) bathroom

(c) dining area

(d) common area

(e) other *(specify)*: ...

2. Did the resident shower for too long?

   (a) yes

   (b) no

3. Did the resident forget to turn off the tap?

   (a) yes

   (b) no

4. Did the resident forget to flush the toilet?

   (a) yes

   (b) no

5. Did the resident wander around aimlessly?

   (a) yes

   (b) no

6. Did the resident ask for something?

   (a) yes *(specify)*: ...

   (b) no

9.3.4 Results

The deployment in Peacehaven was useful as a technical validation of the system in a real environment with harsh networking conditions and unexpected issues such as residents pulling off sensors. It was also the first long-term deployment, which tested other issues such as sensors batteries. The technical part of this validation has been presented and commented in section 8.2.5. In this part, we are more focused on the human side of the validation. We are notably interested in analysing what can be detected by the framework in real conditions, and what kind of services can consequently be provided to the different stakeholders.

Firstly, we wish to estimate the accuracy of the context detected by the framework. Detection and recognition errors can indeed be incurred at the sensing, low-level event processing and higher-level context recognition steps. The accuracy is *estimated only*, by comparing logs from the system with the log-sheets filled by the caregivers. Results are given based on the analysis of the logs for the two bathroom services described previously, during an uptime period of nine days. We consider *atomic* events first—e.g. the use of taps and shower—that happened 34 times a day in average with a recognition accuracy of 71% (Figure 9.7, left). *Complex* events—which correspond to deviances and the provision of services—happened 7 times a day in average, with an accuracy of service delivery of 70%. This accuracy characterises the ratio between the number of times a service was delivered over the number of times it was needed. Since complex events are derived from atomic events, we conclude that little error is introduced by the higher-level context recognition, most of it being introduced either by sensing issues or low-level processing of the sensors signal. These results were obtained in February 2012, during the second phase of our deployment. After improving the system, notably from the hardware point of view, we estimated again the accuracy of the system in the same room, considering the same services and over a second nine-days period. The improved recognition accuracy obtained was of 83% for the atomic events, up from 71% (Figure 9.7, right). As for the accuracy of service delivery, it followed up to 82%, confirming the low error introduced by the context recognition algorithm. We conclude here that the context recognition algorithm performs well, provided that the sensor events in input are non-noisy. We will see in the results of the next deployment that noisy data is however a big issue for knowledge driven reasoning.

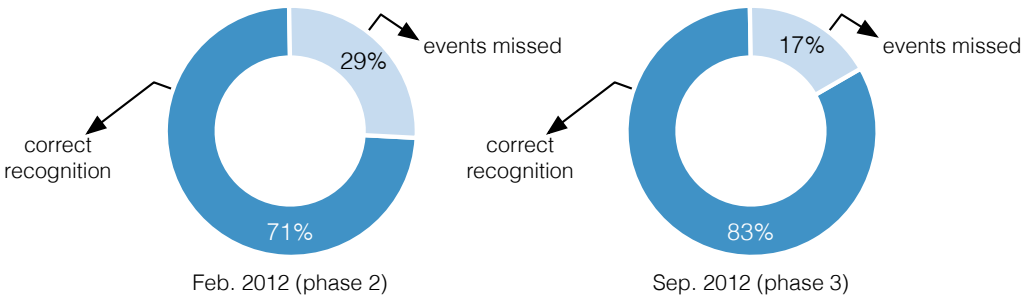


Figure 9.7: Recognition Rate of Atomic Events in Phase 2 & 3 of the Deployment

Beside the specific deviances that we focused on for the services provided in the nursing home, we have also observed the possibility to raise early alerts concerning the deterioration of the condition of a resident. As shown in Figure 9.8, we observed for one of the residents an abnormal increase in the number of reminders delivered in early March 2012. This resident who would usually receive about 2 reminders a day, varying between 0 and 6, up to the 3rd of March, suddenly was sent between 8 and 12 reminders each day for ten consecutive days. The resident eventually was hospitalised after a degradation of her situation. We conclude that the observation of the evolution of a resident's lifestyle overtime enables the early detection of health related issue, and may be helpful in preventing the degradation a one's condition.

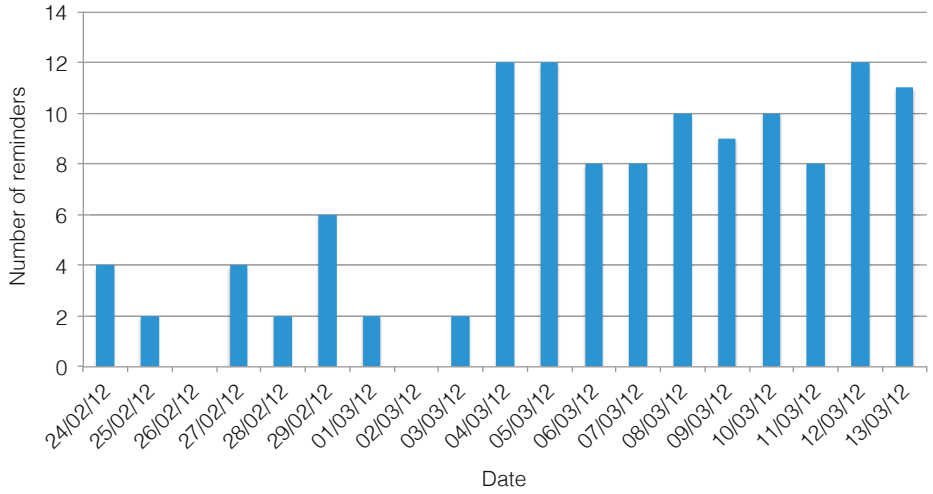


Figure 9.8: Early Detection of the Deterioration of a Resident's Condition

From a more qualitative point of view, the feedback received from the nursing home staff has been very encouraging. The deployment was promising in terms of demonstrated features and capabilities. In view of the expected benefits, the head of the nursing home has remarked that the staff would like to have the full system deployed in every room. This was encouraging since there has been a perceptible change in attitude over the months during which the deployment had taken place. The caregivers have become more adept in the use of smartphones and have appreciated the value of the underlying sensor-based services. Although the staff participated well in filling the manual log-sheets, which was crucial as it was the only form of ground-truth available, they admitted having difficulties to cope systematically with the extra work it represented. We discussed simplifying the logging process by providing a more automated logging media through tablets embedded in the environment and bringing logging down to a few clicks on a touchscreen. Doctors carry a positive attitude towards the deployment and feel that it would go a long way to improve the residents' safety and would add to their well being and comfort.

9.4 Bottom-Up Approach: Individual Private Homes in France

9.4.1 Context of the Deployment

Complementarily to the validation of the top-down approach realized in the nursing home in Singapore, we extend our deployment effort towards individual homes in order to validate the bottom-up approach as well. Therefore, my research contributions were integrated to the Quality of Life (QoL) Chair. The QoL Chair is supported by the Fondation Télécom of the Institut Mines Télécom in France, and by La Mutuelle Générale which figures among the major healthcare insurance companies in France. The project involves research teams from IPAL, Handicom laboratory in Institut Mines Télécom Evry and Age, Imagerie, Modélisation (AGIM) laboratory in Grenoble University Hospital. Finally, it is possible thanks to our close collaboration with Handco, an SME based in Paris region who offers dedicated solutions to people with disabilities or in a dependent situation.

The main goal of the QoL Chair is to maintain the quality of life of ageing people in their own home through the incorporation of ICT. The challenges are to:

– Investigate the methods for the analysis of the use and acceptance of ICT-based assistive homes, using qualitative (on-site observations) and quantitative (analysis of system data) evaluations.

– Deploy a framework in pilot sites to estimate the impact of ICT-based assistive homes on the independence towards ADLs and the quality of life of elders.

– Design and implement algorithms for the comprehension of situational data and the formalisation of contextual knowledge based on a coarse grain hardware deployment.

– Find techniques for the continuous assessment of people's health condition in their own home, as well as for their stimulation towards a healthier lifestyle and a greater social inclusion.

In order to provide a proof of concept that valuable knowledge can be derived—and thus valuable services can be provided—from a coarse sensor deployment in a private home, we have deployed our system in three houses in France. The three residents, women aged over 75 years old, are subject to mild dementia, live alone at home, and receive support from caregivers or family around one hour daily-to-biweekly. Each family was contacted through its insurance company, following a survey about the dependence of elderly living alone, and accepted to take part in the trial. The data gathered was naturally anonymised. This proof of concept is an opportunity to validate the activity inference mechanisms (section 5.4.2) and reasoning architecture (chapter 7) described in this doctoral work. My contribution to this experimentation mainly concerned the adaptation of the reasoning engine to the specific challenges emerging from the coarse situational data available. The partial datasets considered for the validation are summarised in Table 9.3.

Table 9.3: Partial Datasets Used from our Deployment in France

Home	Resident's age	Dataset duration
Home 1	84	39 days
Home 2	79	15 days
Home 3	77	26 days

9.4.2 System and Data Description

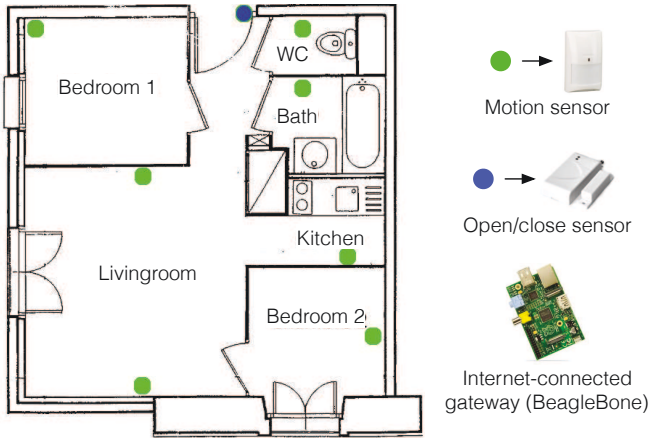


Figure 9.9: QoL Map

The system deployed in France and represented in Figure 9.9 gathers in each house a live sensor stream from a WSN based on the radio version of the industrially standardized X10 communication protocol. Sensor events are received by a X10 module used as gateway and mounted on a BeagleBone

credit-card-sized Linux (Debian) machine that connects to the Internet to post events to a MySQL database. This MySQL database is shared for the three houses and the data of each house is stored in an independent table. Our UbiSMART framework fetches its input from the MySQL database to process the live datastream available. It can alternatively replay a given time window from the database. Each house is processed in a separate instance of the framework, which was setup as a single house processor for the time being. UbiSMART has been tested on a Linux server (Ubuntu 12.10 32-bit) powered by an Intel Core 2 Duo CPU E6550 at 2 x 2.33GHz and 3.8GB RAM. In the three houses, motion sensors were installed on the ceiling of each room, sometimes with two sensors in one room to avoid uncovered areas (blind spots). A reed switch indicating the opening or closing of the entrance door was also deployed. The motion sensors were calibrated to send an "on" event when a motion is detected and an "off" event after one minute of inactivity. After each event, the sensor observes a ten-seconds period of silence. The X10 protocol being energy efficient, we estimate the autonomy of the sensors to several months (three to six months depending on the amount of activity in the house). The dataset used for our validation is coming from the first house (Home 1 in Table 9.3) since this is the "cleanest" dataset, i.e. the data seems to match the most with the specification described above (10s silence and 60s to off signal). A short extract of the dataset is given as example in Table 9.4. The corresponding sensor description table is given in Table 9.5.

Table 9.4: Extract of the Dataset Used for the Validation

Timestamp	Sensor id	State
2012-06-06 07:02:45	A7	on
2012-06-06 07:03:29	A7	on
2012-06-06 07:03:55	A3	on
2012-06-06 07:04:29	A7	off
2012-06-06 07:04:22	A3	on
2012-06-06 07:05:22	A3	off
2012-06-06 07:09:39	A3	on
2012-06-06 07:09:37	A3	on
2012-06-06 07:10:14	A7	on
2012-06-06 07:10:28	A2	on

Table 9.5: Sensor Location for the Dataset Used for the Validation

Sensor id	Sensor type	Location
A2	motion	kitchen
A3	motion	bedroom
A4	motion	bedroom2
A6	motion	bathroom
A7	motion	livingroom
B1	door open/close	main door

9.4.3 Results

The goal of this experimentation was to confirm the feasibility of reasoning in an AAL solution based on a stripped-down hardware deployment. This means that we must demonstrate that valuable knowledge can be extracted about an elderly person's lifestyle from coarse sensor data. Moreover, we must verify

that the scalability of the incurred processing remains within an economical level, where an economy of scale is possible with one server providing enough processing power to cater for hundreds of houses.

Firstly, we observe that the data itself with no further processing already provides significant information about a person's day if it is well visualized. For example, the Figure 9.10 is a "donut" representation of the resident's lifestyle. Each circle corresponds to one day of sensing data with a colour-coded representation of the person's location. This representation is given relatively to the number of events received from the sensors in each room, thus it incorporates a notion of activity in the room and not only presence. In other words, if the resident spends half his time in the bedroom and the other half in the livingroom, but is twice as active in the livingroom as compared to the bedroom, then the livingroom will occupy two thirds of the circumference of the circle. The vertical rectangle beside each circle indicates the total number of events for that day, i.e. it shows whether the resident was at home and his general amount of activity for the day. From this visualization, we can immediately

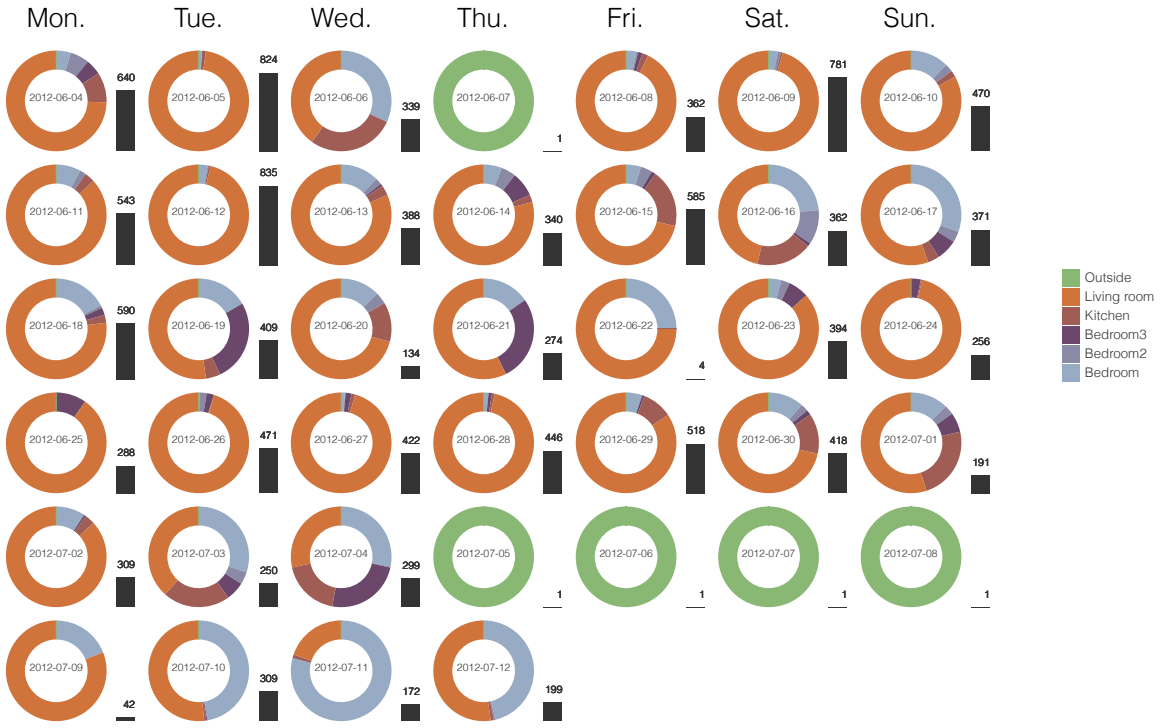


Figure 9.10: Calendar Dashboard: Location Data Visualisation

identify the days when the resident was out the whole day, whether for a hospitalisation or for leisure. These are the days where the rectangle is a line and/or the circle fully coloured in green (e.g. 5^{th} to 8^{th} July). One can also estimate efficiently the amount of activity and most common location for each day, which provides an average of the lifestyle of the resident. For instance, "bluish" days with more activity in the bedroom and less activity in total might indicate that the resident feels sick or weak (e.g. 10^{th} to 12^{th} July), whereas "orangey" days spent moving a lot and mostly in the livingroom most probably correspond to a resident in good spirit (e.g. 24^{th} to 29^{th} June). Most importantly, it helps to visualise the evolution of this lifestyle over the days. We wish to study more in depth the effectiveness of such visualisations for real-time situation awareness as well as continuous and ambient health assessment purpose.

Concerning the actual context comprehension, I provide in Figures 9.11 and 9.12 the results of

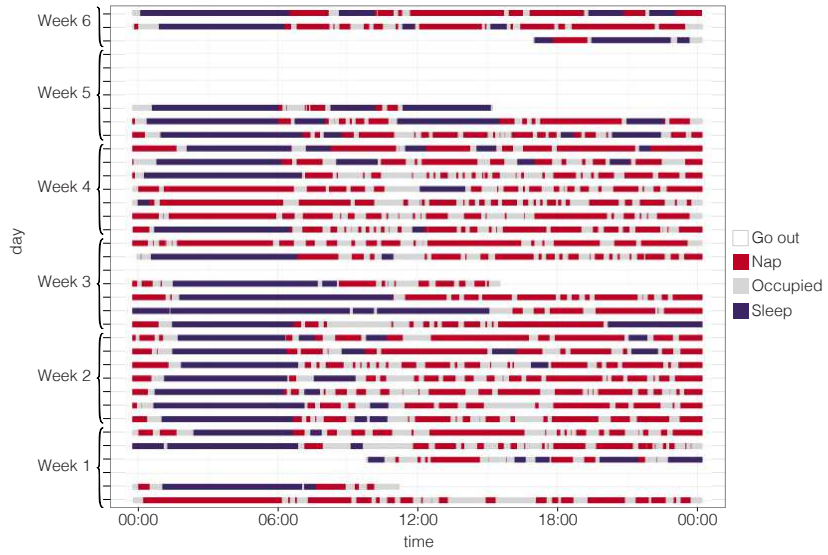


Figure 9.11: Chronological Visualisation of the Result of the Activity Inference

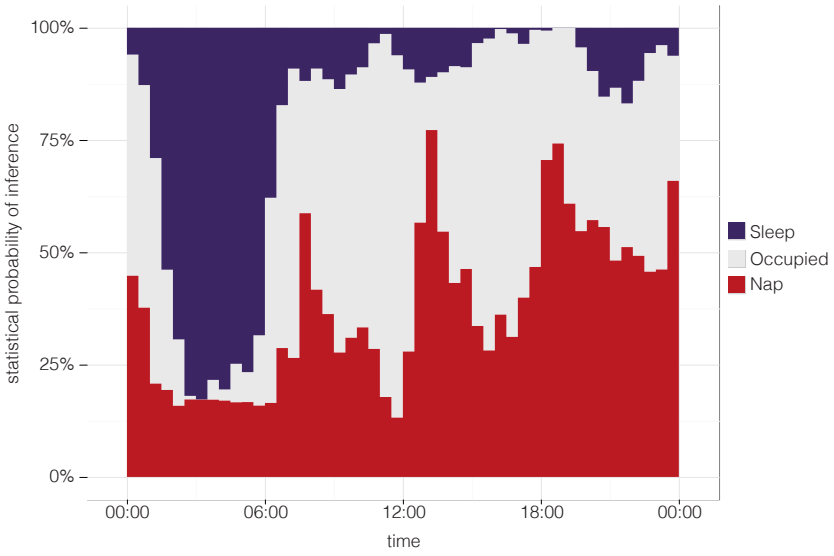


Figure 9.12: Statistical Visualisation of the Result of the Activity Inference

my rule-based activity inference on the first house's dataset in a chronological and statistical manner. The figures represent the inferred activity (colour legend) depending on the time of the day (on the horizontal axis) and respectively the day or the statistical probability of inference (on the vertical axis). These figures illustrate very well the night rhythm of the resident as we detect a clear zone of "sleep" activity between 0:30 and 6:00, despite the fact that some nights seem to be spent in the livingroom and detected as "nap" during the exact same time period. We also see a tendency to start the night by a series of short naps in the livingroom from 18:00 onwards. These naps may actually correspond to the resident watching TV as he would be present in the livingroom without motion (thus understood as having a "nap") and it also matches with the time for evening entertainment shows, news and then movies on French TV. We detect in Figure 9.12 two other major "nap" picks around 8:00 to 9:00 after

some time spent in the bathroom and kitchen (no supporting graphics for location here though) and around 13:00 to 14:00 probably during the news and the early afternoon TV series that target an elder audience at this time on several French channels. In Figure 9.11, we see an additional and unusually long "nap" period on the three first Sundays (week starts on Monday) from 13:00 to 17:00; this matches perfectly with popular Sunday's afternoon-long TV series broadcast on French channel TF1. Finally, we observe two picks of activity (large "occupied" area in Figure 9.12) around 11:00 to 12:00 when the resident might be preparing his lunch and after 15:00. While these results can still be improved, they already provide some elements that can be matched to the resident's profile; which offers an avenue of improvement by modifying the rules accordingly as explained in section 6.2.1. We also believe that such visualisations with the highlight of potentially dangerous or abnormal behaviours would be useful to remotely "keep an eye" on residents, which is a service companies or families have expressed interest for. We are also able to compute some metrics about the resident's habits and track the evolution of these metrics over longer periods of time to lift early alerts about the condition of the person, as we have done in the Peacehaven deployment with the number of reminders sent to one of the residents. One of the main limitations here is the restriction to coarse grain activities such as "occupied". This highlights the need for a finer sensing in the house, possible if using more types of sensors. A trade-off is therefore necessary between the scalability of deployment and the appropriate context granularity in order to reach an optimal marketable solution.

9.5 Lessons Learned

9.5.1 Get Out of the Lab

A lot of technical issues have emerged from deploying technological systems in real living spaces. To perform the validation, these issues had to be dealt with, which is often considered as a "waste of time" by researchers. However, this experience allowed us to learn a lot about the targeted users and stakeholders in general, and it provided us with essential and extremely valuable knowledge related to bringing value out of our research work and making an impact in society. This knowledge is mainly related to the feedback received from the stakeholders, to the acceptance of the solutions, their ease of deployment and maintenance, usage issues, etc. Even though such deployments felt like a burden at some point in time, we can only recommend to researchers in our field to get out of the lab, deploy their solutions, and include stakeholders early in the research work.

9.5.2 The Suitable Sensing Granularity

Looking back on our two approaches (top-down and bottom-up), and analysing the experience and results of the corresponding validation deployments, we feel that we are more apt today to identify the trade-offs they set forth. The main question that emerges from our validation effort is: *What is the most optimal granularity of sensing for an impactful and economical adoption of AAL technologies?*

The purely top-down approach has revealed very wide possibilities for the provision of context-aware services, both in real-time and looking at longer-term analysis of lifestyles. For example, in our deployment in the nursing home, we were able to detect an average of seven critical situations per day for two people sharing the same room (e.g. residents left the tap on, or showered for too long). These types of information are crucial for the caregiver and provides strategic information on the quality of life of end-users living in both the nursing home and their own home. In the case of longer-term services, we have seen the potential of smart homes as tools for the assessment of one's condition and the early detection of its deterioration. The top-down approach is however very complex from both hardware and software perspectives. On the hardware side, the wide range of technologies used makes it difficult to deploy and maintain the system. In term of software, the hardware complexity and specificity can only be leveraged by scenario-based algorithms for the comprehension of the context. Overall, taking

into account the poor scalability of the solution, it seems a bit far fetch to imagine systems of this complexity being deployed and maintained in large scale by third-party service providers.

On the contrary, bottom-up solutions are just right in term of deployment scalability. The reduced technological spread of the system makes it easier to deploy and cheaper to maintain, even in large scale. The skill-set required to handle the maintenance is being greatly reduced, which opens the door to economically viable deployments by third-party service providers. However, in this case the possibilities of the solution in term of services that can be provided are dramatically reduced as well. Only critical situations may be detected and not really in real-time. The reminders and notifications available with the top-down approach are impossible without adding more specific sensors to gather finer grain situational data. The long-term analysis of one's lifestyle is still possible, although it may not be as detailed or accurate.

In conclusion, I believe that the bottom-up approach is the only one that might make an impact on societies in the future. It would be good, however, to make it possible for users to add on to the basic features of the solution in order to enable more targeted services that respond to their specific needs and wishes in a personalised fashion.

Part V

Conclusion

The true function of philosophy is to educate us in the principles of reasoning and not to put an end to further reasoning by the introduction of fixed conclusions.

— George Henry Lewes, 1817–1878

10 Conclusions and Perspectives

10.1 Conclusions

This thesis studies the strategies that can be put in place for the comprehension of context in smart homes for the elderly. Two approaches to AAL are considered: a top-down approach similar to what can be found in the literature, and a novel bottom-up approach that introduces a stripped-down vision of AAL systems with coarse sensing granularity. The specific scientific contributions can be summarised as follows:

- In the first part of this thesis, I make the case of knowledge driven methods and semantic rule-based inference techniques. A functional model for the provision of context-aware services is proposed. This model complements the very exhaustive and declarative ones found in the literature by being more system oriented, and introduces several layers for the representation of the context. It is therefore more usable for the computed inference of activities based on situational data.

- Since semantic web technologies are very wide ranging, I analyse which specific languages and engines are more suitable for AAL systems. Core requirements are gathered, usually emerging from the differences between web applications which semantic web tools have been designed for, and the real-time and pervasive services that AAL systems provide. My resulting choice and recommendation is to use EYE reasoning engine with the N3 ontological language.

- I then propose a rule design based on EYE/N3 which takes into account the different aspects of context comprehension. This design relies on abstract and parameterised rules when possible, and introduces three complementary types of rules. Rules of reasoning and rules of fact contribute to balancing rationalism and empiricism in the inference such that common sense or domain knowledge can be leveraged as much as more deterministic aspects of the contextual knowledge.

- Next, I highlight the needs and propose techniques to incorporate novel aspects into the semantic rule engine. Indeed I extend the reasoning architecture and the model to integrate quality of information metrics on a statement basis, and to enable the use of data driven techniques in the inference.

- I finally propose a cognitively inspired reasoning architecture, and describe in details the mechanisms involved and the integration in the UbiSMART framework.

In addition to the scientific research contributions listed above, this work has been implemented and integrated into a context-aware service framework as described below. As part of this doctoral work, UbiSMART should be considered as a main tangible deliverable, and a tool that enables the validation of the research contributions summarized above. The framework has evolved greatly in its three years of development, from a proof of concept prototype implemented imperatively, to a

semantic framework based on Jena, and later on to an optimized semantic framework based on EYE and enabling proof of value deployments. At its heart, the reasoning mechanism changed from a fully distributed architecture to a hybrid approach where semantic inference is centralized and coupled to more heterogeneous reasoning modules, thus improving both scalability and flexibility of the reasoning in UbiSMART. We have also introduced unique and novel features in the framework, such as our tailor-made triplestore, the semantic plug & play mechanism, and the abstract event bridge. All these more technical contributions are detailed in chapter 8.

Both the research and technical contributions have been validated following a three-fold process:

- The rule design was formally verified along certain properties such as deadlock-freeness or reachability, thus certifying a correct behaviour of the framework under defined circumstances. The verification was done using formal model checking methods and is detailed in section 5.4.3.
- Then, we realized in-situ *ecological* testing of the framework in three complimentary settings, namely in a realistic research facility, in a nursing home in Singapore with the involvement of eight residents and three caregivers, and in three individual homes in France involving three families. Our deployment effort and results are described in details in chapter 9.
- Finally, we performed load tests and performance analysis on the platform to ensure its scalability in charge and confirm the economical processing power needed to analyse hundreds of home. This tests and the results are described in section 8.4.5.

I must highlight the importance of deployments in real settings, since it enables an interdisciplinary research effort by involving stakeholders early in the research. This allows a better design of the solutions, ensuring an optimal acceptance in society. It also raises our framework for AAL from a proof of concept to a proof of value by, e.g., showing the kind of services that can be provided. Despite the low number of users in our validation, we are able to provide today a framework that can be deployed in projects of larger scale. This is something we actually are working on as part of the VHP@interactive research project and the QoL Chair project. Both projects have for objective to demonstrate the economic viability of human and technological services enabling ageing in place, VHP@interactive handling chronic disease related dependencies, and the QoL Chair being focused on dementia related dependencies. In this context, our framework will also allow therapeutic education, rehabilitation and lifestyle coaching services which are essential services to bring into elders' homes in order to assess and advice them towards a healthier lifestyle.

10.2 Perspective Work

The future work at the end of this thesis contains both short-term and long-term perspectives. The short-term research aspect that I wish to study more in depth is the Fuzzy Activation Map Engine (FAME) described in section 6.2.2. I am especially interested in analysing the duality between iFAM and wFAM. This will probably be part of my early post-doctoral research effort. True to our "get out of the lab" spirit, and taking into account the vision expressed in Appendix F, I would also like to get a multi-disciplinary team together working on the design, development and deployment of a smart home in a box solution. My idea is to involve stakeholders as early as possible in the design process and gather continuous feedback throughout the project to ensure a good level of acceptance for the solution proposed.

Concerning the long-term perspectives, I believe that my doctoral work has highlighted two distinct areas in which in-depth studies would be interesting and consistent enough to propose two new doctoral thesis subjects:

- The first aspect is related to context modelling and would be to build a more parametric context model, with multiple facets of representation for the context. As explained in section 4.5, this

would empower reasoning algorithms to automatically understand situations and estimate the danger even for unknown situations. It might help to go beyond scenario-based activity recognition. Moreover it would enable the provision of services without relying on pre-programmed context-service bindings.

– As explained in section 6.2, I believe that the combination of data driven techniques and knowledge driven techniques opens interesting paths for future research. The second thesis subject I would like to propose concerns this combination. It would address the design, implementation and validation of mechanisms for combining both techniques. It should introduce new paradigms for the real-time interchange of data and/or results between semantic and statistical reasoning techniques, as well as for the arbitration of heterogeneous reasoning results.

Part VI

Appendix

Home is a place you grow up wanting to leave, and grow old wanting to get back to.

— John Ed Pearce, 1917–2006

A

Overview of AAL Research Bottlenecks

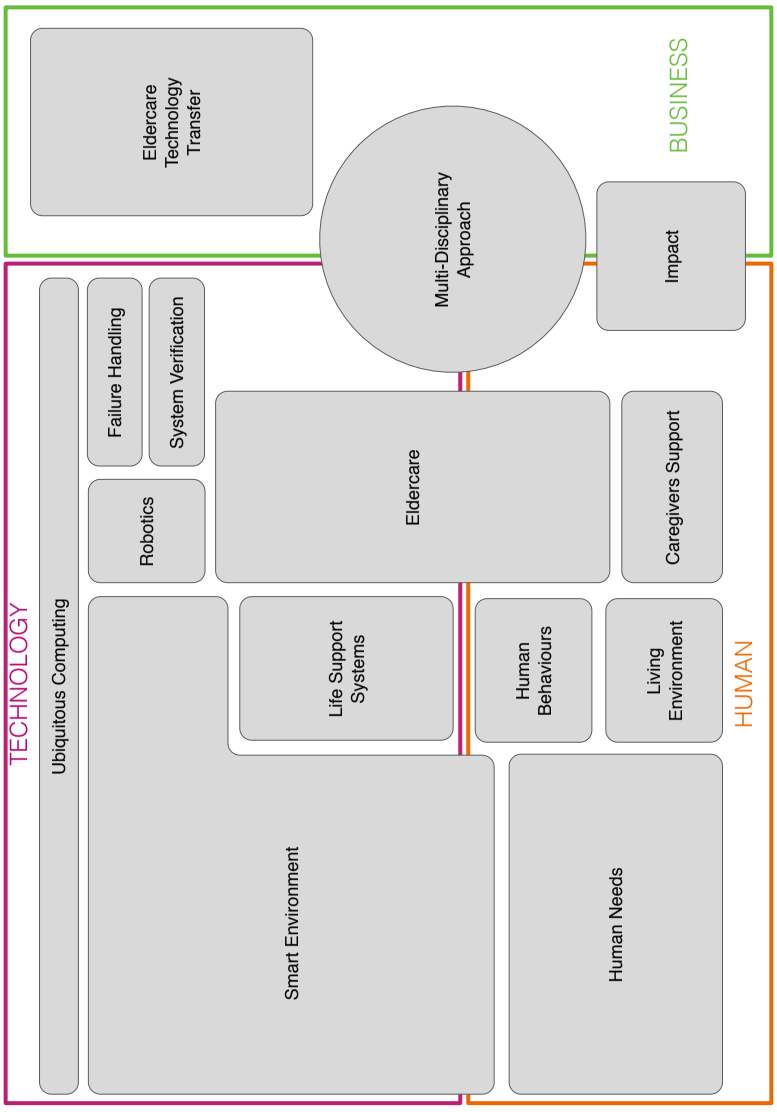


Figure A.1: Coarse-Grain Overview of AAL Research Activities

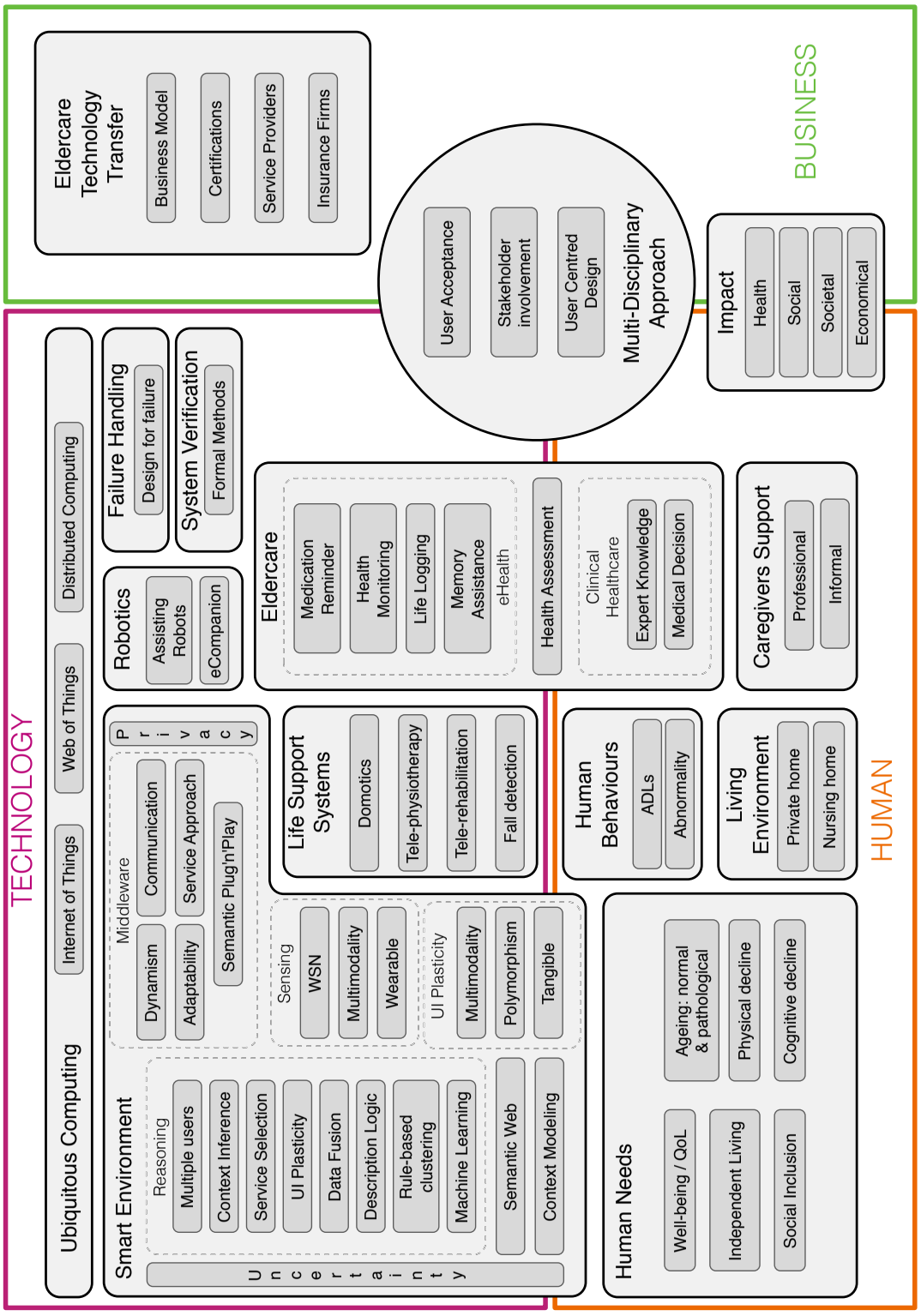


Figure A.2: Fine-Grain Overview of AAL Research Bottlenecks

The art of medicine consists of amusing the patient while nature cures the disease.

— Voltaire, 1694–1778

B

Global Deterioration Scale (GDS)

Some healthcare professionals use the GDS, also called the **Reisberg Scale**, to measure the progression of Alzheimer's disease. This scale divides Alzheimer's disease into seven stages of ability. Table B.1 and Table B.2 were extracted from [14] and provide a description for each of the seven stages.

Table B.1: The Global Deterioration Scale for Assessment of Primary Degenerative Dementia (1)

Level		Clinical characteristics
1	No cognitive decline	No subjective complaints of memory deficit. No memory deficit evident on clinical interview.
2	Very mild cognitive decline (age-associated memory impairment)	Subjective complaints of memory deficit, most frequently in the following areas: (a) forgetting where one has placed familiar objects; (b) forgetting names one formerly knew well. No objective evidence of memory deficit on clinical interview.
3	Mild cognitive decline (mild cognitive impairment)	Earliest clear-cut deficits. Manifestations in more than one of the following areas: (a) patient may have got lost when traveling to an unfamiliar location; (b) co-workers become aware of patients relatively poor performance; (c) word- and name-finding deficits become evident to intimates; (d) patient may read a passage of a book and retain relatively little material; (e) patient may demonstrate decreased facility in remembering names upon introduction to new people; (f) patient may have lost or misplaced an object of value; (g) concentration deficit may be evident on clinical testing. Objective evidence of memory deficit obtained only with an intensive interview. Decreased performance in searching for employment and social settings. Denial begins to become manifest in patient. Mild to moderate anxiety accompanies symptoms.
4	Moderate cognitive decline (mild dementia)	Clear-cut deficit on careful clinical interview. Deficit manifest in the following areas: (a) decreased knowledge of current and recent events; (b) may exhibit some deficit in memory of ones personal history; (c) concentration deficit elicited on serial subtractions; (d) decreased ability to travel, handle finances. Frequently no deficit in following areas: (a) orientation to time and place; (b) recognition of familiar persons and faces; (c) ability to travel to familiar locations. Inability to perform complex tasks. Denial is the dominant defense mechanism. Flattening of affect and withdrawal from challenging situations frequently occur.
		Continued in Table B.2

Table B.2: The Global Deterioration Scale for Assessment of Primary Degenerative Dementia (2)

Continued from Table B.1		
Level		**Clinical characteristics**
5	Moderately severe cognitive decline (moderate dementia)	Patient may no longer survive without assistance. Patient is unable to recall a major relevant aspect of their current lives during an interview, e.g. an address or telephone number of many years, the names of close family members (such as grandchildren), the name of the high school or college from which they graduated. Frequently, some disorientation to time (date, day of week, season, etc.) or to place. An educated person may have difficulty counting back from 40 by 4s or from 20 by 2s. Persons at this stage retain knowledge of many major facts regarding themselves and others. They invariably know their own names and generally know their spouses and childrens names. They require no assistance with toileting and eating, but may have some difficulty choosing the proper clothing to wear.
6	Severe cognitive decline (moderately severe dementia)	May occasionally forget the name of the spouse upon whom they are entirely dependent for survival. Will be largely unaware of all recent events and experiences in their lives. Retain some knowledge of their past lives but this is very sketchy. Generally unaware of their surroundings, the year, the season, etc. May have difficulty counting from 10 both backward and, sometimes, forward. Will require some assistance with activities of daily living, e.g. may become incontinent, will require travel assistance but occasionally will be able to travel to familiar locations. Diurnal rhythm frequently disturbed. Almost always recall their own name. Frequently, continue to be able to distinguish familiar from unfamiliar persons in their environment. Personality and emotional changes occur. These are quite variable and include: (a) delusional behavior, e.g. patients may accuse their spouse of being an impostor, may talk of imaginary figures in the environment, or to their own reflection in the mirror; (b) obsessive symptoms, e.g. person may continually repeat simple cleaning activities; (c) anxiety symptoms, agitation and even previously nonexistent violent behavior may occur; (d) cognitive abulia, i.e. loss of willpower because the individual cannot carry the thought long enough to determine a purposeful course of action.
7	Very severe cognitive decline (severe dementia)	All verbal abilities are lost over the course of this stage. Frequently there is no speech at all only unintelligible utterances and rare emergence of seemingly forgotten words and phrases. Incontinent of urine, requires assistance toileting and feeding. Basic psychomotor skills, e.g. ability to walk, are lost with the progression of this stage. The brain appears to no longer be able to tell the body what to do. Generalized rigidity and developmental neurologic reflexes are frequently present.

In short, we can summarize the stages as follow:

- Stage 1: no cognitive decline, experiences no problems in daily living.
- Stage 2: very mild cognitive decline, forgets names and locations of objects, may have trouble finding words.
- Stage 3: mild cognitive decline, has difficulty travelling to new locations, has difficulty handling problems at work.
- Stage 4: moderate cognitive decline, has difficulty with complex tasks (finances, shopping, planning dinner for guests).
- Stage 5: moderately severe cognitive decline, needs help to choose clothing, needs prompting to bathe.
- Stage 6: severe cognitive decline, loss of awareness of recent events and experiences, requires assistance bathing; may have a fear of bathing, has decreased ability to use the toilet or is incontinent.

– Stage 7: very severe cognitive decline, vocabulary becomes limited, eventually declining to single words, loses ability to walk and sit, requires help with eating.

A man's grammar, like Caesar's wife, should not only be pure, but above suspicion of impurity.

— Edgar Allan Poe, 1809–1849

C Grammars for the Semantic Web

C.1 Jena Rule Grammar

Following is a short rule grammar tutorial handy to understand how Jena rules must be formatted. This annex is inspired by the Jena sourceforge website [168]. A rule for the rule-based reasoner is defined by a Java Rule object with a list of body terms (premises), a list of head terms (conclusions) and an optional name and optional direction. Each term or `ClauseEntry` is either a triple pattern, an extended triple pattern or a call to a built-in primitive. A rule-set is simply a `List` of `Rules`. For convenience a rather simple parser is included with `Rule` which allows rules to be specified in reasonably compact form in text source files. However, it would be perfectly possible to define alternative parsers which handle rules encoded using, say, XML or RDF and generate `Rule` objects as output. It would also be possible to build a real parser for the current text file grammar which offered better error recovery and diagnostics. An informal description of the simplified text rule grammar is provided in Source C.1, where "," separators are optional.

Source C.1: Jena Simplified Grammar

```
Rule := bare-rule or [ bare-rule ] or [ ruleName : bare-rule ]

bare-rule := term, ... term -> hterm, ... hterm // forward rule
  or   bhterm <- term, ... term // backward rule

hterm :=   term or [ bare-rule ]

term := (node, node, node) // triple pattern
  or   (node, node, functor)  // extended triple pattern
  or   builtin(node, ... node) // invoke procedural primitive

bhterm := (node, node, node) // triple pattern

functor := functorName(node, ... node) // structured literal

node := uri-ref // e.g. http://foo.com/eg
  or   prefix:localname // e.g. rdf:type
  or   <uri-ref> // e.g. <myscheme:myuri>
  or   ?varname // variable
  or   'a literal' // a plain string literal
  or   'lex'^^typeURI // a typed literal, xsd:* type names supported
  or   number // e.g. 42 or 25.5
```

The difference between the forward and backward rule syntax is only relevant for the hybrid execution strategy — see website. The `functor` in an extended triple pattern is used to create and access structured literal values. The `functorName` can be any simple identifier and is not related to the execution of built-in procedural primitives, it is just a data-structure. It is useful when a single

semantic structure is defined across multiple triples and allows a rule to collect those triples together in one place. To keep rules readable qname syntax is supported for URI refs. The set of known prefixes is those registered with the `PrintUtil` object. This initially knows about rdf, rdfs, owl, daml, xsd and a test namespace eg, but more mappings can be registered in java code. In addition it is possible to define additional prefix mappings in the rule file, see below. Source C.2 provides some example rules which illustrate most of these constructs.

Source C.2: Jena Rule Examples

```
[allID: (?C rdf:type owl:Restriction), (?C owl:onProperty ?P),
  (?C owl:allValuesFrom ?D) -> (?C owl:equivalentClass all(?P, ?D))]

[all2: (?C rdfs:subClassOf all(?P, ?D)) -> print('Rule for ', ?C)
  [all1b: (?Y rdf:type ?D) <- (?X ?P ?Y), (?X rdf:type ?C) ] ]

[max1: (?A rdf:type max(?P, 1)), (?A ?P ?B), (?A ?P ?C)
  -> (?B owl:sameAs ?C) ]
```

`allID` illustrates the functor use for collecting the components of an OWL restriction into a single data-structure which can then fire further rules. `all2` illustrates a forward rule which creates a new backward rule and also calls the print procedural primitive. `max1` illustrates use of numeric literals.

Rule files may be loaded and parsed using one of the three Java commands in Source C.3.

Source C.3: Jena Rules Loading

```
List rules = Rule.rulesFromURL("file:myfile.rules");

BufferedReader br = /* open reader */ ;
List rules = Rule.parseRules( Rule.rulesParserFromReader(br) );

String ruleSrc = /* list of rules in line */ ;
List rules = Rule.parseRules( rulesSrc );
```

In the first two cases (reading from a URL or a `BufferedReader`) the rule file is pre-processed by a simple processor which strips comments and supports some additional macro commands:

- `# ...` is a comment line.
- `// ...` is a comment line too.
- `@prefix pre: <http://domain/url#>.` defines a prefix "pre" which can be used in the rules. The prefix is local to the rule file.
- `@include <urlToRuleFile>.` includes the rules defined in the given file in this file. The included rules will appear before the user defined rules, irrespective of where in the file the `@include` directive appears. A set of special cases is supported to allow a rule file to include the predefined rules for RDFS and OWL — in place of a real URL for a rule file use one of the keywords `RDFS OWL OWLMicro OWLMini` (case insensitive).

To conclude, Source C.4 is an example complete rule file which includes the RDFS rules and defines a single extra rule.

Source C.4: Jena Example Rule File

```
# Example rule file
@prefix pre: <http://jena.hpl.hp.com/prefix#>.
@include <RDFS>.
[rule1: (?f pre:father ?a) (?u pre:brother ?f) -> (?u pre:uncle ?a)]
```

C.2 N3 Grammar

The aims of the N3 language are:

- to optimize expression of data and logic in the same language,
- to allow RDF to be expressed,
- to allow rules to be integrated smoothly with RDF,
- to allow quoting so that statements about statements can be made, and
- to be as readable, natural, and symmetrical as possible.

The language achieves these with the following features:

- URI abbreviation using prefixes which are bound to a namespace (using `@prefix`) a bit like in XML,
- repetition of another object for the same subject and predicate using a comma ",",
- repetition of another predicate for the same subject using a semicolon ";",
- bnode syntax with a certain properties just put the properties between "[" and "]",
- formulae allowing N3 graphs to be quoted within N3 graphs using "{" and "}",
- variables and quantification to allow rules, etc. to be expressed,
- and, a simple and consistent grammar defined by the context free grammar in Source C.5.

Source C.5: N3 Grammar in N3

```
# Notation3 in Notation3
# Context Free Grammar without tokenization

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix cfg: <http://www.w3.org/2000/10/swap/grammar/bnf#>.
@prefix rul: <http://www.w3.org/2000/10/swap/grammar/bnf-rules#>.
@prefix : <http://www.w3.org/2000/10/swap/grammar/n3#>.
@prefix n3: <http://www.w3.org/2000/10/swap/grammar/n3#>.
@prefix list: <http://www.w3.org/2000/10/swap/list#>.
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@keywords a, is, of.

language a cfg:Language;
  cfg:document  document;
  cfg:whiteSpace  "@@@@@".

document a rul:Used;
  cfg:mustBeOneSequence(
    (
      statements_optional
      cfg:eof
    )
  ).

statements_optional cfg:mustBeOneSequence (() ( statement "." statements_optional ) ).

# Formula does NOT need period on last statement
```

```
formulacontent cfg:mustBeOneSequence (
    ( statementlist )
  ).

statementlist cfg:mustBeOneSequence (
    ( )
    ( statement statementtail )
  ).

statementtail cfg:mustBeOneSequence (
    ( )
    ( "." statementlist )
  ).

statement  cfg:mustBeOneSequence (
    (declaration)
    (universal)
    (existential)
    (simpleStatement)
  ).

universal cfg:mustBeOneSequence (
    (
      "@forAll"
      [ cfg:commaSeparatedListOf symbol ]
    )).

existential cfg:mustBeOneSequence(
    (  "@forSome"
       [ cfg:commaSeparatedListOf symbol ]
    )).

declaration cfg:mustBeOneSequence(
    ( "@base" explicituri  )
    ( "@prefix" prefix explicituri  )
    ( "@keywords" [ cfg:commaSeparatedListOf barename ] )
  ).

simpleStatement cfg:mustBeOneSequence(( subject propertylist )).

propertylist cfg:mustBeOneSequence (
    ( )
    ( predicate  object objecttail propertylisttail )
  ).

propertylisttail cfg:mustBeOneSequence (
    ( )
    ( ";" propertylist )
  ).

objecttail cfg:mustBeOneSequence (
    ( )
    ( ","   object objecttail )
  ).

predicate cfg:mustBeOneSequence (
    ( expression )
    ( "@has" expression )
    ( "@is" expression "@of" )
    ( "@a" )
    ( "=" )
    ( "=>" )
    ( "<=" )
  ).
```

```
subject cfg:mustBeOneSequence ((expression)).

object cfg:mustBeOneSequence ((expression)).

expression cfg:mustBeOneSequence(
    ( pathitem pathtail )
  ).

pathtail cfg:mustBeOneSequence(
    (  )
    ( "!" expression )
    ( "^" expression )
  ).

pathitem cfg:mustBeOneSequence (
    ( symbol )
    ( "{" formulacontent "}" )
    ( quickvariable )
    ( numericliteral )
    ( literal )
    ( "[" propertylist "]"  )
    (  "("  pathlist ")"  )
                ( boolean )

boolean cfg:mustBeOneSequence (
                ( "@true" )
                ( "@false" )
) .

pathlist cfg:mustBeOneSequence (() (expression pathlist)).

symbol cfg:mustBeOneSequence (
    (explicituri)
    (qname)
  ).

numericliteral cfg:mustBeOneSequence (
          ( integer )
          ( rational )
          ( double )
          ( decimal )
) .

rational cfg:mustBeOneSequence (( integer "/" unsignedint)).

literal cfg:mustBeOneSequence(( string dtlang)).

dtlang cfg:mustBeOneSequence(  ()  ("@" langcode)  ("^^" symbol)).
```

C.3 OWL 2 RL Grammar

Source C.6 provides an overview of the expressivity allowed by OWL 2 RL by describing its grammar. It is extracted from the late 2012 W3C recommendation about OWL 2 profiles [101]. The reader should note that the Internationalized Resource Identifier (IRI) is a generalization of the URI which may contain non ASCII characters.

Source C.6: OWL 2 RL Grammar

```
Class := IRI
```

```
Datatype := IRI
ObjectProperty := IRI
DataProperty := IRI
AnnotationProperty := IRI
Individual := NamedIndividual | AnonymousIndividual
NamedIndividual := IRI
AnonymousIndividual := nodeID
Literal := typedLiteral | stringLiteralNoLanguage | stringLiteralWithLanguage
typedLiteral := lexicalForm '^^' Datatype
lexicalForm := quotedString
stringLiteralNoLanguage := quotedString
stringLiteralWithLanguage := quotedString languageTag

ObjectPropertyExpression := ObjectProperty | InverseObjectProperty
InverseObjectProperty := 'ObjectInverseOf' '(' ObjectProperty ')'
DataPropertyExpression := DataProperty

DataRange := Datatype | DataIntersectionOf
DataIntersectionOf := 'DataIntersectionOf' '(' DataRange DataRange { DataRange } ')'

zeroOrOne  := '0' | '1'
subClassExpression :=
    Class other than owl:Thing |
    subObjectIntersectionOf | subObjectUnionOf | ObjectOneOf |
    subObjectSomeValuesFrom | ObjectHasValue |
    DataSomeValuesFrom | DataHasValue
subObjectIntersectionOf := 'ObjectIntersectionOf' '(' subClassExpression
    subClassExpression { subClassExpression } ')'
subObjectUnionOf := 'ObjectUnionOf' '(' subClassExpression subClassExpression {
    subClassExpression } ')'
subObjectSomeValuesFrom :=
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression subClassExpression ')' |
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression owl:Thing ')'
superClassExpression :=
    Class other than owl:Thing |
    superObjectIntersectionOf | superComplementOf |
    superObjectAllValuesFrom | ObjectHasValue | superObjectMaxCardinality |
    DataAllValuesFrom | DataHasValue | superDataMaxCardinality
superObjectIntersectionOf := 'ObjectIntersectionOf' '(' superClassExpression
    superClassExpression { superClassExpression } ')'
superObjectComplementOf := 'ObjectComplementOf' '(' subClassExpression ')'
superObjectAllValuesFrom := 'ObjectAllValuesFrom' '(' ObjectPropertyExpression
    superClassExpression ')'
superObjectMaxCardinality :=
    'ObjectMaxCardinality' '(' zeroOrOne ObjectPropertyExpression [ subClassExpression
        ] ')' |
    'ObjectMaxCardinality' '(' zeroOrOne ObjectPropertyExpression owl:Thing ')'
superDataMaxCardinality := 'DataMaxCardinality' '(' zeroOrOne DataPropertyExpression [
    DataRange ] ')' |
equivClassExpression :=
    Class other than owl:Thing |
    equivObjectIntersectionOf |
    ObjectHasValue |
    DataHasValue
equivObjectIntersectionOf := 'ObjectIntersectionOf' '(' equivClassExpression
    equivClassExpression { equivClassExpression } ')'
ObjectOneOf := 'ObjectOneOf' '(' Individual { Individual }')'
ObjectHasValue := 'ObjectHasValue' '(' ObjectPropertyExpression Individual ')'
DataSomeValuesFrom := 'DataSomeValuesFrom' '(' DataPropertyExpression {
    DataPropertyExpression } DataRange ')'
DataAllValuesFrom := 'DataAllValuesFrom' '(' DataPropertyExpression {
    DataPropertyExpression } DataRange ')'
DataHasValue := 'DataHasValue' '(' DataPropertyExpression Literal ')'
```

```
Axiom := Declaration | ClassAxiom | ObjectPropertyAxiom | DataPropertyAxiom |
    DatatypeDefinition | HasKey | Assertion | AnnotationAxiom

ClassAxiom := SubClassOf | EquivalentClasses | DisjointClasses
SubClassOf := 'SubClassOf' '(' axiomAnnotations subClassExpression superClassExpression
    ')'
EquivalentClasses := 'EquivalentClasses' '(' axiomAnnotations equivClassExpression
    equivClassExpression { equivClassExpression } ')'
DisjointClasses := 'DisjointClasses' '(' axiomAnnotations subClassExpression
    subClassExpression { subClassExpression } ')'

ObjectPropertyAxiom :=
    SubObjectPropertyOf | EquivalentObjectProperties |
    DisjointObjectProperties | InverseObjectProperties |
    ObjectPropertyDomain | ObjectPropertyRange |
    FunctionalObjectProperty | InverseFunctionalObjectProperty |
    IrreflexiveObjectProperty |
    SymmetricObjectProperty | AsymmetricObjectProperty
    TransitiveObjectProperty
SubObjectPropertyOf := 'SubObjectPropertyOf' '(' axiomAnnotations
    subObjectPropertyExpression superObjectPropertyExpression ')'
subObjectPropertyExpression := ObjectPropertyExpression | propertyExpressionChain
propertyExpressionChain := 'ObjectPropertyChain' '(' ObjectPropertyExpression
    ObjectPropertyExpression { ObjectPropertyExpression } ')'
superObjectPropertyExpression := ObjectPropertyExpression
EquivalentObjectProperties := 'EquivalentObjectProperties' '(' axiomAnnotations
    ObjectPropertyExpression ObjectPropertyExpression { ObjectPropertyExpression } ')'
DisjointObjectProperties := 'DisjointObjectProperties' '(' axiomAnnotations
    ObjectPropertyExpression ObjectPropertyExpression { ObjectPropertyExpression } ')'
InverseObjectProperties := 'InverseObjectProperties' '(' axiomAnnotations
    ObjectPropertyExpression ObjectPropertyExpression ')'
ObjectPropertyDomain := 'ObjectPropertyDomain' '(' axiomAnnotations
    ObjectPropertyExpression superClassExpression ')'
ObjectPropertyRange := 'ObjectPropertyRange' '(' axiomAnnotations
    ObjectPropertyExpression superClassExpression ')'
FunctionalObjectProperty := 'FunctionalObjectProperty' '(' axiomAnnotations
    ObjectPropertyExpression ')'
InverseFunctionalObjectProperty := 'InverseFunctionalObjectProperty' '('
    axiomAnnotations ObjectPropertyExpression ')'
IrreflexiveObjectProperty := 'IrreflexiveObjectProperty' '(' axiomAnnotations
    ObjectPropertyExpression ')'
SymmetricObjectProperty := 'SymmetricObjectProperty' '(' axiomAnnotations
    ObjectPropertyExpression ')'
AsymmetricObjectProperty := 'AsymmetricObjectProperty' '(' axiomAnnotations
    ObjectPropertyExpression ')'
TransitiveObjectProperty := 'TransitiveObjectProperty' '(' axiomAnnotations
    ObjectPropertyExpression ')'

DataPropertyAxiom :=
    SubDataPropertyOf | EquivalentDataProperties | DisjointDataProperties |
    DataPropertyDomain | DataPropertyRange | FunctionalDataProperty
SubDataPropertyOf := 'SubDataPropertyOf' '(' axiomAnnotations subDataPropertyExpression
    superDataPropertyExpression ')'
subDataPropertyExpression := DataPropertyExpression
superDataPropertyExpression := DataPropertyExpression
EquivalentDataProperties := 'EquivalentDataProperties' '(' axiomAnnotations
    DataPropertyExpression DataPropertyExpression { DataPropertyExpression } ')'
DisjointDataProperties := 'DisjointDataProperties' '(' axiomAnnotations
    DataPropertyExpression DataPropertyExpression { DataPropertyExpression } ')'
DataPropertyDomain := 'DataPropertyDomain' '(' axiomAnnotations DataPropertyExpression
    superClassExpression ')'
DataPropertyRange := 'DataPropertyRange' '(' axiomAnnotations DataPropertyExpression
    DataRange ')'
FunctionalDataProperty := 'FunctionalDataProperty' '(' axiomAnnotations
```

```
    DataPropertyExpression ')'

DatatypeDefinition := 'DatatypeDefinition' '(' axiomAnnotations Datatype DataRange ')'

HasKey := 'HasKey' '(' axiomAnnotations subClassExpression '(' {
    ObjectPropertyExpression } ')' '(' { DataPropertyExpression } ')' ')'

Assertion :=
    SameIndividual | DifferentIndividuals | ClassAssertion |
    ObjectPropertyAssertion | NegativeObjectPropertyAssertion |
    DataPropertyAssertion | NegativeDataPropertyAssertion
sourceIndividual := Individual
targetIndividual := Individual
targetValue := Literal
SameIndividual := 'SameIndividual' '(' axiomAnnotations Individual Individual {
    Individual } ')'
DifferentIndividuals := 'DifferentIndividuals' '(' axiomAnnotations Individual
    Individual { Individual } ')'
ClassAssertion := 'ClassAssertion' '(' axiomAnnotations superClassExpression Individual
     ')'
ObjectPropertyAssertion := 'ObjectPropertyAssertion' '(' axiomAnnotations
    ObjectPropertyExpression sourceIndividual targetIndividual ')'
NegativeObjectPropertyAssertion := 'NegativeObjectPropertyAssertion' '('
    axiomAnnotations ObjectPropertyExpression sourceIndividual targetIndividual ')'
DataPropertyAssertion := 'DataPropertyAssertion' '(' axiomAnnotations
    DataPropertyExpression sourceIndividual targetValue ')'
NegativeDataPropertyAssertion := 'NegativeDataPropertyAssertion' '(' axiomAnnotations
    DataPropertyExpression sourceIndividual targetValue ')'
```

I think everybody in this country should learn how to program a computer because it teaches you how to think.

— Steve Jobs, 1955–2011

D

UbiSMART v2 Source Code Extracts

In this appendix, we provide the source code written for the reasoning modules inside UbiSMART v2. Each source below corresponds to one of the bundles described in section 8.4.

D.1 Stimulistener

Source D.1: Stimulistener Bundle

```
package sg.ipal.pawm.ubi.stimulistener;

import org.osgi.service.event.Event;
import org.osgi.service.event.EventHandler;

public interface Stimulistener extends EventHandler {
  public void handleEvent(Event event);
}
================================================================================
package sg.ipal.pawm.ubi.stimulistener.internal;

import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;
import java.util.concurrent.LinkedBlockingQueue;

import sg.ipal.pawm.ubi.cogitation.Cogitation;
import sg.ipal.pawm.ubi.eventbridge.EventSubscriber;
import sg.ipal.pawm.ubi.ntriplestore.NTripleStore;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
import org.osgi.service.event.Event;
import org.osgi.util.tracker.ServiceTracker;

import sg.ipal.pawm.tools.toolbox.ConfigR;
import sg.ipal.pawm.tools.toolbox.LogR;

public class Activator implements BundleActivator {
  /** PARAMETERS **/
  private static final String STIMULITOPIC = "stimulistener";
  private static final String LOAD_PREFIX = "load-";
  private static String EULER_DIR;
  private static final String S_BRIDGE = "sg.ipal.pawm.ubi.osgibridge.OSGiSubscriber";
  public static String SYSTEM_LOG;
  private static boolean DEBUG = true;

  /** VARIABLES **/
```

```
  public static BundleContext bc;
  public static NTripleStore n3Store;
  public static LinkedBlockingQueue<Event> eventQueue;
  public static LogR log;
  private ServiceTracker st;
  private StimuliDecoder decoder;


  public void start(BundleContext context) throws Exception {
    /** BUNDLE START **/
    // init variables
    bc = context;
    eventQueue = new LinkedBlockingQueue<Event>();
    ServiceReference ref = Activator.bc.getServiceReference(ConfigR.class.getName());
    ConfigR conf = (ConfigR) Activator.bc.getService(ref);
    SYSTEM_LOG = conf.getProperty("system_log");

    // get logR service
    ref = Activator.bc.getServiceReference(LogR.class.getName());
    log = (LogR) Activator.bc.getService(ref);

    // load n3 files into the triplestore
    loadTripleStore();

    // start the StimuliDecoder thread that process the eventQueue
    decoder = new StimuliDecoder();
    decoder.start();

    // instantiate stimulistener and subscribe to information topic
    // since there is no formal dependency: use service tracker in case service not
        started yet
    st = new ServiceTracker(bc, S_BRIDGE, null) {
      @Override
      public Object addingService(ServiceReference reference) {
        EventSubscriber bridge = (EventSubscriber) bc.getService(reference);
        bridge.subscribe(STIMULITOPIC, new StimulistenerImpl());
        return super.addingService(reference);
      }

      @Override
      public void remove(ServiceReference reference) {
        super.remove(reference);
      }
    };
    st.open();
  }


  public void stop(BundleContext context) throws Exception {
    /** BUNDLE STOP **/
    decoder.requestHalt();

  }


  public void loadTripleStore() throws IOException {
    /** LOAD N3 FILES IN TRIPLE STORE **/
    System.out.println("Loading files in triple store...");

    // retrieve the nTripleStore service reference and get an instance
    ServiceReference ref = Activator.bc.getServiceReference(NTripleStore.class.getName
        ());
    n3Store = (NTripleStore) Activator.bc.getService(ref);
```

```
        // get euler directory
        ref = Activator.bc.getServiceReference(ConfigR.class.getName());
        ConfigR conf = (ConfigR) Activator.bc.getService(ref);
        EULER_DIR = conf.getProperty("euler_dir");

        // get the n3 files from the euler directory
        File dir = new File(EULER_DIR);
        String[] children = dir.list();

        // this filter only returns n3 files with load prefix
        FilenameFilter fileFilter = new FilenameFilter() {
            public boolean accept(File dir, String name) {
                return name.startsWith(LOAD_PREFIX) && name.endsWith(".n3") && !name.
                    endsWith("~");
            }
        };
        children = dir.list(fileFilter);

        // get cogitation service to get owl deductive closure
        ref = Activator.bc.getServiceReference(Cogitation.class.getName());
        Cogitation cogit = (Cogitation) Activator.bc.getService(ref);
        n3Store.load(cogit.passOWL(children));
        if(DEBUG) {
          for(String sale : cogit.passOWL(children)) {
            System.out.println(sale);
          }
        }
      }

}
================================================================================
package sg.ipal.pawm.ubi.stimulistener.internal;

import org.osgi.service.event.Event;
import sg.ipal.pawm.ubi.stimulistener.Stimulistener;

public class StimulistenerImpl implements Stimulistener {

  /** PARAMETERS **/
  private final boolean DEBUG = false;


  @Override
  public void handleEvent(Event event) {
    /** MAIN METHOD CALLED WHEN STIMULISTER RECEIVES AN EVENT FROM SFM **/
    // debug if wanted
    if(DEBUG) {
      System.out.println("---------- NEW EVENT ---------------");
      for(String name : event.getPropertyNames()) {
        if(!name.equalsIgnoreCase("event.topics")) {
          System.out.println(name + " = " + event.getProperty(name));
        }
      }
    }

    // add event to the eventQueue, to be decoded by the StimuliDecoder thread
    try {
      Activator.eventQueue.put(event);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

}
```

```
=============================================================================================
package sg.ipal.pawm.ubi.stimulistener.internal;

import org.osgi.framework.ServiceReference;
import org.osgi.service.event.Event;

import sg.ipal.pawm.tools.toolbox.DateR;
import sg.ipal.pawm.ubi.cortex.Cortex;
import sg.ipal.pawm.ubi.ntriplestore.NTripleStore;


public class StimuliDecoder extends Thread {

  /** PARAMETERS **/
  String HOME_NS = "hom:";
  String MODEL_NS = "qol:";
  String CALENDAR = HOME_NS+"calendar";
  String CLOCK = HOME_NS+"clock";
  boolean DEBUG = false;

  /** VARIABLES **/
  private Event event;
  private NTripleStore n3Store;
  private boolean halt;


  public StimuliDecoder() {
    /** CONSTRUCTOR **/
    n3Store = Activator.n3Store;
    halt = false;
  }


  public void run() {
    /** DECODE EVENT IN A THREAD **/
    String sensor, value, time, dayOfWeek;
    boolean isState, isTime;

    // init variables
    dayOfWeek = "";

    // get dater service
    ServiceReference ref = Activator.bc.getServiceReference(DateR.class.getName());
    DateR dater = (DateR) Activator.bc.getService(ref);

    while(!halt) {
      try {
        event = Activator.eventQueue.take();
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      long startTime = System.nanoTime();

      // extract values from event
      sensor = (String) event.getProperty("sensor");
      value = (String) event.getProperty("value");
      time = (String) event.getProperty("time");

      if(DEBUG) {
        System.out.println("sensor:"+sensor+" | value:"+value+" | time:"+time);
      }
```

```
228         // check kind of sensor: state or value
            isState = n3Store.exist(getSensorURI(sensor), MODEL_NS+"hasPossibleState", "*");
            isTime = n3Store.exist(getSensorURI(sensor), MODEL_NS+"type", HOME_NS+"time");
            if(isState) {
              // update sensor current state
233           n3Store.updateObject(getSensorURI(sensor), MODEL_NS+"hasCurrentState",
                  getSensorStateURI(sensor, value));
              n3Store.updateSubject(getSensorURI(sensor), MODEL_NS+"hasLastUpdate", "true");
              // update time based on sensor event
              n3Store.updateObject(CLOCK, MODEL_NS+"hasValue", dater.getN3Time(time));
              // extract day of week and update if needed
238           if(!dater.getDayOfWeek(time).equalsIgnoreCase(dayOfWeek)) {
                n3Store.updateObject(CALENDAR, MODEL_NS+"hasValue", "\""+dater.getDayOfWeek(
                    time)+"\"^^qol:dayOfWeek");
              }
            } else if(isTime) {
              // update time value
243           value = time;
              n3Store.updateObject(getSensorURI(sensor), MODEL_NS+"hasValue", dater.getN3Time
                  (value));
    //          // extract day of week and update if needed
    //          if(!dater.getDayOfWeek(value).equalsIgnoreCase(dayOfWeek)) {
    //            n3Store.updateObject(CALENDAR, MODEL_NS+"hasValue", "\""+dater.getDayOfWeek
        (value)+"\"^^qol:dayOfWeek");
248 //          }
            } else {
              // update sensor value
              n3Store.updateObject(getSensorURI(sensor), MODEL_NS+"hasValue", value);
            }
253
            // update last update time
            n3Store.updateObject(getSensorURI(sensor), MODEL_NS+"lastUpdate", dater.getN3Time
                (time));

            long endTime = System.nanoTime();
258         long duration = endTime - startTime;
            Activator.log.append("Stimulistener processing duration, " + duration, Activator.
                SYSTEM_LOG, false);

            // launch inference
            startTime = System.nanoTime();
263         ref = Activator.bc.getServiceReference(Cortex.class.getName());
            Cortex cortex = (Cortex) Activator.bc.getService(ref);

            // pass isState to cortex in order to log for clustering only on sensor state
                change
            cortex.think(isState);
268         endTime = System.nanoTime();
            duration = endTime - startTime;
            Activator.log.append("Cortex processing duration, " + duration, Activator.
                SYSTEM_LOG, false);
            Activator.log.append("Event queue size, " + Activator.eventQueue.size(),
                Activator.SYSTEM_LOG, false);
          }
273     }


        private String getSensorURI(String sensorId) {
          /** GIVE SENSOR URI FOR GIVEN SENSOR ID **/
278       return HOME_NS+sensorId.toLowerCase();
        }


        private String getSensorStateURI(String sensorId, String value) {
```

```
283     /** GIVE SENSOR STATE URI FOR GIVEN SENSOR ID & VALUE **/
        return HOME_NS+sensorId.toLowerCase()+"_"+value.toLowerCase();
      }


288   public void requestHalt() {
        /** ENDS THE DECODER THREAD **/
        halt = true;
      }

293 }
```

D.2 Cortex

Source D.2: Cortex Bundle

```
package sg.ipal.pawm.ubi.cortex;
2
public interface Cortex
{
    void think(boolean isState);
    void registerCerebration(String bundleClass);
7   void unregisterCerebration(String bundleClass);
}
================================================================================
package sg.ipal.pawm.ubi.cortex.internal;

12 import java.util.Dictionary;
import java.util.Hashtable;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
17 import org.osgi.framework.ServiceReference;

import sg.ipal.pawm.tools.toolbox.ConfigR;
import sg.ipal.pawm.tools.toolbox.LogR;
import sg.ipal.pawm.ubi.cortex.Cortex;
22


public final class Activator implements BundleActivator {
    /** PARAMETERS **/
  public static String SYSTEM_LOG;
27
  /** VARIABLES **/
  public static BundleContext bc;
  public static int eventCount;
  public static LogR log;
32


  public void start( BundleContext context ) throws Exception {
    /** BUNDLE START **/
    bc = context;
37  eventCount = 0;

    // parse config file
    ServiceReference ref = Activator.bc.getServiceReference(ConfigR.class.getName());
    ConfigR conf = (ConfigR) Activator.bc.getService(ref);
42  SYSTEM_LOG = conf.getProperty("system_log");

    // get logR service
```

```
    ref = Activator.bc.getServiceReference(LogR.class.getName());
    log = (LogR) Activator.bc.getService(ref);

      // register services
      Dictionary<Object, Object> props = new Hashtable<Object, Object>();
    bc.registerService( Cortex.class.getName(), new CortexImpl(), props );
    }

  public void stop( BundleContext bc ) throws Exception {
    /** BUNDLE STOP **/
        // no need to unregister our service - the OSGi framework handles it for us
    }
}


================================================================================
package sg.ipal.pawm.ubi.cortex.internal;

import java.util.HashSet;
import java.util.Set;

import org.osgi.framework.ServiceReference;

import sg.ipal.pawm.ubi.cerebration.Cerebration;
import sg.ipal.pawm.ubi.cogitation.Cogitation;
import sg.ipal.pawm.ubi.cortex.Cortex;

public final class CortexImpl implements Cortex {
  /** PARAMETERS **/

  /** VARIABLES **/
  private Set<String> cerebraList;
  private CortexDecision decidor;


  public CortexImpl() {
    /** CONSTRUCTOR **/
    cerebraList = new HashSet<String>(500);
    decidor = new CortexDecision();
  }

    public void think(boolean isState) {
      /** HANDLE REASONING CYCLES **/

      /* testing tweak below */
      ServiceReference ref;

      // launch cerebrations one by one (+profiling)
      for (String cereBundle : cerebraList) {
        ref = Activator.bc.getServiceReference(cereBundle);
          Cerebration cerebration = (Cerebration) Activator.bc.getService(ref);
          long startTime = System.nanoTime();
        cerebration.think();
          long endTime = System.nanoTime();
        long duration = endTime - startTime;
        Activator.log.append("Cerebration "+cerebration.getName()+" processing time, "
            + duration, Activator.SYSTEM_LOG, false);
      }

      // launch cogitation (+profiling)
      ref = Activator.bc.getServiceReference(Cogitation.class.getName());
```

```
        Cogitation cogit = (Cogitation) Activator.bc.getService(ref);
        long startTime = System.nanoTime();
        String[] res =  cogit.think();
        long endTime = System.nanoTime();
112     long duration = endTime - startTime;
        Activator.log.append("Cogitation processing time, " + duration, Activator.
            SYSTEM_LOG, false);

        // temporary make CortexDecision after cogitation when a sensor state change
//        if(isState) {
117         decidor.decode(res);
//        }
      }


122   public void registerCerebration(String bundleClass) {
      /** REGISTER A CEREBRATION SERVICE **/
        cerebraList.add(bundleClass);
    }

127

      public void unregisterCerebration(String bundleClass) {
      /** UNREGISTER A CEREBRATION SERVICE **/
        cerebraList.remove(bundleClass);
    }
132
}
================================================================================
package sg.ipal.pawm.ubi.cortex.internal;

137 import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.Dictionary;
import java.util.HashMap;
142 import java.util.Hashtable;
import java.util.Map;

import org.osgi.framework.ServiceReference;
import org.osgi.service.event.Event;
147 import org.osgi.util.tracker.ServiceTracker;

import sg.ipal.pawm.tools.toolbox.DateR;
import sg.ipal.pawm.ubi.eventbridge.EventPublisher;

152 public class CortexDecision {

    /** PARAMETERS **/
    private static final String P_BRIDGE = "sg.ipal.pawm.ubi.osgibridge.OSGiPublisher";

157 /** VARIABLES **/
    private ServiceTracker st;
    private EventPublisher bridge;
    private boolean bridgeReady;

162

    public CortexDecision() {
      /** CONSTRUCTOR **/

      // initialize variables
167   bridgeReady = false;

      st = new ServiceTracker(Activator.bc, P_BRIDGE, null) {
        @Override
```

```
      public Object addingService(ServiceReference reference) {
172     bridge = (EventPublisher) Activator.bc.getService(reference);
        bridgeReady = true;
        return super.addingService(reference);
      }

177   @Override
      public void remove(ServiceReference reference) {
        bridgeReady = false;
        super.remove(reference);
      }
182 };
    st.open();
  }


187 public void decode(String[] res) {
    /** DECODE COGITATION RESULT LINE BY LINE **/
    // set local variables
    boolean logToSend = false;
    Map<String, Integer> topicMap = new HashMap<String, Integer>();
192 int nbOfTopics = 0;
    ArrayList<String> topics = new ArrayList<String>();
    ArrayList<String> logs = new ArrayList<String>();
    ArrayList<String> clocks = new ArrayList<String>();
    Activator.eventCount++;
197
    // decoding...
    for(int i=0; i<res.length; i++) {

      /* logging */
202   if(res[i].contains("lgr:") && !res[i].contains("@prefix")) {
        // get triple to log
        int beginIndex = res[i].indexOf("{");
        int endIndex = res[i].indexOf("}");
        String logtriple = res[i].substring(beginIndex+1, endIndex);
207
        // get topic for this log and add to map if needed
        String topic = res[i].split(":")[1].split("\\ ")[0];
        if(topicMap.get(topic) == null) {
          // new topic: create log line and related topic and default system clock
212       topicMap.put(topic, nbOfTopics);
          topics.add(topic);
          logs.add(logtriple);
          clocks.add(new SimpleDateFormat("yyyy-MM-dd_HH").format(new Date()));
          nbOfTopics++;
217       } else {
          // known topic: compile triple into its log line
          String newLine = logs.get(topicMap.get(topic)).concat(" . "+logtriple);
          logs.set(topicMap.get(topic), newLine);
        }
222
        // get time of log to replace default system clock
        if(logtriple.contains("hom:clock qol:hasValue")) {
          ServiceReference ref = Activator.bc.getServiceReference(DateR.class.getName()
              );
          DateR dater = (DateR) Activator.bc.getService(ref);
227       String clock = new SimpleDateFormat("yyyy-MM-dd_HH").format(dater.getDate(
              logtriple.split("\\ ")[2]));
          clocks.set(topicMap.get(topic), clock);
        }

        // set flag to send compiled log
232     logToSend = true;
```

```
      }

      /* decisions */
      else if(true) {
        // TODO
      }
    }

    // send log if needed
    if(logToSend && bridgeReady) {
      for(int i=0; i<nbOfTopics; i++) {
        Dictionary<String, Object> props = new Hashtable<String, Object>();
        props.put("log", logs.get(i));
        props.put("clock", clocks.get(i));
        props.put("topic", topics.get(i));
        props.put("eventcount", Activator.eventCount);
        bridge.send(new Event("log/"+topics.get(i), props));
      }
    }
  }

}
```

D.3 Cogitation

Source D.3: Cogitation Bundle

```
package sg.ipal.pawm.ubi.cogitation;

public interface Cogitation
{
    String[] think();
    String[] passOWL(String[] loadFiles);
}
================================================================================
package sg.ipal.pawm.ubi.cogitation.internal;

import java.util.Dictionary;
import java.util.Hashtable;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;

import sg.ipal.pawm.ubi.cogitation.Cogitation;
import sg.ipal.pawm.tools.toolbox.ConfigR;
import sg.ipal.pawm.tools.toolbox.LogR;


public final class Activator implements BundleActivator {
    /** PARAMETERS **/
  public static String SYSTEM_LOG;

  /** VARIABLES **/
  public static BundleContext bc;
  public static LogR log;


  public void start( BundleContext context ) throws Exception {
    /** BUNDLE START **/
```

```
        // init variables
        bc = context;

        // get ConfigR service
        ServiceReference ref = Activator.bc.getServiceReference(ConfigR.class.getName());
        ConfigR conf = (ConfigR) Activator.bc.getService(ref);
        SYSTEM_LOG = conf.getProperty("system_log");

        // get logR service
        ref = Activator.bc.getServiceReference(LogR.class.getName());
        log = (LogR) Activator.bc.getService(ref);

          // register services
          Dictionary<String, Object> props = new Hashtable<String, Object>();
        bc.registerService( Cogitation.class.getName(), new CogitationImpl(), props );
        }


      public void stop( BundleContext bc ) throws Exception {
        /** BUNDLE STOP **/
            // no need to unregister our service - the OSGi framework handles it for us
        }

}
================================================================================
package sg.ipal.pawm.ubi.cogitation.internal;

import java.io.File;
import java.io.FilenameFilter;
import java.io.IOException;

import sg.ipal.pawm.ubi.eyereasoner.EyeReasoner;
import sg.ipal.pawm.ubi.ntriplestore.NTripleStore;

import org.osgi.framework.ServiceReference;

import sg.ipal.pawm.tools.toolbox.ConfigR;
import sg.ipal.pawm.ubi.cogitation.Cogitation;


public final class CogitationImpl implements Cogitation {
  /** PARAMETERS **/
  private String EULER_DIR;
  private String DUMP_FILE;
  private String INFER_PREFIX = "infer-";
  private String QUERY_PREFIX = "query-";

  /** VARIABLES **/
  private NTripleStore n3Store;
  private EyeReasoner eye;
  private EyeDecoder decoder;


  public CogitationImpl() {
    /** CONSTRUCTOR **/
    // get triple store service
    ServiceReference ref = Activator.bc.getServiceReference(NTripleStore.class.getName
        ());
    n3Store = (NTripleStore) Activator.bc.getService(ref);

    // get eye reasoner service
    ServiceReference ref2 = Activator.bc.getServiceReference(EyeReasoner.class.getName
        ());
    eye = (EyeReasoner) Activator.bc.getService(ref2);
```

```
96
        // get euler directory
        ref = Activator.bc.getServiceReference(ConfigR.class.getName());
        ConfigR conf = (ConfigR) Activator.bc.getService(ref);
        EULER_DIR = conf.getProperty("euler_dir");
101     DUMP_FILE = EULER_DIR + conf.getProperty("dump_file");

        // create decoder
        decoder = new EyeDecoder(n3Store);
      }
106

      public String[] think() {
        /** HANDLE RULE-BASED REASONING USING EULER **/
        // get number of triples for profiling
111     Activator.log.append("N3Store triple count, " + n3Store.countTriples(), Activator.
            SYSTEM_LOG, false);

        // dump the triplestore into the dump.n3 file
        long startTime = System.nanoTime();
        try {
116       n3Store.write(DUMP_FILE);
        } catch (IOException e) {
          e.printStackTrace();
        }
        long endTime = System.nanoTime();
121     long duration = endTime - startTime;
        Activator.log.append("N3Store dump duration, " + duration, Activator.SYSTEM_LOG,
            false);

        File dir = new File(EULER_DIR);
        String[] rulesFiles = dir.list();
126     String[] queryFiles = dir.list();

        // filter n3 files with infer prefix
        FilenameFilter fileFilter = new FilenameFilter() {
          public boolean accept(File dir, String name) {
131         return name.startsWith(INFER_PREFIX) && name.endsWith(".n3") && !name.endsWith(
                "~");
          }
        };
        rulesFiles = dir.list(fileFilter);

136     // filter n3 files with query prefix
        fileFilter = new FilenameFilter() {
          public boolean accept(File dir, String name) {
            return name.startsWith(QUERY_PREFIX) && name.endsWith(".n3") && !name.endsWith(
                "~");
          }
141     };
        queryFiles = dir.list(fileFilter);

        // create arguments to call eye
        String[] eulerargs = new String[rulesFiles.length + queryFiles.length + 4];
146     eulerargs[0] = DUMP_FILE;
        for(int i=0 ; i<rulesFiles.length ; i++) {
          eulerargs[i+1] = EULER_DIR + rulesFiles[i];
        }
        eulerargs[rulesFiles.length + 1] = "--think";
151     eulerargs[rulesFiles.length + 2] = "--query";
        for(int i=0 ; i<queryFiles.length ; i++) {
          eulerargs[rulesFiles.length + i + 3] = EULER_DIR + queryFiles[i];
        }
        eulerargs[rulesFiles.length + queryFiles.length + 3] = "--nope";
```

```
156
        System.out.println("---------- START REASONING ----------");

        startTime = System.nanoTime();
        String[] res = eye.think(eulerargs);
161     endTime = System.nanoTime();
        duration = endTime - startTime;
        Activator.log.append("EyeBundle reasoning duration, " + duration, Activator.
            SYSTEM_LOG, false);

        System.out.println("---------- REASONER OUTPUT ----------");
166
        //decode the reasoner output
        startTime = System.nanoTime();
        decoder.decode(res);
        endTime = System.nanoTime();
171     duration = endTime - startTime;
        Activator.log.append("EyeDecoder processing time, " + duration, Activator.
            SYSTEM_LOG, false);
        System.out.println("-----------------------------------");
        return res;
      }
176

      public String[] passOWL(String[] loadFiles) {
        /** INFER WITH OWL RULES AND PASS ALL DEDUCTIVE CLOSURES **/
        // create arguments to call eye
181     String[] eulerargs = new String[loadFiles.length + 4];
        for(int i=0 ; i<loadFiles.length ; i++) {
          eulerargs[i] = EULER_DIR + loadFiles[i];
        }
        eulerargs[loadFiles.length] = EULER_DIR + "infer-owl.n3";
186     eulerargs[loadFiles.length + 1] = "--think";
        eulerargs[loadFiles.length + 2] = "--pass";
        eulerargs[loadFiles.length + 3] = "--nope";

        // infer and pass all
191     System.out.println("Generating owl deductive closure for load files...");
        return eye.think(eulerargs);
      }

    }
196 ================================================================================
    package sg.ipal.pawm.ubi.cogitation.internal;

    import java.util.Collection;

201 import sg.ipal.pawm.ubi.ntriplestore.NTriple;
    import sg.ipal.pawm.ubi.ntriplestore.NTripleStore;

    public class EyeDecoder {
      /** PARAMETERS **/
206
      /** VARIABLES **/
      NTripleStore n3Store;


211   public EyeDecoder(NTripleStore n3Store) {
        /** CONSTRUCTOR **/
        this.n3Store = n3Store;
      }

216
      public void decode(String[] res) {
```

```
    /** DECODE EYE OUTPUT LINE BY LINE **/
    // each line of result is contained separately in res list
    for(int i=0; i<res.length; i++) {

      /* updates to triple store */
      if(res[i].contains("ts:n3store ts:")) {
        //print the reasoner output
        int beginIndex = res[i].indexOf("{");
        int endIndex = res[i].indexOf("}");
        Collection<NTriple> tripleSet = n3Store.getNTriples(res[i].substring(beginIndex
            +1, endIndex));
        System.out.println("N3 STORE >>> " + res[i].substring(beginIndex+1, endIndex));
        NTriple[] triples = new NTriple[tripleSet.size()];
        tripleSet.toArray(triples);

        for(int j= 0; j<triples.length; j++)
        {
          if(res[i].contains("ts:update"))
          {
            n3Store.updateObject(triples[j].getSubject(), triples[j].getPredicate(),
                triples[j].getObject());
          }
          else if(res[i].contains("ts:add"))
          {
            n3Store.add(triples[j].getSubject(), triples[j].getPredicate(), triples[j].
                getObject());
          }
          else if(res[i].contains("ts:remove"))
          {
            n3Store.remove(triples[j].getSubject(), triples[j].getPredicate(), triples[
                j].getObject());
          }
        }
      }

      /* decisions */
      else if(true) {
        // TODO
      }
    }
  }

}
```

D.4 EyeReasoner

Source D.4: EyeReasoner Bundle

```
package sg.ipal.pawm.ubi.eyereasoner;

public interface EyeReasoner
{
  String[] think(String[] eulerArgs);
}
================================================================================
package sg.ipal.pawm.ubi.eyereasoner.internal;

import java.util.Dictionary;
import java.util.Properties;
```

```
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;

import sg.ipal.euler.ProofEngineService;
import sg.ipal.pawm.ubi.eyereasoner.EyeReasoner;

public final class Activator implements BundleActivator {
    /** PARAMETERS **/

  /** VARIABLES **/
  public static BundleContext bc;
  public static ProofEngineService proofEngine;


    public void start(BundleContext context) throws Exception {
      /** BUNDLE START **/
      bc = context;

      //install euler bin from a specific directory
      new Euler().installEuler();

        // Register our services implementation in the OSGi service registry
        Dictionary<?, ?> props = new Properties();
        bc.registerService( EyeReasoner.class.getName(), new EyeReasonerImpl(), props )
            ;

        ServiceReference ref = bc.getServiceReference(ProofEngineService.class.getName
            ());
    proofEngine = (ProofEngineService) bc.getService(ref);

    }


    public void stop(BundleContext bc) throws Exception {
      /** BUNDLE STOP **/

    }
}
================================================================================
package sg.ipal.pawm.ubi.eyereasoner.internal;

import sg.ipal.pawm.ubi.eyereasoner.EyeReasoner;


public final class EyeReasonerImpl implements EyeReasoner
{

  public String[] think(String[] eulerArgs) {
    /** LAUNCH EYE INFERENCE ONE TIME **/
    return Activator.proofEngine.runProofEngine(eulerArgs).split("\n");
  }
}
================================================================================
package sg.ipal.pawm.ubi.eyereasoner.internal;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import org.osgi.framework.ServiceReference;
```

```
import sg.ipal.pawm.tools.toolbox.ConfigR;

public class Euler {
  /** PARAMETERS **/
  public static String EYE_BIN; //euler bin folder for installation

  /** VARIABLES **/


    public void config() {
      ServiceReference ref = Activator.bc.getServiceReference(ConfigR.class.getName());
      ConfigR conf = (ConfigR) Activator.bc.getService(ref);
      EYE_BIN = conf.getProperty("euler_bin");
    }


    public void installEuler() {
      /** INSTALLS EYE ON MACHINE **/

    // get parameters from config file
      config();

      // install depending on machine OS
      boolean windows = System.getProperty("os.name").startsWith("Windows");
    if(windows) {
      String tmpdir = System.getProperty("java.io.tmpdir");
      String sep = System.getProperty("file.separator");
      if (!tmpdir.endsWith(sep)) {
        tmpdir += sep;
      }
      File tmpFolder = new File(tmpdir+"eye");
      if(!tmpFolder.exists()) {
        System.out.println("Installing eye in "+tmpdir+"eye");
        File srcFolder = new File(EYE_BIN);
        try {
          copyDirectory(srcFolder, tmpFolder);
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    } else /* UNIX */ {
      File tmpFolder = new File("/tmp/eye/");
      if(!tmpFolder.exists()) {
        System.out.println("Installing eye in /tmp/eye/");
        File srcFolder = new File(EYE_BIN);
        try {
          copyDirectory(srcFolder, tmpFolder);
        } catch (IOException e) {
          e.printStackTrace();
        }
      }
    }
    }


    private void copyDirectory(File srcPath, File dstPath) throws IOException {
      /** COPY GIVEN DIRECTORY TO GIVEN LOCATION **/
    if (srcPath.isDirectory()) {
      if (!dstPath.exists()) {
        dstPath.mkdir();
      }
      String files[] = srcPath.list();
      for(int i = 0; i < files.length; i++) {
```

```
139           copyDirectory(new File(srcPath, files[i]), new File(dstPath, files[i]));
          }
        } else {
          if(!srcPath.exists()) {
            System.out.println("File or directory does not exist.");
144           System.exit(0);
          } else {
            InputStream in = new FileInputStream(srcPath);
                OutputStream out = new FileOutputStream(dstPath);
            // Transfer bytes from in to out
149             byte[] buf = new byte[1024];
            int len;
                while ((len = in.read(buf)) > 0) {
              out.write(buf, 0, len);
            }
154         in.close();
                out.close();
                // keep files executable
                if(srcPath.canExecute()) {
                  dstPath.setExecutable(true);
159             }
          }
        }
      }

164 }
```

D.5 Cerebration

Source D.5: Cerebration Bundle (abstract class)

```
1   package sg.ipal.pawm.ubi.cerebration;

    public interface Cerebration
    {
        void think();
6       String getName();
    }
    ================================================================================
    package sg.ipal.pawm.ubi.cerebration.internal;

11  import java.util.Dictionary;
    import java.util.Hashtable;

    import org.osgi.framework.BundleActivator;
    import org.osgi.framework.BundleContext;
16
    import sg.ipal.pawm.ubi.cerebration.Cerebration;


    public final class Activator implements BundleActivator {
21      /** PARAMETERS **/

      /** VARIABLES **/
      public static BundleContext bc;


26
      public void start( BundleContext context ) throws Exception {
        /** BUNDLE START **/
        bc = context;
```

```
31        // register services
          Dictionary<String, Object> props = new Hashtable<String, Object>();
        bc.registerService( Cerebration.class.getName(), new CerebrationImpl(), props );
        }

36
      public void stop( BundleContext bc ) throws Exception {
        /** BUNDLE STOP **/
            // no need to unregister our service - the OSGi framework handles it for us
        }
41  }
    ================================================================================
    package sg.ipal.pawm.ubi.cerebration.internal;

    import sg.ipal.pawm.ubi.cerebration.Cerebration;
46
    public final class CerebrationImpl implements Cerebration {
      /** ABSTRACT CLASS **/
      public CerebrationImpl() {}

51
        public void think() {
          /** ABSTRACT METHOD **/
          System.out.println("This is a test of Cerebration.think()");
        }
56

      public String getName() {
        /** ABSTRACT METHOD **/
          return "AbstractCerebration";
61    }

    }
```

D.6 MotionEstimator

Source D.6: MotionEstimator Bundle (Cerebration example)

```
    package sg.ipal.pawm.ubi.cerebration.motionestimator;
2
    import sg.ipal.pawm.ubi.cerebration.Cerebration;

    /**
     * Public API representing an example OSGi service
7    */
    public interface MotionEstimator extends Cerebration
    {
      void think();
    }
12  ================================================================================
    package sg.ipal.pawm.ubi.cerebration.motionestimator.internal;

    import java.util.Dictionary;
    import java.util.Hashtable;
17
    import org.osgi.framework.BundleActivator;
    import org.osgi.framework.BundleContext;
    import org.osgi.framework.ServiceReference;
```

```
22  import sg.ipal.pawm.ubi.cortex.Cortex;
    import sg.ipal.pawm.ubi.cerebration.motionestimator.MotionEstimator;


    public final class Activator implements BundleActivator {
27      /** PARAMETERS **/

      /** VARIABLES **/
      public static BundleContext bc;
      private Cortex cortex;
32


      public void start( BundleContext context ) throws Exception {
        /** BUNDLE START **/
        bc = context;
37
          // register services
          Dictionary<String, Object> props = new Hashtable<String, Object>();
        bc.registerService( MotionEstimator.class.getName(), new MotionEstimatorImpl(),
            props );

42      // register to cortex
        ServiceReference ref = Activator.bc.getServiceReference(Cortex.class.getName());
        cortex = (Cortex) Activator.bc.getService(ref);
        cortex.registerCerebration(MotionEstimator.class.getName());
        }
47


      public void stop( BundleContext bc ) throws Exception {
            /** BUNDLE START **/
            // no need to unregister our service - the OSGi framework handles it for us
52
        // unregister from cortex
        cortex.unregisterCerebration(MotionEstimator.class.getName());
        }

57  }
    ================================================================================
    package sg.ipal.pawm.ubi.cerebration.motionestimator.internal;

    import java.util.ArrayList;
62  import java.util.Date;
    import java.util.Iterator;

    import sg.ipal.pawm.ubi.ntriplestore.NTriple;
    import sg.ipal.pawm.ubi.ntriplestore.NTripleStore;
67  import org.osgi.framework.ServiceReference;

    import sg.ipal.pawm.tools.toolbox.ConfigR;
    import sg.ipal.pawm.tools.toolbox.DateR;
    import sg.ipal.pawm.ubi.cerebration.motionestimator.MotionEstimator;
72
    public final class MotionEstimatorImpl implements MotionEstimator {
        /** PARAMETERS **/
      private int TIME_WINDOW;
      private String NAME = "MotionEstimator";
77
      /** VARIABLES **/
      private NTripleStore n3Store;
      private String house;
      private ArrayList<Room> rooms;
82    private ArrayList<Sensor> sensors;
```

```
  public MotionEstimatorImpl() {
    /** CONSTRUCTOR **/
87  // read time window in config file
    ServiceReference ref = Activator.bc.getServiceReference(ConfigR.class.getName());
    ConfigR conf = (ConfigR) Activator.bc.getService(ref);
    TIME_WINDOW = conf.getIntProperty("motion_window_min") * 60 * 1000; // in
        milliseconds

92  // get triple store service
    ref = Activator.bc.getServiceReference(NTripleStore.class.getName());
    n3Store = (NTripleStore) Activator.bc.getService(ref);

    // get house URI
97  house = (String) n3Store.searchURIs("?", "a", "qol:House").toArray()[0];

    // get rooms URIs and fill room list
    rooms = new ArrayList<Room>();
    Iterator<String> roomURIs = n3Store.searchURIs("?", "a", "qol:Room").iterator();
102 while(roomURIs.hasNext()) {
      rooms.add(new Room(roomURIs.next()));
    }

    // get sensors URIs and fill sensor list (room by room)
107 sensors = new ArrayList<Sensor>();
    for (int i=0; i<rooms.size(); i++) {
      Iterator<NTriple> sensorTriples = n3Store.searchNTriples("*", "qol:deployedIn",
          rooms.get(i).getURI()).iterator();
      while(sensorTriples.hasNext()) {
        Sensor s = new Sensor(sensorTriples.next().getSubject());
112     s.setRoom(i);
        sensors.add(s);
      }
    }
  }
117

  public void think() {
    /** ESTIMATE THE MOTION DETECTED IN EACH ROOM AND IN THE HOUSE **/
    // reset motion of each room
122 for (int i=0; i<rooms.size(); i++) {
      rooms.get(i).setMotion(0);
    }

    // get dater service
127 ServiceReference ref = Activator.bc.getServiceReference(DateR.class.getName());
    DateR dater = (DateR) Activator.bc.getService(ref);

    // get clock time
    String sClock = (String) n3Store.searchURIs("hom:clock", "qol:hasValue", "?").
        toArray()[0];
132 Date clock = dater.getDate(sClock);

    // remove on states out of time window and compute motion in each room (sensor by
        sensor)
    for (int i=0; i<sensors.size(); i++) {
      ArrayList<Date> on = sensors.get(i).getOnStates();
137   for (int j=0; j<on.size(); j++) {
        // remove if too old
        if(clock.getTime() - on.get(j).getTime() > TIME_WINDOW) {
          on.remove(j);
        }
142   }
      sensors.get(i).setOnStates(on);
      int roomID = sensors.get(i).getRoom();
```

```
      rooms.get(roomID).incrementMotion(sensors.get(i).getOnStates().size());
    }

    // check latest sensor update and update motion estimation (sensor by sensor)
    for (int i=0; i<sensors.size(); i++) {
      // get last update from triple store
      String slU = (String) n3Store.searchURIs(sensors.get(i).getURI(), "qol:lastUpdate
          ", "?").toArray()[0];
      Date lU = dater.getDate(slU);
      // get arraylist of previous on states
      ArrayList<Date> pon = sensors.get(i).getOnStates();
      // check if last update different from the one in arraylist
      if(pon.size() != 0) {
        if(lU.getTime() - pon.get(pon.size()-1).getTime() != 0) {
          String state = (String) n3Store.searchURIs(sensors.get(i).getURI(), "qol:
              hasCurrentState", "?").toArray()[0];
          // check if state is on
          if(state.endsWith("on")) {
            // add to arraylist
            pon.add(lU);
            sensors.get(i).setOnStates(pon);
            rooms.get(sensors.get(i).getRoom()).incrementMotion(1);
          }
        }
      } else {
        String state = (String) n3Store.searchURIs(sensors.get(i).getURI(), "qol:
            hasCurrentState", "?").toArray()[0];
        // check if state is on
        if(state.endsWith("on")) {
          // add to arraylist
          pon.add(lU);
          sensors.get(i).setOnStates(pon);
          rooms.get(sensors.get(i).getRoom()).incrementMotion(1);
        }
      }
    }

    // sum motion of whole house and update triple store
    int homeMotion = 0;
    for (int i=0; i<rooms.size(); i++) {
      int roomMotion = rooms.get(i).getMotion();
      n3Store.updateObject(rooms.get(i).getURI(), "qol:motionMeasured", Integer.
          toString(roomMotion));
      homeMotion = homeMotion + roomMotion;
    }
    n3Store.updateObject(house, "qol:motionMeasured", Integer.toString(homeMotion));
  }


  public String getName() {
    return NAME;
  }

}
================================================================================
package sg.ipal.pawm.ubi.cerebration.motionestimator.internal;

public class Room {
  /** VARIABLES **/
  private String uri;
  private int motion;


  public Room(String uri) {
```

```
    /** CONSTRUCTOR **/
    this.uri = uri;
  }


  public void incrementMotion(int i) {
    motion = motion + i;
  }


  /** GETTERS & SETTERS **/
  public void setMotion(int motion) {
    this.motion = motion;
  }

  public int getMotion() {
    return motion;
  }

  public String getURI() {
    return uri;
  }

}
================================================================================
package sg.ipal.pawm.ubi.cerebration.motionestimator.internal;

import java.util.ArrayList;
import java.util.Date;

public class Sensor {
  /** VARIABLES **/
  private String uri;
  private int roomID;
  private ArrayList<Date> onStates;


  public Sensor(String uri) {
    /** CONSTRUCTOR **/
    this.uri = uri;
    onStates = new ArrayList<Date>();
  }


  /** GETTERS & SETTERS **/
  public void setRoom(int room) {
    this.roomID = room;
  }

  public int getRoom() {
    return roomID;
  }

  public String getURI() {
    return uri;
  }

  public ArrayList<Date> getOnStates() {
    return onStates;
  }

  public void setOnStates(ArrayList<Date> onStates) {
    this.onStates = onStates;
  }
```

```
}
```

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

— John F. Woods

E

Ontological Models and Rules

E.1 Ontology for the Service Delivery Aspect

Source E.1: Model (TBox) for the service delivery, in Notation3

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix ske: <skeleton#>.


## CLASSES ##
ske:Person a rdfs:Class.
ske:Resident a rdfs:Class;
  rdfs:subClassOf ske.People.
ske:Caregiver a rdfs:Class;
  rdfs:subClassOf ske.People.
ske:Environment a rdfs:Class.
ske:Context a rdfs:Class.
ske:Service a rdfs:Class.
ske:Reminder a rdfs:Class;
  rdfs:subClassOf ske:Service.
ske:Notification a rdfs:Class;
  rdfs:subClassOf ske:Service.
ske:Activity a rdfs:Class;
  rdfs:subClassOf ske:Context.
ske:Location a rdfs:Class;
  rdfs:subClassOf ske:Context.
ske:Deviance a rdfs:Class;
  rdfs:subClassOf ske:Context.
ske:EnvironmentState a rdfs:Class;
  rdfs:subClassOf ske:Context.
ske:Device a rdfs:Class;


## OBJECT PROPERTIES ##
ske:hasContext a owl:ObjectProperty;
  rdfs:domain ske:Resident;
  rdfs:range ske:Context.
ske:hasState a owl:ObjectProperty;
  rdfs:domain ske:Environment;
  rdfs:range ske:EnvironmentState.
ske:helpsWith a owl:ObjectProperty;
  rdfs:domain ske:Service;
  rdfs:range ske:Context.
```

```
ske:startFor a owl:ObjectProperty;
  rdfs:domain ske:Service;
  rdfs:range ske:Resident.
ske:stopFor a owl:ObjectProperty;
  rdfs:domain ske:Service;
  rdfs:range ske:Resident.
ske:runningFor a owl:ObjectProperty;
  rdfs:domain ske:Service;
  rdfs:range ske:Resident.
ske:solvedBy a owl:ObjectProperty;
  rdfs:domain ske:Deviance;
  rdfs:range ske:Context.
ske:mutuallyExclusiveWith a owl:ObjectProperty;
  rdfs:domain ske:Context;
  rdfs:range ske:Context.
ske:watchesAfter a owl:ObjectProperty;
  rdfs:domain ske:Caregiver;
  rdfs:range ske:Resident.
ske:usedBy a owl:ObjectProperty;
  rdfs:domain ske:Device;
  rdfs:range ske:Person.
ske:onDevice a owl:ObjectProperty;
  rdfs:domain ske:Service;
  rdfs:range ske:Device.
ske:deployedIn a owl:ObjectProperty;
  rdfs:domain ske:Device;
  rdfs:range ske:Location.
ske:hasAckService a owl:ObjectProperty;
  rdfs:domain ske:Service;
  rdfs:range ske:Service.
ske:escalateTo a owl:ObjectProperty;
  rdfs:domain ske:Reminder;
  rdfs:range ske:Notification.


## DATATYPE PROPERTIES ##
ske:osgiClassName a owl:DatatypeProperty;
  rdfs:domain ske:Service;
  rdfs:range xsd:string.
ske:name a owl:DatatypeProperty;
  rdfs:range xsd:string.
ske:snoozeTime a owl:DatatypeProperty;
  rdfs:domain ske:Resident;
  rdfs:range xsd:int.
ske:busy a owl:DatatypeProperty;
  rdfs:domain ske:Caregiver;
  rdfs:range xsd:boolean.
ske:stageForAlert a owl:DatatypeProperty;
  rdfs:domain ske:Resident;
  rdfs:range xsd:int.
ske:needHands a owl:DatatypeProperty;
  rdfs:domain ske:Context;
  rdfs:range xsd:boolean.
ske:solved a owl:DatatypeProperty;
  rdfs:domain ske:Deviance;
  rdfs:range xsd:boolean.
ske:asserted a owl:DatatypeProperty;
  rdfs:domain log:Formula;
  rdfs:range xsd:boolean.
ske:handheld a owl:DatatypeProperty;
  rdfs:domain ske:Device;
  rdfs:range xsd:boolean.
ske:repeat a owl:DatatypeProperty;
  rdfs:domain ske:Service;
```

```
  rdfs:range xsd:boolean.
ske:timeSent a owl:DatatypeProperty;
  rdfs:domain ske:Service;
  rdfs:range xsd:dateTime.
ske:stage a owl:DatatypeProperty;
  rdfs:domain ske:Service;
  rdfs:range xsd:int.
ske:acknowledgement a owl:DatatypeProperty;
  rdfs:domain ske:Service;
  rdfs:range xsd:string.
ske:ackHandled a owl:DatatypeProperty;
  rdfs:domain ske:Service;
  rdfs:range xsd:boolean.
```

Source E.2: Example ABox for the service delivery, in Notation3

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix env: <environment#>.
@prefix ske: <skeleton#>.


## INITIAL DATA ##
## People
env:patient1 a ske:Resident;
  ske:name """John"""@en.
env:patient2 a ske:Resident;
  ske:name """Jane"""@en.
env:unknown a ske:Resident;
  ske:name """Someone"""@en.
env:caregiver1 a ske:Caregiver;
  ske:name """Tom"""@en;
  ske:watchesAfter env:patient1, env:patient2.

## Location or environments
env:bathroom a ske:Environment;
  a ske:Location.
env:bedroom a ske:Location.

## Devices
env:speaker1 a ske:Device;
  ske:deployedIn env:bathroom.
env:speaker2 a ske:Device;
  ske:deployedIn env:bedroom.
env:iphone a ske:Device;
  ske:usedBy env:caregiver1.



## SCENARIOS ##
## Tap left on
env:tapOff a ske:EnvironmentState.
env:leftTapOn a ske:Deviance;
  ske:solvedBy env:tapOff;
  ske:mutuallyExclusiveWith env:tapOff.
env:tapOnReminder a ske:Reminder;
  ske:helpsWith env:leftTapOn;
  ske:escalateTo env:tapOnNotif;
  ske:osgiClassName """sg.ipal.aal.tapon.TapOnReminder"""@en.
env:tapOnNotif a ske:Notification;
  ske:helpsWith env:leftTapOn;
```

```
  ske:osgiClassName """sg.ipal.aal.tapon.TapOnNotif"""@en.

## Shower too long
env:showerEmpty a ske:EnvironmentState.
env:showerTooLong a ske:Deviance;
  ske:solvedBy env:showerEmpty;
  ske:mutuallyExclusiveWith env:showerEmpty.
env:showerTooLongReminder a ske:Reminder;
  ske:helpsWith env:showerTooLong;
  ske:escalateTo env:showerTooLongNotif;
  ske:osgiClassName """sg.ipal.aal.showertoolong.ShowerTooLongReminder"""@en.
env:showerTooLongNotif a ske:Notification;
  ske:helpsWith env:showerTooLong;
  ske:osgiClassName """sg.ipal.aal.showertoolong.ShowerTooLongNotif"""@en.

## Wandering at night
env:sleeping a ske:Activity;
  ske:mutuallyExclusiveWith env:showerTooLong.
env:wanderingTooLong a ske:Deviance;
  ske:solvedBy env:sleeping;
  ske:mutuallyExclusiveWith (env:showerTooLong env:sleeping).
env:wanderingTooLongNotif a ske:Notification;
  ske:helpsWith env:wanderingTooLong;
  ske:osgiClassName """sg.ipal.aal.wandering.WanderingNotif"""@en.

## Unknown
env:unknownContext a ske:Context;
  ske:mutuallyExclusiveWith (env:sleeping env:wanderingTooLong env:showerTooLong).
```

Source E.3: Rules for the service delivery, in Notation3

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rul: <rules#>.
@prefix ske: <skeleton#>.


## RULES ##
@forAll :u, :c, :s, :sc, :es.

## Infer services to start
{:u ske:hasContext :c. :c ske:hasService :s. </home/eye/nh-jan12/input.n3>!log:
    semantics log:notIncludes {:s ske:runingForUser :u}} => {:s ske:startForUser :u}.

## Infer services to stop by environmental context
{:s ske:runingForUser :u. :c ske:hasService :s. :c ske:solvedBy :es. ?environment ske:
    hasState :es} => {:s ske:stopForUser :u. {:u ske:hasContext :c} ske:asserted "false
    "^^xsd:boolean}.

## Infer services to stop by personal context
{:s ske:runingForUser :u. :c ske:hasService :s. :c ske:solvedBy :sc. :u ske:hasContext
    :sc} => {:s ske:stopForUser :u. {:u ske:hasContext :c} ske:asserted "false"^^xsd:
    boolean}.


## CONVENIENCE RULES ##
## Mutual exclusivity is symetric (could use existing owl properties and rules for
    symmetry)
{?x ske:mutuallyExclusiveWith ?y} => {?y ske:mutuallyExclusiveWith ?x}.
```

Source E.4: Queries for the service delivery, in Notation3

```
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix string: <http://www.w3.org/2000/10/swap/string#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix ske: <skeleton#>.


## QUERIES ##
@forAll :u, :s, :scn, :f, :name.

## Any service to be started?
{:s ske:startForUser :u. :u ske:name :name. :s ske:osgiClassName :scn. ("Start service
    " :s " for " :u " - service class = " :scn " - user name = " :name) string:
    concatenation ?print} => {:s log:outputString ?print}.

## Any service to be stoped?
{:s ske:stopForUser :u. :u ske:name :name. :s ske:osgiClassName :scn. ("Stop service "
    :s " for " :u " - service class = " :scn " - user name = " :name) string:
    concatenation ?print} => {:s log:outputString ?print}.

## Asserted formula to be added to input file
{:f ske:asserted "true"^^xsd:boolean} => {:f ske:asserted "true"^^xsd:boolean}.
{:f ske:asserted "false"^^xsd:boolean} => {:f ske:asserted "false"^^xsd:boolean}.
```

E.2 Ontology for the Activity Recognition Aspect

Source E.5: Model (TBox) for the activity recognition, in Notation3

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:    <http://www.w3.org/2002/07/owl#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.

@prefix qol:  <load-model#>.



## CLASSES ##
## ------- ##
qol:Person a rdfs:Class.
qol:Resident a rdfs:Class;
  rdfs:subClassOf qol:Person.
qol:Environment a rdfs:Class.
qol:House a rdfs:Class;
  rdfs:subClassOf qol:Environment.
qol:Outside a rdfs:Class;
  rdfs:subClassOf qol:Environment.
qol:Room a rdfs:Class;
  rdfs:subClassOf qol:Environment.
qol:Bedroom a rdfs:Class;
  rdfs:subClassOf qol:Room.
qol:Livingroom a rdfs:Class;
  rdfs:subClassOf qol:Room.
qol:Kitchen a rdfs:Class;
  rdfs:subClassOf qol:Room.
qol:Bathroom a rdfs:Class;
```

```
    rdfs:subClassOf qol:Room.
qol:Toilet a rdfs:Class;
    rdfs:subClassOf qol:Room.
qol:Object a rdfs:Class.
qol:Furniture a rdfs:Class;
    rdfs:subClassOf qol:Object.
qol:Door a rdfs:Class;
    rdfs:subClassOf qol:Furniture.
qol:Sensor a rdfs:Class.
qol:SensorState a rdfs:Class.
qol:SensorType a rdfs:Class.
qol:Activity a rdfs:Class.
qol:Deviance a rdfs:Class;
    rdfs:subClassOf qol:Activity;
    rdfs:comment "Problematic activity"@en.
qol:DayOfWeek a rdfs:Class;
    rdfs:comment "we did not find any equivalent"@en;
    rdfs:comment "values: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday"
        @en.


## OBJECT PROPERTIES ##
## ----------------- ##
qol:liveIn a owl:ObjectProperty;
    rdfs:comment "House where the resident live."@en;
    rdfs:domain qol:Resident;
    rdfs:range qol:Environment.

qol:detectedIn a owl:ObjectProperty;
    rdfs:comment "Room where the resident is detected."@en;
    rdfs:domain qol:Resident;
    rdfs:range qol:Environment.

qol:useNow a owl:ObjectProperty;
    rdfs:comment "Object a person is currently using."@en;
    rdfs:domain qol:Resident;
    rdfs:range qol:Object.

qol:believedToDo a owl:ObjectProperty;
    rdfs:comment "Activity a resident is believed to be doing."@en;
    rdfs:domain qol:Resident;
    rdfs:range qol:Activity.

qol:cameFrom a owl:ObjectProperty;
    rdfs:comment "Room the resident was in before the current one."@en;
    rdfs:domain qol:Resident;
    rdfs:range qol:Environment.

qol:partOf a owl:ObjectProperty;
    a owl:TransitiveProperty;
    rdfs:comment "Describe inclusion of environments."@en;
    rdfs:domain qol:Environment;
    rdfs:range qol:Environment.

qol:locatedIn a owl:ObjectProperty;
    rdfs:comment "Location of a door in the environment."@en;
    rdfs:domain qol:Object;
    rdfs:range qol:Environment.

qol:deployedIn a owl:ObjectProperty;
    rdfs:comment "Deployment location of a sensor."@en;
    rdfs:domain qol:Sensor;
    rdfs:range qol:Environment.
```

```
qol:attachedTo a owl:ObjectProperty;
  rdfs:comment "Describe the binding of sensor to a furniture."@en;
  rdfs:domain qol:Sensor;
  rdfs:range qol:Object.

qol:hasPossibleState a owl:ObjectProperty;
  rdfs:comment "Possible state of a sensor."@en;
  rdfs:domain qol:Sensor;
  rdfs:range qol:SensorState.

qol:type a owl:ObjectProperty;
  rdfs:comment "Type of a sensor."@en;
  rdfs:domain qol:Sensor;
  rdfs:range qol:SensorType.

qol:hasCurrentState a owl:ObjectProperty;
  rdfs:comment "Current state of a sensor."@en;
  rdfs:domain qol:Sensor;
  rdfs:range qol:SensorState.

qol:hasDurationEquivalent a owl:ObjectProperty;
  rdfs:comment "Links a date predicate with its equivalent duration predicate."@en;
  rdfs:domain owl:DatatypeProperty;
  rdfs:range owl:DatatypeProperty.


## DATATYPE PROPERTIES ##
## ------------------- ##
qol:isAlone a owl:DatatypeProperty;
  rdfs:comment "Is the resident alone in the environment?"@en;
  rdfs:domain qol:Resident;
  rdfs:range xsd:boolean.

qol:inRoomSince a owl:DatatypeProperty;
  rdfs:comment "The time when the resident entered his current location"@en;
  rdfs:domain qol:Resident;
  rdfs:range xsd:dateTime;
  qol:hasDurationEquivalent qol:inRoomFor.

qol:inRoomFor a owl:DatatypeProperty;
  rdfs:comment "The duration since the resident entered his current location, in
      seconds"@en;
  rdfs:domain qol:Resident;
  rdfs:range xsd:duration.

qol:doesActivitySince a owl:DatatypeProperty;
  rdfs:comment "The time when the resident supposedly started an activity"@en;
  rdfs:domain qol:Resident;
  rdfs:range xsd:dateTime;
  qol:hasDurationEquivalent qol:doesActivityFor.

qol:doesActivityFor a owl:DatatypeProperty;
  rdfs:comment "The duration since the resident supposedly started an activity, in
      seconds"@en;
  rdfs:domain qol:Resident;
  rdfs:range xsd:duration.

qol:motionMeasured a owl:DatatypeProperty;
  rdfs:comment "Measurement of the number of sensor activations in a given space during
       a given time window."@en;
  rdfs:domain qol:Environment;
  rdfs:range xsd:int.
```

```
qol:hasValue a owl:DatatypeProperty;
  rdfs:comment "value provided by the sensors which dont have fixed state."@en;
  rdfs:domain qol:Sensor.

qol:lastUpdate a owl:DatatypeProperty;
  rdfs:comment "Date and time of the last update of a sensor state."@en;
  rdfs:domain qol:Sensor;
  rdfs:range xsd:dateTime.

qol:lastUsed a owl:DatatypeProperty;
  rdfs:comment "Date and time of the last time an object was used."@en;
  rdfs:domain qol:Object;
  rdfs:range xsd:dateTime;
  qol:hasDurationEquivalent qol:doesActivityFor.

qol:notUsedFor a owl:DatatypeProperty;
  rdfs:comment "Duration since an object was last used."@en;
  rdfs:domain qol:Object;
  rdfs:range xsd:duration.

qol:hasLastUpdate a owl:DatatypeProperty;
  rdfs:comment "Indicate whether the sensor is the last one updated."@en;
  rdfs:domain qol:Sensor;
  rdfs:range xsd:boolean.

qol:indicateLocation a owl:DatatypeProperty;
  rdfs:comment "Whether SensorState indicate the resident location."@en;
  rdfs:domain qol:SensorState;
  rdfs:range xsd:boolean.

qol:indicateUse a owl:DatatypeProperty;
  rdfs:comment "Whether SensorState indicate the use of an object."@en;
  rdfs:domain qol:SensorState;
  rdfs:range xsd:boolean.

qol:getRBConfidenceScore a owl:DatatypeProperty;
  rdfs:comment "Rule-Based confidence score obtained by an activity, given between 0
      and 100."@en;
  rdfs:domain qol:Activity;
  rdfs:range xsd:decimal.
```

Source E.6: Example ABox for the activity recognition, in Notation3

```
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.

@prefix qol:  <load-model#>.
@prefix hom:  <load-home#>.


## GENERATED DATA ##
## -------------- ##

## House ##
hom:johndoe a qol:Resident;
  qol:liveIn hom:house1203001.

hom:france a qol:Environment.
hom:notAtHome a qol:Outside;
  qol:partOf hom:france.
```

```
hom:house1203001 a qol:House;
  qol:partOf hom:france.

hom:salon a qol:Livingroom;
  qol:partOf hom:house1203001.
hom:chambre a qol:Bedroom;
  qol:partOf hom:house1203001.
hom:chambre2 a qol:Bedroom;
  qol:partOf hom:house1203001.
hom:chambre3 a qol:Bedroom;
  qol:partOf hom:house1203001.
hom:toilettes a qol:Toilet;
  qol:partOf hom:house1203001.
hom:cuisine a qol:Kitchen;
  qol:partOf hom:house1203001.
hom:portedentree a qol:Door;
  qol:locatedIn hom:house1203001.

## Sensors ##
hom:pir a qol:SensorType.

hom:a2_on a qol:SensorState;
  qol:indicateLocation true.
hom:a2_off a qol:SensorState.
hom:a2 a qol:Sensor;
  qol:type hom:pir;
  qol:deployedIn hom:cuisine;
  qol:hasPossibleState hom:a2_on;
  qol:hasPossibleState hom:a2_off.

hom:a3_on a qol:SensorState;
  qol:indicateLocation true.
hom:a3_off a qol:SensorState.
hom:a3 a qol:Sensor;
  qol:type hom:pir;
  qol:deployedIn hom:chambre;
  qol:hasPossibleState hom:a3_on;
  qol:hasPossibleState hom:a3_off.

hom:a4_on a qol:SensorState;
  qol:indicateLocation true.
hom:a4_off a qol:SensorState.
hom:a4 a qol:Sensor;
  qol:type hom:pir;
  qol:deployedIn hom:chambre2;
  qol:hasPossibleState hom:a4_on;
  qol:hasPossibleState hom:a4_off.

hom:a5_on a qol:SensorState;
  qol:indicateLocation true.
hom:a5_off a qol:SensorState.
hom:a5 a qol:Sensor;
  qol:type hom:pir;
  qol:deployedIn hom:chambre3;
  qol:hasPossibleState hom:a5_on;
  qol:hasPossibleState hom:a5_off.

hom:a6_on a qol:SensorState;
  qol:indicateLocation true.
hom:a6_off a qol:SensorState.
hom:a6 a qol:Sensor;
  qol:type hom:pir;
  qol:deployedIn hom:toilettes;
  qol:hasPossibleState hom:a6_on;
```

```
    qol:hasPossibleState hom:a6_off.

  hom:a7_on a qol:SensorState;
    qol:indicateLocation true.
89 hom:a7_off a qol:SensorState.
  hom:a7 a qol:Sensor;
    qol:type hom:pir;
    qol:deployedIn hom:salon;
    qol:hasPossibleState hom:a7_on;
94  qol:hasPossibleState hom:a7_off.

  hom:reed a qol:SensorType.

  hom:b4_on a qol:SensorState.
99 hom:b4_off a qol:SensorState;
    qol:indicateUse true.
  hom:b4 a qol:Sensor;
    qol:type hom:reed;
    qol:attachedTo hom:portedentree;
104 qol:hasPossibleState hom:b4_on;
    qol:hasPossibleState hom:b4_off.

  hom:time a qol:SensorType.

109 hom:clock a qol:Sensor;
    qol:type hom:time.


  ## Activities ##
114 hom:getUp a qol:Activity.
  hom:goToilet a qol:Activity.
  hom:hygiene a qol:Activity.
  hom:cookMeal a qol:Activity.
  hom:eatMeal a qol:Activity.
119 hom:clearMeal a qol:Activity.
  hom:occupied a qol:Activity.
  hom:sleep a qol:Activity.
  hom:nap a qol:Activity.
  hom:goOut a qol:Activity.
124 hom:runAway a qol:Deviance.
  hom:comeHome a qol:Activity.
  hom:fall a qol:Deviance.
  hom:socialize a qol:Activity.
```

Source E.7: Rules for the activity recognition, in Notation3

```
  @prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
3 @prefix owl:  <http://www.w3.org/2002/07/owl#>.
  @prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
  @prefix log:  <http://www.w3.org/2000/10/swap/log#>.
  @prefix math:   <http://www.w3.org/2000/10/swap/math#>.
  @prefix func:   <http://www.w3.org/2007/rif-builtin-function#>.
8 @prefix e:    <http://eulersharp.sourceforge.net/2003/03swap/log-rules#>.

  @prefix qol:  <load-model#>.
  @prefix hom:  <load-home#>.
  @prefix ts:   <infer-triplestore#>.
13 @prefix prof: <profile#>.
  @prefix :   <infer-qol#>.
```

```
:getScore a owl:DatatypeProperty;
  rdfs:comment "Indicates that a rule is in favor of recognizing an activity with a
      score."@en;
  rdfs:comment "the score should be from -10 to 10."@en;
  rdfs:domain qol:Activity;
  rdfs:range xsd:int.

:getFinalScore a owl:DatatypeProperty;
  rdfs:comment "Computed total score over all single-rule scores."@en;
  rdfs:domain qol:Activity;
  rdfs:range xsd:int.



## RULES ##
## ----- ##

## tracks resident location in house [persistent]
{?se qol:hasCurrentState ?st. ?se qol:hasLastUpdate true. ?st qol:indicateLocation true
    . ?se qol:deployedIn ?r. ?r qol:partOf ?h. ?u qol:liveIn ?h. ?u qol:detectedIn ?r2.
     ?r log:notEqualTo ?r2. ?se qol:lastUpdate ?t} => {?u qol:detectedIn ?r. ?u qol:
    cameFrom ?r2. ?u qol:inRoomSince ?t. ts:n3store ts:update {?u qol:detectedIn ?r. ?u
     qol:cameFrom ?r2. ?u qol:inRoomSince ?t}}.

## detects if resident goes outside [persistent]
{?u qol:liveIn ?h. ?h qol:motionMeasured 0. ?d a qol:Door. ?d qol:notUsedFor ?du. ?du
    math:lessThan 300. ?u qol:detectedIn ?r. ?o a qol:Outside. ?d qol:lastUsed ?t} =>
    {?u qol:detectedIn ?o. ?u qol:cameFrom ?r. ?u qol:inRoomSince ?t. ts:n3store ts:
    update {?u qol:detectedIn ?o. ?u qol:cameFrom ?r. ?u qol:inRoomSince ?t}}.

## tracks usage of objects [live + persistent]
{?se qol:hasCurrentState ?st. ?st qol:indicateUse true. ?se qol:attachedTo ?o. ?o qol:
    locatedIn ?h. ?u qol:liveIn ?h. ?se qol:lastUpdate ?t} => {?u qol:useNow ?o}.
{?se qol:hasCurrentState ?st. ?se qol:hasLastUpdate true. ?se qol:attachedTo ?o. ?se
    qol:lastUpdate ?t} => {?o qol:lastUsed ?t. ts:n3store ts:update{?o qol:lastUsed ?t
    }}.

## infer durations [live]
{?since qol:hasDurationEquivalent ?for. ?x ?since ?start. hom:clock qol:hasValue ?now.
    (?now ?start) math:difference ?duration} => {?x ?for ?duration}.


## ACTIVITY RECOGNITION (SCORE SYSTEM) ##
## ---------------------------------- ##

## get up (+ max duration)
{?u qol:believedToDo hom:sleep. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!prof:
    minSleepDuration. ?u qol:detectedIn ?r. ?r qol:motionMeasured ?m. ?m math:
    notLessThan ?u!prof:maxSleepMotion} => {hom:getUp :getScore 8}.
{?u qol:believedToDo hom:getUp. ?u qol:doesActivityFor ?d. ?d math:lessThan ?u!prof:
    getupDuration} => {hom:getUp :getScore 5}.

## go to the toilet (+ max duration)
{?u qol:detectedIn ?r. ?r a qol:Toilet. ?u qol:inRoomFor ?d. ?d math:lessThan ?u!prof:
    maxToiletDuration} => {hom:goToilet :getScore 7}.

## hygiene activities (+ max duration)
{?u qol:detectedIn ?r. ?r a qol:Bathroom. ?u qol:inRoomFor ?d. ?d math:lessThan ?u!prof
    :maxHygieneDuration} => {hom:hygiene :getScore 9}.

## cook
##{false} => {hom:cookMeal :getScore 0}. >> too coarse data

## have meal
```

```
##{false} => {hom:eatMeal :getScore 0}. >> too coarse data

## clear table and wash dishes
##{false} => {hom:clearMeal :getScore 0}. >> too coarse data

## occupied (+ max duration)
{?u qol:liveIn ?h. ?h qol:motionMeasured ?m. ?m math:notLessThan ?u!prof:
    minOccupiedMotion} => {hom:occupied :getScore 2}.
{?u qol:believedToDo hom:occupied. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!
    prof:maxOccupiedDuration} => {hom:occupied :getScore -2}.

## sleep (+ max duration)
{?u qol:detectedIn ?r. ?r a qol:Bedroom. ?u qol:inRoomFor ?d. ?d math:notLessThan ?u!
    prof:minSleepInitiation. ?r qol:motionMeasured ?m. ?m math:lessThan ?u!prof:
    maxSleepMotion} => {hom:sleep :getScore 6}.
{?u qol:believedToDo hom:sleep. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!prof:
    maxSleepDuration} => {hom:sleep :getScore -4}.

## take a nap (+ max duration)
{?u qol:detectedIn ?r. ?r a qol:Livingroom. ?u qol:inRoomFor ?d. ?d math:notLessThan
    600. ?r qol:motionMeasured ?m. ?m math:lessThan ?u!prof:maxNapMotion} => {hom:nap :
    getScore 6}.
{?u qol:believedToDo hom:nap. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!prof:
    maxNapDuration} => {hom:nap :getScore -4}.

## go out of home (+ max duration, see run away rule)
{?u qol:detectedIn ?o. ?o a qol:Outside} => {hom:goOut :getScore 6}.

## come home (+ max duration)
{?u qol:believedToDo hom:goOut. ?u qol:useNow ?o. ?o a qol:Door} => {hom:comeHome :
    getScore 5}.
{?u qol:believedToDo hom:comeHome. ?u qol:doesActivityFor ?d. ?d math:lessThan ?u!prof:
    comehomeDuration} => {hom:comeHome :getScore 5}.

## run away
{?u qol:useNow ?d. ?d a qol:Door. hom:clock qol:hasValue ?t. ?t func:hours-from-
    dateTime ?h. ?h math:notLessThan ?u!prof:outTooLate. ?h math:lessThan ?u!prof:
    outTooEarly} => {hom:runAway :getScore 9}.
{?u qol:detectedIn ?o. ?o a qol:Outside. ?u qol:inRoomFor ?d. ?d math:notLessThan ?u!
    prof:outTooLong} => {hom:runAway :getScore 9}.

## fall
{?u qol:believedToDo hom:nothing. ?u qol:doesActivityFor ?d. ?d math:notLessThan ?u!
    prof:maxInactiveDuration} => {hom:fall :getScore 8}.

## meet people at home
{?u qol:isAlone false} => {hom:socialize :getScore 8}.


## must add scores and give confidence, if nothing up a given threshold then believe in
     hom:nothing!
{?a :getScore ?x. ?SCOPE e:findall (?sc {?a :getScore ?sc} ?list). ?list math:sum ?
    total} => {?a :getFinalScore ?total}.
{?SCOPE e:findall (?sc {?a :getScore ?sc} ?list). ?list math:sum ?grandtotal. ?
    grandtotal math:lessThan 1} => {hom:nothing :getScore 10}.
{?SCOPE e:findall (?sc {?a :getFinalScore ?sc} ?list). ?list math:sum ?grandtotal. ?
    grandtotal math:notLessThan 0.1. ?a0 :getFinalScore ?fs. (?fs ?grandtotal) math:
    quotient ?cs} => {?a0 qol:getRBConfidenceScore ?cs}.

## infer most probable activity
{?SCOPE e:findall (?rbc {?a qol:getRBConfidenceScore ?rbc} ?list). ?list e:max ?maxrbc.
     ?a qol:getRBConfidenceScore ?maxrbc. ?u qol:liveIn ?h. hom:clock qol:hasValue ?now
    } => {?u qol:believedToDo ?a. ?u qol:doesActivitySince ?now. ts:n3store ts:update
    {?u qol:believedToDo ?a. ?u qol:doesActivitySince ?now}}.
```

Emancipate yourself from mental slavery,
None but ourselves can free our minds.
— Nesta Robert Marley, 1945–1981

F Discussion: Business Models for iAAL

"Despite the overwhelming technology advancements in the recent years, the diffusion of smart home products and services are still far from common reality, and a large-scale commercialization cannot be observed." This is the conclusion reached in 2011 by Solaimani et al. after an extensive review and qualitative meta-analysis of the smart home landscape [169]. The Service, Technology, Organisation, Finance (STOF) model provides a "holistic" view on business models organized around four interrelated perspectives [170]. *Service* concerns the value proposition and customers, *Technology* is related to ICT, applications and platforms, *Organization* has to do with the actors, resources and value network, while *Finance* involves investments, cost and revenues. One of the reasons raised by Solaimani et al. for the status of smart home services is the disequilibrium in the number of *Technology*-focused publications against publications in the other aspects of the STOF model.

In this discussion chapter, I want to think in a novel way about pushing AAL technologies to the market in order to increase the impact on society. I wish that some of the ideas I propose below, maybe a bit too "creative", certainly too immature, could spark some new initiatives in the marketplace. The challenge I try to raise is to bring **independent Ambient Assisted Living (iAAL)** to society, approaching it from the point of view of the possible business models to ensure profitability and acceptance, while taking into account the peculiarities of stakeholders in different regions, e.g. Europe and Asia. Indeed, if big institutions such as insurance companies may trigger iAAL in Europe, it would probably be more of a family-driven evolution in Asian countries (see section 3.1.2).

F.1 Smart Home in a Box

Smart home in a box is a term that was first coined by Sumi Helal from University of Florida as part of the ICOST conference. I find it to be an excellent tangible way to deliver smart home products if a reduced hardware complexity can be achieved. As such, it is extremely compatible with our bottom-up approach and stripped-down vision of AAL solutions. I see the box itself purely as a gateway to push sensor data to a cloud-based storage where it can be processed by server-side applications. Hence the purchasing of the box should include the necessary subscriptions and configuration tools to access cloud services. Moreover, I believe the end-user services should be designed as web applications for which interfaces (e.g. a dashboard) can be implemented for any internet-connected terminal. The priority should be given to services that address health concerns in the lifestyle of elders, while raising implication and visibility for families, and professional caregivers. Below, I try to gather some technical requirements that emerge from my vision of a smart home in a box.

F.1.1 A Box as Gateway in Each Home

The box's main processing element can be built around a credit-card-sized linux machine such as the BeagleBone or the Raspberry Pi. We identify in the following the box to this linux machine. It must

integrate wireless communication modules for one or a few protocols used by sensors and activators (e.g. ZigBee, ANT, X10). It should act as gateway for the data between the local hardware entities on binary protocols and the cloud storage and services over a REST protocol (see section 8.1.3). Both the binary and REST communication protocols must be enhanced along efficiency and security aspects. On the software side, we can leverage the small processing power to perform light pre-processing of the data locally if needed. I imagine that the box could centralize all the configuration processes needed to setup an environment. It could hold account information and handle the various authentication layers.

F.1.2 Web Browser as the Main Interface

As we cannot imagine a user logging in to a command line interfaced linux to access the settings, I believe a lightweight web server can be installed on the box and the management features provided on the local network as web service. These features can then be accessed via any web browser on another machine, or on a dedicated app on mobile platforms. Dynamic management interfaces can be implemented using AJAX for example. With such interfaces, we can also leverage the semantic plug & play mechanism for the dynamic configuration of sensors and actuators (see section 8.3.5). From the implementation point of view, the idea would be something similar, although at a different configuration level, to the kickstarter project called *Twine* from design firm *Supermechanical*, based in Cambridge, Massachusetts (`http://supermechanical.com`). A current status report can also be implemented summarizing in a graphical way the real-time status of each of the sensor/actuator configured so far.

F.1.3 Server-Side Processing and Applications

The only way to scale things up while ensuring a possible maintenance and support for numerous homes is to centralize the data storage and processing on servers. This ensures the safety of crucial data against hardware failure or environmental problems. It makes it easier to share data driven services with family members. It ensures that enough processing power is available for the data analysis while keeping the cost for individual homes low. The cost of processing is instead transferred towards the service provider who is able to realize economies of scale. Server-side processing also reduces the time needed for upgrade cycles on the application side as everything is centralized. Thus every user automatically uses the latest available version, which reduces the support effort as well. We can imagine that each box's configuration would be saved on server as well, hence allowing for remote debugging and support services. Finally, dedicated applications could be implemented to analyse the consistency of the data coming from each sensor, enabling early alerts when a sensor becomes faulty.

F.1.4 Sensors and Actuators

What a smart home in a box needs is a handful of "killer sensors and actuators" to start-up with, while making it a breeze to plug new hardware components depending on one's needs and wishes. Here, no trade-off can be made: no level of wiring at all is acceptable from the end-user point of view, which means most of the components are going to be battery-powered. Software aside, adding or changing sensors and actuators should be as simple as hanging a frame or changing a light bulb. On the sensing side, the basic sensor needed to monitor the amount of activity of a person is the motion sensor (passive infra-red). It should be running months on batteries and attached to the ceiling of each room following some placement guidelines. A good complementarity to the very general motion sensor can be found in the shake sensor (accelerometer and gyroscope), as we used in Peacehaven (see section 9.3). The shake sensor, which can possibly be miniaturized greatly, must also run months on batteries and is meant for integration on any object to detect particular events or activities. For instance, we have used it on soap dispensers, pipes, water jugs, medicine boxes and it can also be attached to doors, phone handsets, etc. This part of the system would be similar to *Lively*, a new service available in the United States only (`http://www.mylively.com`). On the actuator side, a wide range of home control applications

can be achieved simply by network-controlled electricity plugs and/or light bulbs. Additionally we have observed the value of speakers as a rich and pervasive interaction modality. Since speakers, and similarly microphones, are interesting components requiring more power, I suggest to integrate into a single component a microphone for ambient sound analysis (level in decibel only at first), a speaker for pervasive interaction, and a controlled power plug which enables both the continuous power supply to the microphone and speaker and some home control applications. A research team at I2R in Singapore is coming up with such a device, omitting the controlled power plug part, and integrating advanced sound event classification techniques. The device is called soundeye (`http://www.sound-eye.com`).

Taking into account the power consumption issue, a major challenge resides in the power management of the battery-powered components. Firstly, it is known that most of the power is usually consumed for wireless communication. Hence, the protocol used must be adapted to transfer only useful states and make the messages overhead minimal. A study must be planned to choose between adding redundancy to the signal or using acknowledgement messages. We can also look into sleep control techniques for the devices. Finally, a study could compare the different wireless network topologies from the ease of use, efficiency and power consumption aspects.

A significant add-on to the sensors mentioned above would be to embrace the numerous vital signs monitoring devices such as bed pressure mats, blood pressure monitors, connected weighing machines. In this aspect, the most efficient way would be to make our box interoperable with other systems through a shared standard. Continua Health Alliance is a pioneer in establishing industry standards for connected health technologies such as smartphones, gateways and remote monitoring devices. I believe that following the guidelines and standard proposed by Continua would be the simplest way to leverage and connect to the work of thousands around the world.

F.2 The SmartStore Project

Mounir Mokhtari, my thesis supervisor, was saying back in 2001 that "technology is beneficial only when shared by all". Thus, let us consider that the smart home in a box idea receives a good acceptance from the stakeholders. How do we drive its adoption among elder people? What sales channels are most suitable? How do we build a community of users? In a nutshell, the SmartStore idea consists in opening a store where senior citizens may buy iAAL products on a component basis, where components can be integrated at home with little support. Components should tackle specific needs and remain lowly featured, thus keeping the technology understandable for elders and empowering them with tools towards their own independence. The components' integration in a home gateway increases the value by creating collaborative system behaviours where a software service can be added like a plugin to provide a new specific assistance based on the available knowledge of the user's context. The store would also feature a workshop area where customers can discover products, get some hands-on support, share their experience, and thus, get socially involved. The SmartStore is the result of a creativity group project at ICOST 2011 summer school and is described in details below.

F.2.1 Motivation

Our motivation towards the creation of a SmartStore is articulated around two main axes: getting smart home products out of research laboratories, and working on the perception of iAAL technology as an empowering, trendy, and non-stigmatizing movement. For the first aspect, the SmartStore is a way to accelerate the commercialization process of iAAL products by identifying projects with business opportunities and offering them an easy access to a community of users, thus skipping the need for investors' research, marketing and advertising. To increase the technology acceptance, I believe in focusing on answering specific needs to empower elders. As explained previously, products and systems would be sold in the store on a component basis, each component being integrated through a home gateway in a modular way, thus supporting iterative deployments natively. The integration is seen to

be as automatic as possible in order to provide optimal ease of use for the elderly. Each component having a specific role to play, the technology does not seem too complex and the elderly should feel able to take control of their own independent living. The idea is really to pass the choice and the control of the system over to the end-users, which would naturally feel less invasive and more rewarding [6]. By proposing specific components for specific needs, elderly should better foresee the impact of the technology in their daily life and understand what they are paying for. Coupled with the relative affordability of the technology, we target at enhancing the acceptance of iAAL. I explain in the next section what kind of needs the components could correspond to.

F.2.2 Concept Development

The Components

I have mentioned that components should integrate in a collaborative and modular way, but what exactly are "components"? I explain in this section what components can be, what needs they can fulfil, at which level of intervention, etc. First of all, components can be hardware or software. A hardware component could be a sensor attached with a microchip for embedded processing and a wireless communication module, or perhaps a modality of ambient interaction like a small light source changing colour depending on wireless inputs. Software components would provide back-end or front-end services. A back-end service could for example gather knowledge about the user's context, infer more from it and provide a high-level representation of this knowledge to other services. One of the other services could be a front-end service detecting dangerous situations and reacting by turning off the gas automatically, turning that small light red and sending a reminder to the TV or a warning to a caregiver via SMS. These components can be adapted to different level of cognitive impairment, by sensing and recognizing more fine-grain context information, or providing a more explicit user interface with video instead of coloured light.

We can also design components for different fields of applications: assistance with daily activities, help to forgetful people, health assessment, social link at home, leisure, and this is not comprehensive. Figure F.1 illustrates some example products of each of these application fields. The CookStop system in Figure F.1(a) assures safety of living while performing daily activities [171], some simple ambient reminder systems like in Figure F.1(b) were developed as part of this doctoral work, the RFID slippers in Figure F.1(c) enables heath assessment [172], Mazadoo project in Figure F.1(d) brings Facebook to the television and is designed especially for elders [173], finally testimonies about elders playing Nintendo Wii games like in Figure F.1(e) are getting more frequent lately. We can also imagine components to control the system, like Bell Labs' card board from the Casensa project [174] in Figure F.2(a), or even to personify the system with avatars, like Karrotz [175] in Figure F.2(b).

(a) CookStop (b) TV reminder (c) RFID slippers (d) Mazadoo (e) Wii games

Figure F.1: Example Products for Diverse Applications

With the system set up and the components running, we can imagine adding tiny applications that would make use of existing hardware and information from other software components to provide additional services of miscellaneous kinds. As we stand today in a "there is an app for that" world

(a) Casensa project

(b) Nabaztag RFID rabbit

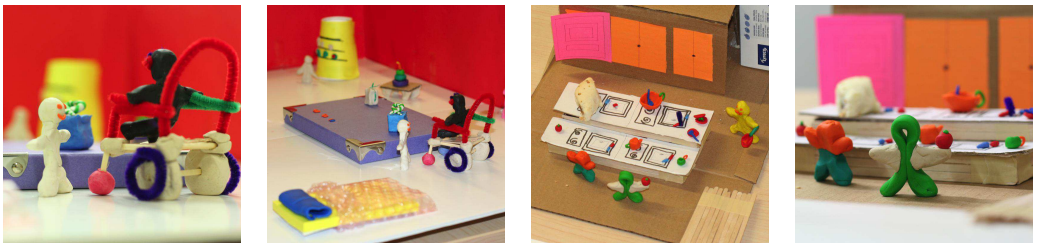
Figure F.2: Products to Control or Personify the System

[176], we have the vision of a sort of AppStore for iAAL environments where free or few dollars apps would be easily accessible to add value to our solution. We try to make sense here of the fact that people do not buy technology for the sake of technology but for the services that come along with it. An iAAL framework offering an AppStore with apps fitted for every task, every need and every person would definitely gather some interest.

The Store

Figure F.3: Welcome to the SmartStore

The store as we imagine it, and as we modelled it (see Figure F.3 & F.4), follows the Apple Store model, with a bright and spacious setting. It is a place to discover the technology, get some hands-on experience and have face-to-face exchange with knowledgeable consultants. We want people to receive good assistance when buying their iAAL components and go home with clear ideas about their purchase. Figure F.4(a) shows an elder in a wheelchair in a conversation with a sales consultant in the store area of Figure F.4(b). Moreover, we wish to introduce an educational aspect in our store in order to enhance the acceptance of components. Therefore we designed the store coupled with a workshop area as represented in Figure F.4(c). This area aims to be a social, collaborative and educational space where elders can use the technology together, with assistance when necessary and where free thematic workshops are held for a few weeks to promote best practices with the technology. For example, the Figure F.4(d) shows elders collaboratively cooking in a smart kitchen, assisted by iAAL components. We work here on making the store a social place to meet users and receive assistance, thus increasing the acceptance of iAAL and enhancing social link among elders.



(a) (b) (c) (d)

Figure F.4: Pictures of the Store's Model

F.2.3 Self-Review for the Summer School

As part of the summer school, the team was supposed to propose a crazy idea like this one, as well as a short review of its feasibility. The main idea behind this project is to provide something different from what we have seen throughout the scientific literature and business news. It resonates with the analysis of Solaimani et al. by proposing a tangible solution with an almost immediate impact, as opposed to the longer-term research efforts that are more common in our field. Moreover, there is no contradiction with long-term research and the SmartStore can even support it.

Business Factors

The SmartStore project is clearly business oriented; it is about moving from exploration to exploitation of the technology [169]. No company has really made its name in the AAL market yet; there is an obvious opportunity. Risks must not be overlooked but the market is available, and investors making the first move along a good vision might become market leaders within ten years. Our team had no background legitimating an opinion concerning the viability of the project, but we feel that it is a motivating, eye-catching idea, which might get the attention of potential investors.

Technology Factors

Though it is energizing, our project is also limited by its ambition. Is technology ready for this? The solution might be more complex than how we picture it, especially in term of integration. It is however close to the focus in the current state of the art and we believe that starting on a new research phase of two to three years with a freshly envisioned mind-set would ready the team and technology for spin-off. One of our technological strength is the capitalization of the state of the art, coupled with an iterative approach creating an avenue for future advancements.

Human Factors in the Ageing Process

We see our solution as potentially having a high societal impact within a reasonable time. For this, we take care of the acceptance of iAAL technology. We strongly believe that passing the choice and the control of the system over to the elderly by proposing understandable iAAL would be rewarding for them and help in this matter. We also work on creating a community, and a space for this community to meet, discover and receive advices. We have to be careful with the amount of technology available being overwhelming even for people with engineering background, even more so for the elderly. Therefore, a real work is needed in collaboration with designers and marketers to plan how to bring iAAL to the market. We must provide non-technical description of products using simple language and graphics, use appropriate accessibility information and visualization [177]; finally the design must be kept non-stigmatizing.

Politeness is the poison of collaboration.

— Edwin Land, 1909–1991

Research & Development in Singapore

G.1 Introduction

In this appendix, an overview of the research context in Singapore is presented. The content was adapted from the report by Christelle Gervasoni and Walid Benzarti [178] for the scientific section at the French Embassy in Singapore.

Singapore is a small country with limited natural resources and an ever-growing energy consumption due to the rapid urban development. The country is therefore facing difficulties in its development, but is thinking about adapting itself and improving its urban organisation for a better expansion rather than accepting the limits of this expansion. Hence, it is crucial for Singapore to seek and adopt modern technologies to ensure the durability of its urban development. With this vision, big investments are made on innovation, research and education, and the reliance on foreign talents is also important. This approach has enabled institutes of higher education and research among the world's best, as well as internationally renowned scientists, to settle in Singapore. The presence of these institutes and scientists also consolidates the local research capabilities. It favors the emergence of innovating solutions by leveraging their strong and complementary expertise, especially in the domains of transport, energy and urban technologies.

G.2 Singapore's Research Organisation

G.2.1 Hierarchy of Singapore's Research Institutions

In absence of a Ministry of Research, Singapore's research is under the administrative co-supervision of the Ministry of Trade & Industry (MTI) and the Ministry of Education (MOE). The funding of research is handled mainly through the National Research Foundation (NRF), the MTI and the MOE; and sometimes through the Ministry of Health (MOH) and the Ministry of Defence (MINDEF). The orientation of Singapore's research is defined in the five-year "Science & Technology" plans of the Research, Innovation and Enterprise Council (RIEC). The last plan covers the 2011–2015 time period. The intermediary funding organs — such as the Economic Development Board (EDB), the Agency for Science, Technology and Research (A*STAR), the National Medical Research Council (NMRC) and the Academic Research Fund (AcRF) — are making the link with the operators of research like universities, research institutes of A*STAR and enterprises. The NRF is placed at the same decisional level as the MOE and the MTI. A summary is given in Figure G.1.

G.2.2 The National Research Foundation (NRF)

The NRF was created on the 1st of January 2006, as a department to the office of the Prime Minister. A S$5bn (€3.1bn) budget was assigned to the NRF for 5 years with two main objectives:

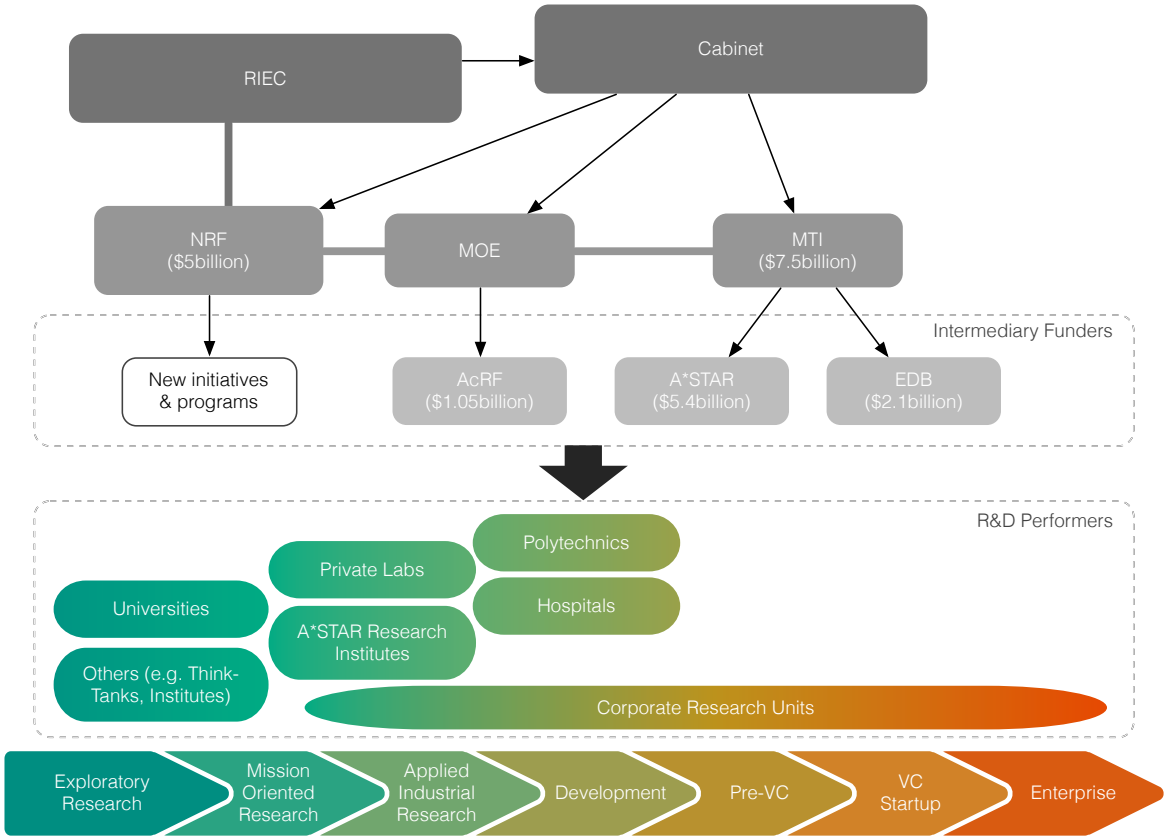


Figure G.1: Singapore's Research Organisation & Budget for 2006-2010

- transform Singapore into a Research & Development (R&D) hub in order to support an economy focused on innovation, entrepreneurship and knowledge,
- transform Singapore into a pole of attraction for scientific excellence and innovation.

The foundation sets the national R&D orientation by developing several policies, road-maps and strategies for research, innovation and entrepreneurship. It builds the country's research capabilities through the education of local talents and the attraction of renowned foreign scientists. It funds ambitious research programs found to be strategic for long-term, while defining global research axes at the national level and coordinating the different national agencies and research operators.

G.3 Orientation of Singapore's Research

G.3.1 A Research Strategy Defined in Five-Year Plans

Since 1991, four "Science & Technology" plans have been developed by Singapore's government in order to provide a common direction to all the actors of the research of the country focusing their efforts on common objectives that have been given the highest priority at a national level. In the early 2000s, the challenges were to provide more resources to R&D, define which domains of research would represent major economic stakes, stimulate private R&D, find a balance between fundamental and applied research, and strengthen the bounds between research players and enterprises.

G.3.2 Research Priorities at the 2015 Horizon

The fifth "Science & Technology" plan at horizon 2015 was launched in 2011. It is designed to foster knowledge creation, and to develop innovation and corporate spirit. Its budget is S$16.1bn (€10bn), which is an increase of 20% compared to the previous five-year plan and its S$13.6bn (€8.5bn) budget.

This new plan continues to support heavily the research activity of Singapore, trying to encourage private investments in R&D and to position research towards new markets. The awarding of research funds is getting more and more competitive, aiming at galvanizing innovative, collaborative and multidisciplinary research efforts between all research players in Singapore. The development of public-private research partnership is for example stimulated. New funding schemes are destined to technological transfer, to incite switching from fundamental research to commercialization. In parallel, Singapore pursues its efforts in welcoming foreign scientific talents and grants are available to facilitate the development of young scientific talents, locals or foreigners. The RIEC has also launched the "National Innovation Challenge" to develop solutions to national stakes such as the improvement of energetic management towards the island's autonomy, a sustainable development and a controlled urbanization.

This funding effort aims at supporting prosperity through the creation of high added value employment for Singaporeans. Innovation and entrepreneurship maintain the competitiveness of companies, opening doors to new markets and allowing the economy to grow through its high intellectual capital. The previous five-year plan targeted a dedication of 3% of the country's Gross Domestic Product (GDP) to R&D by 2010. The current plan is targeting at 3.5% by 2015, backed up by the growth in private R&D activities.

Bibliography

[1] United Nations, *World Population Ageing: 1950-2050*. Population Division, Department of Economic and Social Affairs, United Nations New York, NY, USA (http://www.un.org/esa/population/publications/worldageing19502050/), 2010.

[2] M. Prince and J. Jackson, *World Alzheimer Report 2009*. Alzheimer's Disease International, 2009.

[3] United Nations, *World Population Prospects: The 2010 Revision*. Population Division, Department of Economic and Social Affairs, United Nations New York, NY, USA (http://esa.un.org/unpd/wpp/index.htm), 2010.

[4] R. Cliquet and M. Nizamuddin, *Population Ageing: Challenges for Policies and Programmes in Developed and Developing Countries*. United Nations Population Fund: Population and Family Study Centre, 1999.

[5] K. Kinsella and V. A. Velkoff, *An Aging World: 2001: International Population Reports*. US Cencus Bureau, Economic and Statistics Administration, Department of Commerce, 2001.

[6] S. Routhier, "Aging health and aging needs," in *ICOST 2011 Summer School Talks*, 2011.

[7] K. W. Schaie, "What can we learn from longitudinal studies of adult development?," *Research in human development*, vol. 2, no. 3, pp. 133–158, 2005.

[8] T. A. Salthouse *et al.*, "The processing-speed theory of adult age differences in cognition," *Psychological review*, vol. 103, no. 3, pp. 403–427, 1996.

[9] U. Lindenberger, P. B. Baltes, *et al.*, "Sensory functioning and intelligence in old age: A strong connection," *Psychology and aging*, vol. 9, pp. 339–339, 1994.

[10] M. L. Johnson, V. L. Bengtson, and P. G. Coleman, *The Cambridge handbook of age and ageing*. Cambridge University Press, 2005.

[11] W. C. Mann, "The aging population and its needs," *Pervasive Computing, IEEE*, vol. 3, no. 2, pp. 12–14, 2004.

[12] A. Wimo and M. J. Prince, *World Alzheimer Report 2010: the global economic impact of dementia*. Alzheimer's Disease International, 2010.

[13] J. Diamond, *A report on Alzheimer's disease and current research*. Alzheimer Society of Canada, 2006.

[14] B. Reisberg, S. H. Ferris, M. J. de Leon, and T. Crook, "The global deterioration scale for assessment of primary degenerative dementia.," *The American journal of psychiatry*, 1982.

[15] R. Brookmeyer, E. Johnson, K. Ziegler-Graham, and H. M. Arrighi, "Forecasting the global burden of alzheimers disease," *Alzheimer's and Dementia*, vol. 3, no. 3, pp. 186–191, 2007.

[16] "Gerontechnology." Wikipedia, the free encyclopedia (http://en.wikipedia.org/wiki/Gerontechnology).

[17] M. Mokhtari, H. Aloulou, T. Tiberghien, J. Biswas, D. Racoceanu, and P. Yap, "New trends to support independence in persons with mild dementia – a mini-review," in *International Journal of Experimental, Clinical, Behavioural, Regenerative and Technological Gerontology*, vol. 58, pp. 554–563, Karger Publishers, 2012.

[18] T. Harrington and M. Harrington, "Gerontechnology. why and how.," tech. rep., Shaker, Maastricht, 2000.

[19] C. McCreadie and A. Tinker, "The acceptability of assistive technology to older people," *Ageing and Society*, vol. 25, no. 01, pp. 91–110, 2005.

[20] "Ambient assisted living joint programme: Ict for ageing well." http://www.aal-europe.eu.

[21] L. C. for Aging Services Technologies (CAST), "Imagine: the future of ageing (video)." http://www.leadingage.org/Imagine-the-Future-of-Aging.aspx, 2010.

[22] TechRepublic and ZDNet, "The executive's guide to the internet of things." http://www.zdnet.com/the-executives-guide-to-the-internet-of-things-free-ebook-7000009589/.

[23] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[24] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, "Connecting the physical world with pervasive networks," *Pervasive Computing, IEEE*, vol. 1, no. 1, pp. 59–69, 2002.

[25] H. Ishii and B. Ullmer, "Tangible bits: towards seamless interfaces between people, bits and atoms," in *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*, pp. 234–241, ACM, 1997.

[26] M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Personal communications*, vol. 8, no. 4, pp. 10–17, 2001.

[27] M. Hariz, *Mechanism for handling user interface plasticity in ambient assistive living*. PhD thesis, Institut National des Télécommunications, France, 2009.

[28] D. Thevenin and J. Coutaz, "Plasticity of user interfaces: Framework and research agenda," in *Human-computer Interaction, INTERACT'99: IFIP TC. 13 International Conference on Human-Computer Interaction*, p. 110, IOS Press, 1999.

[29] J. Viterbo, L. Mazuel, Y. Charif, M. Endler, N. Sabouret, K. Breitman, A. E. F. Seghrouchni, and J.-P. Briot, "Managing distributed and heterogeneous context for ambient intelligence," *Context-Aware Self Managing Systems, CRC Studies in Informatics Series*, pp. 79–128, 2010.

[30] J. Lindenberg, W. Pasman, K. Kranenborg, J. Stegeman, and M. A. Neerincx, "Improving service matching and selection in ubiquitous computing environments: a user study," *Personal and Ubiquitous Computing*, vol. 11, no. 1, pp. 59–68, 2007.

[31] A. K. Dey, "Understanding and using context," *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[32] G. Chen, D. Kotz, *et al.*, "A survey of context-aware mobile computing research," tech. rep., Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[33] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*, pp. 304–307, Springer, 1999.

[34] G. J. Jones, "Challenges and opportunities of context-aware information access," in *Ubiquitous Data Management, 2005. UDM 2005. International Workshop on*, pp. 53–60, IEEE, 2005.

[35] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *First International Workshop on Advanced Context Modelling, Reasoning and Management*, 2004.

[36] C. Rougier, E. Auvinet, J. Rousseau, M. Mignotte, and J. Meunier, "Fall detection from depth map video sequences," in *Toward Useful Services for Elderly and People with Disabilities*, vol. 6719 of *Lecture Notes in Computer Science*, pp. 121–128, Springer, 2011.

[37] A. Sixsmith and N. Johnson, "A smart sensor to detect the falls of the elderly," *Pervasive Computing, IEEE*, vol. 3, no. 2, pp. 42–47, 2004.

[38] M. Alwan, P. J. Rajendran, S. Kell, D. Mack, S. Dalal, M. Wolfe, and R. Felder, "A smart and passive floor-vibration based fall detector for elderly," in *Information and Communication Technologies, 2006. ICTTA'06. 2nd*, vol. 1, pp. 1003–1007, IEEE, 2006.

[39] M. Floeck, L. Litz, and T. Rodner, "An ambient approach to emergency detection based on location tracking," in *Toward Useful Services for Elderly and People with Disabilities*, vol. 6719 of *Lecture Notes in Computer Science*, pp. 296–302, Springer, 2011.

[40] M. A. Stelios, A. D. Nick, M. T. Effie, K. M. Dimitris, and S. C. Thomopoulos, "An indoor localization platform for ambient assisted living using uwb," in *Proceedings of the 6th international conference on advances in mobile computing and multimedia*, pp. 178–182, ACM, 2008.

[41] "Jawbone up fitness bracelet." `https://jawbone.com/up`.

[42] P. Rashidi and D. J. Cook, "Keeping the resident in the loop: Adapting the smart home to the user," *IEEE Transactions on Systems, Man and Cybernetics. Part A: Systems and Humans*, vol. 39, no. 5, pp. 949–959, 2009.

[43] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen, "The gator tech smart house: A programmable pervasive space," *Computer*, vol. 38, no. 3, pp. 50–60, 2005.

[44] F. Doctor, H. Hagras, and V. Callaghan, "A fuzzy embedded agent-based approach for realizing ambient intelligence in intelligent inhabited environments," *IEEE Transactions on Systems, Man and Cybernetics. Part A: Systems and Humans*, vol. 35, no. 1, pp. 55–65, 2005.

[45] G. D. Abowd and E. D. Mynatt, "Designing for the human experience in smart environments," *Smart environments: technologies, protocols, and applications*, pp. 151–174, 2005.

[46] H. Aloulou, M. Mokhtari, T. Tiberghien, J. Biswas, C. Phua, J. H. K. Lin, and P. Yap, "Deployment of assistive living technology in a nursing home environment: methods and lessons learned," *BMC Medical Informatics and Decision Making*, vol. 13, no. 1, p. 42, 2013.

[47] Z. Jiaqi, L. V. Yen, J. Biswas, M. Mokhtari, T. Tiberghien, H. Aloulou, *et al.*, "Context-aware reasoning engine with high level knowledge for smart home," in *1st International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*, pp. 292–297, 2011.

[48] L. Chen and C. Nugent, "Ontology-based activity recognition in intelligent pervasive environments," *International Journal of Web Information Systems*, vol. 5, no. 4, pp. 410–430, 2009.

[49] T. Tiberghien, M. Mokhtari, H. Aloulou, and J. Biswas, "Semantic reasoning in context-aware assistive environments to support ageing with dementia," in *Proceedings of the 11th International Semantic Web Conference (ISWC)*, pp. 212–227, Springer, 2012.

[50] M. Ros, M. Delgado, A. Vila, H. Hagras, and A. Bilgin, "A fuzzy logic approach for learning daily human activities in an ambient intelligent environment," in *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8, IEEE, 2012.

[51] N. Roy, A. Roy, and S. K. Das, "Context-aware resource management in multi-inhabitant smart homes a nash h-learning based approach," in *Fourth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 158–169, IEEE, 2006.

[52] A. Tolstikov, X. Hong, J. Biswas, C. Nugent, L. Chen, and G. Parente, "Comparison of fusion methods based on dst and dbn in human activity recognition," *Journal of Control Theory and Applications*, vol. 9, no. 1, pp. 18–27, 2011.

[53] P. Olivier, G. Xu, A. Monk, and J. Hoey, "Ambient kitchen: designing situated services using a high fidelity prototyping environment," in *Proceedings of the 2nd International Conference on Pervasive Technologies Related to Assistive Environments*, p. 47, ACM, 2009.

[54] V. Jiménez-Mixco, J. L. Villalar González, A. Arca, M. F. Cabrera-Umpierrez, M. T. Arredondo, P. Manchado, and M. García-Robledo, "Application of virtual reality technologies in rapid development and assessment of ambient assisted living environments," in *Proceedings of the 1st ACM SIGMM international workshop on Media studies and implementations that help improving access to disabled users*, pp. 7–12, ACM, 2009.

[55] C. Orwat, A. Graefe, and T. Faulwasser, "Towards pervasive computing in health care–a literature review," *BMC Medical Informatics and Decision Making*, vol. 8, no. 1, p. 26, 2008.

[56] R. Orpwood, C. Gibbs, T. Adlam, R. Faulkner, and D. Meegahawatte, "The design of smart homes for people with dementia: user-interface aspects," *Universal Access in the Information Society*, vol. 4, no. 2, pp. 156–164, 2005.

[57] J. Chin, V. Callaghan, and G. Clarke, "Soft-appliances: A vision for user created networked appliances in digital homes," *Journal of Ambient Intelligence and Smart Environments*, vol. 1, no. 1, pp. 69–75, 2009.

[58] V. Callaghan, G. Clarke, and J. Chin, "Some socio-technical aspects of intelligent buildings and pervasive computing research," *Intelligent Buildings International*, vol. 1, no. 1, pp. 56–74, 2009.

[59] M. Ball, V. Callaghan, M. Gardner, and D. Trossen, "Achieving human-agent teamwork in ehealth based pervasive intelligent environments," in *4th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pp. 1–8, IEEE, 2010.

[60] "Random house kernerman webster's college dictionary." http://www.kdictionaries-online.com.

[61] J. Hong and J. Landay, "An infrastructure approach to context-aware computing," *Human-Computer Interaction*, vol. 16, no. 2, pp. 287–303, 2001.

[62] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Modeling context information in pervasive computing systems," *Pervasive Computing*, pp. 79–117, 2002.

[63] A. Ranganathan, J. Al-Muhtadi, and R. Campbell, "Reasoning about uncertain contexts in pervasive computing environments," *IEEE Pervasive Computing*, pp. 62–70, 2004.

[64] H. Lei, D. Sow, J. Davis II, G. Banavar, and M. Ebling, "The design and applications of a context service," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 45–55, 2002.

[65] P. Gray and D. Salber, "Modelling and using sensed context information in the design of interactive applications," *Engineering for Human-Computer Interaction*, pp. 317–335, 2001.

[66] A. Artikis, M. Sergot, and G. Paliouras, "A logic programming approach to activity recognition," in *Proceedings of the 2nd ACM international workshop on Events in multimedia*, pp. 3–8, ACM, 2010.

[67] G. Antoniou, "Rule-based activity recognition in ambient intelligence," in *Proceedings of the 5th international conference on Rule-based reasoning, programming, and applications*, pp. 1–1, Springer-Verlag, 2011.

[68] W. Hu, D. Xie, T. Tan, and S. Maybank, "Learning activity patterns using fuzzy self-organizing neural network," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 34, no. 3, pp. 1618–1626, 2004.

[69] T. Van Kasteren, A. Noulas, G. Englebienne, and B. Kröse, "Accurate activity recognition in a home setting," in *Proceedings of the 10th international conference on ubiquitous computing*, pp. 1–9, ACM, 2008.

[70] D. J. Patterson, D. Fox, H. Kautz, and M. Philipose, "Fine-grained activity recognition by aggregating abstract object usage," in *Proceedings of the 9th IEEE International Symposium on Wearable Computers*, pp. 44–51, IEEE, 2005.

[71] D. L. Vail, M. M. Veloso, and J. D. Lafferty, "Conditional random fields for activity recognition," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, p. 235, ACM, 2007.

[72] T.-y. Wu, C.-c. Lian, and J. Y.-j. Hsu, "Joint recognition of multiple concurrent activities using factorial conditional random fields," in *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*, 2007.

[73] Y.-X. Hung, C.-Y. Chiang, S. J. Hsu, and C.-T. Chan, "Abnormality detection for improving elders daily life independent," in *Aging Friendly Technology for Health and Independence (ICOST)*, pp. 186–194, Springer, 2010.

[74] P. Rashidi, D. J. Cook, L. B. Holder, and M. Schmitter-Edgecombe, "Discovering activities to recognize and track in a smart environment," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 4, pp. 527–539, 2011.

[75] J. Loane, B. O'Mullane, B. Bortz, and R. B. Knapp, "Interpreting presence sensor data and looking for similarities between homes using cluster analysis," in *5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, pp. 438–445, IEEE, 2011.

[76] T. Gu, S. Chen, X. Tao, and J. Lu, "An unsupervised approach to activity recognition and segmentation based on object-use fingerprints," *Data & Knowledge Engineering*, vol. 69, no. 6, pp. 533–544, 2010.

[77] X. Wang, J. S. Dong, C. Chin, S. R. Hettiarachchi, and D. Zhang, "Semantic space: An infrastructure for smart spaces," *Computing*, vol. 1, no. 2, pp. 67–74, 2002.

[78] F. Razzak, "Semantic web technologies role in smart environments," in *On the Move to Meaningful Internet Systems: OTM 2012 Workshops*, pp. 54–58, Springer, 2012.

[79] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J. Burgelman, "Ambient intelligence: From vision to reality," *IST Advisory Group Draft Report, European Commission*, 2003.

[80] P. Carreira, V. Amaral, and B. Barroca, "The case for a systematic development of building automation systems," in *2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe)*, pp. 1–8, IEEE, 2011.

[81] C. Nugent, D. Finlay, R. Davies, H. Wang, H. Zheng, J. Hallberg, K. Synnes, and M. Mulvenna, "homeml–an open standard for the exchange of data within smart environments," *Pervasive Computing for Quality of Life Enhancement*, pp. 121–129, 2007.

[82] J. Hallberg, C. Nugent, R. Davies, K. Synnes, M. Donnelly, D. Finlay, and M. Mulvenna, "Homeruleml-a model for the exchange of decision support rules within smart environments," in *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*, pp. 513–520, IEEE, 2007.

[83] U. Akdemir, P. Turaga, and R. Chellappa, "An ontology based approach for activity recognition from video," in *Proceeding of the 16th ACM international conference on Multimedia*, pp. 709–712, ACM, 2008.

[84] M. Rodríguez, C. Curlango, and J. García-Vázquez, "An agent-based component for identifying elders' at-home risks through ontologies," in *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*, pp. 168–172, Springer, 2009.

[85] F. Arab, *Quelles ressources pour le sujet vieillissant? Les ontologies, une perspective pour la conception et l'évaluation des aides capacitantes*. PhD thesis, Institut Telecom - Université de Sherbrooke, 2010.

[86] F. Paganelli and D. Giuli, "An ontology-based context model for home health monitoring and alerting in chronic patient care networks," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 2, pp. 838–845, IEEE, 2007.

[87] O. Lassila and R. R. Swick, "Resource description framework (rdf) model and syntax specification." W3C Recommendation, http://www.w3.org/TR/PR-rdf-syntax/, 1999.

[88] T. R. Gruber *et al.*, "A translation approach to portable ontology specifications," *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[89] D. Brickley, R. V. Guha, and B. McBride, "Rdf vocabulary description language 1.0: Rdf schema." W3C Recommendation, http://www.w3.org/TR/rdf-schema/, 2004.

[90] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "Owl 2 web ontology language primer," *W3C recommendation, http://www.w3.org/TR/owl2-primer/*, vol. 27, pp. 1–123, 2009.

[91] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, "Ep-sparql: a unified language for event processing and stream reasoning," in *Proceedings of the 20th international conference on World wide web*, pp. 635–644, ACM, 2011.

[92] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus, "C-sparql: Sparql for continuous querying," in *Proceedings of the 18th international conference on World wide web*, pp. 1061–1062, ACM, 2009.

[93] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth, "A native and adaptive approach for unified processing of linked streams and linked data," in *The Semantic Web–ISWC 2011*, pp. 370–388, Springer, 2011.

[94] P. Nurmi and P. Floréen, "Reasoning in context-aware systems." Helsinki Institute for Information Technology, 2004.

[95] M. Luther, T. Liebig, S. Böhm, and O. Noppens, "Who the heck is the father of bob?," *The Semantic Web: Research and Applications*, pp. 66–80, 2009.

[96] J. W. Lloyd, "Practical advantages of declarative programming," in *Joint Conference on Declarative Programming, GULP-PRODE*, vol. 94, p. 94, 1994.

[97] G. Antoniou and F. Van Harmelen, *Semantic Web Primer*. the MIT Press, 2004.

[98] A. Shehzad, H. Q. Ngo, K. A. Pham, and S. Lee, "Formal modeling in context aware systems," in *Proceedings of the First International Workshop on Modeling and Retrieval of Context*, Citeseer, 2004.

[99] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung, "Ontology based context modeling and reasoning using owl," in *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pp. 18–22, IEEE, 2004.

[100] D. L. McGuinness, F. Van Harmelen, *et al.*, "Owl web ontology language overview," *W3C recommendation, http://www.w3.org/TR/owl-features/*, vol. 10, no. 2004-03, p. 10, 2004.

[101] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz, "Owl 2 web ontology language: Profiles," *W3C recommendation, http://www.w3.org/TR/owl2-profiles/*, vol. 27, p. 61, 2009.

[102] T. Berners-Lee and D. Connolly, "Notation3 (n3): a readable rdf syntax." W3C Team Submission, http://www.w3.org/TeamSubmission/n3, 2011.

[103] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, and J. Hendler, "N3logic: A logical framework for the world wide web," *Theory and Practice of Logic Programming*, vol. 8, no. 3, pp. 249–269, 2008.

[104] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, *et al.*, "Swrl: A semantic web rule language combining owl and ruleml," *W3C Member submission, http://www.w3.org/Submission/SWRL/*, vol. 21, p. 79, 2004.

[105] H. Boley, S. Tabet, and G. Wagner, "Design rationale of ruleml: A markup language for semantic web rules," in *International Semantic Web Working Symposium (SWWS)*, pp. 381–402, 2001.

[106] W. Dargie, *Context-aware computing and self-managing systems*. Chapman and Hall/CRC, 2010.

[107] H. S. Goldberg, M. Vashevko, A. Postilnik, K. Smith, N. Plaks, and B. M. Blumenfeld, "Evaluation of a commercial rule engine as a basis for a clinical decision support service," in *AMIA Annual Symposium Proceedings*, vol. 2006, p. 294, American Medical Informatics Association, 2006.

[108] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *Pervasive Computing, IEEE*, vol. 1, no. 4, pp. 74–83, 2002.

[109] H. L. Chen, *An intelligent broker architecture for pervasive context-aware systems*. PhD thesis, University of Maryland, Baltimore County, 2004.

[110] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *The Knowledge Engineering Review*, vol. 18, no. 03, pp. 197–207, 2003.

[111] D. Bottazzi, R. Montanari, and A. Toninelli, "Context-aware middleware for anytime, anywhere social networks," *Intelligent Systems, IEEE*, vol. 22, no. 5, pp. 23–32, 2007.

[112] B. Mrohs, M. Luther, R. Vaidya, M. Wagner, S. Steglich, W. Kellerer, and S. Arbanowski, "Owl-sf–a distributed semantic service framework," in *Proc. of the Workshop on Context Awareness for Proactive Systems (CAPS05), Helsinki*, pp. 67–77, Citeseer, 2005.

[113] G. Meditskos and N. Bassiliades, "Dlejena: A practical forward-chaining owl 2 rl reasoner combining jena and pellet," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, no. 1, pp. 89–94, 2010.

[114] G. Meditskos, S. Dasiopoulou, V. Efstathiou, and I. Kompatsiaris., "Ontology patterns for complex activity modelling," in *7th International Web Rule Symposium: Research Based and Industry Focused (RuleML)*, vol. 8035 of *LNCS*, pp. 144–157, Springer, 2013.

[115] H. Knublauch, J. Hendler, and K. Idehen, "Spin - overview and motivation." W3C member submission, http://www.w3.org/Submission/spin-sparql/, 2011.

[116] L. Chen, C. Nugent, and H. Wang, "A knowledge-driven approach to activity recognition in smart homes," *Knowledge and Data Engineering, IEEE Transactions on*, no. 99, pp. 1–1, 2011.

[117] V. Foo Siang Fook, S. C. Tay, M. Jayachandran, J. Biswas, and D. Zhang, "An ontology-based context model in monitoring and handling agitation behavior for persons with dementia," in *Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on*, pp. 5–pp, IEEE, 2006.

[118] A.-Y. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Möller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, and T. Weithöner, "Dig2. 0–towards a flexible interface for description logic reasoners," in *Proc. of the OWL Experiences and Directions Workshop at the ISWC*, vol. 6, 2006.

[119] T. Weithoner, T. Liebig, M. Luther, and S. Bohm, "Whats wrong with owl benchmarks?," in *Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2006)*, p. 101.

[120] M. Knorr, J. J. Alferes, and P. Hitzler, "Local closed world reasoning with description logics under the well-founded semantics," *Artificial Intelligence*, vol. 175, no. 9, pp. 1528–1554, 2011.

[121] C. Patel, J. Cimino, J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas, "Matching patient records to clinical trials using ontologies," in *The Semantic Web*, pp. 816–829, Springer, 2007.

[122] D. J. Plas, M. Verheijen, H. Zwaal, and M. Hutschemaekers, "Manipulating context information with swrl," tech. rep., Ericsson Telecommunicatie B.V., 2006.

[123] T. Berners-Lee, "Design issues: Linked data." http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[124] A. Caragliu, C. Del Bo, and P. Nijkamp, *Smart cities in Europe*. Vrije Universiteit, Faculty of Economics and Business Administration, 2009.

[125] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pp. 74–83, ACM, 2004.

[126] S. Singh and R. Karwayun, "A comparative study of inference engines," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pp. 53–57, IEEE, 2010.

[127] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[128] V. Haarslev and R. Möller, "Description of the racer system and its applications," *Description Logics*, vol. 49, 2001.

[129] J. De Roo, "Euler proof mechanism – eye." http://eulersharp.sourceforge.net/, 1999-2013.

[130] T. Osmun and J. De Roo, "Linear relationship benchmark." Results available at http://eulersharp.sourceforge.net/2003/03swap/dtb-2010.txt and sources at http://eulersharp.sourceforge.net/2009/12dtb/.

[131] C. Zins, "Conceptual approaches for defining data, information, and knowledge," *Journal of the American Society for Information Science and Technology*, vol. 58, no. 4, pp. 479–493, 2007.

[132] D. P. Wallace, *Knowledge management: Historical and cross-disciplinary themes*. Libraries unlimited, 2007.

[133] G. W. Leibniz, *La Monadologie*. 1714.

[134] T. Berners-Lee, "Semantic web application platform - swap." http://www.w3.org/2000/10/swap/, 2000.

[135] W. K. Edwards and R. E. Grinter, "At home with ubiquitous computing: seven challenges," in *Ubicomp 2001: Ubiquitous Computing*, pp. 256–272, Springer, 2001.

[136] Y. Liu, X. Zhang, J. S. Dong, Y. Liu, J. Sun, J. Biswas, and M. Mokhtari, "Formal analysis of pervasive computing systems," in *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*, pp. 169–178, IEEE, 2012.

[137] M. Arapinis, M. Calder, L. Dennis, M. Fisher, P. Gray, S. Konur, A. Miller, E. Ritter, M. Ryan, S. Schewe, *et al.*, "Towards the verification of pervasive systems," *Electronic Communications of the EASST*, vol. 22, 2010.

[138] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model checking*. MIT press, 1999.

[139] K. Du, D. Zhang, X. Zhou, and M. Hariz, "Handling conflicts of context-aware reminding system in sensorised home," *Cluster Computing*, vol. 14, no. 1, pp. 81–89, 2011.

[140] M. M. Kokar, C. J. Matheus, and K. Baclawski, "Ontology-based situation awareness," *Information fusion*, vol. 10, no. 1, pp. 83–98, 2009.

[141] A. Singh, D. Juneja, and A. Sharma, "A fuzzy integrated ontology model to manage uncertainty in semantic web: the fiom," *International Journal on Computer Science and Engineering (IJCSE)*, vol. 3, no. 3, pp. 1057–1062, 2011.

[142] G. Stoilos, G. B. Stamou, V. Tzouvaras, J. Z. Pan, and I. Horrocks, "Fuzzy owl: Uncertainty and the semantic web.," in *OWLED*, 2005.

[143] J. Z. Pan, G. Stoilos, G. Stamou, V. Tzouvaras, and I. Horrocks, "f-swrl: A fuzzy extension of swrl," in *Journal on Data Semantics VI*, pp. 28–46, Springer, 2006.

[144] T. Gu, H. K. Pung, D. Q. Zhang, H. K. Pung, and D. Q. Zhang, "A bayesian approach for dealing with uncertain contexts," in *Austrian Computer Society*, Citeseer, 2004.

[145] R. N. Carvalho, R. Haberlin, P. C. G. Costa, K. B. Laskey, and K. Chang, "Modeling a probabilistic ontology for maritime domain awareness," in *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, pp. 1–8, IEEE, 2011.

[146] A. Bellenger and S. Gatepaille, "Uncertainty in ontologies: Dempster-shafer theory for data fusion applications," *arXiv preprint arXiv:1106.3876*, 2011.

[147] H. Aloulou, *Framework for Ambient Assistive Living: Handling Dynamism and Uncertainty in Real Time Semantic Services Provisioning*. PhD thesis, Institut Mines-Télécom, EDITE, 2013.

[148] J. F. Sequeda and O. Corcho, "Linked stream data: A position paper," 2009.

[149] A. Rula, M. Palmonari, A. Harth, S. Stadtmüller, and A. Maurino, "On the diversity and availability of temporal information in linked open data," in *The Semantic Web–ISWC 2012*, pp. 492–507, Springer, 2012.

[150] N. Noy, A. Rector, P. Hayes, and C. Welty, "Defining n-ary relations on the semantic web." W3C Working Group Note, http://www.w3.org/TR/swbp-n-aryRelations/, 2006.

[151] G. Yang and M. Kifer, "Reasoning about anonymous resources and meta statements on the semantic web," in *Journal on Data Semantics I*, pp. 69–97, Springer, 2003.

[152] C. J. Matheus, "Position paper: Using ontology-based rules for situation awareness and information fusion.," in *Rule Languages for Interoperability*, 2005.

[153] S. Freud, "The ego and the id. standard edition, 19: 12-66," 1961.

[154] D. Westen, "The scientific status of unconscious processes: is freud really dead?," *Journal of the American Psychoanalytic Association*, vol. 47, no. 4, pp. 1061–1106, 1999.

[155] H. Aloulou, M. Mokhtari, T. Tiberghien, J. Biswas, and P. Yap, "Real world deployment of assistive living technologies for cognitively impaired people in singapore: Demonstration guidelines," in *IEEE Journal of Biomedical and Health Informatics (J-BHI)*, IEEE, 2013.

[156] T. Erl, "Introducing soa design patterns," *SOA World Magazine*, vol. 8, no. 6, pp. 2–7, 2008.

[157] O. Alliance, *Osgi service platform, release 3*. IOS Press, Inc., 2003.

[158] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S. Hallsteinsen, J. Lorenzo, A. Mamelli, and U. Scholz, "Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments," in *Software engineering for self-adaptive systems*, pp. 164–182, Springer, 2009.

[159] D.-M. Han and J.-H. Lim, "Design and implementation of smart home energy management systems based on zigbee," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 3, pp. 1417–1425, 2010.

[160] Z. Etzioni, J. Keeney, R. Brennan, and D. Lewis, "Supporting composite smart home services with semantic fault management," in *Future Information Technology (FutureTech), 2010 5th International Conference on*, pp. 1–8, IEEE, 2010.

[161] T. Perumal, A. R. Ramli, C. Y. Leong, S. Mansor, and K. Samsudin, "Interoperability among heterogeneous systems in smart home environment," in *Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on*, pp. 177–186, IEEE, 2008.

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

[163] xmpp.org, "Xmpp standards foundation." http://xmpp.org.

[164] P. Saint-Andre, "Streaming XML with Jabber/XMPP," *IEEE Internet Computing*, pp. 82–89, 2005.

[165] igniterealtime.org, "Openfire project under ignite realtime community." http://www.igniterealtime.org/projects/openfire/.

[166] R. Verborgh, "Node-n3: Lightning fast, asynchronous, streaming of turtle/n3/rdf." GitHub https://github.com/RubenVerborgh/node-n3, 2012.

[167] H. Aloulou, M. Mokhtari, T. Tiberghien, J. Biswas, and J. H. K. Lin, "A semantic plug&play based framework for ambient assisted living," in *Impact Analysis of solutions for chronic disease Prevention and Management*, Lecture Notes in Computer Science, Springer, 2012.

[168] jena.sourceforge.net, "Jena - a semantic web framework for java." http://jena.sourceforge.net/inference/.

[169] S. Solaimani, H. Bouwman, and N. Baken, "The smart home landscape: A qualitative meta-analysis," in *Toward Useful Services for Elderly and People with Disabilities* (B. Abdulrazak, S. Giroux, B. Bouchard, H. Pigot, and M. Mokhtari, eds.), vol. 6719 of *Lecture Notes in Computer Science*, pp. 192–199, Springer, 2011.

[170] H. Bouwman, E. Faber, T. Haaker, B. Kijl, and M. De Reuver, "Conceptualizing the stof model," in *Mobile service innovation and business models*, pp. 31–70, Springer, 2008.

[171] "Cookstop: The leader in kitchen fire prevention." http://www.cookstop.com/.

[172] T. Fujinami, M. Miura, R. Takatsuka, and T. Sugihara, "A study of long term tendencies in residents' activities of daily living at a group home for people with dementia using rfid slippers," in *Toward Useful Services for Elderly and People with Disabilities* (B. Abdulrazak, S. Giroux, B. Bouchard, H. Pigot, and M. Mokhtari, eds.), vol. 6719 of *Lecture Notes in Computer Science*, pp. 303–307, Springer, 2011.

[173] C. Bothorel, C. Lohr, A. Thépaut, F. Bonnaud, and G. Cabasse, "From individual communication to social networks: Evolution of a technical platform for the elderly," in *Toward Useful Services for Elderly and People with Disabilities* (B. Abdulrazak, S. Giroux, B. Bouchard, H. Pigot, and M. Mokhtari, eds.), vol. 6719 of *Lecture Notes in Computer Science*, pp. 145–152, Springer, 2011.

[174] K. De Voegt, M. Feki, L. Claeys, J. Criel, P. Zontrop, M. Godon, M. Roelands, M. Geerts, and L. Trappeniers, "Augmented photoframe for interactive smart space," *Aging Friendly Technology for Health and Independence*, pp. 60–66, 2010.

[175] "Karotz: Your smart rabbit." http://www.karotz.com/.

[176] M. Alwan, "Eldercare technology transfer," in *ICOST 2011 Summer School Talks*, 2011.

[177] R. Mendonca and R. Smith, "Effects of providing medical product accessibility information to individuals with disabilities," *Archives of Physical Medicine and Rehabilitation*, vol. 91, no. 10, pp. e36–e37, 2010.

[178] C. Gervasoni and W. Benzarti, "Create, un campus innovant pour l'excellence en recherche et les entreprises technologiques." http://www.ambafrance-sg.org, 2011.

List of Figures

List of Tables

Acronyms

3C Callaghan-Clarke-Chin. 21

A*STAR Agency for Science, Technology and Research. 199

AAL Ambient Assisted Living. i, ii, 9, 10, 15–20, 23, 25, 47–51, 66, 76, 81, 102, 122, 127, 130, 135, 136, 193, 198

ABox Assertional Box. 31, 32, 34–36, 105

AcRF Academic Research Fund. 199

ADL Activities of Daily Living. 5–7, 9, 16, 19, 30, 42, 119, 126

AGIM Age, Imagerie, Modélisation. 125

AmI Ambient Intelligence. 14, 15, 29, 32, 42, 46, 47, 55, 61, 84

AMUPADH Activity Monitoring and UI Plasticity for supporting Ageing with mild Dementia at Home. 115, 118

API Application Programming Interface. 50, 83, 84, 89, 90, 95, 105

C-SPARQL Continuous SPARQL. 38

CAST LeadingAge Center for Aging Services Technologies. 10

CoBrA Context Broker Architecture. 46

CQELS Continuous Query Evaluation over Linked Streams. 38

CRC Cyclic Redundancy Check. 94

CRF Conditional Random Fields. 23

CRUD Create Read Update Delete. 95

CWA Closed World Assumption. 48, 49

cwm closed-world machine. 45, 52

DBN Dynamic Bayesian Network. 16, 85

DIG DL Implementation Group. 47, 51

DIKW "Data-Information-Knowledge-Wisdom". 53, 56, 95

DL Description Logic. 43, 46–48, 51–53, 55, 94

DPWS Devices Profile for Web Services. 102, 104

DSL Domain Specific Languages. 23, 42, 46, 47

EDB Economic Development Board. 199

EP-SPARQL Event Processing SPARQL. 38

EYE Euler YAP Engine. 45, 49, 51, 52, 57, 58, 71, 73, 74, 76, 94, 95, 98, 100, 101, 105, 110–112, 135, 136

FAM Fuzzy Activation Map. 64

FAME Fuzzy Activation Map Engine. 64, 136

FOAF friend-of-a-friend. 46

GDP Gross Domestic Product. 201

GDS Global Deterioration Scale. 8, 119, 143

GUI Graphical User Interface. 14

HCI Human Computer Interaction. 14, 15

HLM Higher Level Modelling. 29

HMM Hidden Markov Models. 23, 24, 66

HTTP Hypertext Transfer Protocol. 97, 102

I/O inputs/outputs. 75, 76, 108

I2R Institute for Infocomm Research. 115, 116, 195

iAAL independent Ambient Assisted Living. 193, 195–198

ICT Information & Communications Technology. 9, 10, 13, 125, 126, 193

iFAM instant Fuzzy Activation Map. 64, 136

IM Instant Messaging. 88

IoT Internet of Things. 13, 14

IPAL Image & Pervasive Access Laboratory. 115, 125

IRI Internationalized Resource Identifier. 151

IT Information Technology. 13

JSON JavaScript Object Notation. 85

KB Knowledge Base. 48, 49, 59, 71–75, 81, 84, 85, 87–90, 95, 98, 101, 103, 104, 112

KTPH Khoo Teck Puat Hospital. 118

LLM Lower Level Modelling. 29

LTL Linear Temporal Logic. 59

M2M Machine to Machine Communication. 13

MINDEF Ministry of Defence. 199

MOE Ministry of Education. 199

MOH Ministry of Health. 199

MTI Ministry of Trade & Industry. 199

MUC Multi-User Chat. 88, 89, 102

N3 Notation3. 31–33, 35, 44, 45, 51, 52, 55, 57, 58, 67, 68, 73, 74, 76, 95, 98, 100, 101, 104, 135, 149

N3Logic Notation3 Logic. 45

NAF Negation As Failure. 48

NMRC National Medical Research Council. 199

NRF National Research Foundation. 199

NUS National University of Singapore. 115, 118

OECD Organisation for Economic Co-Operation and Development. 4

OSGi Open Service Gateway initiative. 73, 74, 82–84, 87–89, 97, 100, 102, 103, 105, 108, 110

OWA Open World Assumption. 48, 49

OWL Web Ontology Language. 32, 43–47, 51, 52, 89, 94, 151

POJI Plain Old Java Interfaces. 83

POJO Plain Old Java Objects. 83

QoI Quality of Information. 22, 24, 26, 63, 66–69

QoL Quality of Life. 16, 125, 136

QoS Quality of Service. 82

R&D Research & Development. 200, 201

RBC Rule-Based Clustering. 64

RBCS Rule-Based Confidence Score. 57, 58

RDF Resource Description Framework. 31, 32, 38, 43–47, 49, 51, 67, 87, 89, 98, 100, 149

RDF-S RDF Schema. 32, 43

REST Representational State Transfer. 84, 85, 95, 97, 194

RFID Radio-Frequency Identification. 117, 121

RIEC Research, Innovation and Enterprise Council. 199, 201

RuleML Rule Markup Language. 46

S2C Smart Space Composer. 77, 94

SERC Science & Engineering Research Council. 115, 116

SNAF Scoped Negation As Failure. 49, 51

SOA Service Oriented Architecture. 81, 82, 84, 85, 89, 102

SOAP Simple Object Access Protocol. 104

SPARQL SPARQL Protocol And RDF Query Language. 38, 47, 49, 50, 89

SQL Structured Query Language. 42, 43

STOF Service, Technology, Organisation, Finance. 193

SVM Support Vector Machines. 23, 112

SWAP Semantic Web Application Platform. 58

SWRL Semantic Web Rule Language. 46, 49, 51

TBox Terminological Box. 31, 32, 35, 36, 67, 76, 104

TCP/IP Transmission Control Protocol - Internet Protocol. 85

UbiSMART Ubiquitous Service MAnagement & Reasoning archiTecture. 73, 74, 76, 77, 81, 85, 89, 93–95, 98, 101, 102, 105, 112, 115, 116, 120, 127, 135, 136, 155

UI User Interface. 87, 89, 98

UIP User Interface Plasticity. 87

URI Uniform Resource Identifier. 32, 89, 93, 105, 108, 149, 151

W3C World Wide Web Consortium. 38, 43–45, 52, 58, 151

Web World Wide Web. 13, 48

wFAM windowed Fuzzy Activation Map. 64, 136

WS Web Services. 104

WSDL Web Services Description Language. 104

WSN Wireless Sensor Network. 77, 93, 126

XML Extensible Markup Language. 31, 44, 46, 149

XMPP Extensible Messaging and Presence Protocol. 85, 88, 89, 97, 102, 117, 121

YAP Yet Another Prolog. 45, 52

Index